Section 3

CSCI E-22

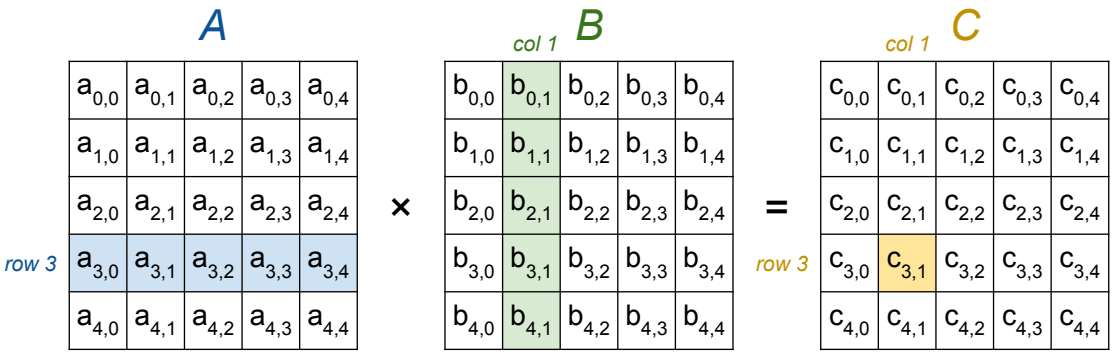
Will Begin Shortly

Big O notation

- Constant multipliers do not “affect” big O
  - $600n^2$ and $0.00005n^2$ are both $O(n^2)$
- Lower order terms do not “affect” big O
  - $2^n + n^{100}$ is still $O(2^n)$
- General ordering of functions, from slowest-growing to fastest:
  - $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^k) < O(c^n) < O(n!)$
    $c > 1$, $k > 2$ constant

Matrix multiplication

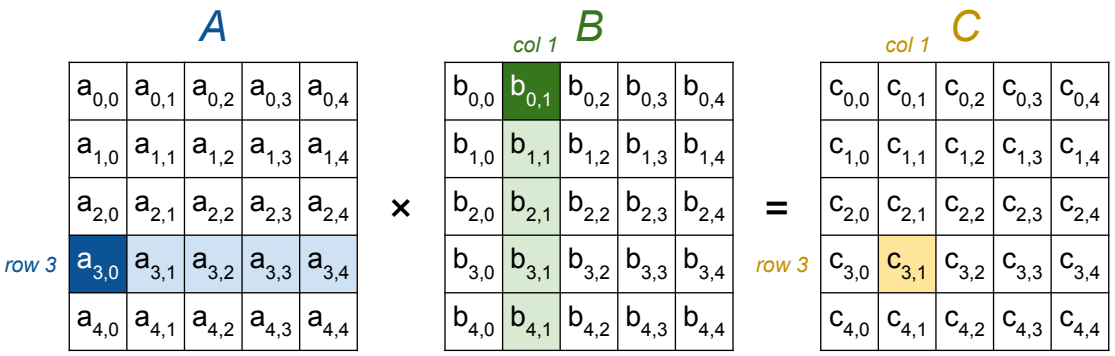
- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



$$c_{3,1} = a_{3,0} \times b_{0,1} + a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} + a_{3,3} \times b_{3,1} + a_{3,4} \times b_{4,1}$$

Matrix multiplication

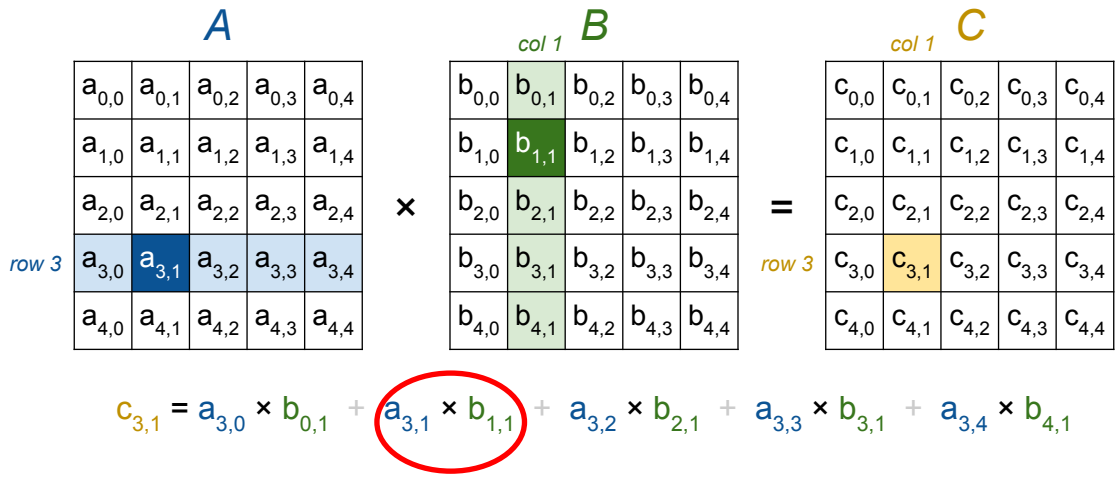
- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



$$c_{3,1} = a_{3,0} \times b_{0,1} + a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} + a_{3,3} \times b_{3,1} + a_{3,4} \times b_{4,1}$$

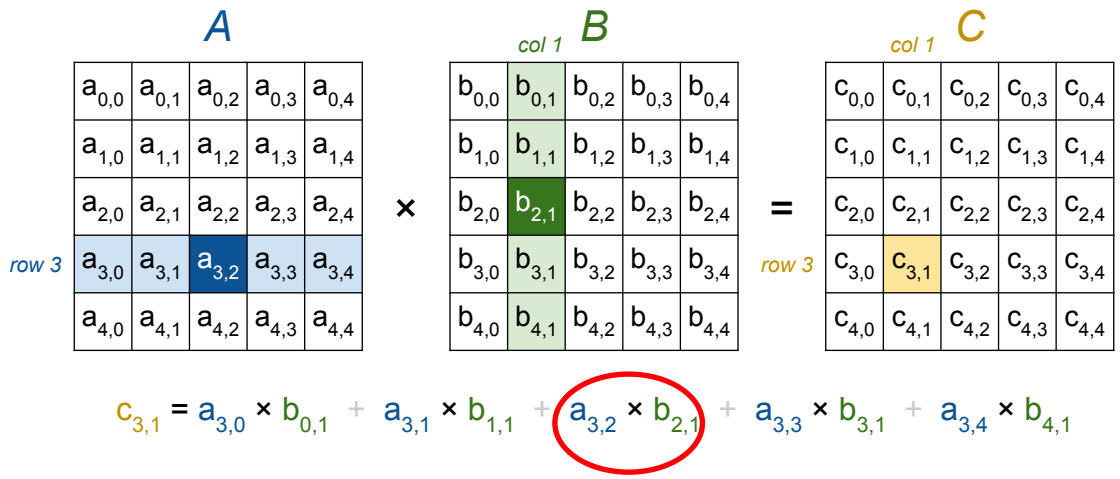
Matrix multiplication

- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



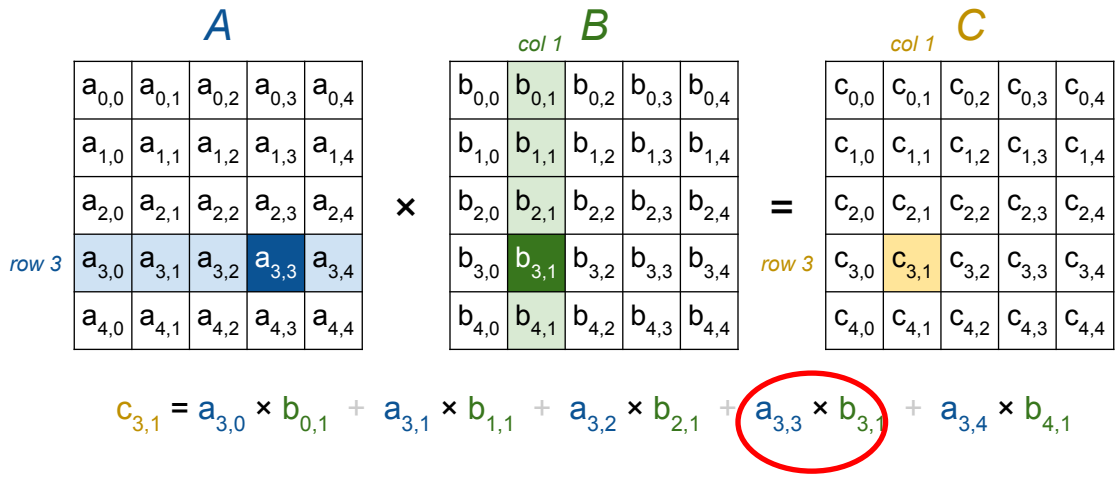
Matrix multiplication

- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



Matrix multiplication

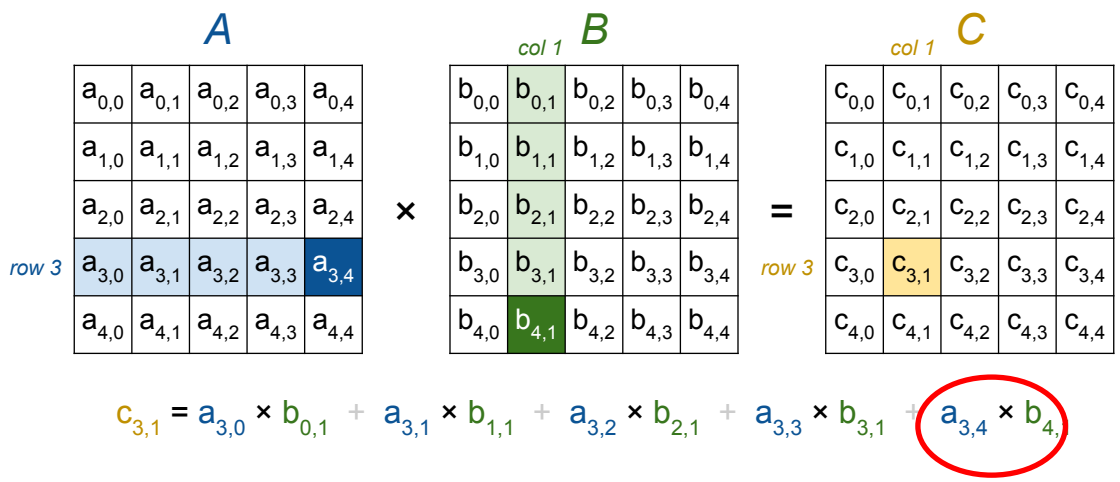
- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



$$c_{3,1} = a_{3,0} \times b_{0,1} + a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} + a_{3,3} \times b_{3,1} + a_{3,4} \times b_{4,1}$$

Matrix multiplication

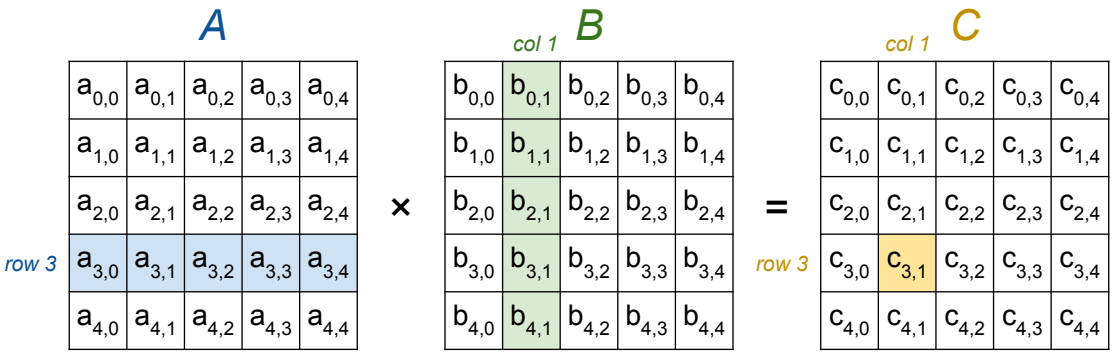
- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



$$c_{3,1} = a_{3,0} \times b_{0,1} + a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} + a_{3,3} \times b_{3,1} + a_{3,4} \times b_{4,1}$$

Matrix multiplication

- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:



individual multiplications are added together

$$c_{3,1} = a_{3,0} \times b_{0,1} + a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} + a_{3,3} \times b_{3,1} + a_{3,4} \times b_{4,1}$$

Matrix multiplication

- For two $n \times n$ matrices A and B, the product C is another $n \times n$ matrix where each element is obtained by multiplying elements of A row-wise with B column-wise, adding up the individual products:

$c_{0,0}$	$c_{0,1}$	$c_{0,2}$	...	$c_{0,n-1}$
$c_{1,0}$	$c_{1,1}$	$c_{1,2}$	...	$c_{1,n-1}$
$c_{2,0}$	$c_{2,1}$	$c_{2,2}$	...	$c_{2,n-1}$
⋮	⋮	⋮	$c_{i,j}$	⋮
$c_{n-1,0}$	$c_{n-1,1}$	$c_{n-1,2}$	...	$c_{n-1,n-1}$

$$c_{i,j} = a_{i,0} \times b_{0,j} + a_{i,1} \times b_{1,j} + a_{i,2} \times b_{2,j} + \ldots + a_{i,n-1} \times b_{n-1,j}$$

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \times b_{k,j}$$

row *col*

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```


Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:


```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

outer i-loop gets values 0, 1, …, n - 1

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

outer `i`-loop gets values 0, 1, …, n - 1

for each `i`, inner `j`-loop gets values 0, 1, …, n - 1

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

outer `i`-loop gets values 0, 1, …, n - 1

for each `i`, inner `j`-loop gets values 0, 1, …, n - 1

this code runs for every $c_{i,j}$ matrix element

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

how many multiplications are done for each element $c_{i,j}$?

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

how many multiplications are done for each element $c_{i,j}$?
***n** multiplications*

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

how many multiplications are done for each element $c_{i,j}$?
***n** multiplications*

overall, how many multiplications are required to compute the final result?

Matrix multiplication

- In Java, if we represent a matrix as a two-dimensional array, we can determine the entries of the matrix *AB* using nested for loops in the following way:

```
int[][] matrixA = new int[n][n];     // A is an n-by-n matrix
int[][] matrixB = new int[n][n];     // B is another n-by-n matrix
int[][] matrixC = new int[n][n];     // C will hold the matrix product A * B

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        int total = 0;

        for (int k = 0; k < n; k++) {
            total += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = total;
    }
}
```

how many multiplications are done for each element $c_{i,j}$?
***n** multiplications*

overall, how many multiplications are required to compute the final result?
n×n elmnts × n multiplications =
n × n × n = n^3 multiplications

General case of nested loops

- Nested loops, both dependent on **n**: statements inside the inner loop will run exactly **n** × **n** times:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // statements here are executed n^2 times
    }
}
```

General case of nested loops

- Nested loops, both dependent on **n**: statements inside the inner loop will run exactly **n** × **n** times:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // statements here are executed n^2 times
    }
}
```

- Three nested loops, all dependent on **n**: statements inside the innermost loop will run exactly **n** × **n** × **n** times:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            // statements here are executed n^3 times
        }
    }
}
```

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop runs a constant number of times (here, only 3 times)?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 3; j++) {
        // statements here are executed ??? times
    }
}
```

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop runs a constant number of times (here, only 3 times)?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 3; j++) {
        // statements here are executed 3n times
    }
}
```

n iterations of outer loop × 3 iterations of inner loop = 3**n**

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop runs a constant number of times (here, only 3 times)?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 3; j++) {
        // statements here are executed 3n times
    }
}
```

n iterations of outer loop × 3 iterations of inner loop = 3**n** = still O(*n*)

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

General case of nested loops

- What if the outer loop is dependent on **`n`** and the inner loop is dependent on the current value of **`i`**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

0 < 0, false

when `i` = 0 and `j` = 0, the inner loop does not run. no statements are executed

i	j	exec?
0	0	???

General case of nested loops

- What if the outer loop is dependent on **`n`** and the inner loop is dependent on the current value of **`i`**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

when `i` = 0 and `j` = 0, the inner loop does not run. no statements are executed

i	j	exec?
0	0	✗

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

i	j	exec?
0	0	❌
1	0	???

when i = 0, statements are executed 0 times
when i = 1, statements are executed ??? times

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

when i = 1, the inner loop is allowed to run once

0 < 1, true

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

i	j	exec?	
0	0	❌	
1	0	✔	*1 time*
1	1	❌	
2	0	???	

when i = 0, statements are executed 0 times
when i = 1, statements are executed 1 time
when i = 2, statements are executed ??? times

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

when i = 0, statements are executed 0 times
when i = 1, statements are executed 1 time
when i = 2, statements are executed 2 times
when i = 3, statements are executed 3 times

i	j	exec?	
0	0	✗	
1	0	✓	1 time
1	1	✗	
2	0	✓	2 times
2	1	✓	
2	2	✗	
3	0	✓	3 times
3	1	✓	
3	2	✓	
3	3	✗	
4	...	...	

General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

when i = 0, statements are executed 0 times
when i = 1, statements are executed 1 time
when i = 2, statements are executed 2 times
when i = 3, statements are executed 3 times
when i = 4, statements are executed 4 times
...

how many total executions for a given value of n?

i	j	exec?	
0	0	✗	
1	0	✓	1 time
1	1	✗	
2	0	✓	2 times
2	1	✓	
2	2	✗	
3	0	✓	3 times
3	1	✓	
3	2	✓	
3	3	✗	
4	0	✓	4 times
4	1	✓	
4	2	✓	
4	3	✓	
4	4	✗	
5	0	✓	
...	...	...	

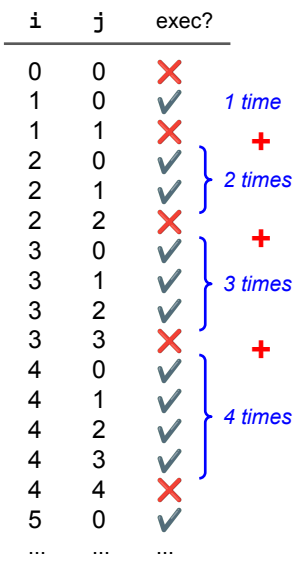
General case of nested loops

- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

*when **i** = 0, statements are executed 0 times*
*when **i** = 1, statements are executed 1 time*
*when **i** = 2, statements are executed 2 times*
*when **i** = 3, statements are executed 3 times*
*when **i** = 4, statements are executed 4 times*
...

how many total executions for a given value of n?
1 + 2 + 3 + 4 + ... + n - 1



General case of nested loops

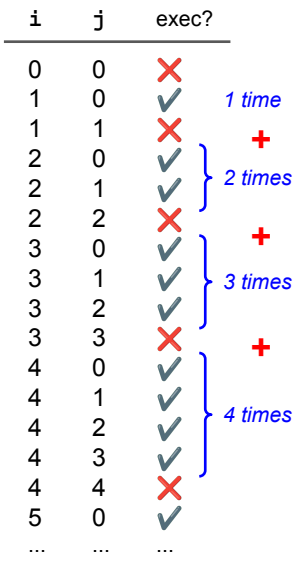
- What if the outer loop is dependent on **n** and the inner loop is dependent on the current value of **i**?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // statements here are executed ??? times
    }
}
```

*when **i** = 0, statements are executed 0 times*
*when **i** = 1, statements are executed 1 time*
*when **i** = 2, statements are executed 2 times*
*when **i** = 3, statements are executed 3 times*
*when **i** = 4, statements are executed 4 times*
...

how many total executions for a given value of n?

*1 + 2 + 3 + 4 + ... + n - 1 = **n(n - 1) / 2***



n^2 versus $n(n - 1) / 2$

Both loops dependent on *n*

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed n times*
when `i` *= 1, statements are executed n times*
when `i` *= 2, statements are executed n times*
when `i` *= 3, statements are executed n times*
...
when `i` *= n - 1, statements are excd n times*

$n \times n = n^2$ $O(n^2)$

Outer loop dependent on *n*, inner loop dependent on current value of `i`

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed 0 times*
when `i` *= 1, statements are executed 1 times*
when `i` *= 2, statements are executed 2 times*
when `i` *= 3, statements are executed 3 times*
...
when `i` *= n - 1, statements are excd n - 1 times*

$n(n - 1) / 2$

n^2 versus $n(n - 1) / 2$

Both loops dependent on *n*

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed n times*
when `i` *= 1, statements are executed n times*
when `i` *= 2, statements are executed n times*
when `i` *= 3, statements are executed n times*
...
when `i` *= n - 1, statements are excd n times*

$n \times n = n^2$ $O(n^2)$

Outer loop dependent on *n*, inner loop dependent on current value of `i`

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed 0 times*
when `i` *= 1, statements are executed 1 times*
when `i` *= 2, statements are executed 2 times*
when `i` *= 3, statements are executed 3 times*
...
when `i` *= n - 1, statements are excd n - 1 times*

$n(n - 1) / 2$
$n^2 / 2 - n / 2$

n^2 versus $n(n - 1) / 2$

Both loops dependent on *n*

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed n times*
when `i` *= 1, statements are executed n times*
when `i` *= 2, statements are executed n times*
when `i` *= 3, statements are executed n times*
...
when `i` *= n - 1, statements are excd n times*

$n \times n = n^2$ $O(n^2)$

Outer loop dependent on *n*, inner loop dependent on current value of `i`

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // ...
    }
}
```

when `i` *= 0, statements are executed 0 times*
when `i` *= 1, statements are executed 1 times*
when `i` *= 2, statements are executed 2 times*
when `i` *= 3, statements are executed 3 times*
...
when `i` *= n - 1, statements are excd n - 1 times*

$n(n - 1) / 2$
$n^2 / 2 - n / 2$
$\frac{1}{2}n^2 - \frac{1}{2}n$

also $O(n^2)$

Sorting algorithms

Selection sort
Insertion sort
Quicksort

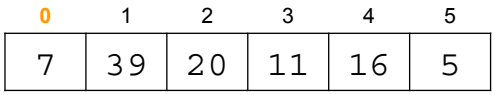
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

0	1	2	3	4	5
7	39	20	11	16	5

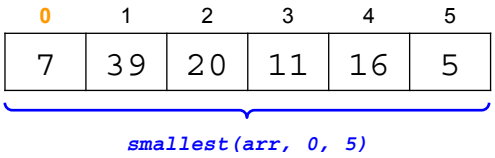
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

0	1	2	3	4	5
7	39	20	11	16	5

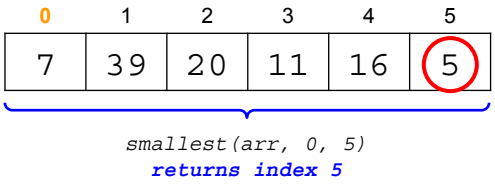
```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```




```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

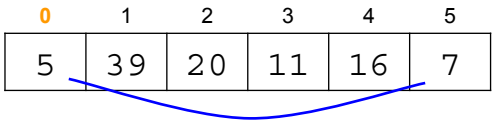



```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



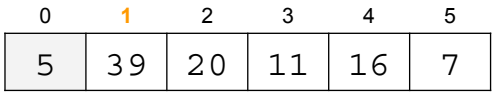
smallest(arr, 0, 5)
returns index 5

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

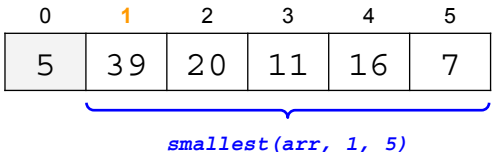


swap(arr, 0, 5)

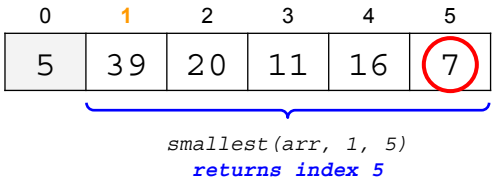
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



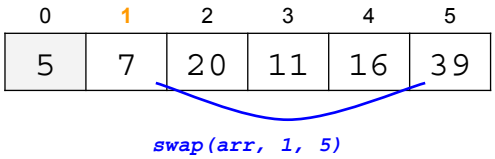

```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



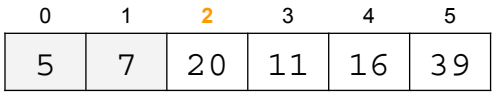
```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



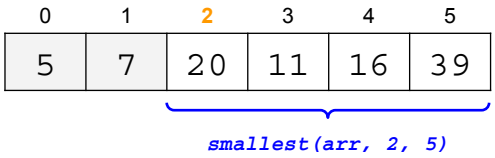

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



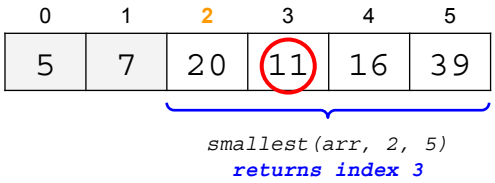
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



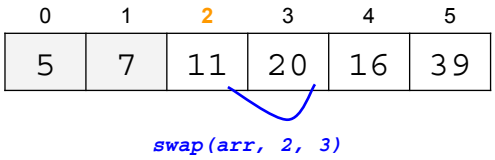

```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

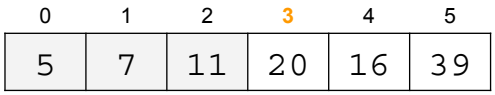
```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



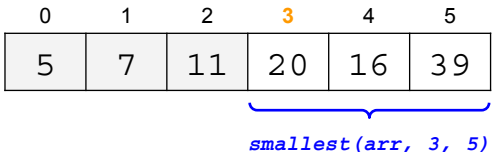

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



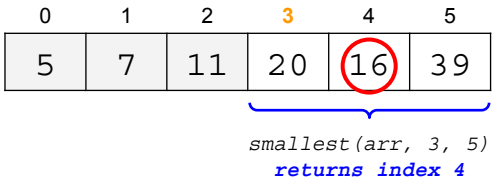
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



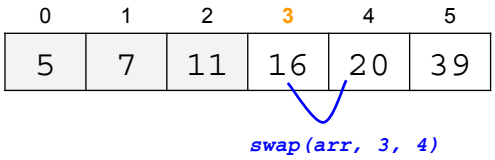

```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



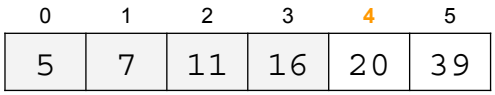
```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



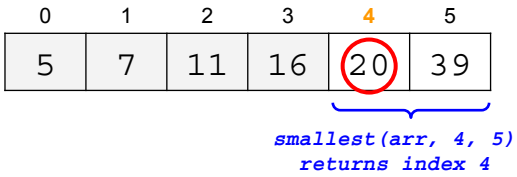

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



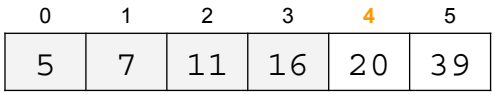
```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



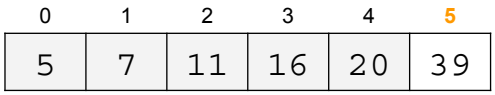

```java
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```




```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```



5 < 5, false

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

0	1	2	3	4	5
5	7	11	16	20	39

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

0	1	2	3	4	5
7	39	20	11	16	5
5	39	20	11	16	**7**
5	**7**	20	11	16	**39**
5	7	**11**	**20**	16	39
5	7	11	**16**	**20**	39
5	7	11	16	**20**	39

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	???	???
	O(???)	O(???)
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n(n - 1) / 2$	???
	O(???)	O(???)
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n(n - 1) / 2$	???
	$O(n^2)$	O(???)
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n(n - 1) / 2$	$3(n - 1)$
	$O(n^2)$	O(???)
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n(n - 1) / 2$ $O(n^2)$	$3(n - 1)$ $O(n)$
worst case	O(???)	O(???)
average case	O(???)	O(???)


```
private static int smallest(
  int[] arr, int lower, int upper
) {
    int indexMin = lower;
    for (int i = lower + 1; i <= upper; i++) {
        if (arr[i] < arr[indexMin]) {
            indexMin = i;
        }
    }

    return indexMin;
}

public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int j = smallest(arr, i, arr.length - 1);
        swap(arr, i, j);
    }
}
```


	comparisons	moves (swap is 3 moves)
best case	$n(n - 1) / 2$ $O(n^2)$	$3(n - 1)$ $O(n)$
worst case	$O(n^2)$	$O(n)$
average case	$O(n^2)$	$O(n)$

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 1 toInsert

j

0	1	2	3	4	5
7	39	20	11	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

arr[1] < arr[0]

i 1 toInsert

j

0	1	2	3	4	5
7	39	20	11	16	5

arr[1] < arr[0]?

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

arr[1] < arr[0]

i 1 toInsert

j

0	1	2	3	4	5
7	39	20	11	16	5

39 < 7?

any moves necessary to put 39 into sorted order with the already sorted subarray {7}?

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

arr[1] < arr[0]

39 < 7

false

i 1 toInsert

j

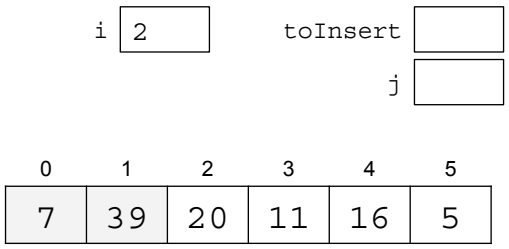
0	1	2	3	4	5
7	39	20	11	16	5

39 < 7?

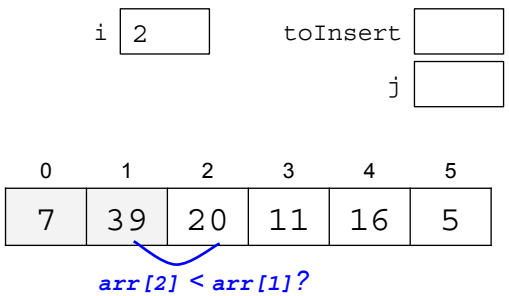
any moves necessary to put 39 into sorted order with the already sorted subarray {7}?

no moves necessary

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```



```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

arr[2] < arr[1]
20 < 39
true

i 2 toInsert
j

0	1	2	3	4	5
7	39	20	11	16	5

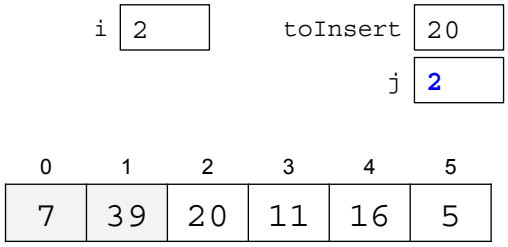
arr[2] < arr[1]?
20 < 39
need to move 20 left into the sorted subarray (need to end up with {7, 20, 39})

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

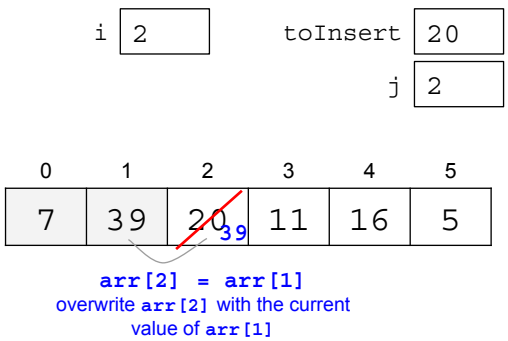
i 2 toInsert **20**
j

0	1	2	3	4	5
7	39	20	11	16	5

```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```



```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 2 toInsert 20

j 2

0	1	2	3	4	5
7	39	**39**	11	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 2 toInsert 20

j **1**

0	1	2	3	4	5
7	39	39	11	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

1 > 0, true

i 2 toInsert 20

j 1

0	1	2	3	4	5
7	39	39	11	16	5


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

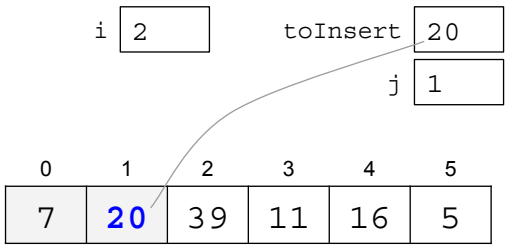

toInsert < arr[j - 1]
20 < 7
false

i 2 toInsert 20

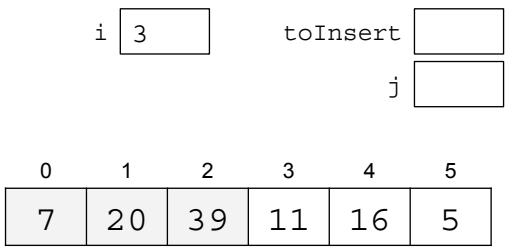
j 1

0	1	2	3	4	5
7	39	39	11	16	5

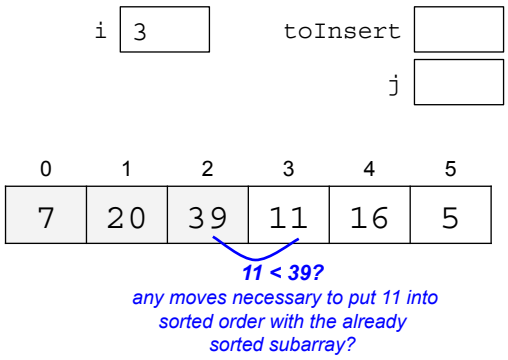
```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

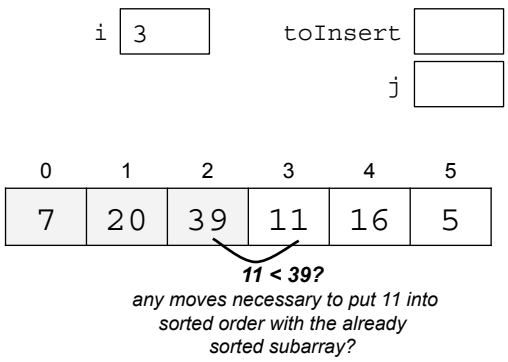


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

arr[3] < arr[2]
11 < 39
true



```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 3 toInsert **11**

j **3**

0	1	2	3	4	5
7	20	39	11	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 3 toInsert 11

j 3

0	1	2	3	4	5
7	20	39	**39**	16	5

arr[3] = arr[2]

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 3 toInsert 11

j 2

0	1	2	3	4	5
7	20	39	39	16	5


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


2 > 0, *true*

i 3 toInsert 11

j 2

0	1	2	3	4	5
7	20	39	39	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

toInsert < arr[j - 1]
11 < 20
true

i 3 toInsert 11
j 2

0	1	2	3	4	5
7	20	39	39	16	5


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


i 3 toInsert 11
j 2

0	1	2	3	4	5
7	20	**20**	39	16	5

arr[2] = arr[1]

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 3 toInsert 11

j 1

0	1	2	3	4	5
7	20	20	39	16	5


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


1 > 0, true

i 3 toInsert 11

j 1

0	1	2	3	4	5
7	20	20	39	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

toInsert < arr[j - 1]
11 < 7
false

i 3 toInsert 11
j 1

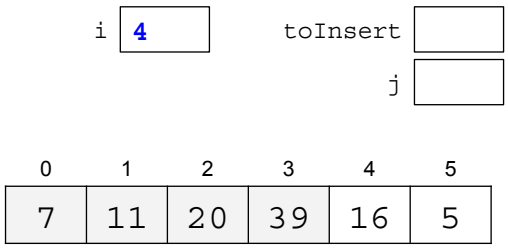
0	1	2	3	4	5
7	20	20	39	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

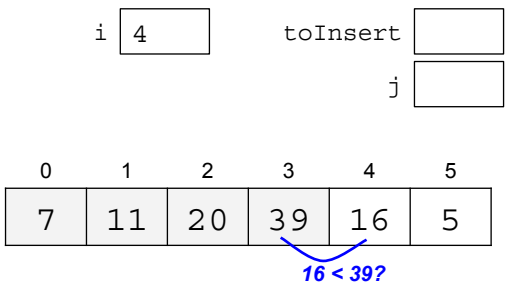
i 3 toInsert 11
j 1

0	1	2	3	4	5
7	**11**	20	39	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```



```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 4 toInsert 16

j 4

0	1	2	3	4	5
7	11	20	39	16	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 4 toInsert 16

j 3

0	1	2	3	4	5
7	11	20	39	**39**	5

arr[4] = arr[3]

```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 4 toInsert 16

j 3

0	1	2	3	4	5
7	11	20	39	39	5

toInsert < arr[j - 1]
16 < 20
true


```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


i 4 toInsert 16

j 2

0	1	2	3	4	5
7	11	20	**20**	39	5

arr[3] = arr[2]

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 4 toInsert 16

j 2

0	1	2	3	4	5
7	11	20	20	39	5

toInsert < arr[j - 1]

16 < 11

false

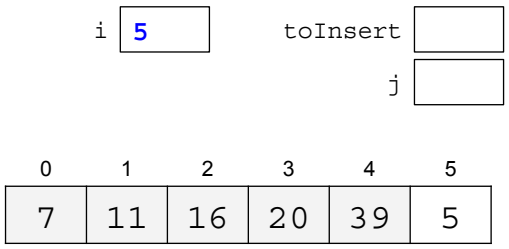

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


i 4 toInsert 16

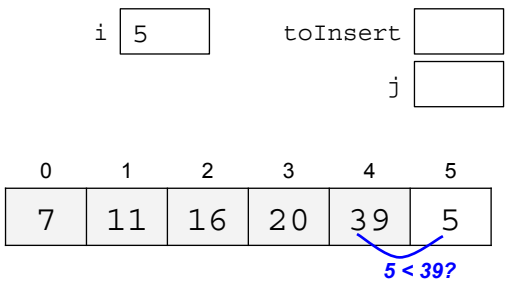
j 2

0	1	2	3	4	5
7	11	**16**	20	39	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```




```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```



```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 5

0	1	2	3	4	5
7	11	16	20	39	5

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 4

0	1	2	3	4	5
7	11	16	20	39	39

arr[5] = arr[4]

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 4

0	1	2	3	4	5
7	11	16	20	39	39

toInsert < arr[j - 1]

5 < 20

true

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 3

0	1	2	3	4	5
7	11	16	20	**20**	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 3

0	1	2	3	4	5
7	11	16	20	20	39

toInsert < arr[j - 1]

5 < 16

true

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j **2**

0	1	2	3	4	5
7	11	16	**16**	20	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 2

0	1	2	3	4	5
7	11	16	16	20	39

toInsert < arr[j - 1]

5 < 11

true

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 5 toInsert 5

j 1

0	1	2	3	4	5
7	11	**11**	16	20	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i	5

toInsert	5

j	1

0	1	2	3	4	5
7	11	11	16	20	39

toInsert < arr[j - 1]
5 < 7
true

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i	5

toInsert	5

j	0

0	1	2	3	4	5
7	7	11	16	20	39

```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

j > 0
0 > 0
false

i	5

toInsert	5

j	0

0	1	2	3	4	5
7	7	11	16	20	39

```java
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i	5

toInsert	5

j	0

0	1	2	3	4	5
5	7	11	16	20	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i 6 toInsert

j

0	1	2	3	4	5
5	7	11	16	20	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i toInsert

j

0	1	2	3	4	5
5	7	11	16	20	39

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

i ☐ toInsert ☐

j ☐

0	1	2	3	4	5
7	39	20	11	16	5
7	**20**	**39**	11	16	5
7	**11**	**20**	**39**	16	5
7	11	**16**	**20**	**39**	5
5	**7**	**11**	**16**	**20**	**39**


```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```


	comparisons	moves (swap is 3 moves)
best case	???	???
	O(???)	
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n - 1$ $O(n)$	0
worst case	O(???)	O(???)
average case	O(???)	O(???)

```
public static void insertionSort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            int toInsert = arr[i];
            int j = i;

            do {
                arr[j] = arr[j - 1];
                j = j - 1;
            } while (
                j > 0 &&
                toInsert < arr[j - 1]
            );

            arr[j] = toInsert;
        }
    }
}
```

	comparisons	moves (swap is 3 moves)
best case	$n - 1$ $O(n)$	0
worst case	$O(n^2)$	$O(n^2)$
average case	$O(n^2)$	$O(n^2)$

Quicksort

- Recursive, divide & conquer
- Achieves better average-case time complexity than O(n^2)
- Elements are partitioned into two *subarrays*
  - Elements in left subarray are *less than or equal to* elements in right subarray
- To partition, pick a pivot value, then repeatedly swap elements between subarrays to satisfy above condition
- After partitioning is done, recursively quicksort the subarrays
- Say we want to call `partition()` on the array below. What would be the pivot value?

7	39	20	11	16	5

Quicksort

- Recursive, divide & conquer
- Achieves better average-case time complexity than O(n^2)
- Elements are partitioned into two *subarrays*
  - Elements in left subarray are *less than or equal to* elements in right subarray
- To partition, pick a pivot value, then repeatedly swap elements between subarrays to satisfy above condition
- After partitioning is done, recursively quicksort the subarrays
- Say we want to call `partition()` on the array below. What would be the pivot value?

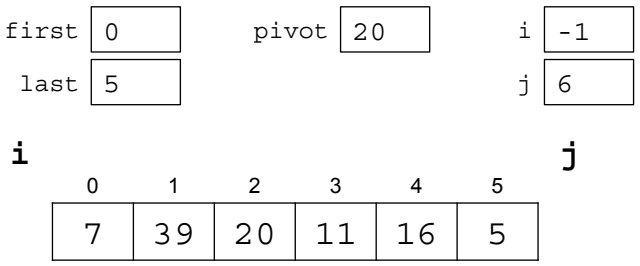
`arr[(first + last)/2]`

7	39	**20**	11	16	5

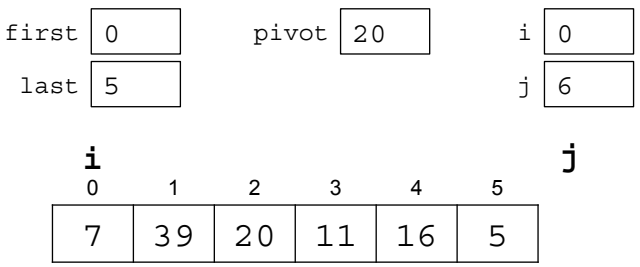
`first` 0

`last` 5

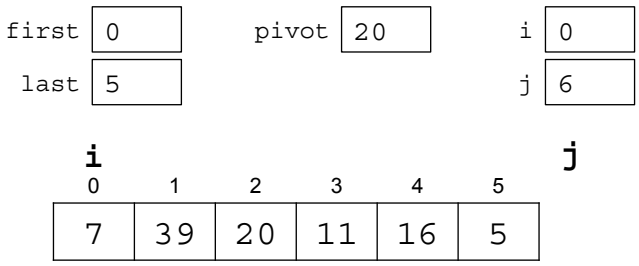
```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



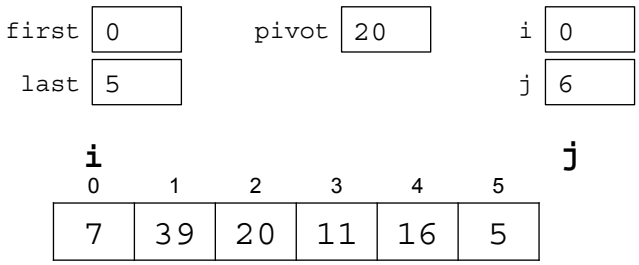

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



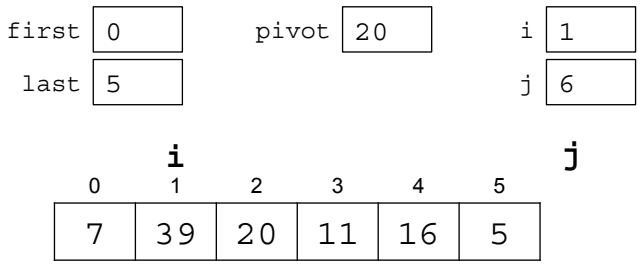
```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



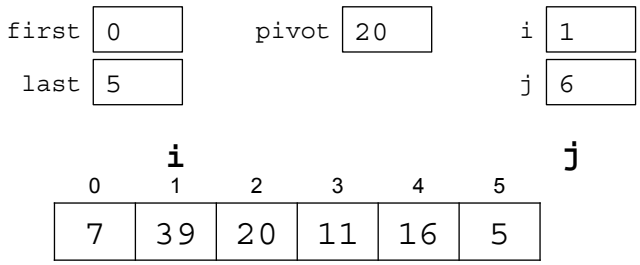

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);
                 arr[0] < pivot
        do {          7 < 20
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

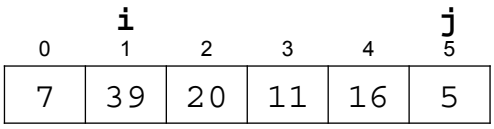



```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);
                arr[1] < pivot
        do {        39 < 20
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

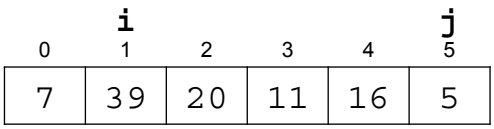
first	0	pivot	20	i	1
last	5			j	5



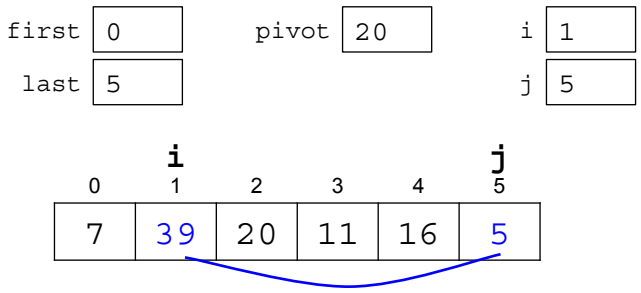

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```


arr[5] > pivot
5 > 20

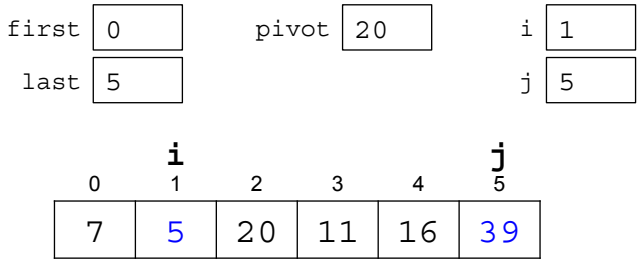
first	0	pivot	20	i	1
last	5			j	5



```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```




```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	20	i	2
last	5			j	5

		i			j
0	1	2	3	4	5
7	5	20	11	16	39


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);
                arr[2] < pivot
        do {        20 < 20
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	20	i	2
last	5			j	5

		i			j
0	1	2	3	4	5
7	5	20	11	16	39

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	20	i	2
last	5			j	4

		i		j	
0	1	2	3	4	5
7	5	20	11	16	39

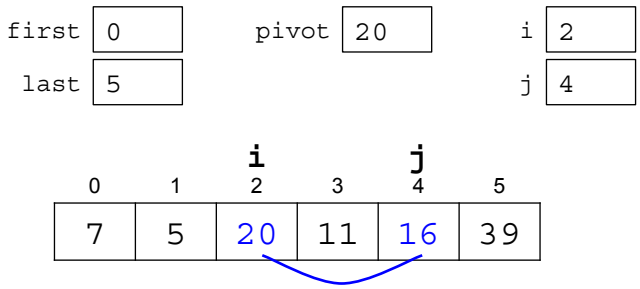

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```


arr[4] > pivot
16 > 20

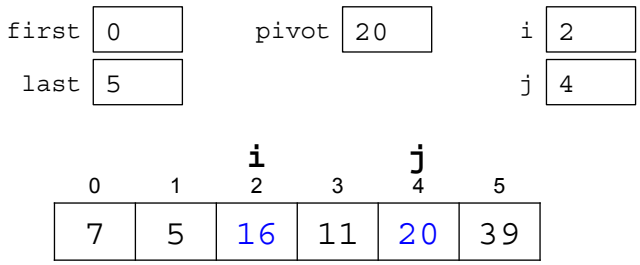
first	0	pivot	20	i	2
last	5			j	4

		i		j	
0	1	2	3	4	5
7	5	20	11	16	39

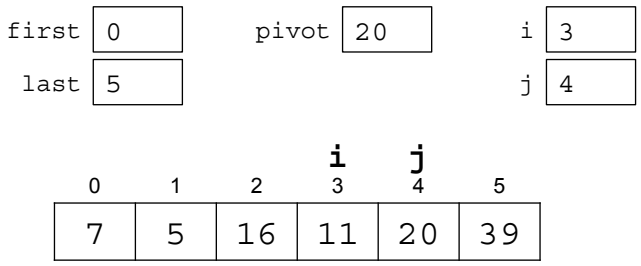
```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



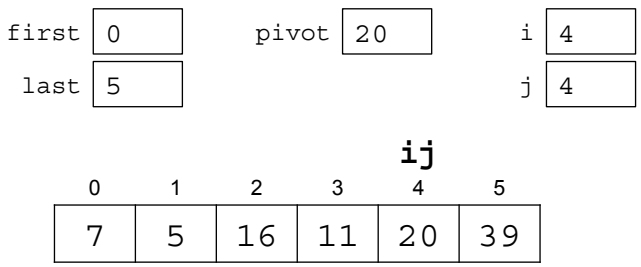

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

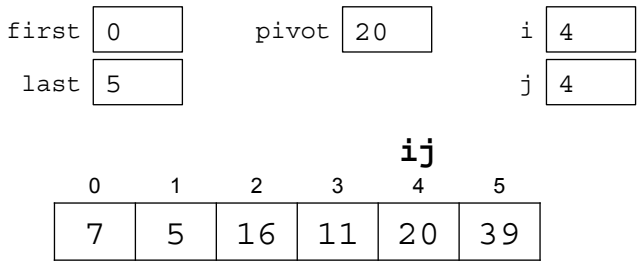
```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



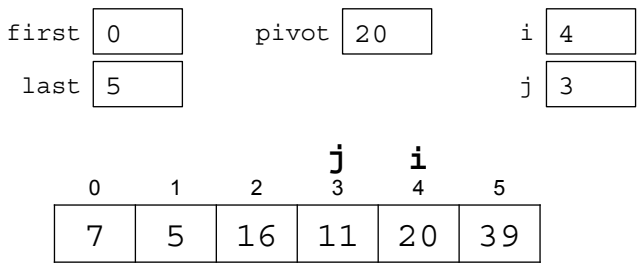

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);
                 arr[4] < pivot
        do {         20 < 20
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

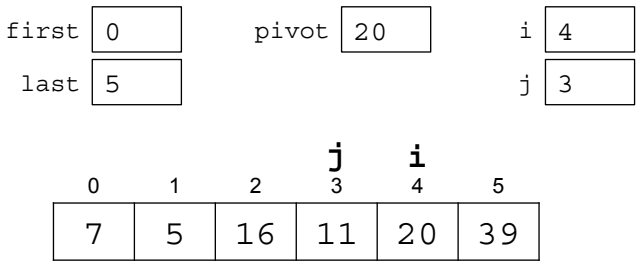



```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



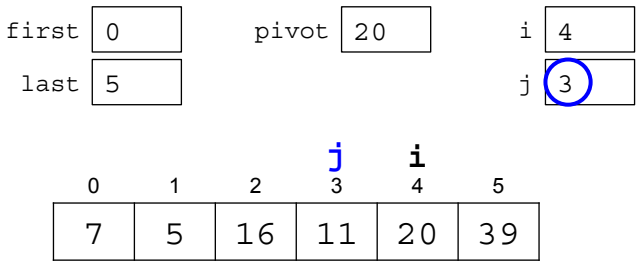
```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

arr[3] > pivot
11 > 20




```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

index of last element in left subarray



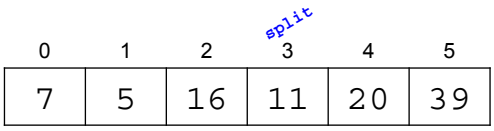
```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

index of last element in left subarray

first 0 split 3

last 5



qSort(0, 5)


```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

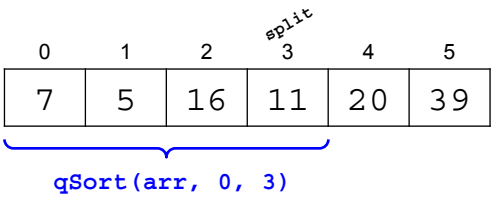

0 < 3


```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

index of last element in left subarray

first 0 split 3

last 5



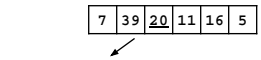
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 last 3 pivot ? i j

what is the pivot value for this subarray?

0	1	2	3	4	5
7	5	16	11	20	39



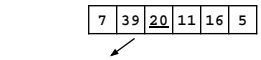
partition(0, 3)
qSort(0, 3)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```


first 0 last 3 pivot 5 i j

what are the initial values of `i` and `j`?

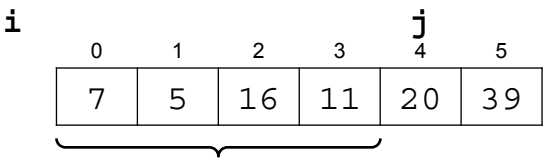
0	1	2	3	4	5
7	5	16	11	20	39

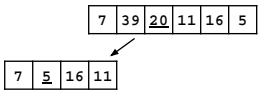


partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	5	i	-1
last	3			j	4

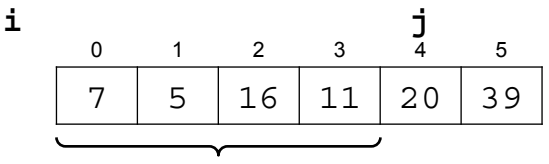




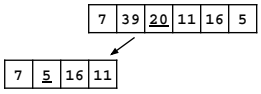
partition(0, 3)
qSort(0, 3)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	5	i	-1
last	3			j	4



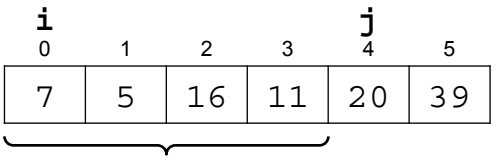
where does `i` stop?



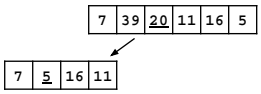
partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	0	pivot	5	i	0
last	3			j	4



where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

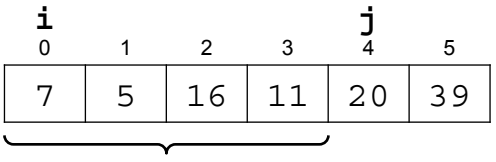


partition(0, 3)
qSort(0, 3)
qSort(0, 5)

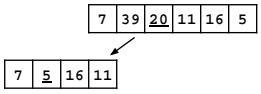

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```


arr[0] < pivot
7 < 5

first	0	pivot	5	i	0
last	3			j	4

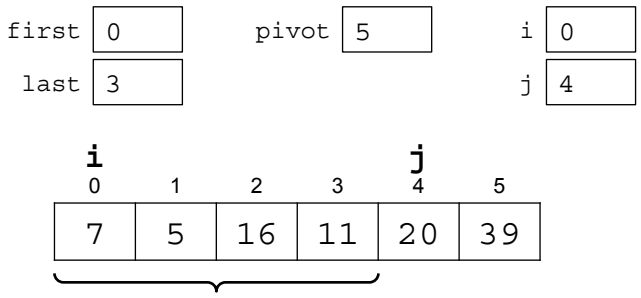


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*



partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

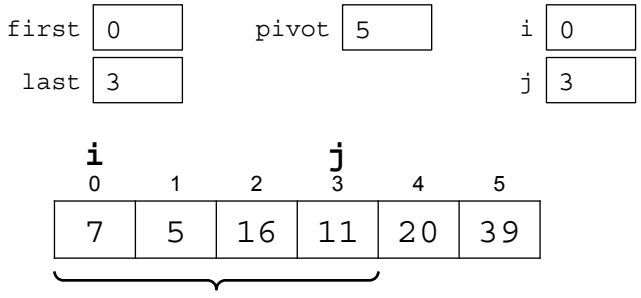


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop?

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

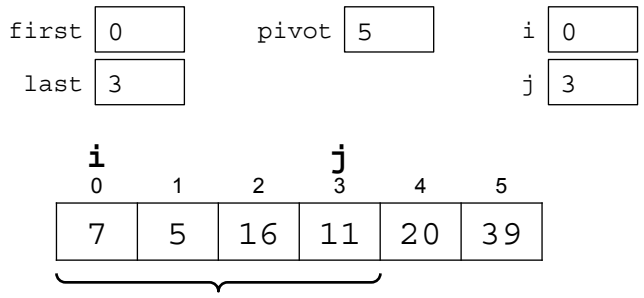


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

                 arr[3] > pivot
        do           11 > 5
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

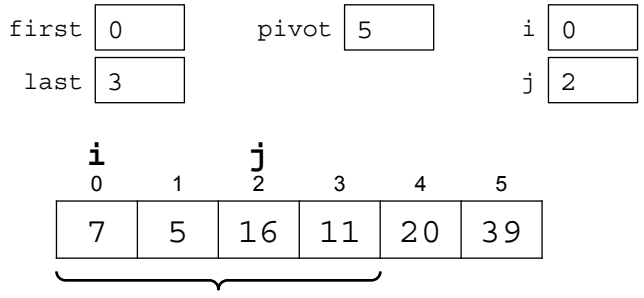


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

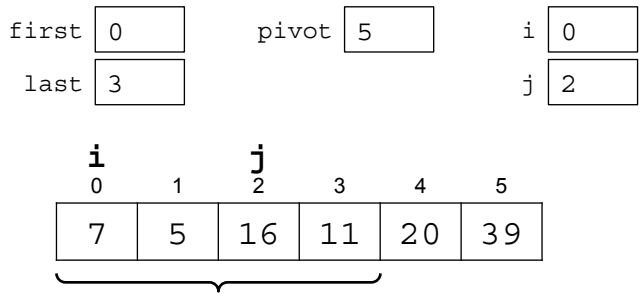


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

                 arr[2] > pivot
        do           16 > 5
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

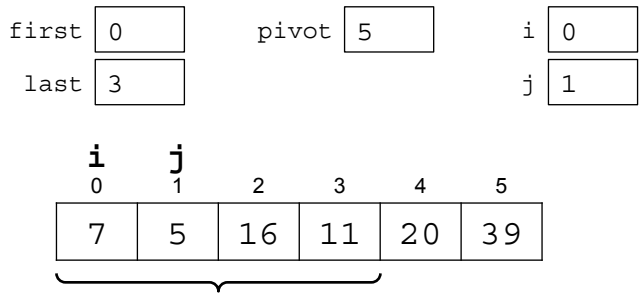


where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



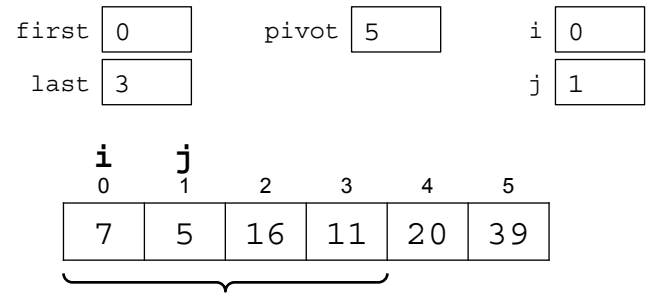
where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

arr[1] > pivot
5 > 5

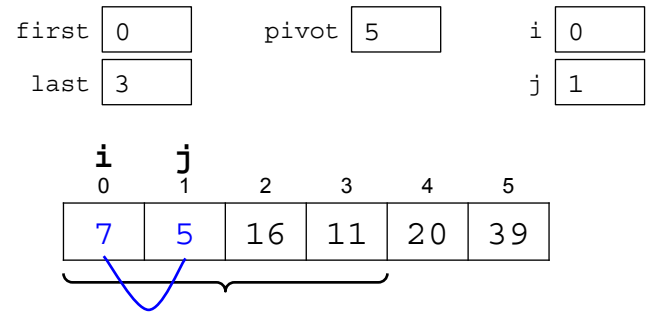


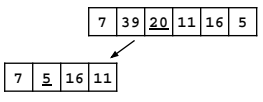
where does `i` stop? — *on the **first** element that is **not** less than the pivot value (stopping condition of do-while loop)*

where does `j` stop? — *on the **first** element that is **not** greater than the pivot value (stopping condition of do-while loop)*

partition(0, 3)
qSort(0, 3)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

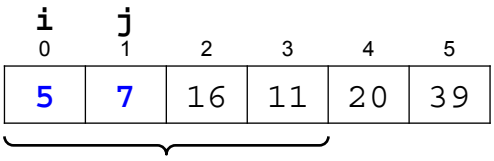


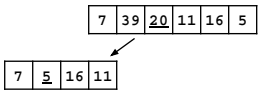
partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 pivot 5 i 0

last 3 j 1



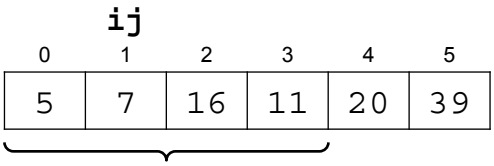


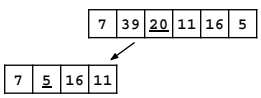
partition(0, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 pivot 5 i 1

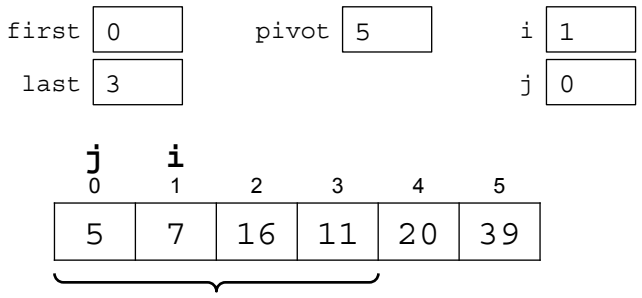
last 3 j 1

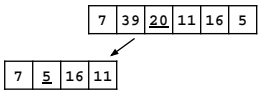




partition(0, 3)
qSort(0, 3)
qSort(0, 5)

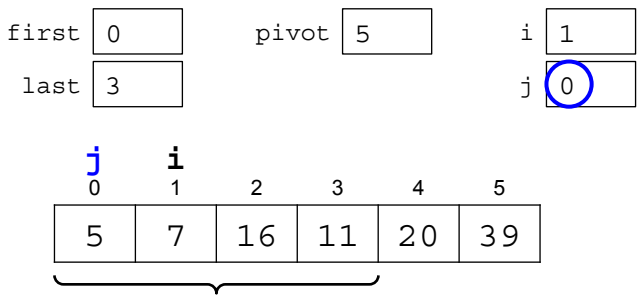
```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

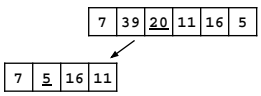




partition(0, 3)
qSort(0, 3)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```



partition(0, 3)
qSort(0, 3)
qSort(0, 5)

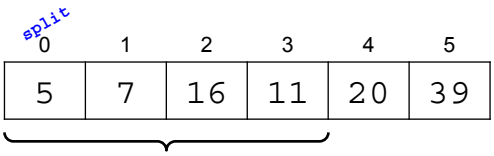
```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

0 < 0

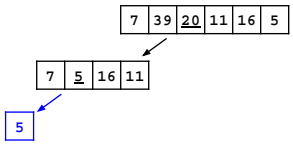
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 split 0

last 3



no call to `qSort()` for subarrays with one element



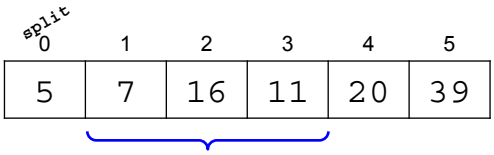
qSort(0, 3)
qSort(0, 5)


```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

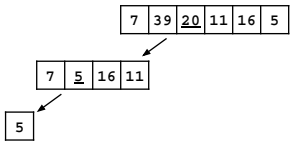
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 split 0

last 3



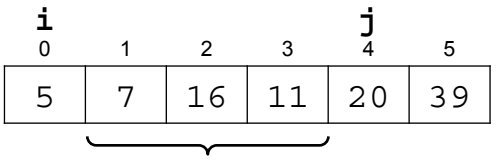
recurse on the right!

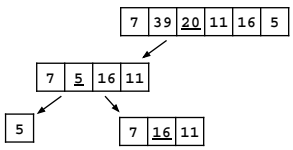


qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	16	i	0
last	3			j	4

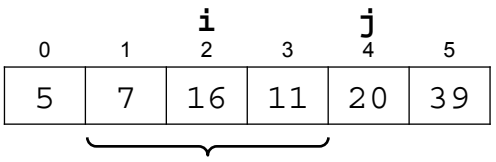


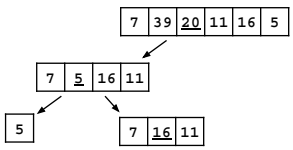


partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

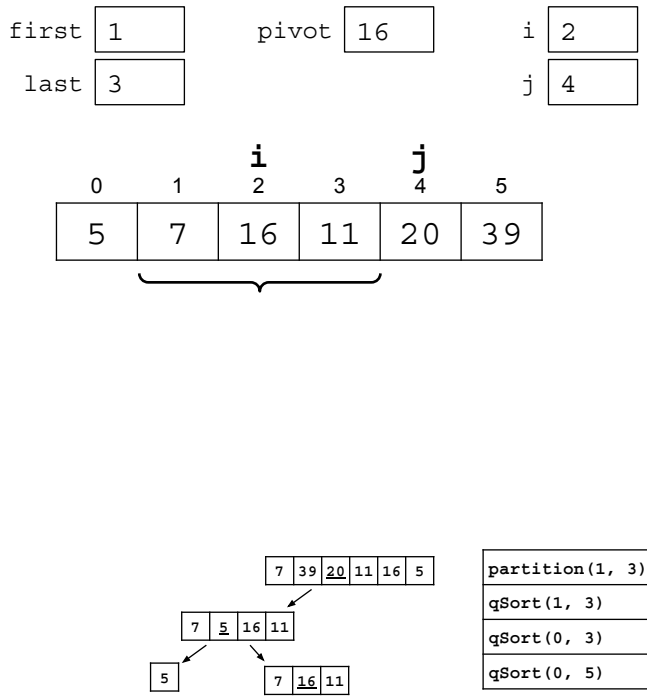
first	1	pivot	16	i	2
last	3			j	4



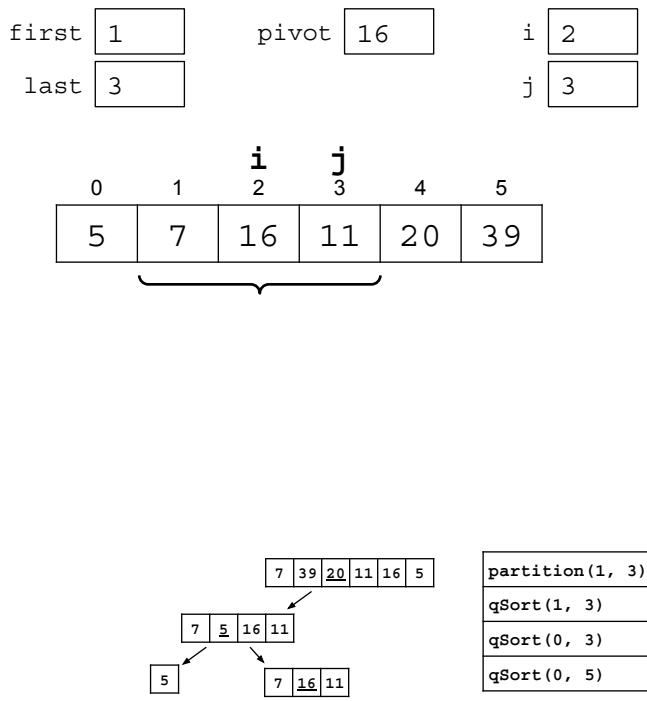


partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

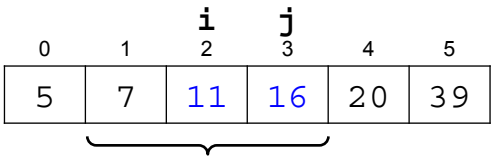


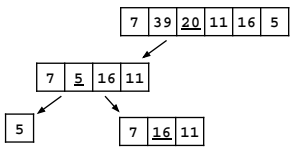

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	16	i	2
last	3			j	3

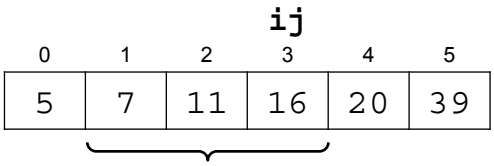


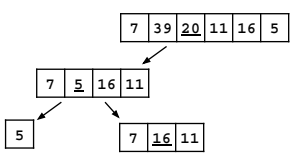


partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	16	i	3
last	3			j	3

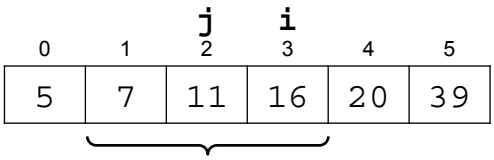


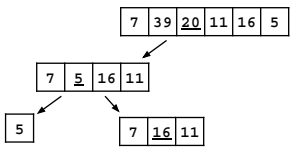


partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	16	i	3
last	3			j	2

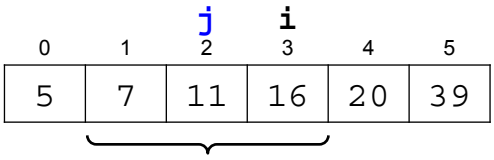


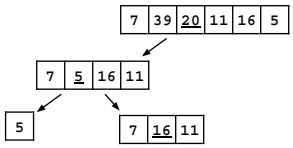


partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```


first	1	pivot	16	i	3
last	3			j	2





partition(1, 3)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

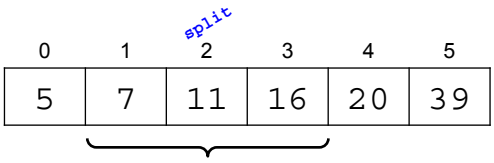
```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

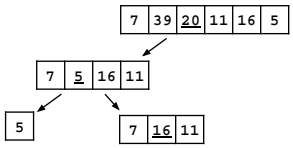
1 < 2

```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 1 split 2

last 3





qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

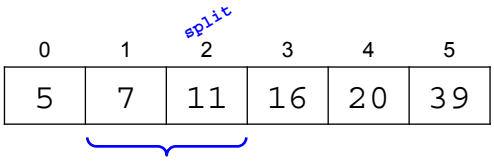

```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

1 < 2

```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

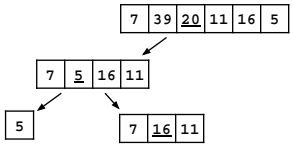
first 1 split 2

last 3



qSort(arr, 1, 2)

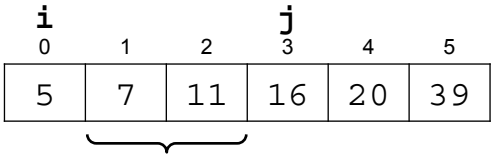
recurse on the left!

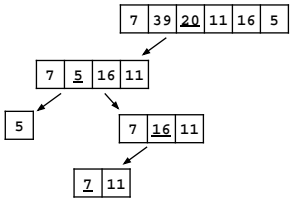


qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	7	i	0
last	2			j	3

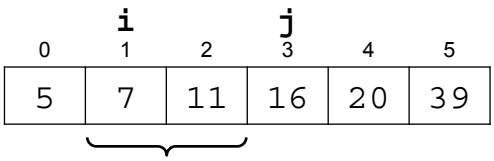


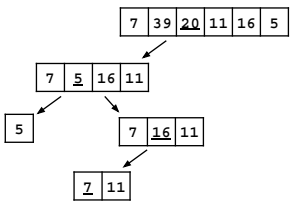


partition(1, 2)
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1	pivot	7	i	1
last	2			j	3

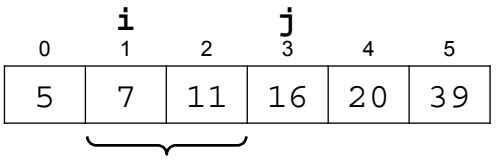


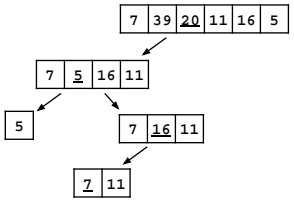


partition(1, 2)
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1		pivot	7	i	1
last	2				j	3

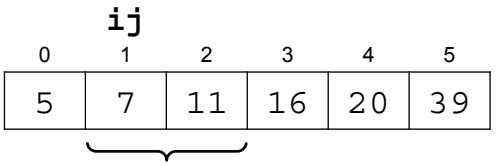


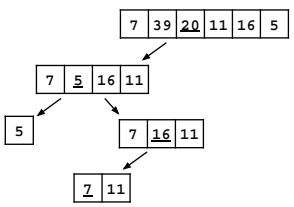


partition(1, 2)
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first	1		pivot	7	i	1
last	2				j	1





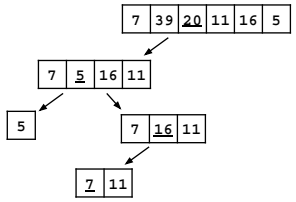
partition(1, 2)
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 1 last 2 pivot 7 i 1 j 1

ij

0	1	2	3	4	5
5	7	11	16	20	39



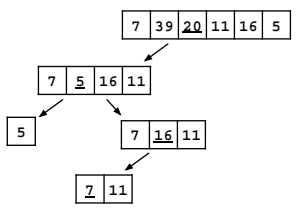
partition(1, 2)
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

first 1 last 2 split 1

split

0	1	2	3	4	5
5	7	11	16	20	39



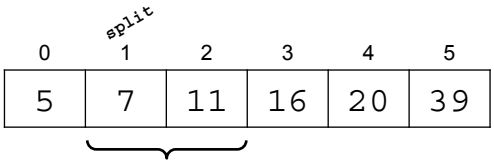
qSort(1, 2)
qSort(1, 3)
qSort(0, 3)
qSort(0, 5)

```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

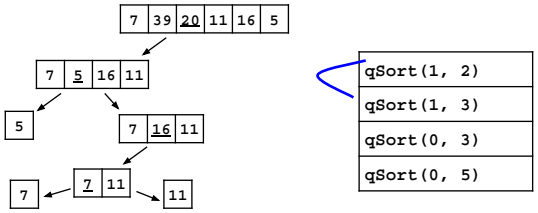
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 1 split 1

last 2



both of these checks are false, so the one-element subarrays with 7 and 11 are sorted, and we return back to `qSort(1, 3)`

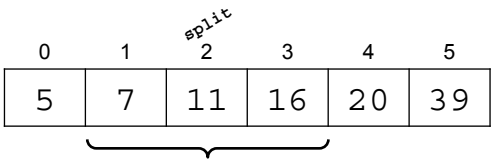



```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

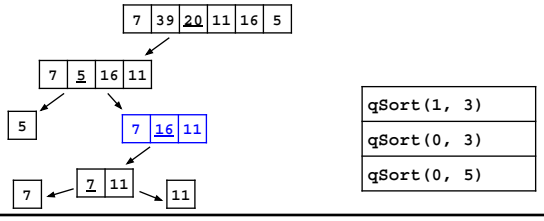
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 1 split 2

last 3



in `qSort(1, 3)` we were waiting on the left recursive call... should we do the right recursive call now?



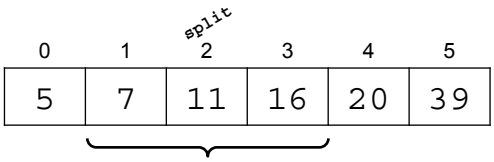
```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

3 > 3

```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

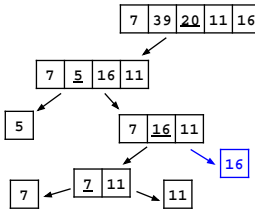
first 1 split 2

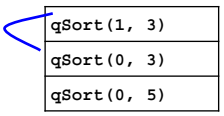
last 3



in `qSort(1, 3)` we were waiting on the left recursive call... should we do the right recursive call now?

***no**, the right subarray only contains 16, so return from `qSort(1, 3)`*



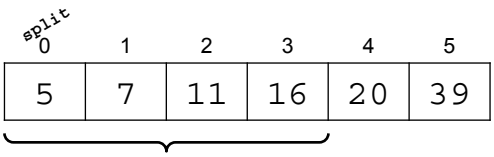



```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

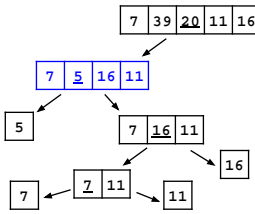
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

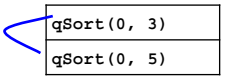
first 0 split 0

last 3



in `qSort(0, 3)` we were waiting on the right recursive call, no more work to do! return back to `qSort(0, 5)`



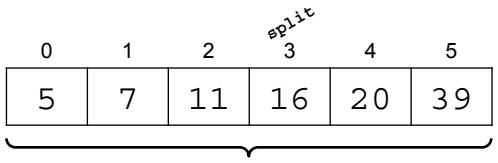


```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

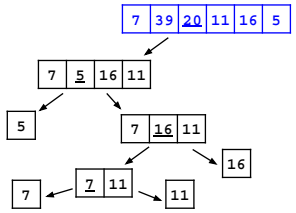
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 split 3

last 5



in qSort(0, 5) we were waiting on the left recursive call, now recurse right



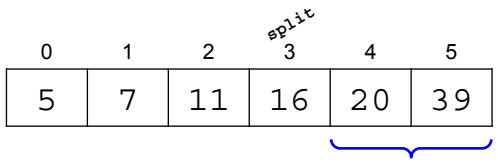
qSort(0, 5)


```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

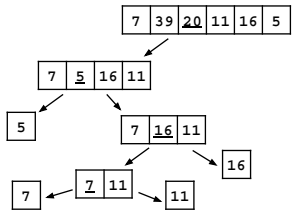
first 0 split 3

last 5



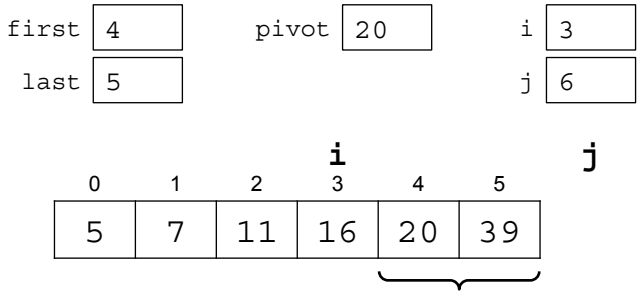
qSort(arr, 4, 5)

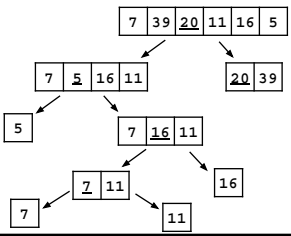
in qSort(0, 5) we were waiting on the left recursive call, now recurse right



qSort(0, 5)

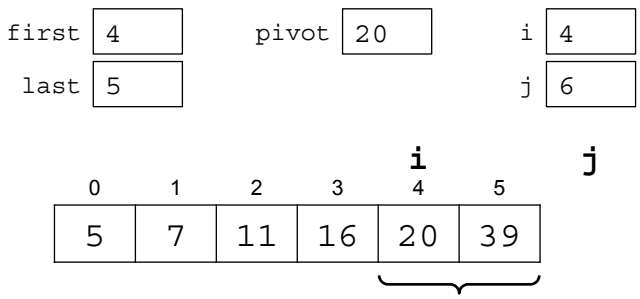
```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

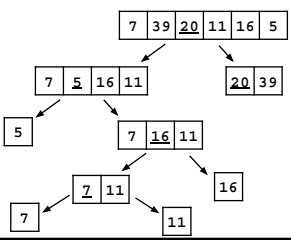




partition(4, 5)
qSort(4, 5)
qSort(0, 5)

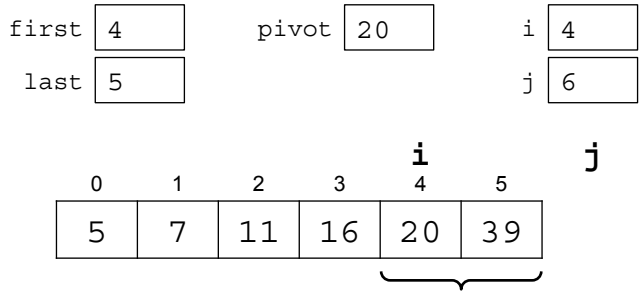

```java
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

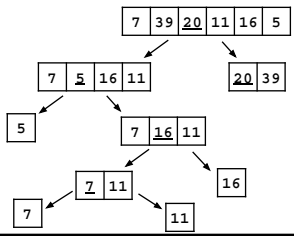





partition(4, 5)
qSort(4, 5)
qSort(0, 5)

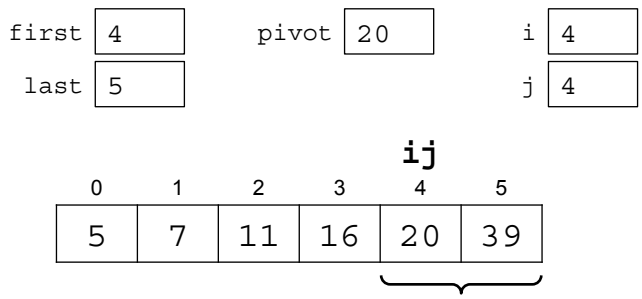
```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

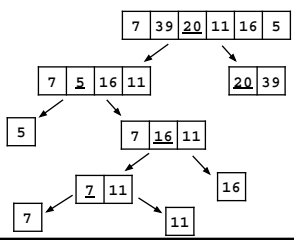




partition(4, 5)
qSort(4, 5)
qSort(0, 5)


```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

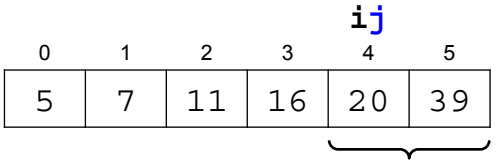


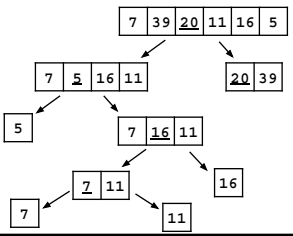
partition(4, 5)
qSort(4, 5)
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 4 pivot 20 i 4

last 5 j 4



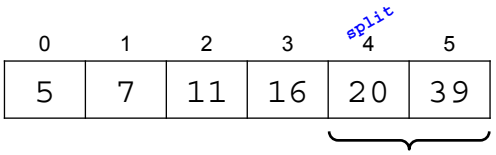


partition(4, 5)
qSort(4, 5)
qSort(0, 5)

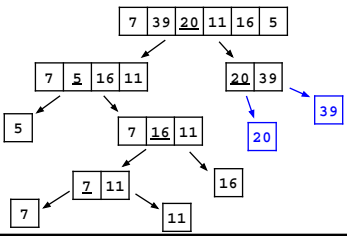
```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

first 4 split 4

last 5



both of these checks are false, so the one-element subarrays with 20 and 39 are sorted, and we return back to `qSort(0, 5)`



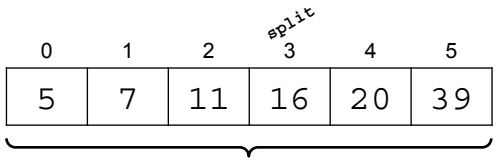
qSort(4, 5)
qSort(0, 5)

```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```

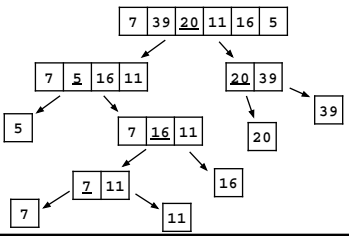
```
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

first 0 split 3

last 5



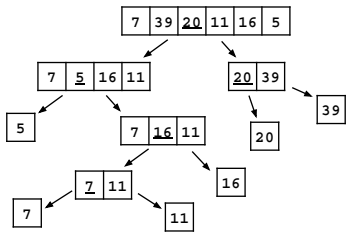
qSort(0, 5) was waiting on the right recursive call, so it has no more work to do, and the array is sorted



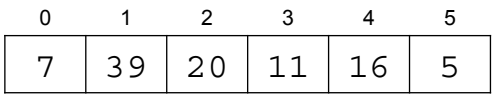
qSort(0, 5)

```
public static int partition(
  int[] arr, int first, int last
) {
    int pivot = arr[(first + last)/2];
    int i = first - 1;
    int j = last + 1;

    while (true) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i < j) {
            swap(arr, i, j);
        } else {
            return j;
        }
    }
}
```

```
private static void qSort(
  int[] arr, int first, int last
) {
    int split = partition(arr, first, last);

    if (first < split) {
        qSort(arr, first, split);
    }
    if (last > split + 1) {
        qSort(arr, split + 1, last);
    }
}
```



5 7 11 16 20 39



first

last

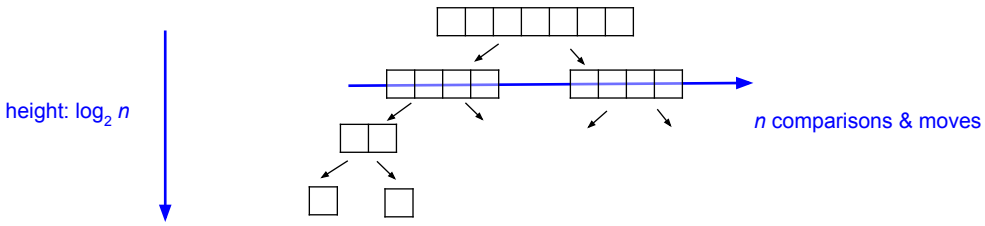
split

Quicksort

- What is the time complexity of quicksort
  - in the best case?

Quicksort

- What is the time complexity of quicksort
  - in the best case?
    Each time we partition, we divide the current subarray in half. The call tree will then have a height of $\log_2 n$, since $\log_2 n$ is an upper bound on the number of times we can divide n in half before reaching 1, and the recursion stops when we get down to subarrays of size 1. At each of the $\log_2 n$ levels of the call tree, we perform n comparisons. The number of moves per level will vary, but is still $O(n)$. The overall best-case time complexity is therefore $\mathbf{O(n \cdot \log_2 n)}$.

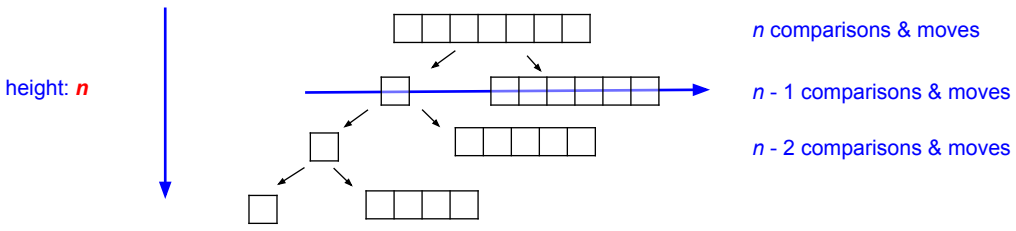


Quicksort

- What is the time complexity of quicksort
  - in the worst case?

Quicksort

- What is the time complexity of quicksort
  - in the worst case?
    Each time we partition, we divide the current subarray into a subarray containing just a single element and another subarray containing all the remaining elements. In this case, the call tree will have a height of *n*, since we're reducing the problem size by **only 1 element at a time**. The number of comparisons we perform starts at *n* and goes down by 1 for each level, giving us $O(n^2)$ comparisons. We perform at most 1 swap per level, so there are $O(n)$ moves, but the overall worst-case complexity is still $\mathbf{O(n^2)}$.



Quicksort

- What is the time complexity of quicksort
  - in the average case?
    We get a mix of balanced and unbalanced partitionings, and so the height of our call tree will
    be > $\log_2 n$ but < n. It can be proved that quicksort is still $O(n \cdot \log_2 n)$ on average, and that its average case is much closer to its best case than its worst case.
- How would you characterize the performance of quicksort in the example we just stepped through? Was it an example of best case, worst case, or average case performance? Why?
  - Our input size n is 6 and the height of our call tree was 4. $\log_2(6) \approx 2.5$, so it is best described as an example of average performance.