Section 10

CSCI E-22

Will Begin Shortly

Heaps

Recall:

- *Heap*: a complete binary tree in which each interior node is greater than or equal to its children
- The maximum value is in the root node
- The minimum value can be in any one of the leaf nodes
- Often used to implement a priority queue
  - Can efficiently retrieve the highest priority value (root of max-at-top heap)
- Since a heap is a complete tree, we can use an array as a compact representation
  - For a parent node at index i, the left child stored in index 2*i + 1
    and right child is stored in index 2*i + 2

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

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

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree

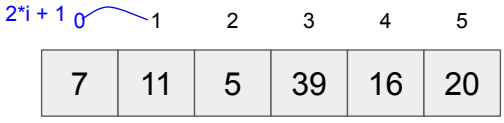
0	1	2	3	4	5
7	11	5	39	16	20

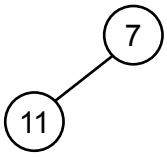
Use the child formulas to figure out what the parent-child edges are

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree

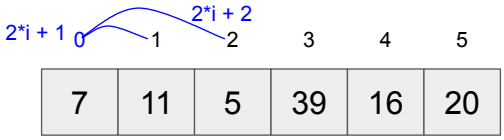


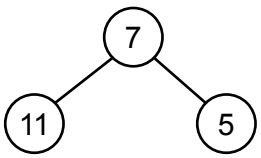


Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree

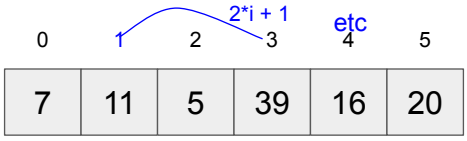


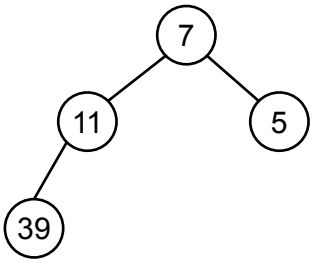


Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree

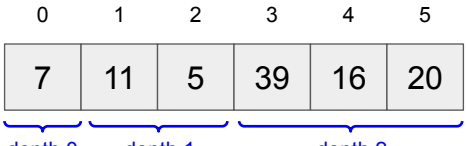


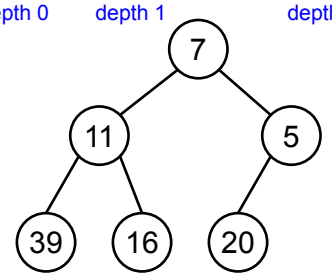


Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree





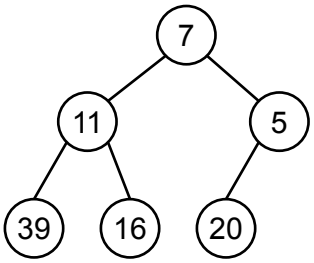
Can also fill in the complete tree from top-to-bottom, left-to-right as you read the array

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree

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



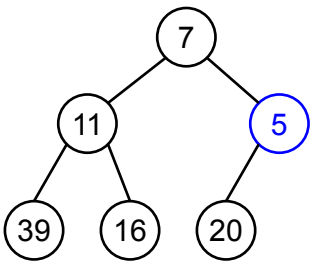
(Remember that in a complete tree there can be gaps in the last level, as long as they are on the right)

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



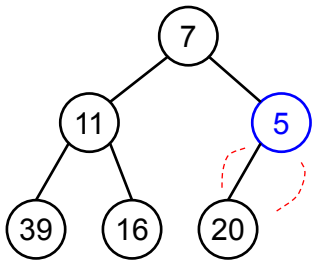
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



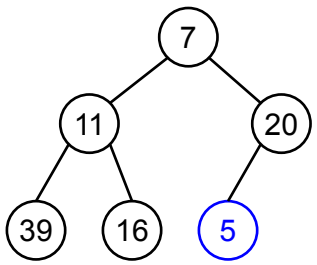
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

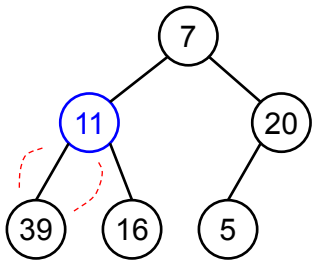
Can't sift 5 down any further

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



If both children are greater than the parent, choose the greatest

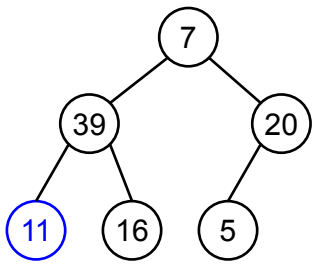
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



Can't sift 11 down any further

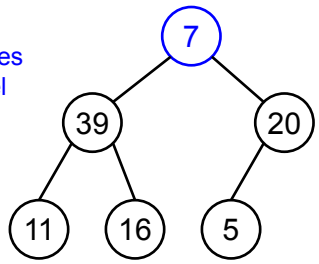
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

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

Step 1: represent as complete tree
Step 2: heapify the tree



Once done sifting all of the nodes in a given level, move up a level

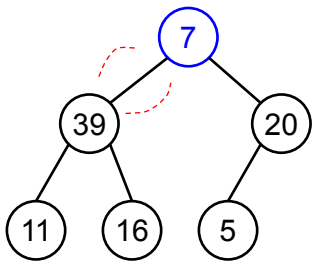
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

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

Step 1: represent as complete tree
Step 2: heapify the tree



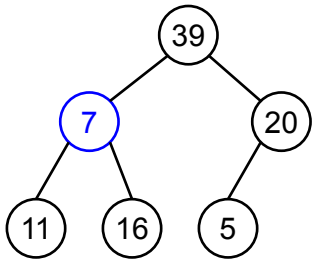
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



Recursively continue sifting 7 down

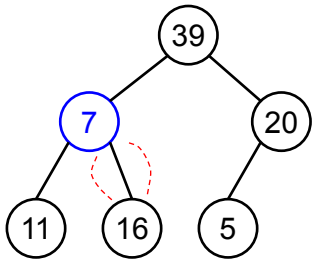
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



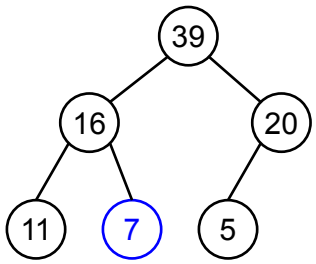
Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



Recursively sift down all interior nodes, starting with the parent of the *last* leaf node

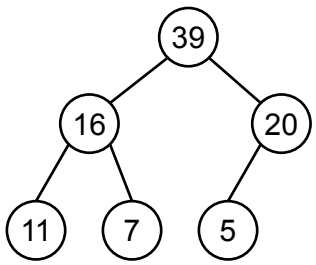
Can't sift 7 down any further

Heaps

We will now convert the following 6-element array into a valid max-heap. We start by taking our array and representing it as a heap in the usual manner, with the first element as the root.

Step 1: represent as complete tree
Step 2: heapify the tree

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



Heaps

How long does it take to create a heap from an array?

Heaps

How long does it take to create a heap from an array?

Each sift operation takes O(log n) time to sift items through the height of the heap.

There are O(n) overall sift operations.

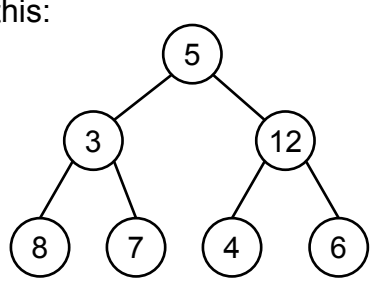
It takes O(n log n) time to create a heap from an array.

Heapsort

We will now run through an example of heapsort. Suppose we have the following numbers in an array:

0	1	2	3	4	5	6
5	3	12	8	7	4	6

We want to sort the array in ascending order. If we interpret the array as a complete tree, it looks like this:



Heapsort

What's the first step of heapsort?

Heapsort

What's the first step of heapsort? Turn the array into a heap.

Heapsort

What's the first step of heapsort? Turn the array into a heap.

Should we create a min-heap or a max-heap?

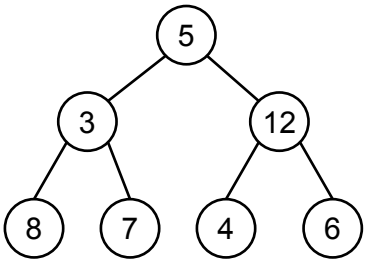
Heapsort

What's the first step of heapsort? Turn the array into a heap.

Should we create a min-heap or a max-heap? We create a **max-heap** since we will be removing the largest element each time and filling the array right to left.

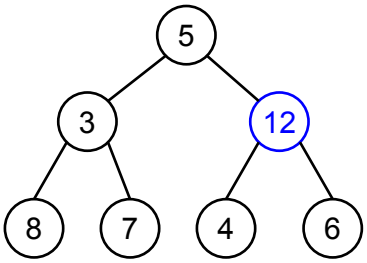
Heapsort

What's the first step of heapsort? Turn the array into a heap.



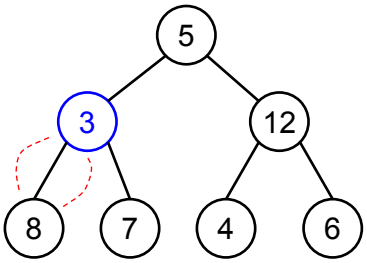
Heapsort

What's the first step of heapsort? Turn the array into a heap.



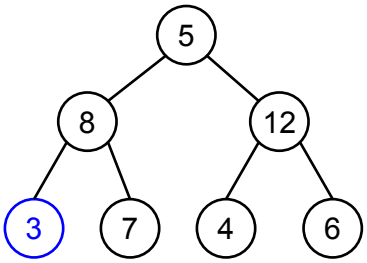
Heapsort

What's the first step of heapsort? Turn the array into a heap.



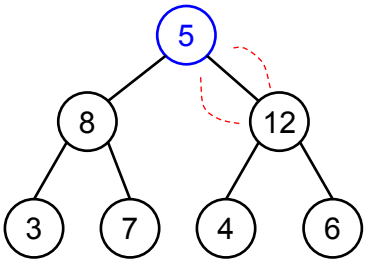
Heapsort

What's the first step of heapsort? Turn the array into a heap.



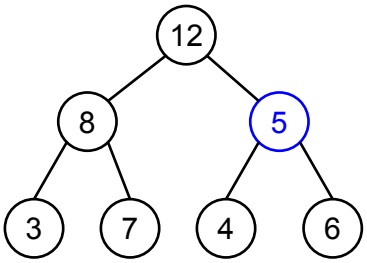
Heapsort

What's the first step of heapsort? Turn the array into a heap.



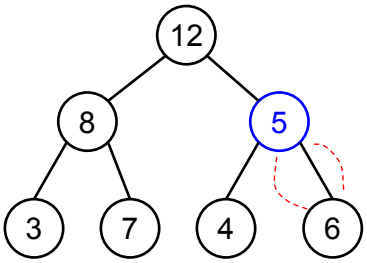
Heapsort

What's the first step of heapsort? Turn the array into a heap.



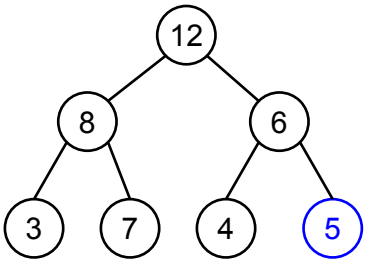
Heapsort

What's the first step of heapsort? Turn the array into a heap.



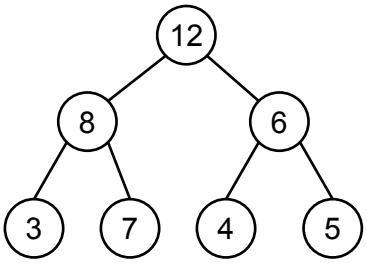
Heapsort

What's the first step of heapsort? Turn the array into a heap.



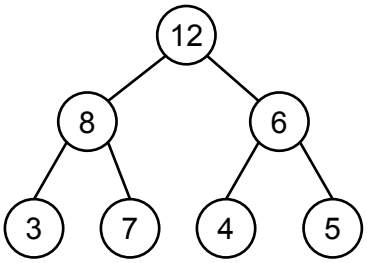
Heapsort

What's the first step of heapsort? Turn the array into a heap.



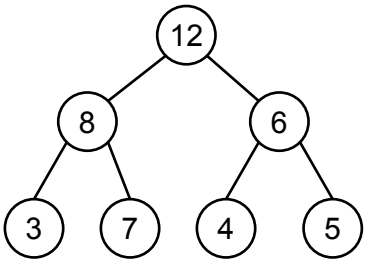
Heapsort

What is the next step?

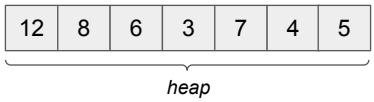


Heapsort

What is the next step? Successively remove the largest element, reheapify, and place the largest element in the correct location in the array.

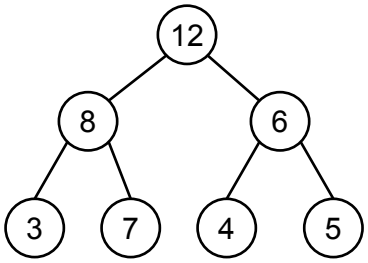


Heapsort

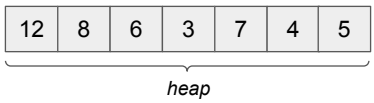


Algorithm:

Repeat:
- Remove largest element
- Re-heapify
- Place removed element into place in array

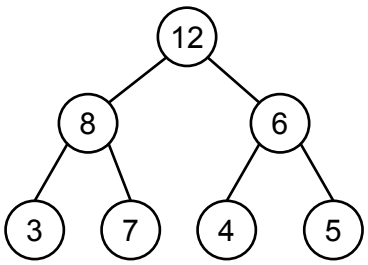


Heapsort

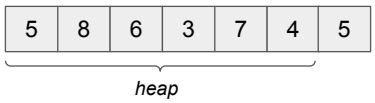


Algorithm:

Repeat:
- Remove largest element
- Re-heapify
- Place removed element into place in array



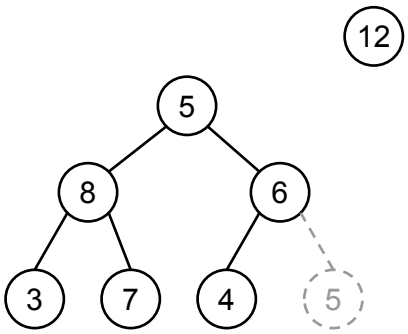
Heapsort



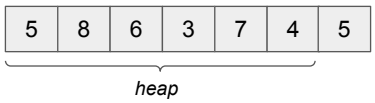
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



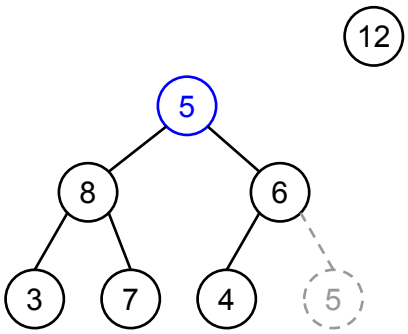
Heapsort



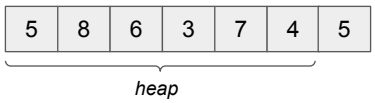
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



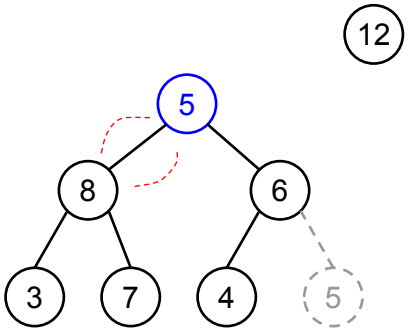
Heapsort



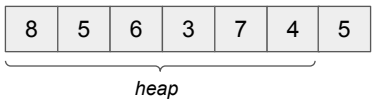
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



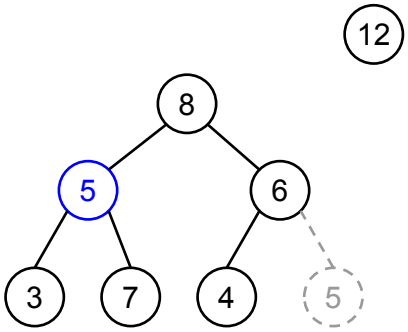
Heapsort



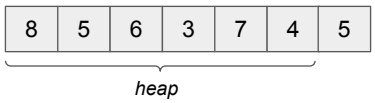
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



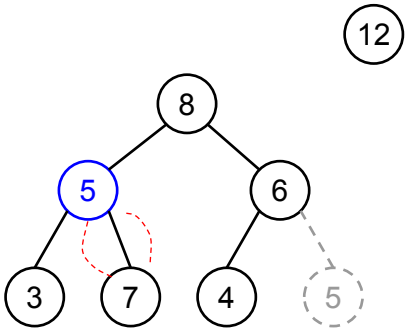
Heapsort



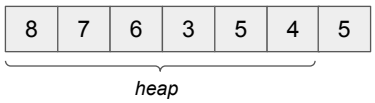
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



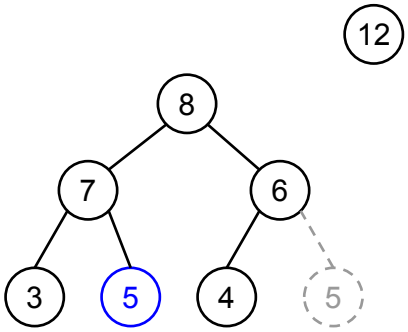
Heapsort



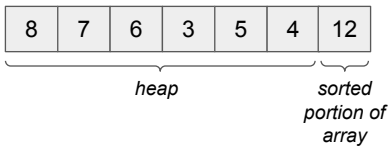
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



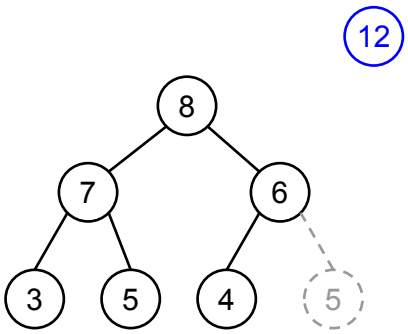
Heapsort



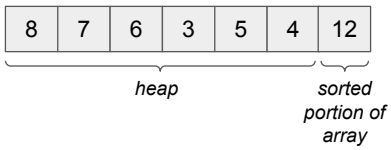
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



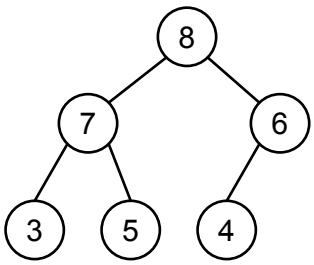
Heapsort



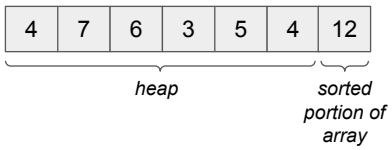
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



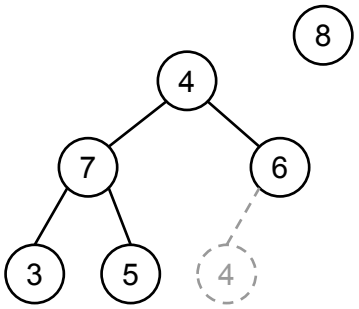
Heapsort



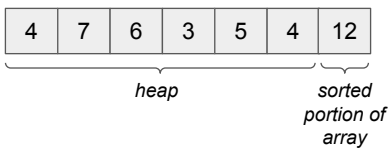
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



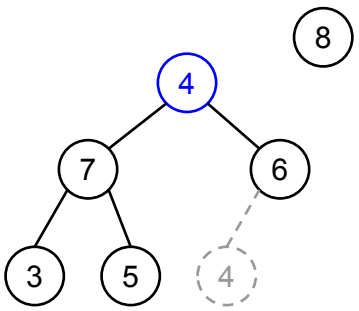
Heapsort



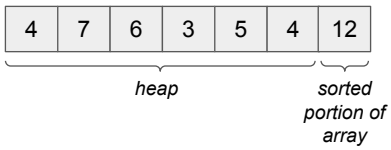
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



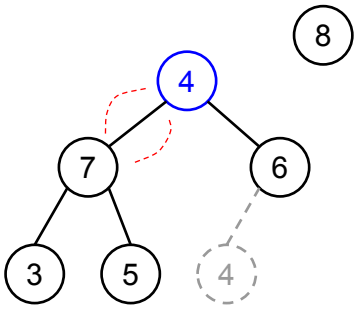
Heapsort



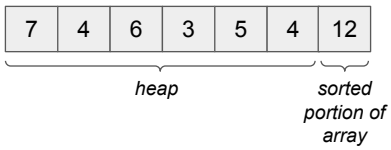
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



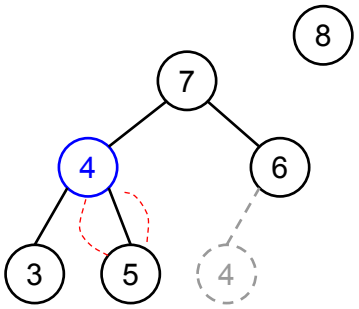
Heapsort



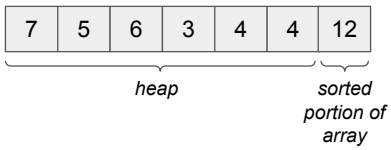
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



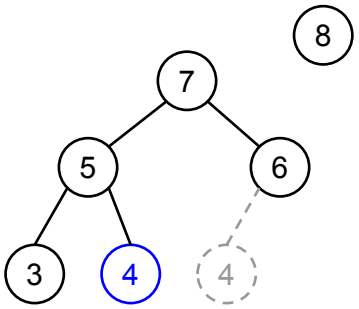
Heapsort



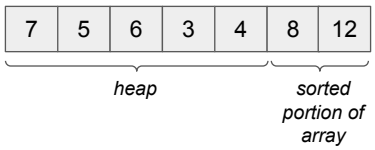
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



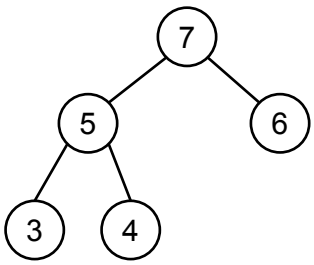
Heapsort



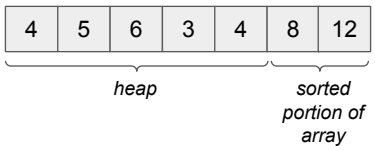
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



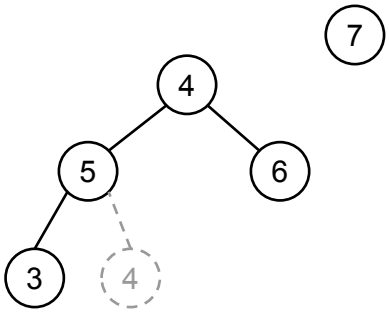
Heapsort



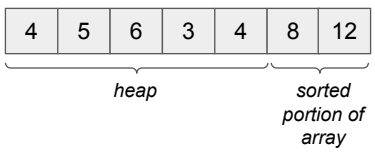
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



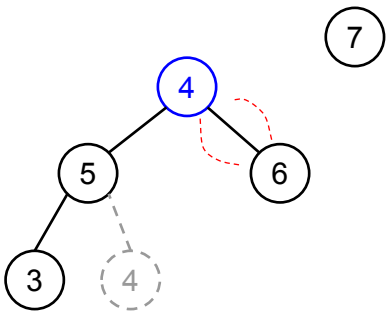
Heapsort



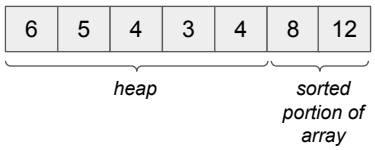
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



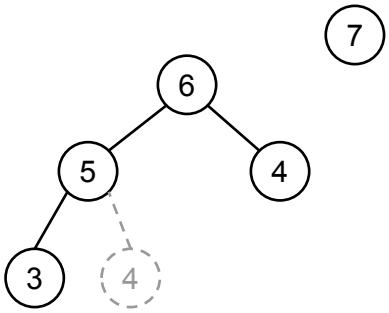
Heapsort



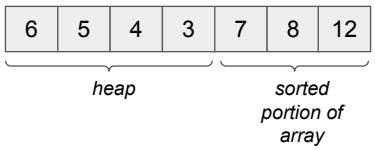
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



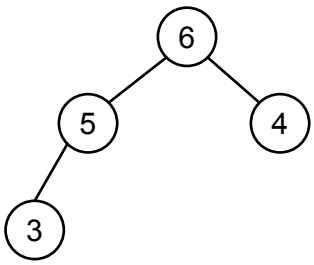
Heapsort



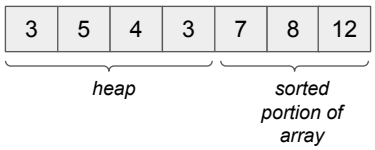
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



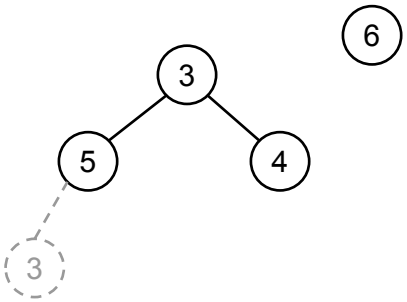
Heapsort



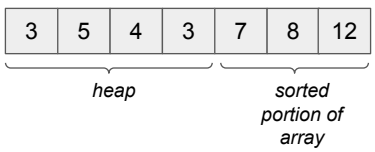
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



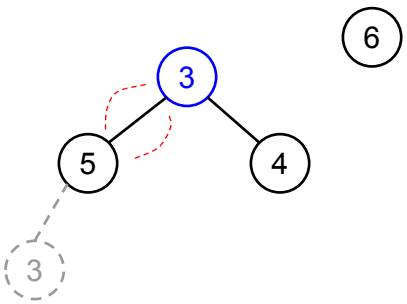
Heapsort



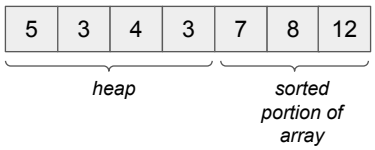
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



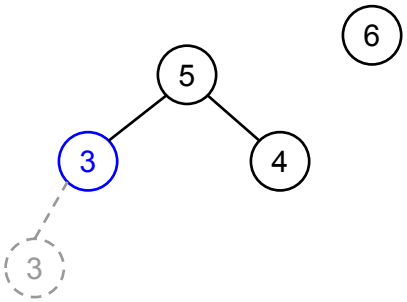
Heapsort



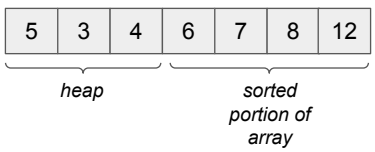
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



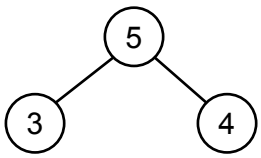
Heapsort



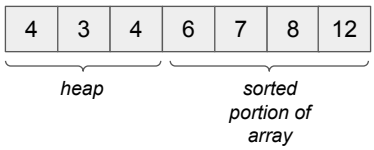
Algorithm:

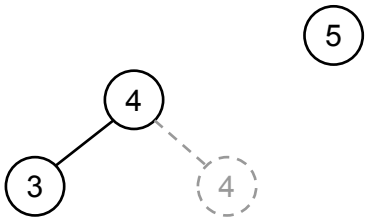
Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



Heapsort



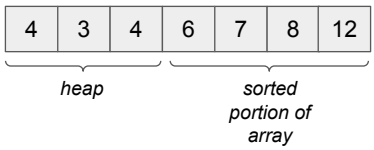


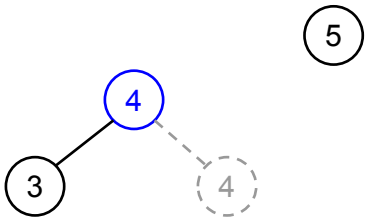
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort



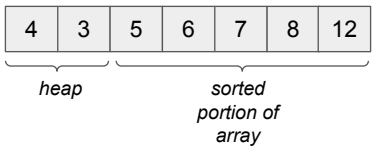


Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

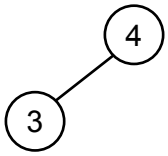
Heapsort



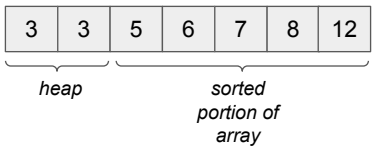
Algorithm:

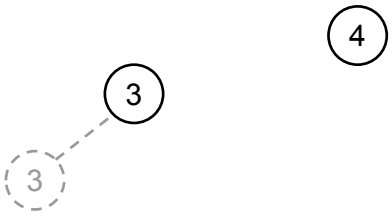
Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array



Heapsort



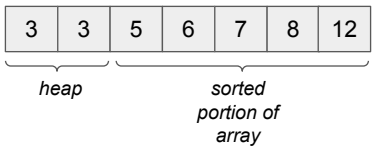


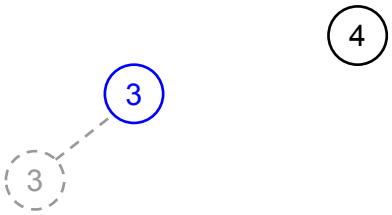
Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort



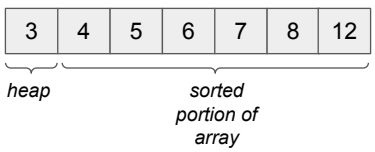


Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort

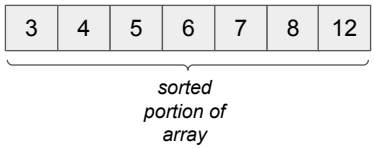


Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort

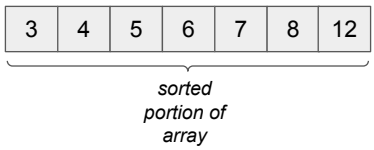


Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort



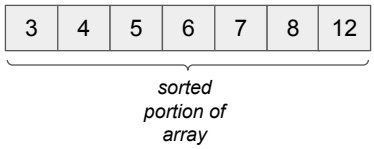
What is the runtime of Heapsort?

Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

Heapsort



Algorithm:

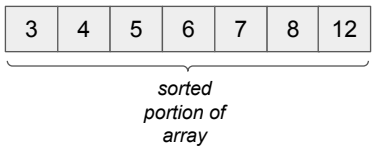
Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

What is the runtime of Heapsort?

O(n * log(n)) for heapifying the initial array

Heapsort



Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

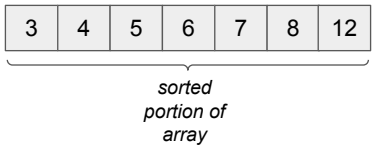
What is the runtime of Heapsort?

O(n * log(n)) for heapifying the initial array

+

O(n * log(n)) for n successive sift operations

Heapsort



Algorithm:

Repeat:

- Remove largest element
- Re-heapify
- Place removed element into place in array

What is the runtime of Heapsort?

O(n * log(n)) for heapifying the initial array

+

O(n * log(n)) for n successive sift operations

O(n * log(n)) overall

Hashing

Suppose we have a 7-element hash table, and we wish to insert following words:

`apple, cat, anvil, boy, bag, dog, cup, down`

We'll use the following hash function:

h(key): index related to first letter of the key (a = 0, b = 1, ...)

Let's insert these keys into an initially empty hash table using *linear* probing and count the total length of the probes. Then, do the same exercise, but use *quadratic* probing.

Hashing

Suppose we have a 7-element hash table, and we wish to insert following words:

```
apple, cat, anvil, boy, bag, dog, cup, down
```

We'll use the following hash function:

h(key): index related to first letter of the key (a = 0, b = 1, ...)

Let's insert these keys into an initially empty hash table using *linear* probing and count the total length of the probes. Then, do the same exercise, but use *quadratic* probing.

Linear probe sequence: h(x), h(x) + 1, h(x) + 2, h(x) + 3, …, h(x) + n-1
Quadratic probe sequence: h(x), h(x) + 1^2, h(x) + 2^2, h(x) + 3^2, …, h(x) + $(n-1)^2$

Hashing

Suppose we have a 7-element hash table, and we wish to insert following words:

```
apple, cat, anvil, boy, bag, dog, cup, down
```

We'll use the following hash function:

h(key): index related to first letter of the key (a = 0, b = 1, ...)

Let's insert these keys into an initially empty hash table using *linear* probing and count the total length of the probes. Then, do the same exercise, but use *quadratic* probing.

Wrap around as necessary with mod!

Linear probe sequence: h(x), h(x) + 1, h(x) + 2, h(x) + 3, …, h(x) + n-1
Quadratic probe sequence: h(x), h(x) + 1^2, h(x) + 2^2, h(x) + 3^2, …, h(x) + $(n-1)^2$

apple, cat, anvil, boy, bag, dog, cup, down

Linear probing

0	
1	
2	
3	
4	
5	
6	

apple, cat, anvil, boy, bag, dog, cup, down

Linear probing

h(apple) = 0

0	apple
1	
2	
3	
4	
5	
6	

apple, cat, anvil, boy, bag, dog, cup, down

Linear probing

0	apple
1	
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

apple, cat, anvil, boy, bag, dog, cup, down

Linear probing

0	apple
1	
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Linear probing

0	apple
1	anvil
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Linear probing

0	apple
1	anvil
2	cat
3	
4	
5	
6	

h(apple) = 0

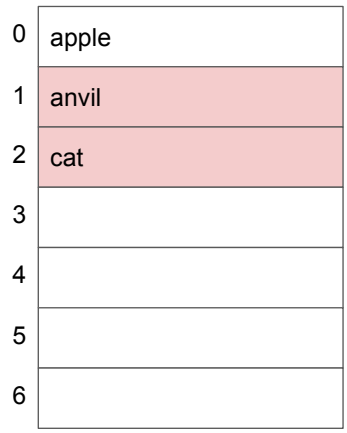
h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Linear probing



h(apple) = 0

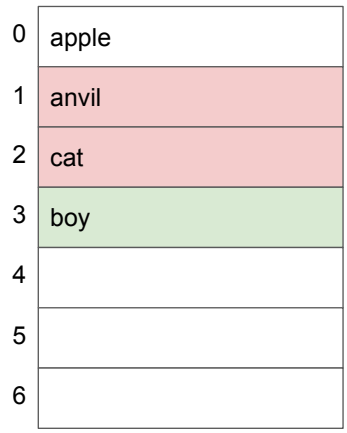
h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Linear probing



h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) = 4

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) = 4

h(dog) = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) = 4

h(dog) = 3
…
h(dog) = 5

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) = 4

h(dog) = 3
…
h(dog) = 5

h(cup) = 2

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) + 3 = 4

h(dog) = 3
…
h(dog) + 2 = 5

h(cup) = 2
…
h(cup) + 4 = 6

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) + 3 = 4

h(dog) = 3
…
h(dog) + 2 = 5

h(cup) = 2
…
h(cup) + 4 = 6

h(down) = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) + 3 = 4

h(dog) = 3
…
h(dog) + 2 = 5

h(cup) = 2
…
h(cup) + 4 = 6

h(down) = 3
…
h(down) + 6 = 9 **% 7** = 2

`apple, cat, anvil, boy, bag, dog, cup, down`

Linear probing

0	apple
1	anvil
2	cat
3	boy
4	bag
5	dog
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1 = 1

h(boy) = 1
h(boy) + 1 = 2
h(boy) + 2 = 3

h(bag) = 1
…
h(bag) + 3 = 4

h(dog) = 3
…
h(dog) + 2 = 5

h(cup) = 2
…
h(cup) + 4 = 6

h(down) = 3
…
h(down) + 6 = 9 **% 7** = 2

Linear probing total probe length: 26

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	
1	
2	
3	
4	
5	
6	

apple, cat, anvil, boy, bag, dog, cup, down

Quadratic probing

h(apple) = 0

0	apple
1	
2	
3	
4	
5	
6	

apple, cat, anvil, boy, bag, dog, cup, down

Quadratic probing

h(apple) = 0

h(cat) = 2

0	apple
1	
2	cat
3	
4	
5	
6	

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5

```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3


```
apple, cat, anvil, boy, bag, dog, cup, down
```

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2
h(cup) + 1^2 = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2
h(cup) + 1^2 = 3
h(cup) + 2^2 = 6

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2
h(cup) + 1^2 = 3
h(cup) + 2^2 = 6

h(down) = 3

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2
h(cup) + 1^2 = 3
h(cup) + 2^2 = 6

h(down) = 3
…
h(down) + 6^2 = 39 % 7 = 4

`apple, cat, anvil, boy, bag, dog, cup, down`

Quadratic probing

0	apple
1	anvil
2	cat
3	bag
4	dog
5	boy
6	cup

h(apple) = 0

h(cat) = 2

h(anvil) = 0
h(anvil) + 1^2 = 1

h(boy) = 1
h(boy) + 1^2 = 2
h(boy) + 2^2 = 5

h(bag) = 1
h(bag) + 1^2 = 2
h(bag) + 2^2 = 5
h(bag) + 3^2 = 10 % 7 = 3

h(dog) = 3
h(dog) + 1^2 = 4

h(cup) = 2
h(cup) + 1^2 = 3
h(cup) + 2^2 = 6

h(down) = 3
…
h(down) + 6^2 = 39 % 7 = 4

Quadratic probing total probe length: 23

The `probe()` method in our `HashTable` class

The return value of the `probe()` method is an integer.

- If the key is in the table, the `probe()` method returns the index it is stored in
- If the key is *not* in the table, the `probe()` method returns the index of the first empty or removed cell encountered during the search for the key

The hash table to the right has been partially filled using linear probing and the hash function from the previous problem. A gray cell indicates that an item has been removed.

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

The `probe()` method in our `HashTable` class

The return value of the `probe()` method is an integer.

- If the key is in the table, the `probe()` method returns the index it is stored in
- If the key is *not* in the table, the `probe()` method returns the index of the first empty or removed cell encountered during the search for the key

The hash table to the right has been partially filled using linear probing and the hash function from the previous problem. A gray cell indicates that an item has been removed.

One of the items in the table has been inserted incorrectly. Which one?

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

The `probe()` method in our `HashTable` class

The return value of the `probe()` method is an integer.

- If the key is in the table, the `probe()` method returns the index it is stored in
- If the key is *not* in the table, the `probe()` method returns the index of the first empty or removed cell encountered during the search for the key

The hash table to the right has been partially filled using linear probing and the hash function from the previous problem. A gray cell indicates that an item has been removed.

One of the items in the table has been inserted incorrectly. Which one?

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

bear

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

bear

h(bear) = 1

This is a removed cell, so bear could still be elsewhere; so linear probe for it

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

bear

h(bear) = 1

This is a removed cell, so bear could still be elsewhere, so linear probe for it

Found bear; return 3

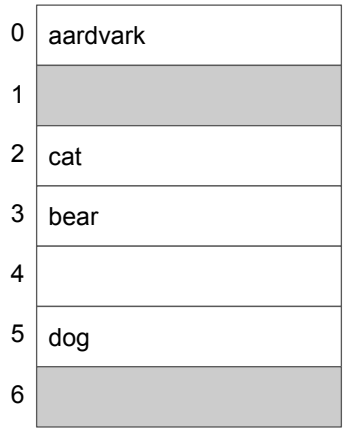
The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

cow

The `probe()` method in our `HashTable` class



Determine the return value of the `probe()` method for:

cow

h(cow) = 2

This cell is occupied, so cow could still be elsewhere; so linear probe for it

The `probe()` method in our `HashTable` class

Determine the return value of the `probe()` method for:

cow

h(cow) = 2

This cell is occupied, so cow could still be elsewhere; so linear probe for it

We have reached an empty cell, so cow can't be elsewhere in the table

Return index 4, so cow can be inserted there

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

buffalo

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

buffalo

h(buffalo) = 1

This is a removed cell, so buffalo could still be elsewhere; so linear probe for it

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

buffalo

h(buffalo) = 1

This is a removed cell, so buffalo could still be elsewhere; so linear probe for it

This is an empty cell, so buffalo can't be elsewhere in the table

Return index 1, since that is the first empty position we encountered, so that buffalo can be inserted

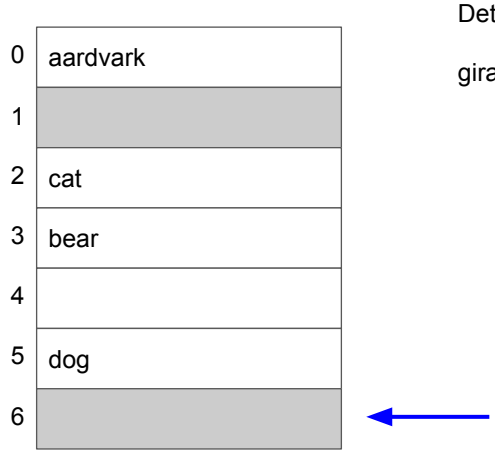
The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

Determine the return value of the `probe()` method for:

giraffe

The `probe()` method in our `HashTable` class



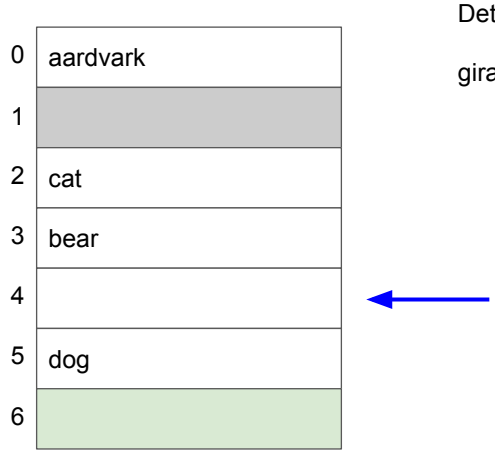
Determine the return value of the `probe()` method for:

giraffe

h(giraffe) = 6

This is a removed cell, so giraffe could still be elsewhere; so linear probe for it

The `probe()` method in our `HashTable` class



Determine the return value of the `probe()` method for:

giraffe

h(giraffe) = 6

This is a removed cell, so giraffe could still be elsewhere; so linear probe for it

This is an empty cell, so giraffe can't be elsewhere in the table

Return index 6, since that is the first empty position we encountered, so that giraffe can be inserted

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

What is the largest probe length that we could have for this table, regardless of its contents?

The `probe()` method in our `HashTable` class

0	aardvark
1	
2	cat
3	bear
4	
5	dog
6	

What is the largest probe length that we could have for this table, regardless of its contents?

7, the size of the table. After 7 positions, the probe sequence repeats, so the probe() method will give up after trying 7 positions.

End of section.

Questions?