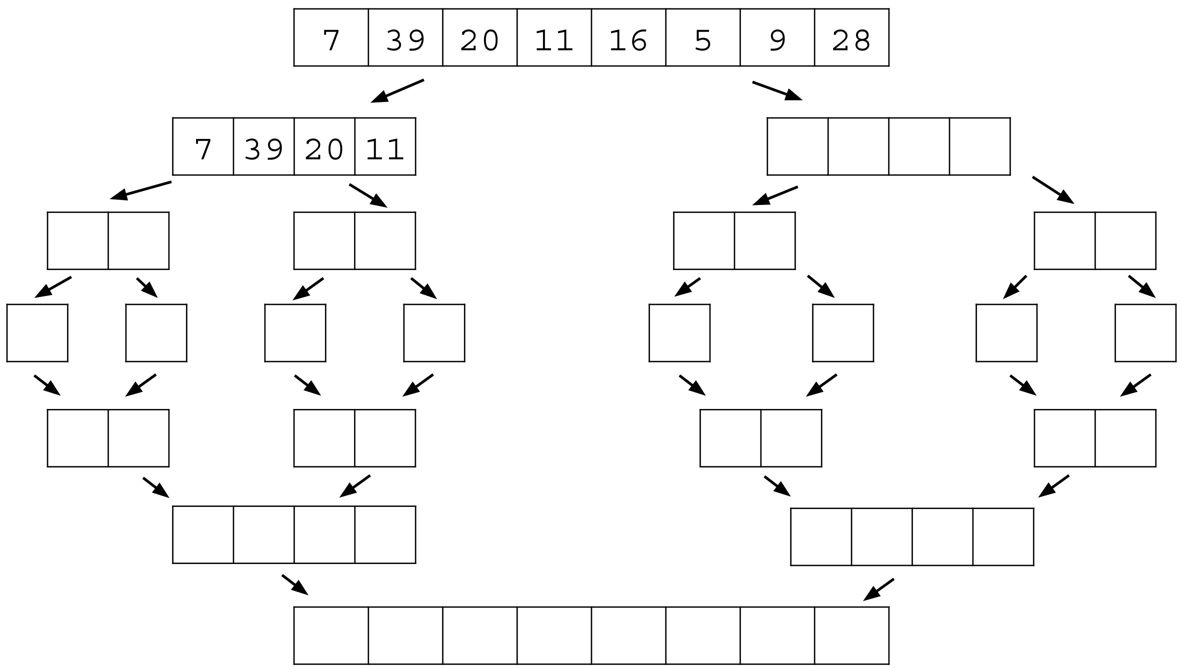
Section 4

CSCI E-22

Will Begin Shortly

Merge sort

- Like quicksort, recursive divide & conquer
- Unlike quicksort, merge sort does not modify the array during the “division” phase of the algorithm
- Instead, sorting is done as subarrays are merged together into a fully sorted subarray
- `merge()` method takes two already sorted subarrays and merges them into a sorted whole



Merge sort

- What major **advantage** does merge sort have over quicksort with respect to **time** complexity?
  - Unlike quicksort, merge sort always divides the current subarray evenly in half, and so its call tree is always perfectly balanced, with height proportional to $\log_2 n$. Therefore, merge sort gives $O(n \cdot \log_2 n)$ performance even in the worst case, whereas quicksort can degenerate into $O(n^2)$ performance if it does not partition evenly and its call tree approaches a height of n.
- What major **disadvantage** does merge sort have compared to quicksort with respect to **space** complexity?
  - Merge sort requires $O(n)$ additional memory on top of the array itself, while quicksort uses only $O(1)$ additional memory.

Radix sort

- Stable, distributive sorting algorithm
- Can be used to sort integers, strings, complex data
- For integers, the algorithm processes individual digits of each element
    - For each element, place it into a “bucket” according to the value of its least significant digit, **maintaining order** in the bucket
    - When you reach the end of the array, repeat the process for the next most significant digit
    - Stop when all elements have been evaluated according to the most significant position of the largest element

41	326	18	1	117	56	86	7	14	221	19	30

first pass (buckets for the ones digit)

0	1	2	3	4	5	6	7	8	9

41	326	18	1	117	56	86	7	14	221	19	30

first pass (buckets for the ones digit)

0	1	2	3	4	5	6	7	8	9
30	41 1 221			14		326 56 86	117 7	18	19

second pass (buckets for the tens digit)

0	1	2	3	4	5	6	7	8	9

41	326	18	1	117	56	86	7	14	221	19	30

first pass (buckets for the ones digit)

0	1	2	3	4	5	6	7	8	9

second pass (buckets for the tens digit)

0	1	2	3	4	5	6	7	8	9
1 7	14 117 18 19	221 326	30	41	56			86	

third pass (buckets for the hundreds digit)

0	1	2	3	4	5	6	7	8	9

Radix sort

- Keeping in mind that radix sort processes its data as a sequence of *m* quantities with *k* possible values, what do *m* and *k* represent in our example?
  - In the example, $m = 3$, because there are 3 positions we're looking at: the 1's position, the 10's position, and the 100's position.
  - In general, if we have one bucket for each value of our given radix, *m* is equal to the positional index of the most significant digit of the largest element. Here, since we're sorting integers in base 10, and we'll have one bucket for each digit 0–9.
  - We can think of *k* as the **number of buckets** we have—that is, it represents the number of possible values we could have for each of *m* quantities. So, as we just mentioned, $k = 10$ in the above example, and in any case in which we're using radix sort to sort integers by significant digit.

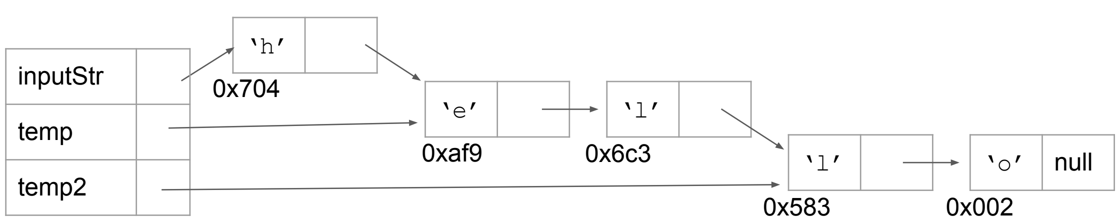
Radix sort

- How many operations did our example above require? How many operations would the example above have required if the elements were already in sorted order? If they were in reverse order?
  - In general radix sort takes $n \cdot m$ steps for an array with *n* elements, since it iterates over all *n* elements of the array a total of *m* times.
  - Our previous example required $12 \cdot 3 = 36$ steps. It would also require 36 steps if it had already been in sorted order, or reverse-sorted order. That is, the number of steps performed by radix sort is dependent only on the values of *n* and *m*, and not on the order of the original array.

Radix sort

- Which sorting method would have been more efficient for sorting the above array: radix sort or merge sort?
  - As discussed in lecture, radix sort is $O(n \cdot m)$, whereas merge sort is $O(n \cdot \log n)$. So radix sort is more efficient than merge sort (and other comparison-based sorting algorithms in $O(n \cdot \log n)$, such as quicksort) when $m < \log n$.
  - Here, $m = 3$ and $\log_2 n = 3.585\ldots$, so radix sort would sort the above example **in fewer steps** than merge sort.

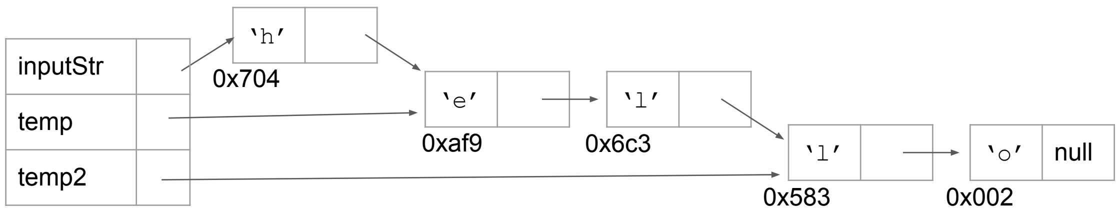
Practice with references



Specify how the following expressions evaluate:

- What is the value of `temp.next`?
- What does `temp.next.ch` evaluate to?
- What does `inputStr.next.next == temp` evaluate to?
- What are some ways we can access the character `'o'`?

Practice with references



How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

End of section.

Questions?

Lecture 5

CSCI E-22

Will Begin Shortly