Searching and Sorting

Searching

- Unordered collection
  - Must check every element
  - Linear-time operation – O(n)

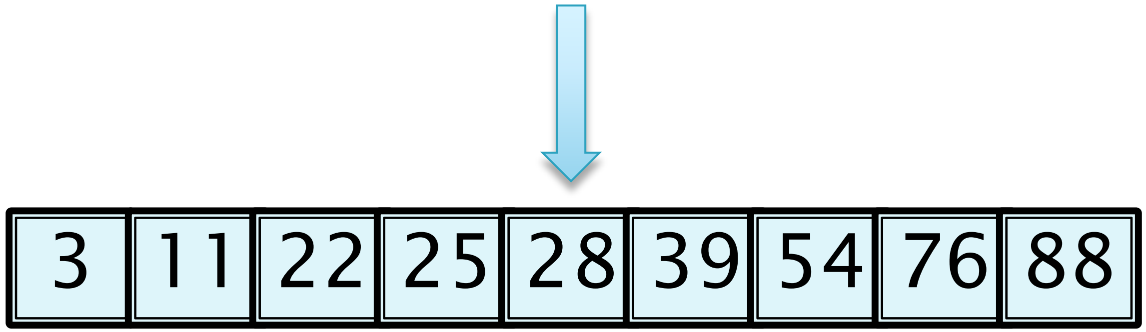
- Ordered collection
  - Exploit order to check only necessary elements
  - Logarithmic-time operation – O(log n)

Binary Search

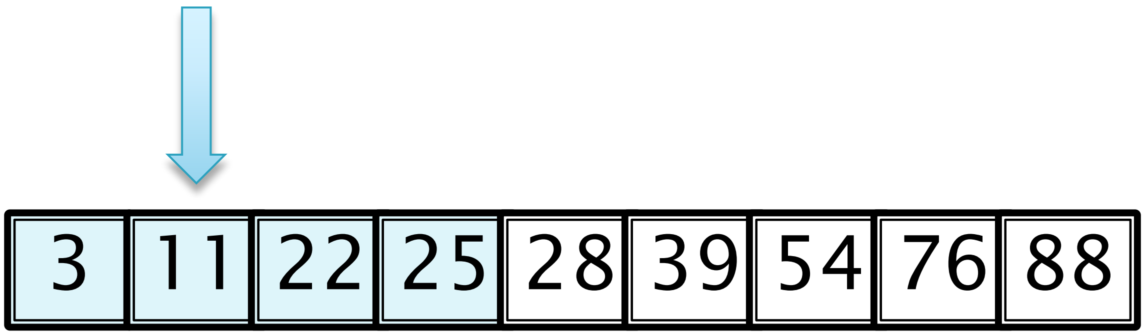
- Like searching a binary search tree
- Elements must be sorted
- Algorithm:
  - Compare the “middle” element with the desired one
  - If the desired element is smaller, search the half of the collection with smaller elements
  - If the desired element is larger, search the half of the collection with larger elements
  - Repeat algorithm with the sub-collection until element found, or sub-collection size reaches zero

3	11	22	25	28	39	54	76	88

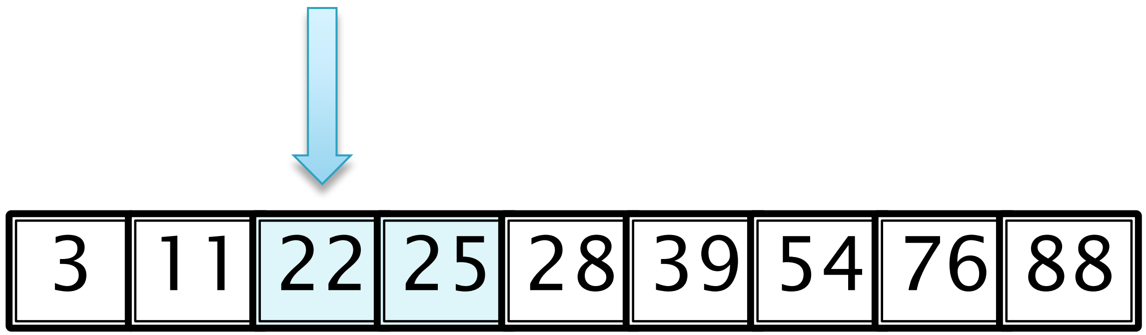
Find element 22



22 < 28, search left



22 > 11, search right



Found element 22

Sorting

- But how do we sort the elements in the first place?
- Isn't it easy to sort things?
  - Human often sort things without exactly knowing how they do it
  - We can scan and recognize patterns that can aid in sorting
  - Computers can only compare two items at once

Isn't it easy?

- Humans often sort things without exactly knowing how they do it
- We can scan and recognize patters that can aid in sorting
- Computers can only compare two items at once

Bubble Sort

- Compare each element with the next one and swap them if needed
- Repeat until no more swaps are required
- Slow ($O(n^2)$ time complexity), but simple

Selection Sort

- Find the largest element not yet sorted
- Swap it with the last element not yet sorted
- Repeat until no more swaps are required

- Some implementations find the smallest element and swap it with the first element

- Also $O(n^2)$ complexity, but more consistent

Insertion Sort

- Sort the last two elements, creating an ordered sublist
- Insert the other elements (one by one) into the sublist so that it grows, while remaining in sorted order

- $O(n^2)$, but faster than Selection or Bubble
- Good when data is already almost sorted
- Good when collection is still receiving elements

Merge Sort

- Repeatedly divide the collection in halves until each sub-collection has only one element
- Merge pairs of adjacent sub-collections such that their elements are sorted

- Has better complexity (O(n logn))
- Can be parallelized to be performed faster
- Typically needs extra memory space to perform merge

Implementation

- Pseudo-code and/or code for algorithms are available on the course website

- Implementing merge sort is left as an exercise