CSE 373
Data Structures and Algorithms

Lecture 6: Searching / Running times in practice

Searching and recursion

- Problem: Given a sorted array of integers and an integer *i*, find the index of any occurrence of *i* if it appears in the array. If not, return -1.
  - We could solve this problem using a standard iterative search; starting at the beginning, and looking at each element until we find *i*
  - What is the runtime of an iterative search?

- Since the array is sorted, we can do better.

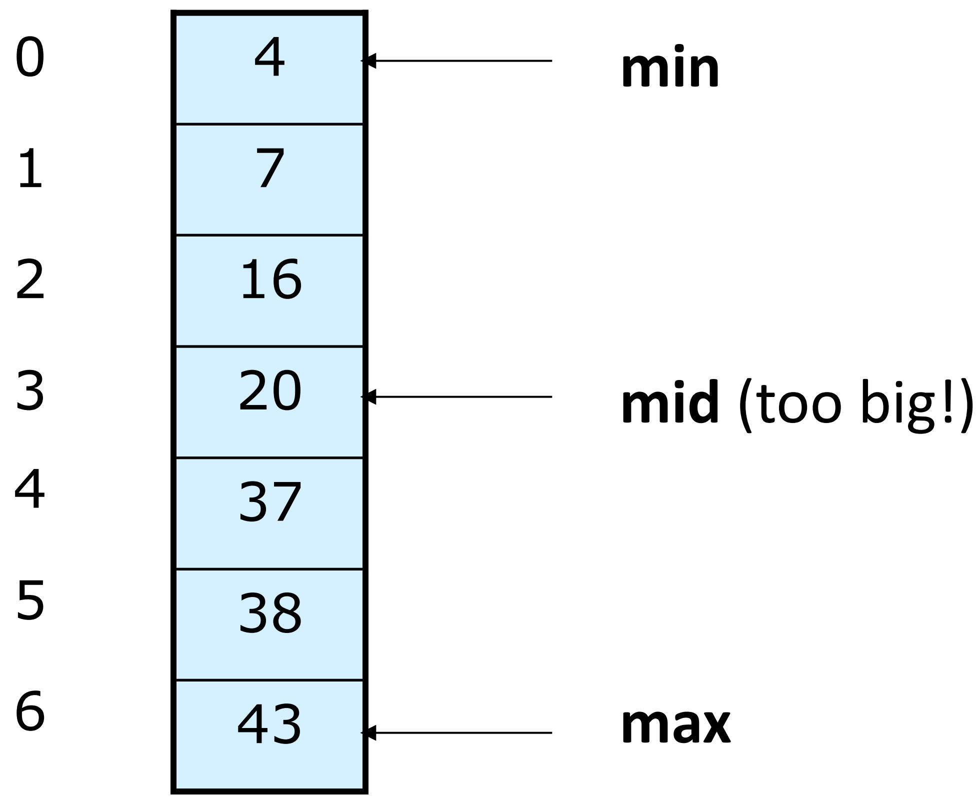
Binary search algorithm

- Algorithm idea: Start in the middle, and only search the portions of the array that might contain the element i. Eliminate half of the array from consideration at each step.
  - Can be written iteratively, but is harder to get right

- Called binary search because it chops the area to examine in half each time
  - Implemented in Java as Arrays.binarySearch in java.util package

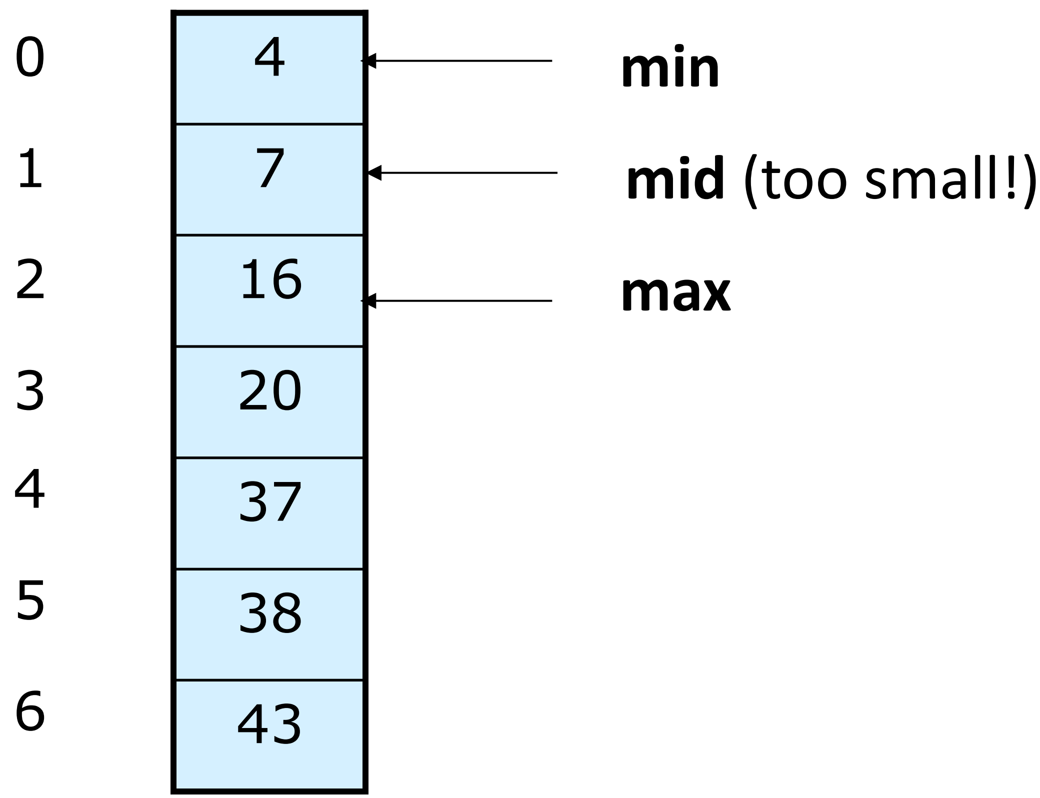
Binary search example

i = 16



Binary search example

i = 16



Binary search example

i = 16

0	4
1	7
2	16
3	20
4	37
5	38
6	43

min, mid, max (found it!)

Binary search pseudocode

binary search array *a* for value *i*:

    if all elements have been searched,

        result is -1.

    examine middle element *a*[*mid*].

    if a[*mid*] equals *i*,

        result is *mid*.

    if *a*[*mid*] is greater than *i*,

        binary search left half of *a* for *i*.

    if *a*[*mid*] is less than *i*,

        binary search right half of *a* for *i*.

Divide-and-conquer

- **divide-and-conquer** algorithm: a means for solving a problem that first separates the main problem into 1 or more smaller problems, then solves each of the smaller problems, then uses those sub-solutions to solve the original problem
  - 1: "divide" the problem up into pieces
  - 2: "conquer" each smaller piece
  - 3: (if necessary) combine the pieces at the end to produce the overall solution

  - binary search is one such algorithm

Runtime of binary search

- How do we analyze the runtime of binary search and recursive functions in general?

- Binary search either exits immediately, when input size <= 1 or value found (base case), or executes itself on 1/2 as large an input (rec. case)
  - $T(1) = c$
  - $T(2) = T(1) + c$
  - $T(4) = T(2) + c$
  - $T(8) = T(4) + c$
  - ...
  - $T(n) = T(n/2) + c$

- How many times does this division in half take place?
  - For more rigorous proof, lookup "recurrence relation" and "Master theorem"

Master Theorem (for reference only)

- A recurrence written in the form

  T(n) = a * T(n / b) + f(n)

  (where f(n) is a function that is O(n^k) for some power k)
  has a solution such that

$$T(n) = \begin{cases} O(n^{\log_b a}), & a > b^k \\ O(n^k \log n), & a = b^k \\ O(n^k), & a < b^k \end{cases}$$

- This form of recurrence is very common for divide-and-conquer algorithms

Runtime (for reference only)

- Binary search is of the correct format:

  $T(n) = a * T(n / b) + f(n)$

  $T(n) = T(n/2) + c$

  $a = 1, b = 2$

  $f(n) = c = O(1) = O(n^0)$ ... therefore $k = 0$

- $a = b^k$

  $1 = 2^0$, therefore:

  $T(n) = O(n^0 \log n) =$ **$O(\log n)$**