CS3000
May 10th (W)

Admin

- Short HW1 due 5/11 9pm
- Long HW1 out 5/11, due 5/16 9pm
- Laney's OH today 12-2
- TA OH on Khoury app

Agenda

1. Divide and Conquer
2. Binary Search
3. Run-time recurrences

Recap

# times test is executed		
	1	if x == y
	n+1	for i = 1 to A.length
	n	for j = A.length downto 2

Selection Sort on $\langle 7, 3, 9, 2 \rangle$
(i = location of next min)
After iteration completes when i is...

1 $\langle \underline{2}, 3, 9, 7 \rangle$
↳ sorted!

2 $\langle \underline{2, 3}, 9, 7 \rangle$
↳ sorted!

$\lceil 7.2 \rceil = \lceil 7.9 \rceil = \lceil 8 \rceil = 8$ ceiling

$\lfloor 7.2 \rfloor = \lfloor 7.9 \rfloor = \lfloor 7 \rfloor = 7$ floor

$\boxed{\text{1. Divide + Conquer}}$

So far...

	Linear Search	Selection Sort
time	$\Theta(n)$ (w.c.)	$\Theta(n^2)$ (b.c. == w.c.)
space	$\Theta(1)$	$\Theta(1)$

Can we do better?

↳ try a known technique!

First technique:

$\boxed{\text{Divide + Conquer}}$

- Have an input, and a problem to solve
- make input smaller
  - remove one thing
  - remove half the things
  - subtract / divide the input

  } divide
- Solve the smallest version } conquer
- put the small solutions together for bigger problem } combine

Solving with D+C also gives us new ~~problem~~ opportunity

- analyze the run-time of a D+C algorithm
- $T(n)$ = # steps to solve problem of size n

↳ $T(n) = T(n-1) + \_\_ + \_\_\_$

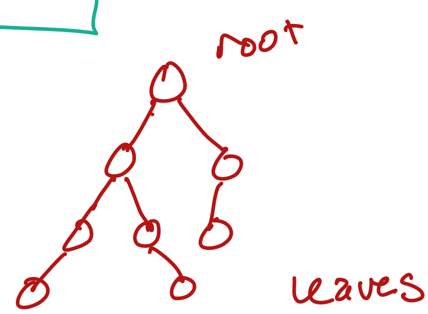
↳ can't bound this! need to solve it

2. Binary Search

↳ search is figured out: BinSearch is the way we search

Binary Search is always D+C

- Data in an array (sorted)
- Data in a Binary Search Tree ☆



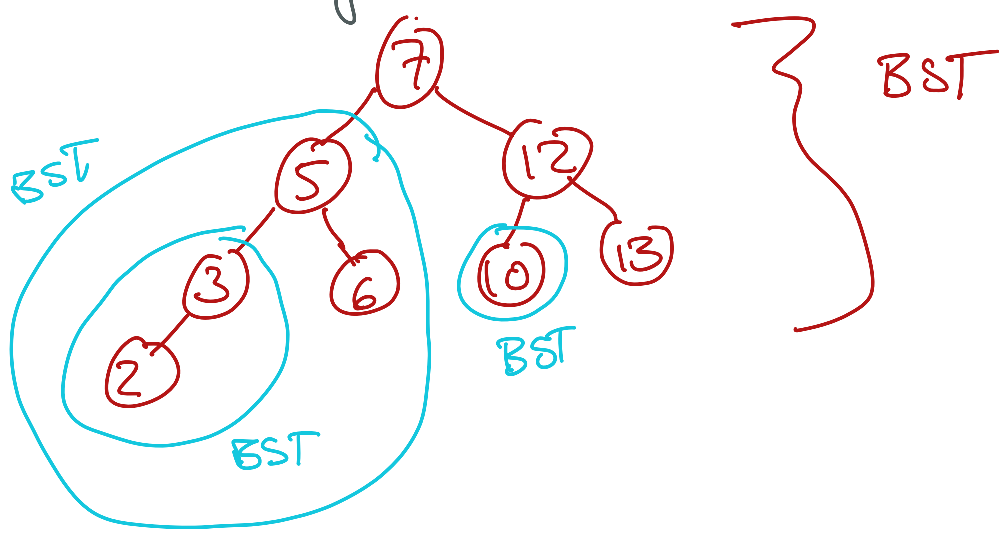
Tree – is a type of graph (DAG) has a root, leaves

Binary Tree – each node has at most 2 children

Binary Search Tree -

every node's value is greater than its left subtree

every node's value is less than its right subtree



Assumptions

- distinct values
- tree already exists
- tree is balanced $\rightarrow$ height is $\Theta(\lg n)$
- refer to tree by its root (x)

    x has a value (x.value)

    x has a left child (x.left)

    x has a right child (x.right)

    empty tree when x is $\boxed{\text{null}}$

High level pseudo code

tree (x)
key → looking for
return boolean

- Is x null?
  If so, done! not there
- Is key == x.value?
  If so, done! there.
- Is key < x.value?
  If so, search left subtree
- Is key > x.value?
  If so, search right subtree

one or the other!

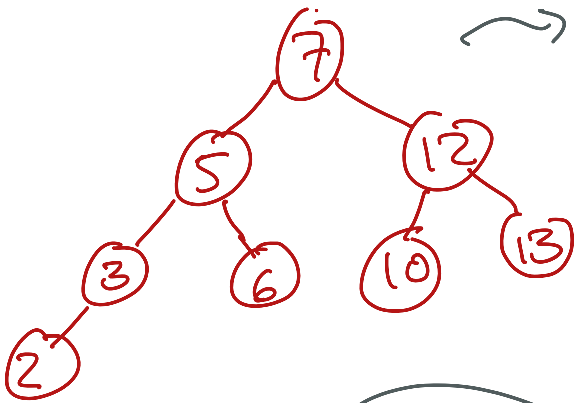
$T(n)$ = # steps to solve problem of size n

$= T(n/2) + c$

first two checks

going left or right

10:50



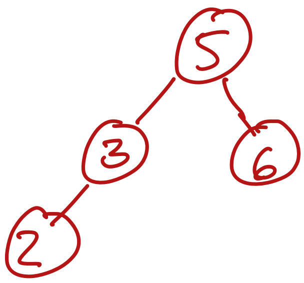
→ a BST

key = 2

compare 2 vs. 7

$2 < 7$

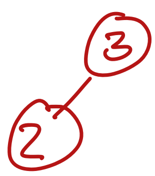
so, search the left subtree



compare 2 vs. 5

$2 < 5$

so, search left subtree



compare 2 vs. 3

$2 < 3$

so, search left subtree

compare 2 vs. 2

done! Found it :)

$T(n) = T(n/2) + c$ ⟶ recurrence

$T(1) = 2$ ⟶ base case

solve the recurrence

+

put a bound on it

technique to solve: $\boxed{\text{iteration method}}$

- plug in smaller values of the problem size
- find a pattern for kth iteration
- pick a value for k to get to the base case

1st iteration: $T(n) = T(n/2) + c$

plug in smaller: $T(n/2) = T(n/4) + c$

2nd iteration: $T(n) = T(n/4) + c + c$

$= T(n/4) + 2c$

plug in smaller: $T(n/4) = T(n/8) + c$

3rd iteration: $T(n) = T(n/8) + c + 2c$

$= T(n/8) + 3c$

...

kth iteration: $T(n) = T\left(\frac{n}{2^k}\right) + k \cdot c$

B.C.

$T(1) = 2$

Choose a value of k s.t. $T(n/2^k) = T(1)$

$\frac{n}{2^k} = 1 \rightsquigarrow$ Solve for k

$\boxed{k = \lg n}$ $\rightsquigarrow \frac{n}{2^k} = 1$

$n = 2^k$

$\lg n = \lg(2^k)$

$\lg n = k$

Plug in lgn for k

$T(n) = T(n/2^k) + k \cdot c$

$= T(n/2^{\lg n}) + \lg n \cdot c$

$= T(n/n) + c \cdot \lg n$

$= T(1) + c \cdot \lg n$

$= 2 + c \cdot \lg n$ → # steps

Bound: $\Theta(\lg n)$

Binary Search is logarithmic in the worst case