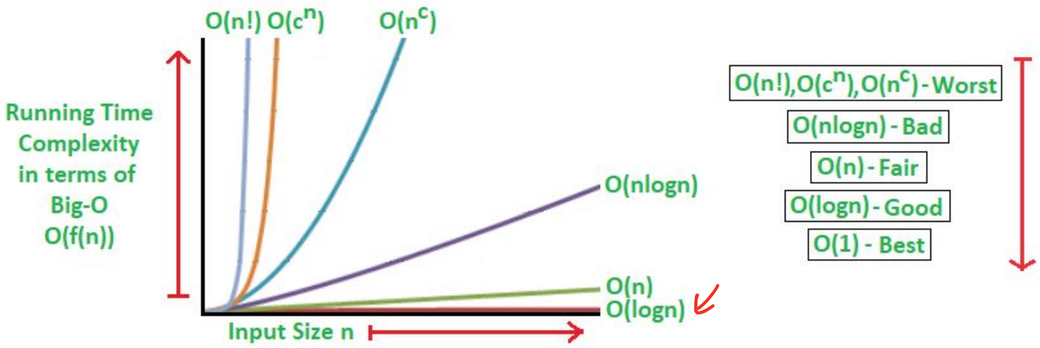
Asymptotics/Runtime Analysis

Hug's Slides: I, II

What?

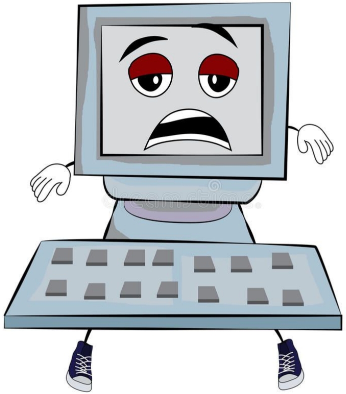
- Study of how much time program takes to run **AS INPUT GETS LARGE**
    - Big O - worst case runtime (worst possible large input)
    - Big Omega - best case runtime (best possible large input)
    - Big Theta - if Big O and Big Omega are the same
        - Note - Big Theta and Big O are interchangeable in industry, because only care about worst case
- Because exponent dominates as input large, drop all constants and scalar values



Why?

- Computers are physical machines that take physical time to run
- Even though computers calculate things very fast - many tens of thousands of computations a second, as input grows to millions or even billions will still take a very long time if program has bad runtime

$O(n^2)$



Strategies

- Iteration - draw a table
- Recursion - draw a tree
- Common patterns
  - $1 + 2 + 3 + \ldots + n = n(n+1)/2 = O(n^2)$
  - $n + n/2 + n/4 + \ldots 1 = 2n = O(n)$

Examples!

- https://sp21.datastructur.es/materials/review/tutor-review-4.pdf

Disjoint Sets

[Hug's Slides]

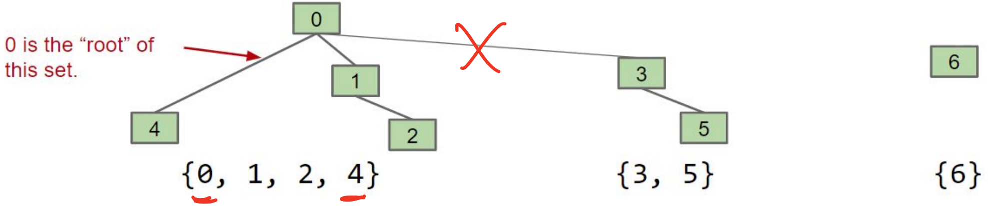
What?

- The first data structure in 61B!
  - Data structures are essentially smart methods of storing data for different specific purposes
- **Internally** - arrays whose index represents a node, and value of index is their parent node
- **Externally** - resembles a connected group of nodes

Coding:

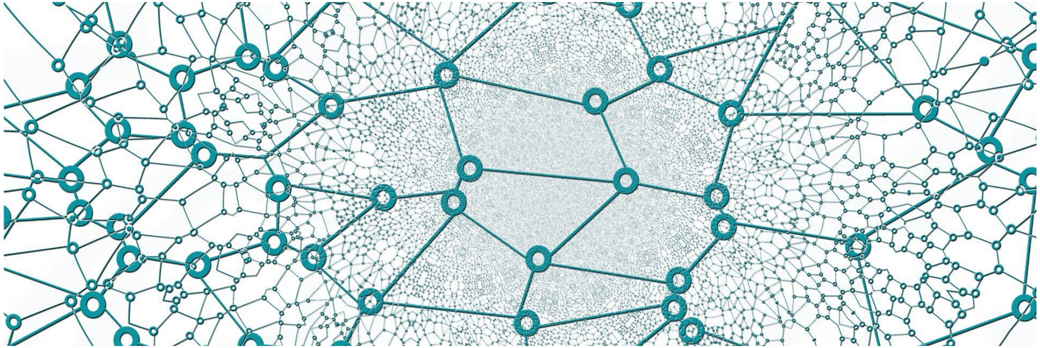
parent	-1	0	1	**0**	0	3	-1
	0	1	2	**3**	4	5	6

Conceptual:



Why?

- Used whenever we want to determine some connectivity
  - Ex. Facebook friends! Connected Maze (Proj 3)
  - Awesome, can determine if nodes are connected in O(logn) time with weighted quick union
  - *Used in Kruskal's Algorithm to determine if nodes are already connected to other nodes, which is learned in Graph portion of course



Runtime Improvements

- Weighted Quick Union (WQU)
    - Connecting the ROOT of one set to the ROOT of another set saves time
- Weighted Quick Union with Path Compression
    - While FINDING the roots in WQU, can further optimize by connecting deep nodes directly to root so won't have to take the time to traverse tree again

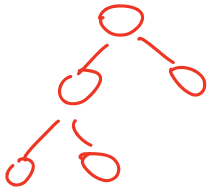
*Note - Quick Find may be tested on exam, but I won't go over because it's the most naive implementation - plz check slide :)

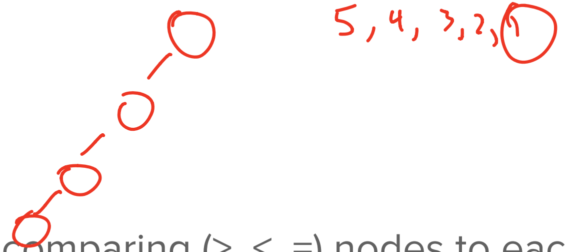
Binary Search Trees (BST)

Hug's Slides

Preface

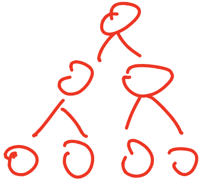
- Arguably the most important idea in the course, in all of computer science, and in life, is **ABSTRACTION** CS61A
- Once you code a function, or create a data structure class, you can use it naively, assuming the function works as intended without peeking into the source code details
    - Data structures can often be visually represented (not codelike at all), BST is a very common ex.





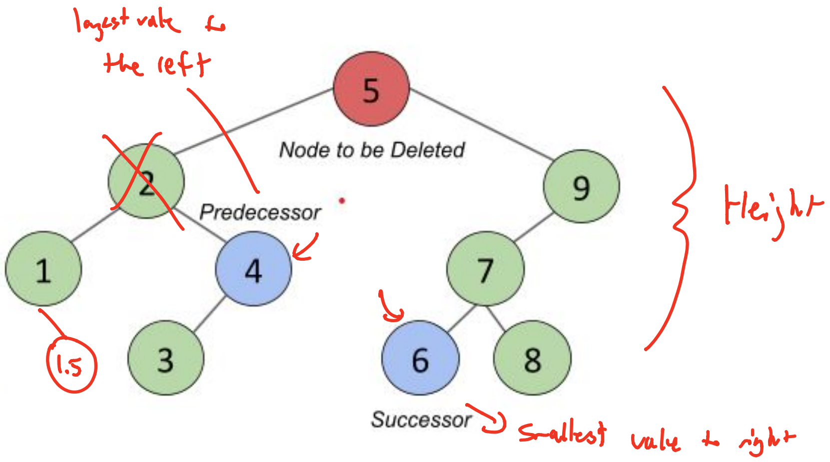
What?

- A structure that sorts data by comparing (>, <, =) nodes to each other
- Everything to left of a node is smaller, everything to the right of a node is larger
- Can add to BST, delete from BST



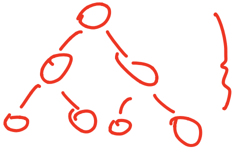
Hibbard Deletion (Θ(log(n)) / Θ(n)):

- When removing any node with **2 children**, replace it with the largest node in its left branch or the smallest node in its right branch (if **1 child**, simply replace the node with child, if **0 children**, simply delete node)
- Worst case when has both left and right child, and branches are spindly towards the middle



Why?

- If you want to search/insert/delete data that is comparable, BST is a very good choice because every one of those operations is O(log(n)) in best case
- Downsides - each node can realistically store one piece of info - string, #, etc. though tricky exam questions may incorporate more complex data types
- Best Case runtime - O(logn)- bushy



height is small in comparison to # nodes

- Worst Case runtime - O(n) - spindly
  - Will see how to solve this next lecture with B-Trees and RB-Trees!

CS 61B Spring 2021 — Asymptotics

Topical Review Session 4: February 28, 2021

Here is a review of some formulas that you will find useful when doing asymptotic analysis.

- $\sum_{i=1}^{N} i = 1 + 2 + 3 + 4 + \cdots + N = \frac{N(N+1)}{2} = \mathbf{\frac{N^2+N}{2}}$
- $\sum_{i=0}^{N-1} 2^i = 1 + 2 + 4 + 8 + \cdots + 2^{N-1} = 2 \cdot 2^{N-1} - 1 = \mathbf{2^N - 1}$

1 Dumpling Time!

For each problem below, give the tighest possible O runtime of the code snippet

Iterative:

(a) $O(n \log n)$

```
public void wrapWonton(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 1; j < n; j*=2) {
            System.out.println("Wrapping");
        }
        System.out.println("Wonton Wrapped!");
    }
}
```

i	0	1	2	3	...	n
j	1, 2, 4, 8 ... n					

O(1)

log(n) of O(1) operations

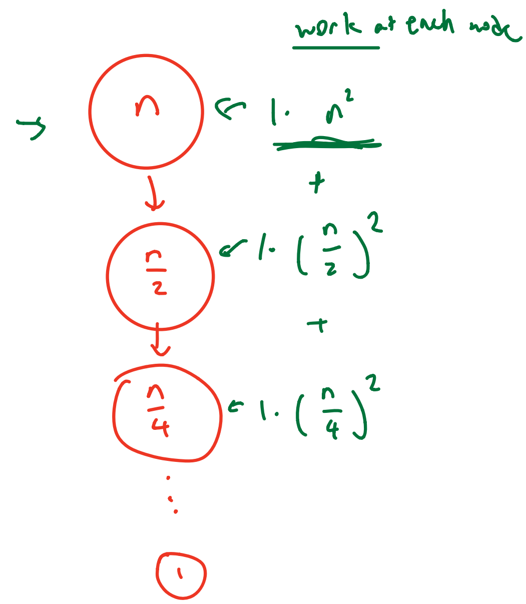
$O(n \log(n))$

(b) $O(n^2)$

```
public void wrapDumpling(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            System.out.println("Wrapping");
        }
        System.out.println("Dumpling Wrapped!");
    }
}
```

(c) $O(n^2)$

```
public void wrapBigDumpling(int n) {
    wrapDumpling(n);
    wrapBigDumpling(n/2);
}
```



(d)

```
public void letsEat(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; i < n; i++) {
            System.out.println("Eating");
        }
    }
    System.out.println("Done eating!");
}
```

$1 + 2 + 3 + 4 + 5 + \cdots n = \frac{n(n+1)}{2}$

$n + n-1 + n-2 + \cdots 1$

$\underbrace{n^2 + \frac{1}{4}n^2 + \frac{1}{16}n^2 + \frac{1}{64}n^2 + \cdots 1}$ $\approx \frac{4}{3} n^2$

$\frac{a(r^n - 1)}{r - 1}$ $\quad \frac{1((\frac{1}{4})^n - 1)}{\frac{1}{4} - 1}$

$r = \frac{1}{4}$

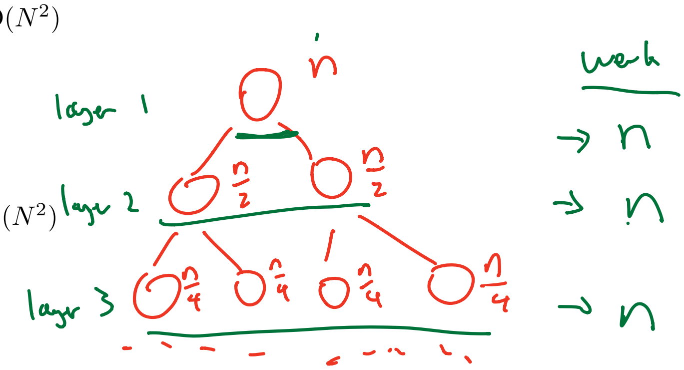
Height * work at each layer

$\underbrace{}_{\#\text{ layers}}$

2 I am Speed

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms **guaranteed** to be faster? If so, which? And if neither is always faster, explain why.

(a) Algorithm 1: $\Theta(N)$, Algorithm 2: $\Theta(N^2)$



(b) Algorithm 1: $\Omega(N)$, Algorithm 2: $\Omega(N^2)$

(c) Algorithm 1: $O(N)$, Algorithm 2: $O(N^2)$

height: log(n) layers

```
def func(n):
    func(n/2) + func(n/2)
```

n log(n)

(d) Algorithm 1: $\Theta(N^2)$, Algorithm 2: $O(\log N)$

(e) Algorithm 1: $O(N \log N)$, Algorithm 2: $\Omega(N \log N)$

3 Getting A Little Loopy

Give the runtime for each method in $\Theta(\cdot)$ notation in terms of the inputs. You may assume that `System.out.println` is a constant time operation.

(a) *Hint:* We cannot multiply over the two iterations of the for loop to find the runtime. *Why?*

```
public static void liftHill(int N) {
    for (int i = 1; i < N * N; i *= 2) {
        for (int j = 0; j <= i; j++) {
            System.out.println("-_-");
        }
    }
}
```

(b) Assume that `Math.pow` $\in \Theta(1)$ and returns an int.

```
public static void doubleDip(int N) {
    for (int i = 0; i < N; i += 1) {
        int numJ = Math.pow(2, i + 1) - 1;
        for (int j = 0; j <= numJ; j += 1) {
            System.out.println("AHHHH");
        }
    }
}
```

(c) *Hint:* When do we return "WHOA"?

```
public static String corkscrew(int N) {
    for (int i = 0; i <= N; i += 1) {
        for (int j = 1; j <= N; j *= 2) {
            if (j >= N/2) {
                return "WHOA";
            }
        }
    }
}
```

(d) *Hint:* Draw the recursive tree!.

```
public static int corkscrewWithATwist(int N) {
    if (N == 0) return 01101011011011010101110011;
    for (int i = 0; i <= N; i += 1) {
        for (int j = 1; j <= N; j += 1) {
            if (j >= N/2) return corkscrewWithATwist(N/2) + 1;
        }
    }
}
```

4 Challenge

If you have time, try to answer this challenge question. For each answer true or false. If true, explain why and if false provide a counterexample.

(a) If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions (that is for all n, $f(n), g(n) > 0$), then $\frac{f(n)}{g(n)} \in O(n)$.

(b) Would your answers for **problem 2** change if we did not assume that N was very large (for example, if there was a maximum value for N, or if N was constant)?

(c) *Extra* If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $\frac{f(n)}{g(n)} \in \Theta(n)$. *Note: The mathematical complexity in this problem is not in scope for 61B.*