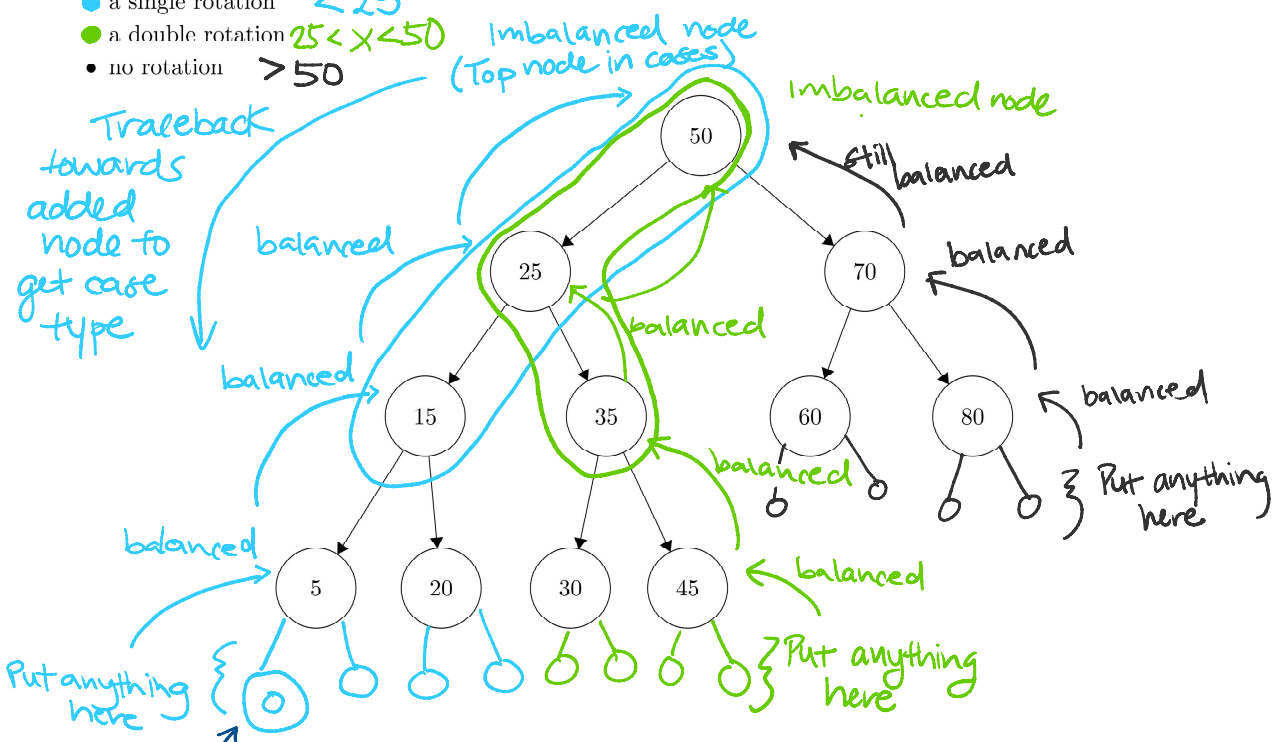
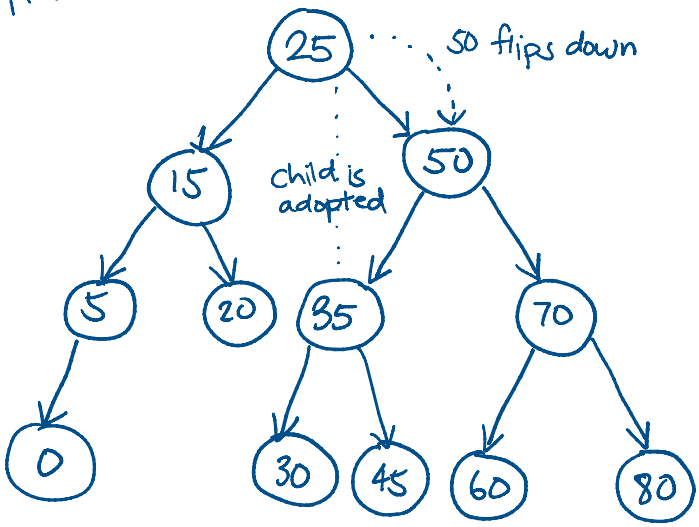
1. (a) For the given AVL tree below, list the range of values that would cause a:

- a single rotation <25
- a double rotation 25<x<50
- no rotation >50



(b) Draw the resulting AVL tree (and intermediary steps to show your work) after inserting the value of 0.

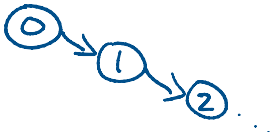
Step 1: Add 0 in correct place
Step 2: Find imbalanced node → 50
Step 3: Find case → line
Step 4: Do rotation



See resources page visual go for AVL practice + section slides as well

(c) Give an **ordered sequence** of 5 values to insert starting from an empty tree, in which you would prefer to use an AVL tree instead of a BST.

[0, 1, 2, 3, 4] → degenerate BST



AVL will balance it

2. Demonstrate that $log(n^2) + 5n(2-n)$ is dominated by n by finding a c and n_0. This is an equivalent question to: show that $log(n^2) + 5n(2-n) \in \mathcal{O}(n)$. Show your work.

Definition Reminder: Dominated By / Big Oh

A function $f(n)$ is dominated by $g(n)$ when...

(a) There exists two constants $c > 0$ and $n_0 > 0$...

(b) Such that for all values of $n \geq n_0$...

(c) $f(n) \leq c \cdot g(n)$ is true.

(d) The previous statements are equivalent to $f(n) \in \mathcal{O}(g(n))$

See quickcheck week 2 for more practice

① Split into seperate terms

log identity

$\log(n^2) \leq c_1 n$ for $n \geq n_1$

$2\log(n) \leq 2n$ for $n \geq 1$

$5n \cdot 2 \leq c_2 n$ for $n \geq n_2$

$10n \leq 10n$ for all n

$5n \cdot (-n) \leq c_3 n$ for $n \geq n_3$

$-5n^2 \leq (1)n$ for $n \geq 1$

② Compile to get c and n_0

$\log(n^2) + 5n(2-n) \leq 2n + 10n + n$

$= 13n$ for all $n \geq 1$

c

$n_0 = \max(n_1, n_2, n_3)$

3. You are a Software Engineer working on a new version of Super Smash Bros. at Nintendo. Your team decides that they want to improve the performance of the game by changing the implementations of the Dictionaries that are currently being used. For each scenario, list some pros and cons of the current implementation, and pick another implementation from the list below that will improve performance. Justify your approach.

Unsorted Array Dictionary, Tree Dictionary, Hash Dictionary

(a) Super Smash Bros. has the ability to assign personalized Mii characters a fighting type (eg. Mii Brawler, Mii Shooter, Mii Sword Fighter). The names of the Mii's are the keys to an Unsorted Array Dictionary, and the fighting types are the values. Users want to be able to quickly find the Mii character that they want by searching for the name.

Pros	Cons
- can take less memory - easy to implement - easy to add/remove	- find() is slow

Hash Dictionary because its fast (assume names are usually unique and there are few collisions

Tree Dictionary because it is sorted and can maybe implement autocomplete

(b) All the characters except for the Mii's are from other franchises (less than 20). A Hash Dictionary with a default initial capacity of 100 stores the franchise name as keys and a List of characters from that franchise as values. No new franchises or characters are being added anymore. We use the Dictionary to print out all the characters' name organized by game in the credits in any order. (Hint: memory usage)

Pros	Cons
- Find is fast	- too much unused memory

Unsorted Array Dictionary because it takes up less space and needs to traverse entire array anyways so find is not important

4. For the following question, we have a hash table with separate chaining. The Hash Table's initial internal capacity is 5. Its buckets are implemented using linked list where new elements append to the end. But in this Hash Table, you are worried about elements being too congested, so you make sure to resize your Hash Table with $\lambda = 0.80$. Resizing your Hash Table will double your initial internal capacity.

   Fill out the table below with the final state of the Hash Table after inserting following key-value pairs in the order given using hashCode function

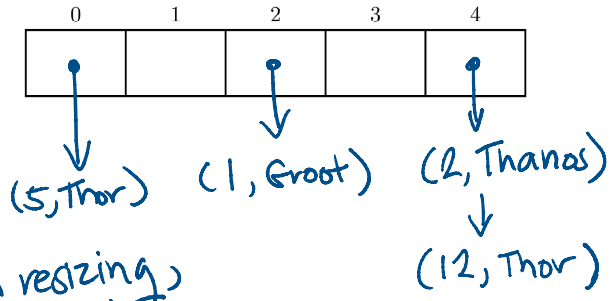
```
1  public int hashCode(int input) {
2      return 2 * input % arr.length;
3  }
```

* checking load before or after insertion doesn't matter

$\lambda = \frac{0}{5} = 0$ $\lambda = \frac{1}{5}$ $\lambda = \frac{2}{5}$ $\lambda = \frac{3}{5}$ $\lambda = \frac{4}{5}$ (Resize!)

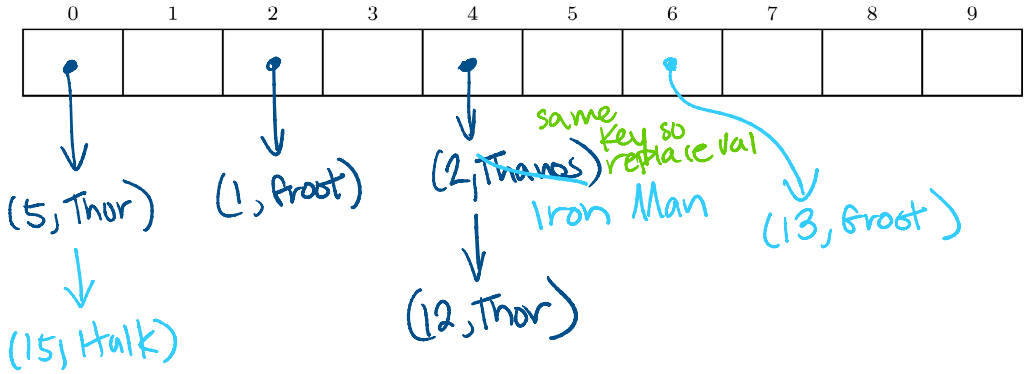
~~(2, Thanos)~~, ~~(12, Thor)~~, ~~(1, Groot)~~, ~~(5, Thor)~~, ~~(15, Hulk)~~, (2, ~~Iron~~ Man), (13, ~~Groot~~)

Before Resizing



When resizing, move everything to new table first

After Resizing



5. Code Modeling. Consider the following *intersection* method that find the intersection between two arrays. Assume that neither array has duplicates.

```
1  public static int[] intersection(int[] A, int[] B) {
2      HashSet<Integer> setOfA = new HashSet<Integer>();
3      HashSet<Integer> result = new HashSet<Integer>();
4      for (int num : A) {                          -> M
5          setOfA.add(num);
6      }
7      for (int num : B) {                          -> N
8          if (setOfA.contains(num)) {
9              result.add(num);
10         }
11     }
12     int[] result_arr = new int[result.size()];
13     int ind = 0;
14     for (int num : result) {                     -> min(M,N) or constant if no intersection
15         result_arr[index] = num;
16         index += 1;
17     }
18     return result_arr;
19 }
```

Answer the following questions about the runtime of the *union_count* method. Consider M is the size of input A and N is the size of input B. Assume the given arrays do not contain duplicates in themselves.

(a) Give simplified tight big-O bounds for the worst-case and best-case runtime of #5 in terms of N and M.

Every addition has a collision $O(M)$ ← worst-case; best-case → $O(1)$ Every addition has no collision

(b) Give simplified tight big-O bounds for the worst-case and best-case runtime of #8 & #9 in terms of N and M?

Contains() does not find num right away collisions during add $O(M+N)$ ← worst-case; best-case → $O(1)$ Contains() finds num right away No collisions in add

(c) Give the overall simplified tight big-O bound of worst-case and best-case runtime of *intersection* in terms of N and M?

Lines 2-6
worst: $c_1 + M * M$
best: $c_1 + M * 1$

Lines 7-11
worst: $c_2 + N(M+N)$
best: $c_2 + N(1)$

Lines 12-18
worst: $c_3 + \min(M,N)$
best: c_3

worst:
$O(c_{all} + M^2 + N(M+N) + \min(M,N)) = O(\max(M^2, N^2))$

best:
$O(c_{all} + M + N) = O(\max(M,N))$

6. General Asymptotics. For each of the following code blocks, give a tight big-Theta bound of the runtime (if Theta doesn't exist, give the Omega and big-O bound instead). Best + Worst case

(a)

```
public static void someMethod1(int n) {
    for (int i = 1; i < n; i *= 2) {
        int j = 0;
        while (j < n) {
            j += 1;
        }
    }
}
```

n → logn

$\Theta(n \log n)$ for both best and worst cases

```
// let n be some number
someMethods1(n);
```

(b)

```
public static void someMethod2(int[] arr) {
    for (int i = 0; i < arr.length; i += 1) {
        int j = i + 1;
        while( j < arr.length) {
            if (arr[i] = arr[j]) {
                return;
            }
            j += 1;
        }
    }
}
```

Method returns if 2 elements are equal

Worst: $\Theta(n^2)$ no duplicates

Best: $\Theta(1)$ duplicates are first 2 elements

```
// let n be the length of arr
someMethods2(arr);
```

(c)

```
public static void someMethod3(int n) {
    int m = (int) ((15 + Math.round(3.2 / 2)) *
(Math.floor(10 / 5.5) / 2.5) * Math.pow(2, 5));
    for (int i = 0; i < m; i++) {
        System.out.println("hi");
    }
}
```

m is a constant / not dependent on n

loop only depends on m

$\Theta(1)$

```
// let n be some number
someMethods3(n);
```

7. Consider the following recurrence:

base case/leaf node work

find last level

input

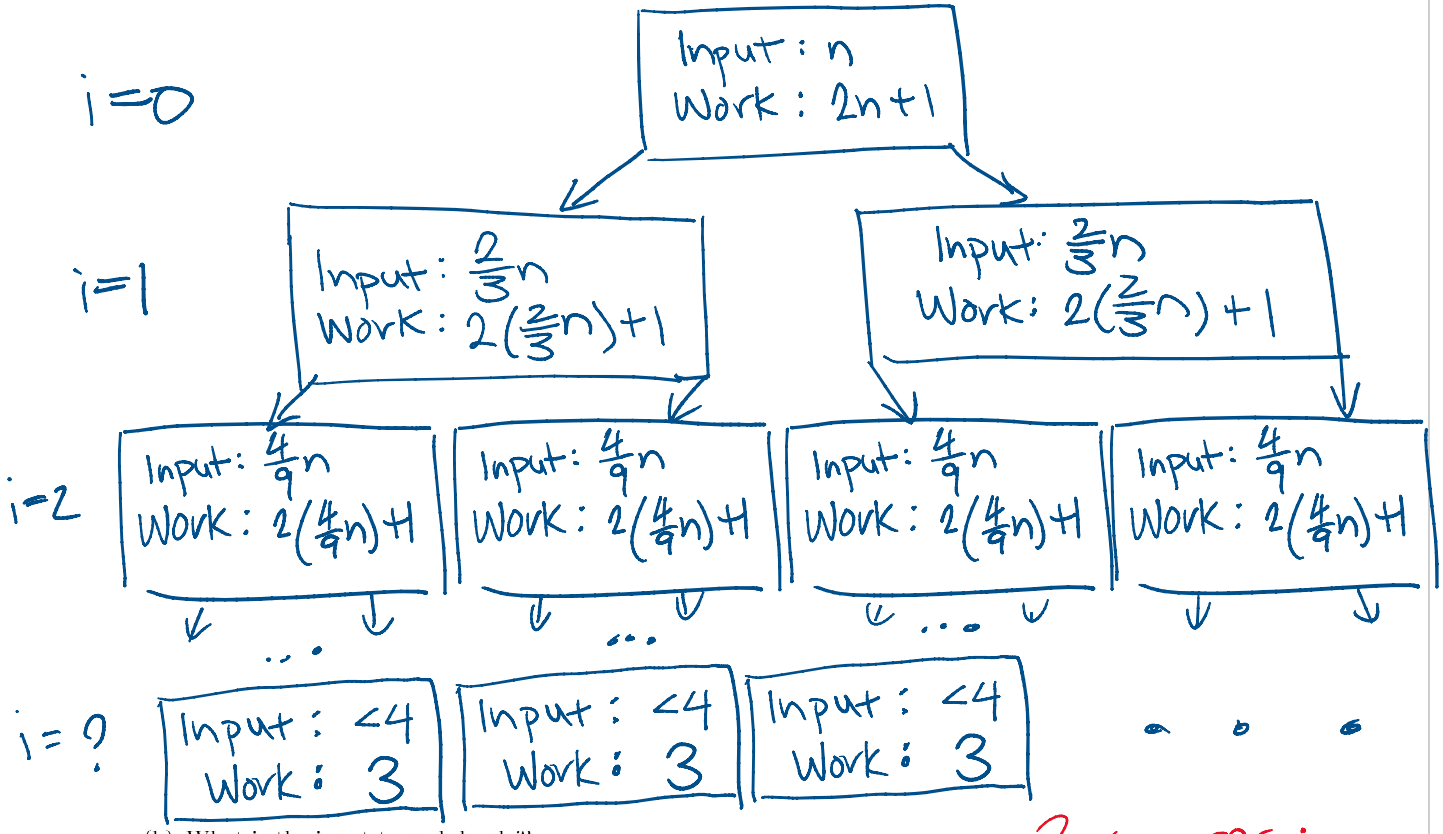
$$T(n) = \begin{cases} 3, & \text{if } x < 4 \\ 2T(2n/3) + 2n + 1, & \text{otherwise} \end{cases} \quad (1)$$

branches

recursive node work

We want you to find an exact closed for this recurrence by using the tree method. Show your work in each part.

(a) Draw the recurrence tree. Your drawing must include the top 3 levels as well as a portion of the final level. Inside each node include the **input** and **work** done for this node. Label the i for each level except for the last one.



(b) What is the input to each level i?

$(2/3)^i n$

(c) How many nodes are at level i?

branches 2^i

(d) What is the total work done per recursive level i?

(# nodes at i)(work/node at i) $= 2^i \left(2\left(\left(\frac{2}{3}\right)^i n\right) + 1\right)$

(e) What value of i does the last level of the tree occur at?

$\left(\frac{2}{3}\right)^i n < 4 \xrightarrow{\text{solve for } i}$ take $\log_2$ of both sides $\Rightarrow i < \frac{\log_2 4 - \log_2 n}{\log_2 2 - \log_2 3} \xrightarrow{\text{calculator}} i < -3.41 + 1.71 \log_2 n$

(f) What is the total work done by the base case (i.e. last level) of the tree?

(# base case nodes) x (base case work/node)

$= 2^{\lceil -3.41 + 1.71 \log_2 n - 1 \rceil} \times 3$

base case level = largest integer that satisfies inequality $= \lceil -3.41 + 1.71 \log_2 n - 1 \rceil$

* Don't need to do anything this hard on exam)

See Resources: Recurrences / Tree-Method + section wksts for practice

(g) Combine your answers to get a final expression with summations for the total work done by the recurrence.

$$\left(\sum_{i=0}^{\lceil -3.41+1.71\log_2 n-1\rceil -1} 2^i\left(2\left(\left(\tfrac{2}{3}\right)^i n\right)+1\right)\right)+2^{\lceil -3.41+1.71\log_2 n-1\rceil}\times 3$$

(h) Simplify your expression from the previous part to a closed form (no need to further simplify once you reach a closed form).

$$2^{\lceil -3.41+1.71\log_2 n-1\rceil}\times 3+\left(\sum_{i=0}^{\lceil -3.41+1.71\log_2 n-1\rceil -1}\frac{2^i\times 2\times 2^i}{3^i}+2^i\right)$$

omitted for now to make shorter

$$=2^{\lceil -3.41+1.71\log_2 n-1\rceil}\times 3+2\sum_{i=0}^{\lceil\cdots\rceil -1}\left(\frac{2^2}{3}\right)^i+\sum_{i=0}^{\lceil\cdots\rceil -1}2^i$$

Finite geometric series identity

$$=2^{\lceil -3.41+1.71\log_2 n-1\rceil}\times 3+2\left(\frac{\left(\frac{4}{3}\right)^{\lceil\cdots\rceil}-1}{\frac{4}{3}-1}\right)+\frac{2^{\lceil\cdots\rceil}-1}{2-1}$$

All done!

No more summations = no need to further simplify.

8. Consider the following recurrence:

$$T(n) = \begin{cases} 19, & \text{if } x <= 1 \\ 3T(n/7) + n^3, & \text{otherwise} \end{cases} \quad (2)$$

(a) Use Master Theorem to find the tight Big-O Bound for this recurrence

$\log_b a \; ? \; c$

$\log_7 3 < 3$

$T(n) \in O(n^3)$

(b) Make a change to this recurrence such that you can't use Master Theorem on it anymore.

Many answers

- changing "a" to be not an number

$$T(n) = \begin{cases} 19 & \text{if } n \leq 1 \\ \log n \times T(n/7) + n^3 & \text{otherwise} \end{cases}$$

- changing input to be not a fraction

$$T(n) = \begin{cases} 19 & \text{if } n \leq 1 \\ 3T(n-7) + n^3 & \text{otherwise} \end{cases}$$

- changing recursive work to be other functions

$$T(n) = \begin{cases} 19 & \text{if } n \leq 1 \\ 3T(n/7) + 2^n & \text{otherwise} \end{cases}$$

and more!