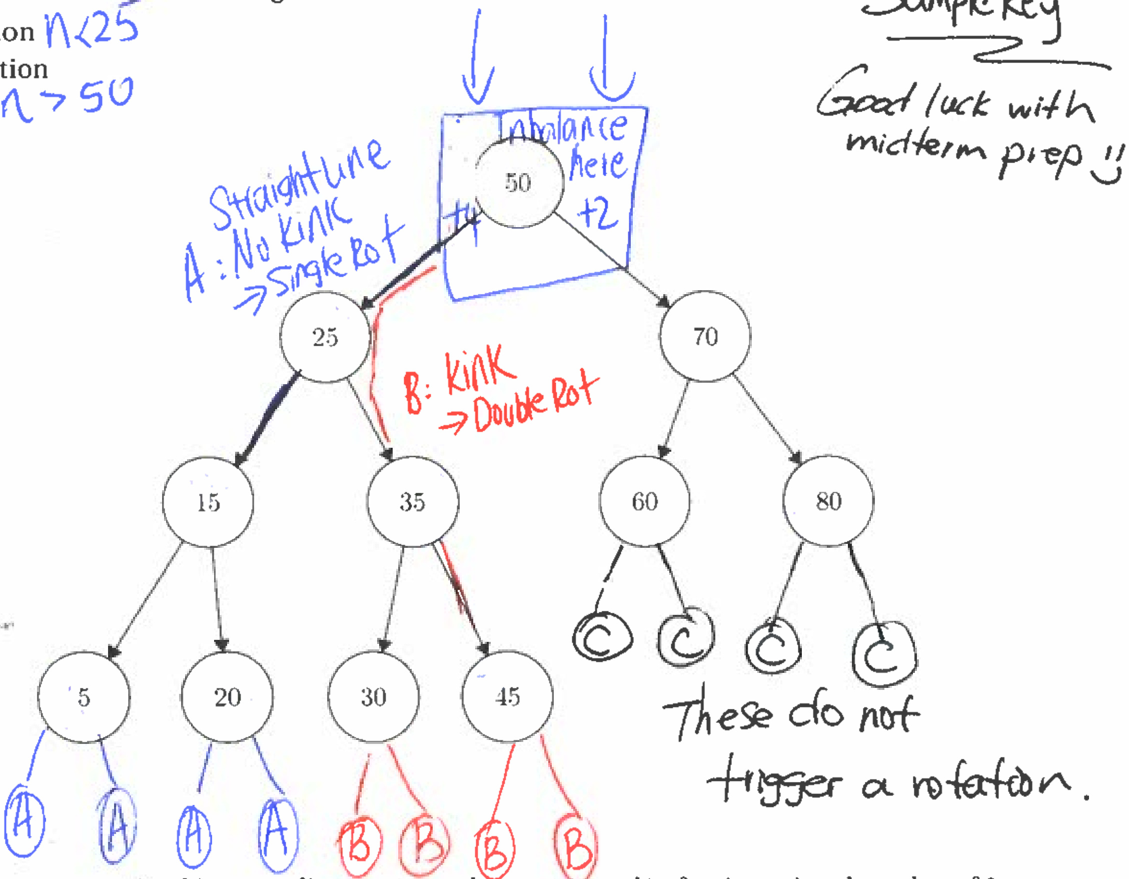
Color correction → ■ 🟥 🟦

Sample Key

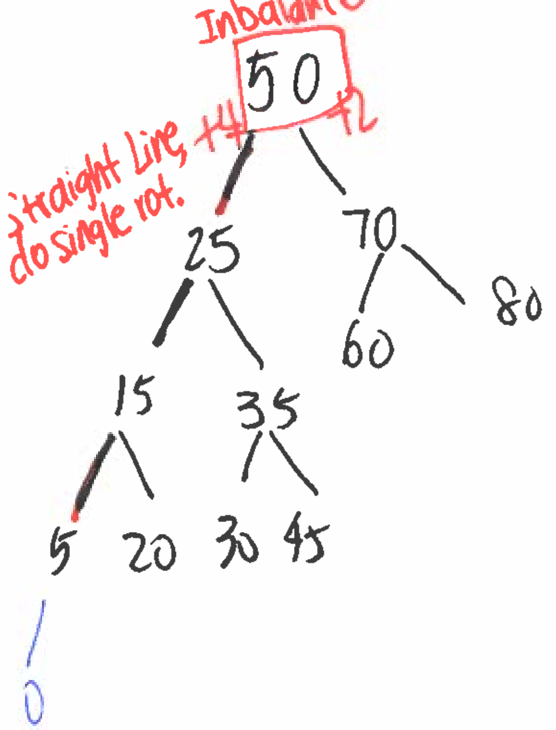
Good luck with midterm prep !! !

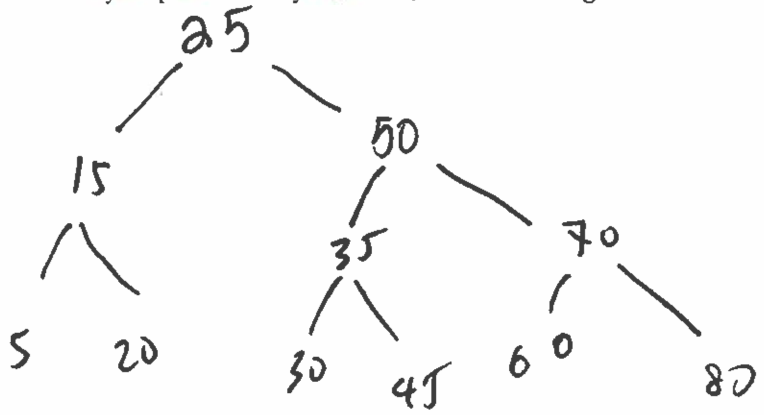
1. (a) For the given AVL tree below, list the range of values that would cause a:

A • a single rotation n<25
B • a double rotation
C • no rotation n>50



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





(c) Give a 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.

Try to cause a degenerate BST: 0, 1, 2, 3, 4, 5

2. 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.

Dictionary use case and searches by key. Currently an AD is used. New insertions / deletions are fast, but for this use case, searches would be $\Theta(n)$.

We should use a hash dictionary. In practice each look up would be $\Theta(1)$.

(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)

With a hash dict, searches are fast, but spaces are wasted if not all buckets are wasted.

Since order doesn't matter, we can use an array dict. w/ a small initial capacity and resize as required.

✓ like 20

3. 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

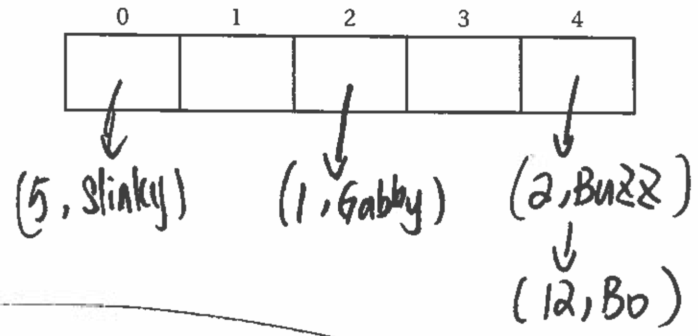
```
public int hashCode(int input) {
    return 2 * input;
}
```

↑ Ex: (2x2)%5 = 4

(2, Buzz), (12, Bo), (1, Gabby), (5, Slinky), (15, Jessie), (2, Woody), (13, Gabby)

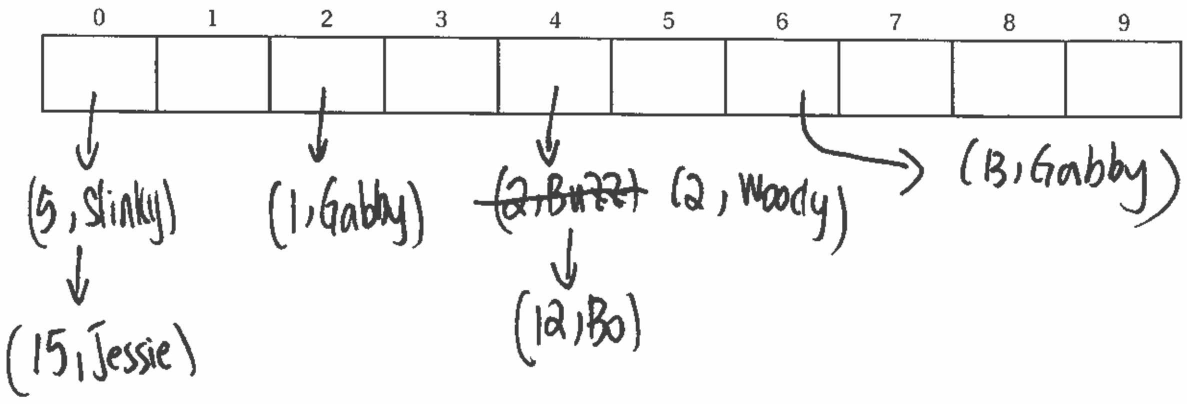
Before Resizing

Resize here because $\frac{4}{5} = 0.8$



Rehash! (2x2)%10 = 4, etc.

After Resizing



4. Code Modeling. Consider the following *intersection* method that find the intersection between two arrays.

```
public static int[] intersection(int[] A, int[] B) {
    DoubleLinkedList<Integer> setOfA = new DoubleLinkedList<Integer>();   +1  +1
    DoubleLinkedList<Integer> setOfB = new DoubleLinkedList<Integer>();   +1  +1

    for (int i = 0; i < A.length; i++) {        } m      m^2
        if (!setOfA.contains(A[i]) {
            setOfA.add(A[i]);
        }
    }

    for (int i = 0; i < B.length; i++) {        } n      n^2
        if (!setOfB.contains(B[i]) {
            setOfB.add(B[i]);
        }
    }

    DoubleLinkedList<Integer> result = new DoubleLinkedList<Integer>();
    Iterator<Integer> itr = setOfA.iterator();
    for (int i = 0; i < setOfA.size(); i++) {                        } m    mn
        int next = itr.next();
        if (setOfB.contains(next)) {
            result.add(next);
        }
    }

    return result;  +1   +1
}
```

Answer the following questions about the runtime of the *union_count* method. Consider m to be the size of input A and n is the size of input B. For all of the questions below, your runtimes may be in terms of n and / or m.

(a) Describe the state of the input for the best case runtime for the first loop, and state the big-Theta runtime of such a best case. Repeat the same for the worst case.

Best case: All values are the same, such that each contains() is $\Theta(1)$. $\Theta(m)$.

Worst case: All values are unique - so setOfA will keep growing in size, and since setOfA.contains will always return false, the contains call will have to do work relative to the current size. $\sum_{i}^{n-1}$ $\Theta(m^2)$.

(b) Describe the state of the input for the best case runtime for the last loop, and state the big-Theta runtime of such a best case. Repeat the same for the worst case.

Best case: A[] and B[] both contain all duplicates, so setOfA and setOfB are constant in size. $\Theta(1)$.

Worst case: All values unique such that A[] and B[] have totally disjoint values. That means for each value in setOfA, search _all_ of setOfB $\Theta(mn)$.

(c) Putting together your answers to the previous questions, what is the big-Theta bound for the best case runtime for the entire method? What is the big-Theta bound for the worst case runtime?

Best Case Annotations in blue

$\Theta(m+n)$

from $2m+n+3$

Worst Case in Red

$\Theta(m^2+n^2)$

from m^2+n^2+mn+3

5. Consider the following recurrence:

Base case work

5 branches

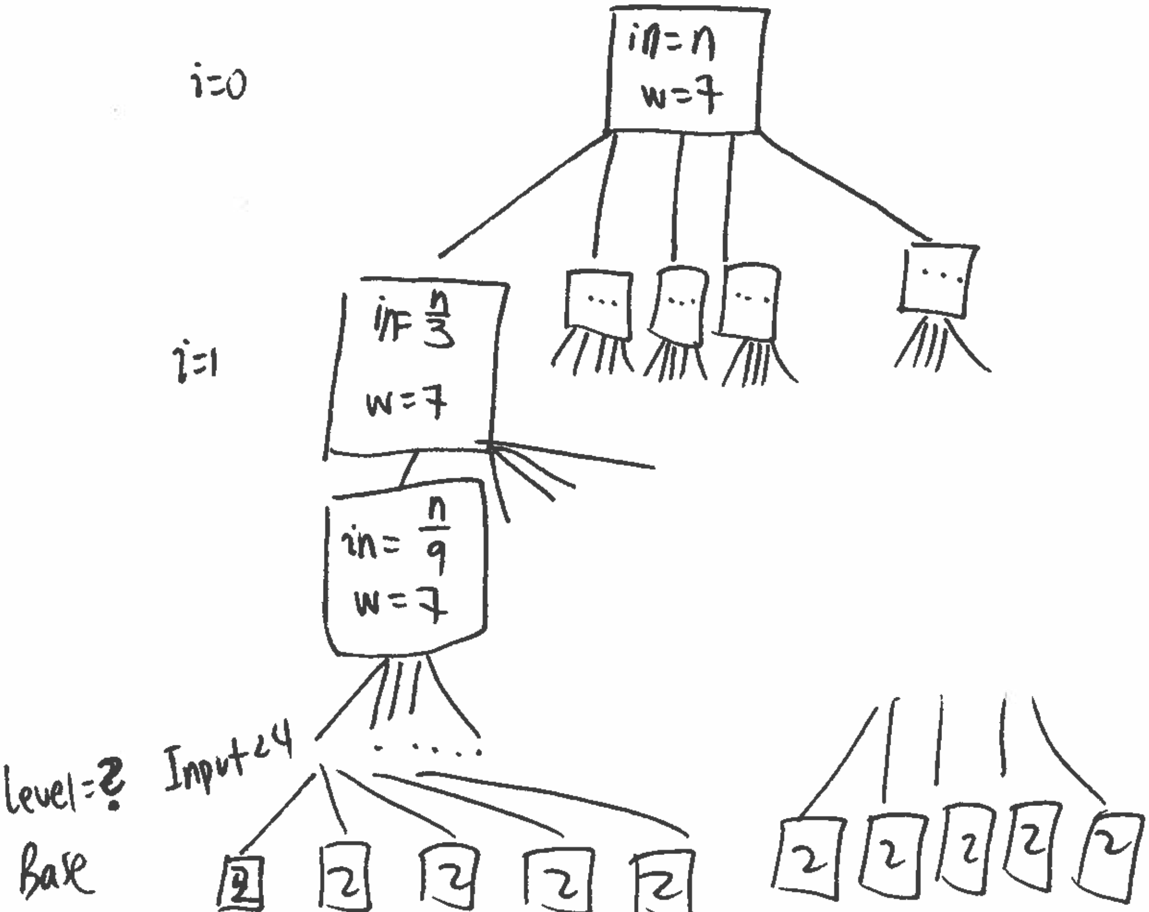
$$T(n) = \begin{cases} 2. & \text{if } n < 4 \\ \boxed{5}T\left(\boxed{\frac{n}{3}}\right) + 7. & \text{otherwise} \end{cases} \qquad (1)$$

Recursive case work

Each level has input divided by 3

We want you to find an exact closed form 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 size of the input to each node at level i? How much work is done per node at level i?

$\frac{n}{3^i}$ 7

(c) How many nodes are at level i?

5^i

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

$7(5^i)$

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

Since $n<4$, look for $n=3$.

$\frac{n}{3^i} = 3 \Rightarrow n = 3 \cdot 3^i = 3^{i+1} \Rightarrow i+1 = \log_3 n \Rightarrow i = \log_3 n - 1$

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

$2 \cdot 5^{\log_3 n - 1} = \frac{2}{5} \cdot 5^{\log_3 n} = \underline{\frac{5}{2} n^{\log_3 5}}$

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

$$\frac{5}{2}n^{\log_3 5} + 7\sum_{i=0}^{\log_3 n-1-1}(5^i)$$

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

$$\frac{5}{2}n^{\log_3 5} + 7\sum_{i=0}^{\log_3 n-2}(5^i) = \frac{5}{2}n^{\log_3 5} + 7\left(\frac{5^{\log_3 n-1}-1}{4}\right)$$

(i) Write a simplified Big-Theta of the run time from the closed form above. No work is needed.

$$\cancel{\frac{5}{2}}n^{\log_3 5} \Rightarrow \Theta(n^{\log_3 5})$$

$$7\left(\frac{5^{\log_3 n-1}-1}{4}\right) \Rightarrow \cancel{7}\left(\frac{\frac{n^{\log_3 5}}{5}\cancel{-1}}{\cancel{4}}\right) \Rightarrow \Theta(n^{\log_3 5})$$

$$\boxed{\Theta(n^{\log_3 5})}$$

6. Consider the following recurrence:

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

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

$\log_7(7^3)$ a b c

$\log_7 3 < 3$

$\Theta(n^3)$

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

not constant $n^2 T(\frac{n}{7}) + n^3$ otherwise

$3T(n-7) + n^3$ otherwise

not by a factor

etc. etc.

STOP

Review session questions end here.

The following questions are excluded from the review session in interest of time. They're still provided here to give you additional practice.

1. 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.

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

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 $f(n) \in \mathcal{O}(g(n))$

$log(n^2) = 2 log n \leq c \cdot n$ for $n_0 = 2$, $c = 1$

$10n \leq c \cdot n$ for $n_0 = 1$, $c = 10$

$-5n^2 \leq c \cdot n$ for $n_0 = 1$, $c = 1$

$log(n^2) + 10n - 5n^2 \leq n + 10n + n \leq 12n$ $\quad \boxed{c = 12}$

when $\boxed{n_0 = 2}$

2. General Asymptotics. For each of the following code blocks, give a big-Theta bound of the runtime in the worst case.

(a)

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

Θ(n logn)

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

(b)

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

Θ(n²)

```
    // 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++) { *C
            System.out.println("hi");
        }
    }
```

} This junk does not depend on n.

Θ(1)

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