University of Washington

CSE 373 | Winter 2020 | Practice Midterm

Full Name:	Sample Solutions	
UWNetID:	cse373-staff	
Name of person to your Left | Right	Anne	Surkhi
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE373 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**		

Do not turn the page until 15:30.

Caveat

- While this practice exam is representative of the format and the topics that might be covered, it hasn't been tested to ensure that it is exactly 50 minutes' worth of work. We think this looks like a 70-75 minute exam, but erred on the side of giving you more practice material 🙂

Advice

- Read the questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Take a deep breath and relax. We believe in you!

UWNetID: ________________

Question 1: Listomania!

What ADT would be best used in the following situations:

(a) You need to move apartments and have rented a moving truck. Once something has been loaded into the truck, it can't be removed until everything that was loaded after it has been removed.

Stack, so the most recently inserted item is removed first.

(b) The TAs want to keep track of who needs help at office hours and they want to help people in the order they arrive.

Queue, so the least recently inserted item is removed first.

(c) The UW is building a new registration system and wants to be able to tell if a student is already registered for a particular class, to prevent them from registering for it twice.

Set, so duplicate registrations only count a single time.

(The set could contain (student, registration) tuples. Acceptable answers could also describe a Map of Sets, such as from each class to a set of registered students)

(d) You are an ER doctor, and you are writing a queueing app that allows you to always take in the patient with the most severe symptoms first. Assume that any two illnesses are comparable.

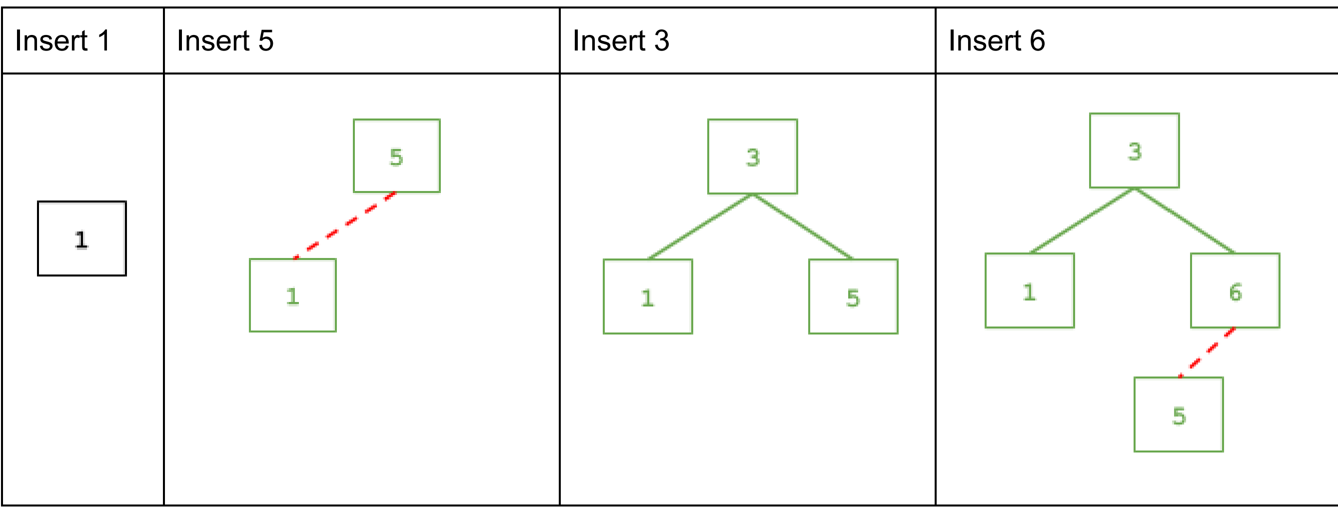
Priority Queue, so when new patients are added they are sorted according to symptom severity.

(e) When grading the midterm, the TAs want to be able to record which student got which score.

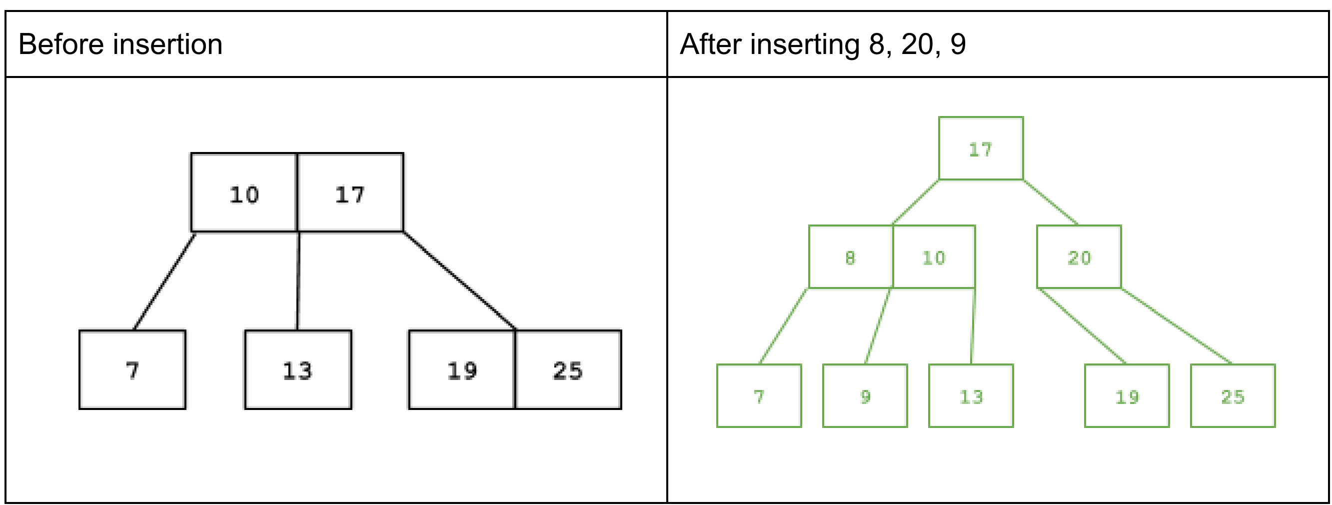
Map, from students to scores.

Question 2: If a Tree Falls in a Forest

a) Draw the entire resulting tree after each of the following insertions into a Left-Leaning Red-Black tree. The first one has been done for you. Indicate red edges in the tree with a dashed line or by labeling them clearly with "**RED**" next to them.



b) Consider the following 2-3 tree. In the box below, draw the final tree after performing the following insertions in order: 8, 20, 9.



c) Suppose we want to store the following elements in a regular binary search tree (i.e. with no balancing properties). Give an order we could insert the elements to create a worst-case tree, and an order to create a best-case tree.

Elements	`8, 12, 6, 9, 5, 2`
Best-Case Insertion Order	`6, 5, 2, 9, 8, 12`
Worst-Case Insertion Order	`2, 5, 6, 8, 9, 12`

UWNetID: ________________

Question 3: Big Oh-No!

(A) Give the simplified worst case runtime in Θ(•) notation, as a function of *n*.

```
public void q1(int n) {
    // Assume TreeMap<K, V> uses an automatically-balanced search tree.
    Map<Integer, Integer> map = new TreeMap<>();

    for (int i = n; i > -n; i--) {
        map.put(i, n);

        for (int j = 0; j < i; j++) {
            System.out.println(j);
        }
    }
}
```

Θ(**n^2**)

The outer loop (with variable i) runs for 2n iterations (Θ(n)). Inside of it, we have an insertion into a balanced tree (which is Θ(log n)), and another loop up to the value of i (which will be a constant factor of n, so it becomes Θ(n)). Since Θ(n) is larger than Θ(log n), we can consider the inside of the loop to just be Θ(n), leading to a total of Θ(n^2).

(B) Give the simplified worst case runtime in Θ(•) notation, as a function of *n*.

```
public int f(int n) {
    if (n < 1) {
        return 0;
    }

    for (int i = 0; i < n; i++) {
        System.out.println("hiya");
    }

    int first = 2 * f(n/3);
    int second = f(n/3);
    int third = f(n/3);

    return first + second + third;
}
```

Θ(**n log n**)

We can "unroll" this recurrence as a tree where each node represents a call to f, which performs n operations and then makes three recursive calls each with n/3 of the work to do. First, we compute the height of this tree: Since we divide by 3 at each level, there will be $\log_3(n)$ levels. Next, we compute the number of operations performed at each level. Each level i (where i=0 is the root) has 3^i nodes, but each node performs $1/(3^i)$ operations in the loop, so each level has a total of n operations to do. This is also true for the base case, in which we have n nodes that each do a constant amount of work. That means our total runtime will be log n levels times n work at each level, or $\Theta(n \log n)$.

(C) The following two sub-questions pertain to this code snippet:

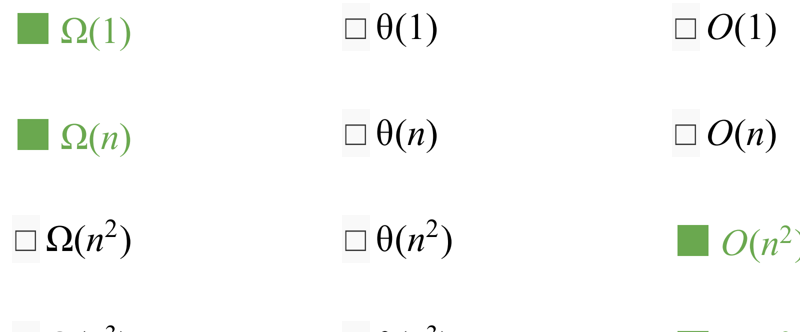
```
public int q3(int n) {
    int tuition = 0;
    if (n < 100000000) {
        for (int i = 0; i < n * n * n; i++) {
            tuition++;
        }
        return tuition;
    }
    if (n % 2 == 0) {
        for (int i = 0; i < n; i++) {
            tuition++;
        }
    } else {
        for (int i = 0; i < n * n; i++) {
            tuition++;
        }
    }
    return tuition;
}
```

a) What set(s) of values will trigger the **best asymptotic runtime** for this function?

Even values of n will trigger the best case behavior.

a) Which of the following claims about the **overall asymptotic** runtime for this function are true? Fill in the boxes next to all choices that apply.

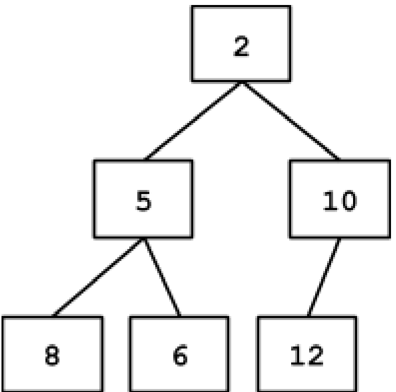
(Hint: Start by finding Ω / θ / O for individual cases like you did for the QuickCheck. Then, try to find Ω / θ / O for the overall function.)



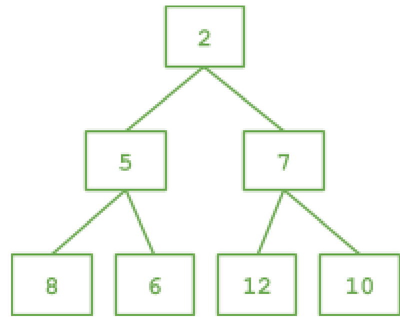
UWNetID: ____________________

Question 4: Sweet Heaps Are Made of This

a) Consider the following Binary Min Heap:



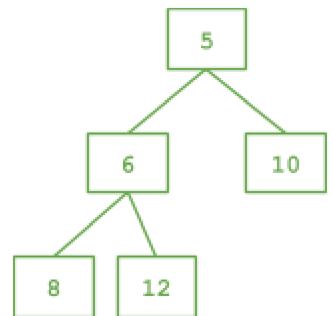
i) Draw the heap after inserting the value 7.



How many swaps are required during this insertion?

1

ii) Draw the heap after removing the minimum value. Start with the heap pictured; do not include the value 7 from part (i).



How many swaps are required during this removal?

3

UWNetID: ________________

b) Suppose we wanted to add each of the following methods to our implementation of a Binary Min Heap. For each method, briefly describe how you would implement it in no more than 2 sentences. You do not need to use specific field names as long as your meaning is clear. Then, give a lower bound for the runtime of that method **in the worst case**, including runtime needed to restore any heap invariants.

i) **removeSecondToMin()** - Removes the second-smallest value in the heap. Does not remove the smallest value.

Implementation (<= 2 sentences)	Lower-Bound Runtime
The root is always the smallest value, so the second-smallest value in the heap must be one of its children. Check both children, remove the smallest and swap the last value in the heap into its place, and finally percolate down the swapped in value as with a normal removal (a logarithmic operation).	$\Omega(\log n)$

ii) **removeArbitrary(int value)** - Removes the given value from the heap (if it exists), otherwise does nothing.

Implementation (<= 2 sentences)	Lower-Bound Runtime
We have to search every element in the heap (a linear time operation) to see if it contains the given value (the heap invariants don't let us "rule out" any branches). Then, if the value is found we remove it, swap in the last element, and percolate down as usual (a logarithmic operation that is dominated by the linear time search).	$\Omega(n)$

iii) **checkHeapInvariants(int[] array)** - Given an arbitrary array, return true if the array satisfies the heap invariants.

Implementation (<= 2 sentences)	Lower-Bound Runtime
To check the heap invariant, we have to check whether every element of the array is smaller than both of its children. Calculating the position of an element's children (by multiplying its index) and comparing with them are constant time operations, but we have to do this for every element (making it linear time).	$\Omega(n)$

iv) **incrementMin()** - Adds 1 to the value of the heap's root node.

Implementation (<= 2 sentences)	Lower-Bound Runtime
Add 1 to the value of the root node, then percolate down. Since the problem does not specify what kind of data is stored in the heap, we can't make any assumptions about how far the node may have to percolate, so in the worst case it's a logarithmic operation.	$\Omega(\log n)$

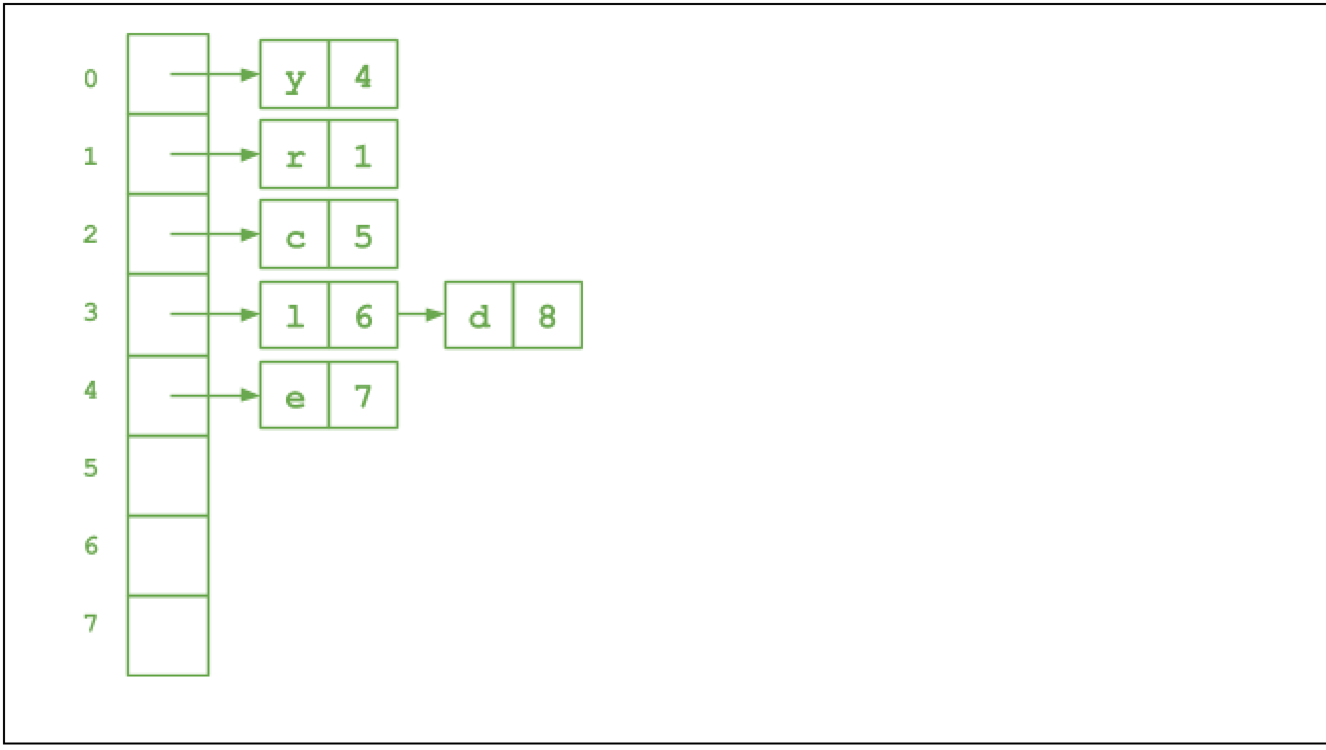
Question 5: Sustainability, It's in Our Nature

Assume we have a hash table the maps from characters to integers. The hash table starts with 4 buckets and resizes by doubling the number of buckets if the insert would cause the load factor to exceed **0.75**. Collision is resolved with separate chaining, and each individual bucket uses linked lists.

(A) Draw an internal representation of the hash map after inserting key-value pairs:

```
('r', 1), ('e', 2), ('c', 3), ('y', 4), ('c', 5), ('l', 6), ('e', 7), ('d', 8)
```

Assume the hash code for 'c' is 2, 'd' is 3, 'e' is 4, 'l' is 11, 'r' is 17, and 'y' is 24. If a key already exists in the table, subsequent inserts with the same key will overwrite its value.



(B) What is the current load factor after all the insertions?

0.75

Question 6: Be Boundless (For Runtime / For The World)

Your friend Morgan says that they have come up with the new greatest idea for a data structure: a hash table, but instead of normal chaining (based on linked lists) they will use a Left-Leaning Red-Black tree in each bucket!

They claim that the runtimes should always be at least as good as for regular hash tables, since the asymptotic bounds are as good or better in both the best and worst cases. You want to fact check their claim.

Assuming that the hash table resizes at a reasonable load factor (for example, 1.5):

(A) Give the **best case** runtime of looking up a key in this data structure, in terms of N, the number of elements in the hash table.

$\Theta($ **1** $)$

(B) Give the **worst case** runtime of looking up a key in this data structure, in terms of N, the number of elements in the hash table.

$\Theta($ **log n** $)$

(C) Is Morgan's claim correct? Briefly justify based on your answers above (in 1-2 sentences).

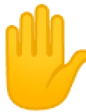
Argument for CORRECT *In the case of looking up a key, Morgan's claim is correct: asymptotically, their new data structure will perform as good or better than regular hash tables.*

Argument for INCORRECT *However, in the case that the hash table needs to resize, the new data structure could perform worse: recreating the LLRB trees in each bucket would run in $\Theta(n \log n)$ time, instead of the $\Theta(n)$ time required by normal hash tables. It would also be acceptable to argue that Morgan cannot claim the data structure is always at least as good just because it is asymptotically better (asymptotic analysis is a useful tool to make generalizations, but there may be specific cases where it is not better).*

UWNetID: ________________

Question 7: It's the End of the Exam as We Know It (and I Feel Fine)

[*Extra Credit*] Which do you like better: B-trees, or LLRB trees? Briefly justify by drawing a picture.



Exam Completed!

Have you written your UWNetID on every page?