CS 61B Fall 2023

Midterm 2

Thursday October 19, 2023

Intro: Welcome to CS61B Midterm 2!

Name: _____ Your SID: _____

Location: _____ Seat Number: _____

SID of Person to your Left: _____ Right: _____

Formatting:

- ◯ indicates only one circle should be filled in. ☐ indicates more than one box may be filled in. **Please fill in the shape completely.** If you change your response, **erase as completely as possible**.
- Anything you write that you ~~cross out~~ will not be graded.
- You may not use ternary operators, lambdas, streams, or multiple assignment.

Tips:

- This midterm is worth **100 points**, and there are a lot of problems on this exam. Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.
- Not all information provided in a problem may be useful, and you may not need all lines.
- **We will not give credit for solutions that go over the number of provided lines or that fail to follow any restrictions given in the problem statement.**
- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- Unless otherwise stated, all given code on this exam compiles. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix.

Write the statement below in the same handwriting you will use on the rest of the exam: "I have neither given nor received help on this exam (or quiz), and have rejected any attempt to cheat. If these answers are not my own work, I may be deducted 9,876,543,210 points on the exam."

_____

_____

_____

_____

_____

Signature: _____

1 Ghost Curry

(16 Points)

(a) Say $f(n) \in \Theta(g(n))$, and $g(n)$ is strictly positive and increasing. Mark all that are always true.

- ☐ $f(n) \in O(g(n))$
- ☐ $f(n) \in \Omega(g(n))$
- ☐ $f(n) \in O(\log(g(n)))$
- ☐ $f(n) \in \Omega(\log(g(n)))$
- ☐ $f(n) \in O(n \cdot g(n))$
- ☐ $f(n) \in \Omega(n \cdot g(n))$
- ○ None of the above

(b) A data structure with a time complexity of $\Theta(1)$ for all its operations always runs faster on all inputs than a data structure with a time complexity of $\Theta(\log n)$ for the same operations.

○ True ○ False

(c) A fully connected disjoint set without path compression has `find()` run in $\Theta(1)$.

○ True ○ False

(d) Inserting into a Binary Search Tree is $O(\log N)$ where N is the number of elements.

○ True ○ False

(e) A 2-3 tree can contain a node with 2 elements as the root.

○ True ○ False

(f) Our HashMap runtime analyses assume `hashCode` runs in constant time.

○ True ○ False

(g) In Hibbard Deletion, you can replace the deleted node with either the right-most node in the left subtree or the left-most node in the right subtree if they exist.

○ True ○ False

(h) Two objects are always equal if they have the same hash, assuming that the `hashCode()` function is valid.

○ True ○ False

(i) Two objects are always unequal if they have different hashes, assuming that the `hashCode()` function is valid.

○ True ○ False

(j) The asymptotic runtime for all methods for a 2-3-4 tree is same as a 2-3 tree.

○ True ○ False

(k) Select all of the valid `hashCode` implementations for the class `Ben`.

```
public class Ben {
    public int x;
    public int y;
    public int z;

    @Override
    public boolean equals(Object o) {
        if (o instanceof Ben b) {
            return this.x == b.x && this.y == b.y;
        }
        return false;
    }

    @Override
    public int hashCode() { return /* answer choices here */ }
}
```

☐ `0;` ☐ `x + y + z;` ☐ `x * 625 - y;` ☐ `z;` ○ None of the above

(l) You are implementing an undo feature in a graphic design software. Users should be able to undo the most recent action they performed. Which data structure is ideal for managing the history of actions?

○ ArrayList ○ LLRB Tree ○ Heap ○ HashMap ○ Stack

(m) You are developing a chat application, which needs to be able to show messages in chronological order. Messages can be inserted after the newest message, and older messages may be deleted or modified regardless of timestamp. Which data structure is best suited for maintaining messages?

○ ArrayList ○ LLRB Tree ○ Heap ○ HashMap ○ Stack

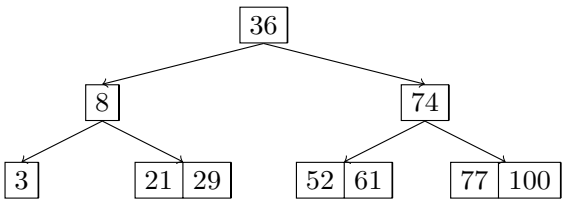
(n) You are building a weather application that needs to store information about cities, including their names and corresponding weather conditions. Users should be able to quickly look up weather data for a specific city by its name. Which data structure is the most appropriate for efficiently organizing this information?

○ ArrayList ○ LLRB Tree ○ Heap ○ HashMap ○ Stack

2 Tree Questions

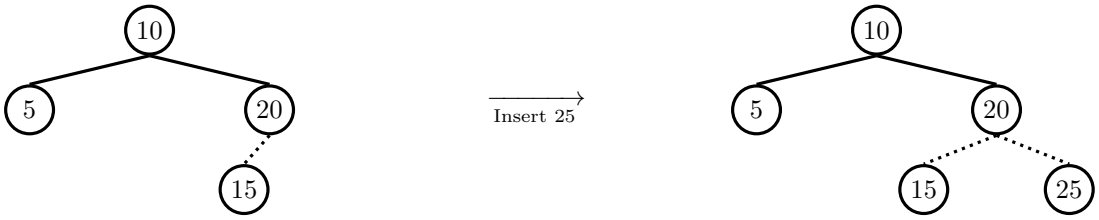
(16 Points)

(a) Given the following 2-3 tree



1. There is an order of insertions that creates this 2-3 Tree such that the 36 was inserted after the 3.
   ○ True ○ False
2. We insert 53, 23, 44, 47 to the above tree, in that order.
   (a) What element(s) are in the root node?
   (b) How many nodes are there with 2 elements?

(b) What sequence of rotations and color flips after inserting 25 would be done to return to a valid LLRB? Dotted links between nodes are red and solid links between nodes are black.

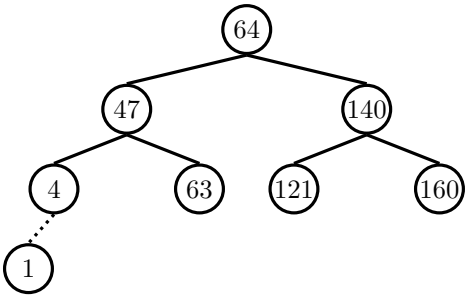


	Operation Type	Node
Operation 1	○ Rotate Left ○ Rotate Right ○ Color Flip ○ None of the above	○ 5 ○ 10 ○ 15 ○ 20 ○ 25 ○ None of the above
Operation 2	○ Rotate Left ○ Rotate Right ○ Color Flip ○ None of the above	○ 5 ○ 10 ○ 15 ○ 20 ○ 25 ○ None of the above
Operation 3	○ Rotate Left ○ Rotate Right ○ Color Flip ○ None of the above	○ 5 ○ 10 ○ 15 ○ 20 ○ 25 ○ None of the above

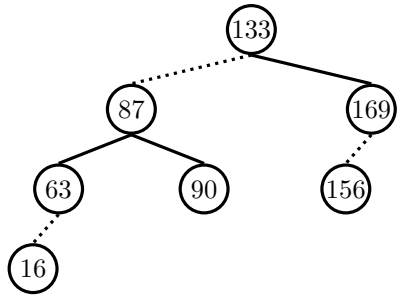
(c) Which of the following are **not** well-formed Left Leaning Red-Black Tree(s)? Select all that apply. Dotted links between nodes are red and solid links between nodes are black.

☐ (i) ☐ (ii) ☐ (iii) ☐ (iv) ☐ (v)

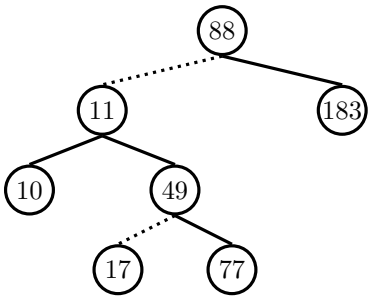
(i)



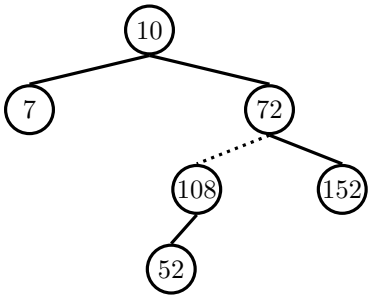
(ii)



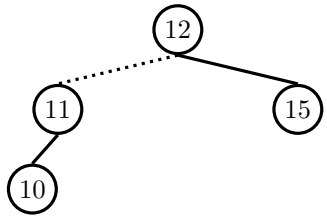
(iii)



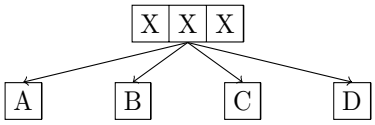
(iv)



(v)



(d) We're now working with a 2-3-4 tree. The values of the 2-3-4 tree are not shown.



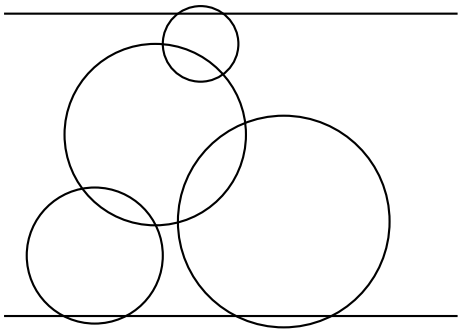
1. What is the maximum number of values that can be added **without** causing the node with 3 Xs to split?

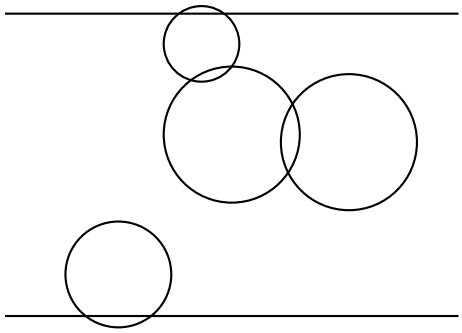
2. What's the minimum number of values that can be added to cause the node with 3 Xs to split?

3 A Hole-y Cheese

(18 Points)

Sam has a hole-y cheese. This cheese is infinitely wide, but has a limited height. This cheese contains numerous circular holes. Sam asked 61B course staff to check if it is possible to go from the bottom surface to the top, without modifying the cheese in any way. If two holes intersect, it is possible to travel between the two. If a hole intersects with the bottom/top surface, it is possible to travel between the surface and the hole.





For instance, in the picture on the left above, it is possible to go from bottom to top. However, in the picture on the right above, it is not possible to go from bottom to top.

Assume we use the following representation for a single Hole.

```
public class Hole {
    // ...implementation not shown...

    // Checks whether this hole intersects with the passed in hole.
    public boolean intersectsWithHole(Hole other) { ...}

    // Checks whether this hole intersects with the bottom.
    public boolean intersectsWithBottom() { ... }

    // Checks whether this hole intersects with the top.
    public boolean intersectsWithTop() { ... }
}
```

Question is on the next page.

Complete the following `topBottomConnected` method, which returns whether it is possible to go from the top to bottom, given a List of `Hole` objects.

```
public boolean topBottomConnected(List<Hole> holes) {
    WeightedQuickUnion q = new WeightedQuickUnion(holes.size() + 2);
    int bottom = holes.size();
    int top = holes.size() + 1;

    for (_____1_____) {

        if (_____2_____) {

            _____3_____;
        }

        if (_____4_____) {

            _____5_____;
        }
    }

    for (_____6_____) {

        for (_____7_____) {

            if (_____8_____) {

                _____9_____;
            }
        }
    }

    return _____10_____;
}
```

4 Heaps of Fun

(16 Points)

1. Say we have a binary MinHeap containing A, B, C, and D such that A < B < C < D.

   (a) Which element(s) could possibly be the root? Select all that apply.

   ☐ A ☐ B ☐ C ☐ D ○ None of the above

   (b) Which element(s) could possibly be one edge away from the root? Select all that apply.

   ☐ A ☐ B ☐ C ☐ D ○ None of the above

   (c) Which element(s) could possibly be two edges away from the root? Select all that apply.

   ☐ A ☐ B ☐ C ☐ D ○ None of the above

2. For this question, assume that we have created a **MinHeap** with the numbers 1, 2, . . . , 1023 inserted, and that the MinHeap is stored as an array with the 0th index empty (as seen in lecture).

   (a) What are the minimum and maximum indices in the underlying heap array that could contain the number 8?

   Min Index: ☐ Min Index: ☐

   (b) If 8 was in its **minimum** possible index, what **other** numbers could only be in exactly one index? List all such numbers and their corresponding index. You may not need to use all boxes provided.

Number										
Index										

   (c) If 8 was in its **maximum** possible index, what **other** numbers could only be in exactly one index? List all such numbers and their corresponding index. You may not need to use all boxes provided.

Number										
Index										

3. For this question, assume that we have created a **MaxHeap** with the numbers 1, 2, . . . , 1023 inserted, and that the MaxHeap is stored as an array with the 0th index empty (as seen in lecture).

   (a) What are the minimum and maximum index in the underlying Heap array that could contain the number 8?

   Min Index: ☐ Min Index: ☐

   (b) If 8 was in its **minimum** possible index, what **other** numbers could only be in exactly one index? List all such numbers and their corresponding index. You may not need to use all boxes provided.

Number										
Index										

   (c) If 8 was in its **maximum** possible index, what **other** numbers could only be in exactly one index? List all such numbers and their corresponding index. You may not need to use all boxes provided.

Number										
Index										

5 BST To MinHeap

(16 Points)

Eric and Justin have been tasked with converting a Binary Search Tree into a MinHeap with the same elements. **The PriorityQueue is implemented like a MinHeap from lecture.** Eric says insertions into the MinHeap with N elements take $O(\log(N))$. Since there are N insertions, the runtime should be $O(N \log(N))$. However, Justin claims that this can be done in $O(N)$ time.

(a) Inserting into a MinHeap can take constant time in certain scenarios. Select all the conditions where MinHeap insertion is a constant-time operation (takes $O(1)$ time):

☐ You are inserting the largest element into the heap

☐ You are inserting the smallest element into the heap

☐ When the heap is balanced and has an equal number of left and right children for all nodes

○ None of the above

(b) Now, implement the instance method `toPQ`. This method must be $O(N)$ for full credit. You may only use publicly-accessible methods when interfacing with the `PriorityQueue`.

```
public class BSTSet<E extends Comparable<E>> {
    class Node {
        E elem;
        Node left; // A subtree containing all elements less than elem
        Node right; // A subtree containing all elements greater than elem
        public Node(E elem) { this.elem = elem; }
    }
    public Node root; // ... binary search tree implementation not shown ...

    public PriorityQueue<E> toPQ() {
        PriorityQueue<E> result = new PriorityQueue<E>();
        helper(this.root, result);
        return result;
    }
    private void helper(Node node, PriorityQueue<E> pq) {

        _____ 1

        _____ 2

        _____ 3

        _____ 4

        _____ 5

        _____ 6

    }
}
```

6 Asymptotics

(18 Points)

Give the best case runtime and worst case runtime for the functions below.

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

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

Worst case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

(b)
```
public static void b(int N) {
    if (N <= 1) {
        return;
    }
    for (int i = 0; i < N; i += 2) {
        System.out.println("b");
    }
    b(N / 3);
    b(N / 3);
    b(N / 3);
}
```

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above Worst case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

(c)
```
public static void c(int N) {
    Random rand = new Random();
    for (int i = 1; i < N; i *= 2) {
        for (int j = 0; j != rand.nextInt(0, i); j += 1) {
            System.out.println("c");
        }
    }
}
```

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above Worst case:
○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

(d)
```
public static void d(int[] arr) {
    int N = arr.length;
    BSTSet<Integer> tree = new BSTSet<>();
    /* Assume that BST implements a binary search tree
    * with no self-balancing optimizations, as seen in lecture */
    for(int i = 0; i < N; i += 1) {
        tree.insert(arr[i]);
    }
}
```

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

Worst case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

(e)
```
public static void e(int N) {
    if (N <= 0) { return; }
    if (N % 2 == 0) { return; }
    e(N - 1);
    e(N - 2);
}
```

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

Worst case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

(f)
```
public static void f(int N) {
    if (N == 1) { return; }
    if (isPowerOfTwo(N)) {
        f(N / 2);
    } else {
        f(N - 1);
    }
}
public static boolean isPowerOfTwo(int N) {
    if (N == 1) {
        return true;
    } else if (N % 2 != 0 || N == 0) {
        return false;
    } else {
        return isPowerOfTwo(N / 2);
    }
}
```

Best case:

○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

Worst case:

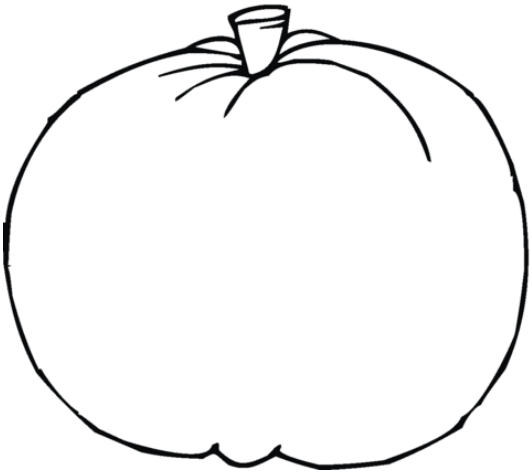
○ $\Theta(1)$ ○ $\Theta(\log(\log N))$ ○ $\Theta(\log N)$ ○ $\Theta((\log N)^2)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$
○ $\Theta(N^2)$ ○ $\Theta(N^2 \log N)$ ○ $\Theta(N^3)$ ○ $\Theta(N^3 \log N)$ ○ $\Theta(N^4)$ ○ $\Theta(N^4 \log N)$
○ Worse than $\Theta(N^4 \log N)$ ○ Never terminates (infinite loop) ○ None of the above

Nothing on this page is worth any points.

Pumpkin Carving

(0 Points)

Carve the pumpkin however you'd like!



61Bonus Question

(0 Points) What is the probability that the worst-case runtime of Q6c occurs in terms of N? Express your answer in the form $1 - 1/(\Theta(f(N)))$, where f is as simple as possible.

Feedback

(0 Points)

Leave any feedback, comments, concerns, or more drawings below!

Random Class API

```
public class Random {
    /** Creates a new random number generator.  Runs in Θ(1) time. */
    public Random() { ... }

    /** Creates a new random number generator using a seed. If two instances of Random are created
    with the same seed, they will generate and return identical sequences of numbers. Runs in Θ(1)
    time. */
    public Random(long seed) { ... }

    /** Returns the next pseudorandom int from this random number generator's sequence. Runs in Θ(1)
    time. */
    public int nextInt() { ... }

    /** Returns a pseudorandom int value between 0 (inclusive) and the specified bound (exclusive),
    drawn from this random number generator's sequence. Runs in Θ(1) time. */
    public int nextInt(int bound) { ... }

    /** Returns a pseudorandom int value between origin (inclusive) and the specified bound (
    exclusive), drawn from this random number generator's sequence. Runs in Θ(1) time. */
    public int nextInt(int origin, int bound) { ... }
}
```

WeightedQuickUnion Class API

```
public class WeightedQuickUnion {
    /** Creates a WeightedQuickUnion with n elements numbered 0 (inclusive) to n - 1 (inclusive). */
    public WeightedQuickUnion(int n) { ... }

    /** Check whether x and y belong to the same set. */
    public boolean isConnected(int x, int y) { ... }

    /** Union the set that x belongs to and the set that y belongs to. */
    public void union(int x, int y) { ... }
}
```

Math Class API

```
public class Math {
    /** Returns the value of the first argument raised to the power of the second argument.
    * Runs in Θ(1) time. */
    public static double pow(double a, double b) { ... }

    /** Returns the positive remainder when dividing x by y. Runs in Θ(1) time. */
    public static long floorMod(long x, long y) { ... }

}
```

PriorityQueue Class API

```
public class PriorityQueue<E extends Comparable<E>> {
    /** Create a PriorityQueue<E> with elements ordered according to their natural
    * ordering as defined by the compareTo method of the elements. Runs in Θ(1) time. */
    public PriorityQueue() { ... }

    /** Inserts the specified element into this priority queue. Runs in O(log(N)) time. */
    public void add(E e) { ... }

    /** Retrieves and removes the head of this queue, or returns null if this queue is empty. The
    head of this queue is the element which is smallest according to the compareTo method.
    * Runs in Θ(log(N)) time. */
    public E poll() { ... }

    /** Determines whether the queue is empty or not. Runs in Θ(1) time. */
    public boolean isEmpty() { ... }

    /** Returns the size of the queue. Runs in Θ(1) time. */
    public int size() { ... }

    /** Retrieves, but does not remove, the head of this queue, or returns null if this queue is
    * empty. Runs in Θ(1) time. */
    public E peek() { ... }
}
```

List Interface API

```
public interface List<E> {
    /** Appends the specified element to the end of this list. Runs in constant time. */
    void add(E e);

    /** Returns the element at the specified position in this list. */
    E get(int index);

    /** Replaces the element at the specified position in this list with the specified element. */
    E set(int index, E element);

    /** Returns the number of elements in this list. */
    int size();

    /** Returns true if this list contains no elements. */
    boolean isEmpty();

    /** Lists are defined to be equal if they contain the same elements in the same order. */
    boolean equals(Object o);
}
```

1c) The disjoint set is a WQU

1f) ... in constant time **relative to the number of elements in the Hashmap.**

2b) If you do not need all operations, select “None of the above” for the respective operation and node.

4.2 and 4.3) The heaps have 1023 distinct elements (which could have been inserted in any order), one for each integer from 1 to 1023 inclusive.

6f) Give the best case and worst case runtime of f(N), not isPowerOfTwo(N)