UC Berkeley – Computer Science
CS61B: Data Structures

Final, Spring 2021 (Final Solutions)

This test has 11 questions across 26 pages worth a total of 3200 points, and is to be completed in 170 minutes. The exam is closed book, except that you are allowed to use unlimited written cheat sheet (front and back). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write the statement out below in the blank provided and sign your name. You may do this before the exam begins.**

"I have neither given nor received any assistance in the taking of this exam."

Signature: _____

#	Points	#	Points
1	0	7	180
2	185	8	160
3	570	9	330
4	125	10	350
5	250	11	190
6	240	12	620
		TOTAL	3200

Name: _____

SID: _____

GitHub Account # : sp21-s_____

Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

Before exam: [☹_____☺].

After exam: [☹_____☺].

1. Welcome to the Final Exam

Many of the problems on this exam consist of multiple choice or fill in the blank questions. Make sure to read directions carefully! Some questions may ask you to select one answer:

◯ you can only pick
◯ one of these

While others may ask you to select all answers that apply:

☐ you can pick
☐ both of these

For fill in the blank questions, your final answer should be a number unless otherwise specified. Do not include units or math expressions in your answer. For example, a valid answer would be "20" (without the quotes), but "20 nodes" or "5 * 4" would be invalid.

There are also a few code writing questions. If you are given skeleton code to fill in, **we will not grade your answer if you alter the skeleton code** unless explicitly permitted. If you are given a line limit, **we will not grade your answer if you go over the line limit**. Additionally, **we will not grade your question if you fail to follow any restrictions given in the problem statement**. Like in previous exams, your code won't be checked by a compiler, but we will take off points for errors that are more than a typo. Please pay attention to detail when answering coding questions!

You can access all exam logistics documents using the exam page on the course website.

While taking the exam, in addition to your handwritten notes, you may use this reference sheet as a resource.

Please check to following boxes to confirm that you understand the exam proctoring policy!

☐ I understand that I may not use any internet resources or communicate with anyone during the exam.
☐ I am screen recording, sharing my entire desktop, unmuted and have quit and closed all tabs and background applications other than this PraireLearn tab, the Exams Ed, Zoom or my local recording software, and the Exam Proctoring Policy.
☐ I understand that I should periodically check the Exam Clarification post on the Exams Ed for important clarifications about the exam. This is my responsibility and failure to do so could result in me not getting all information about the exam.
☐ I understand that the 2 extra minutes I am given are only to **double check** I saved my answers. I understand that any unsaved answers will not be graded, and I should click the "save" button often and always before moving onto a new question.
☐ I understand that if I alter the skeleton code (unless otherwise specified), go over the line limit, or fail to follow any restrictions given in the problem statement of a coding question, then my answer will not be graded.

2. Heaps (185 Points).

1)

a) (25 Points) Suppose we have an array which represents a min heap. What is the worst-case runtime to find the **minimum** item in the array? Assume the items are distinct. Assume we don't know anything else about the array other than that it represents a min heap.

○ θ(1) ○ θ(log N) ○ θ(N) ○ θ(Nlog N) ○ θ(N^2)

b) (25 Points) Suppose we have an array which represents a min heap. What is the worst-case runtime to find the **maximum** item in the array? Assume the items are distinct. Assume we don't know anything else about the array other than that it represents a min heap.

○ θ(1) ○ θ(log N) ○ θ(N) ○ θ(Nlog N) ○ θ(N^2)

c) (25 Points) Suppose we have an array which represents a min heap. What is the worst-case runtime to find the **second smallest** item in the array? Assume the items are distinct. Assume we don't know anything else about the array other than that it represents a min heap.

○ θ(1) ○ θ(log N) ○ θ(N) ○ θ(Nlog N) ○ θ(N^2)

2) Heaps of Strings

a) (40 Points) When considering the runtime of heap operations, we usually assume that the runtime for each comparison is constant time. However, some comparison methods can take non-constant time. For example, code for comparing two strings is given below. You do not need to read and understand this code carefully, though you will need to identify its best and worst case runtime.

```
public int compare(String a, String b) {
    int len1 = a.length();
    int len2 = b.length();
    int lim = Math.min(len1, len2);

    int k = 0;
    while (k < lim) {
        char c1 = a.charAt(k);
        char c2 = b.charAt(k);
        if (c1 != c2) {
            return c1 - c2;
        }
        k += 1;
    }
    return len1 - len2;
}
```

Imagine we have N Strings, each of length N in a min heap. What is the **best case** runtime to insert one additional String of length N? Assume the underlying array doesn't have to resize. Give the best case runtime for this single insert operation.

○ θ(1) ○ θ(log N) ○ θ(N) ○ θ(Nlog N) ○ θ(N^2) ○ θ(2^N)

b) (40 Points) For the same scenario as described in part a, what is the **worst case** runtime of this single insert? Again assume the underlying array doesn't have to resize.

○ θ(1) ○ θ(log N) ○ θ(N) ○ θ(Nlog N) ○ θ(N^2) ○ θ(2^N)

3) Heaps and BSTs

(30 Points) It is possible to create a tree that is both a binary search tree and a heap (max or min). What is the maximum number of nodes in such a tree? If there is no maximum, simply write **"inf"**.

Answer: _____2_____

3. Trees (570 Points)

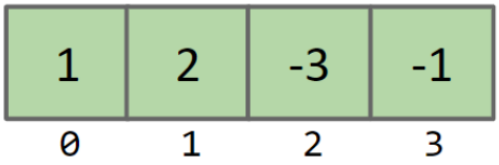
1) Trinary Trees (40 Points)

Recall that, in lecture, we showed how we can represent a complete binary tree as an array. We called this "tree representation 3" in lecture. We used this representation in lecture for storing heaps. In this representation, if we leave index 0 unused (to make the arithmetic easier), then the left child of a node is the index of the node * 2, and the right child of a node is the index of the node * 2 + 1. Imagine we now want to use an array to represent a complete trinary tree, i.e. each node has 3 children. We'll keep index 0 empty just like the binary trees we discussed in lecture. If we call the index of a node , what would the expression be for the index of the third child (i.e., the rightmost child of each node)?

○ 3i - 1 ○ 3i ○ 3i + 1 ○ 2i + 1 ○ i^2 - 1 ○ i^2 + 1 ○ 3^i

2) Trees

a) (40 Points) Suppose we have the WeightedQuickUnion parent array shown below. Assume we're using the WeightedQuickUnion array representation used in lecture and on the HW where each value value represents either the parent or the size of the tree. Which of the following is the reason **this** array is not a valid WeightedQuickUnion parent array? Choose the best option.



○ It has a cycle.
○ The actual weight of the tree rooted in 2 does not match the listed weight.
○ The actual weight of the tree rooted in 3 does not match the listed weight.
○ Its height is impossibly large given the weights of its disjoint sets.

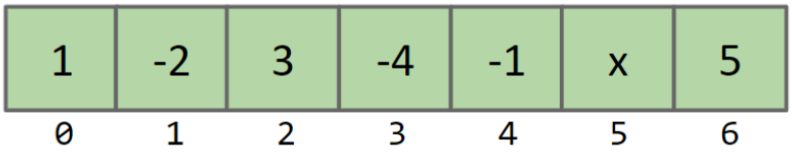
b) (40 Points) Suppose we instead treat the array [1, 2, -3, -1] above as a **complete binary tree** (which we called "tree representation 3" in lecture and as described in problem 1). Suppose we do not leave the leftmost item blank as we did in lecture, i.e. the root of the tree contains the value 1, its left child contains the value 2, etc. Which of the following is the reason that **this** array does not represent a **binary search tree**? Choose the best option.

○ It contains negative numbers.
○ 2 should not be the left child of 1.
○ 2 should not be the left child of -3.
○ The root is not the median value.

c) (40 Points) Just as in part b, suppose we treat the array [1, 2, -3, -1] above as a **complete binary tree** (which we called "tree representation 3" in lecture and as described in problem 1). Suppose we do not leave the leftmost item blank, i.e. the root of the tree contains the value 1, its left child contains the value 2, etc. Which of following is the reason that **this** array does not represent a **max heap**? Choose the best option.

○ It contains negative numbers.
○ -1 should not be below 2.
○ -3 should not be below 1.
○ The root is not the maximum value.

d) (50 Points) Consider the array below.



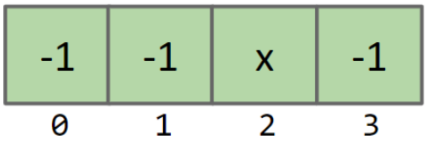
Give **an integer value** for x such that the resulting tree representation 3 interpretation is a valid **BST**. Assume we are not leaving the leftmost item blank. If there is no possible x, write "impossible".

x: _____2_____

e) (50 Points) Now give **an integer value** for x so that the array is a valid WeightedQuickUnion parent array. Assume we're using the array representation used in lecture and on the HW. If there is no possible x, write "impossible".

x: _____3_____

f) (30 Points) Consider the array below.



Give an **integer value for x** such that the resulting tree representation 3 interpretation is a valid **max heap**. If there is no possible x, write "impossible".

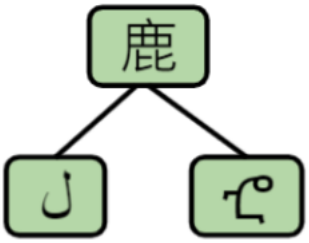
x: _____-1_____

g) (30 Points) Now give an integer value for x such that the array is a valid WeightedQuickUnion parent array. Assume we're using the array representation used in lecture and on the HW. If there is no possible x, write "impossible".

x: _____-1_____

3) BSTs, WQUs and Heaps

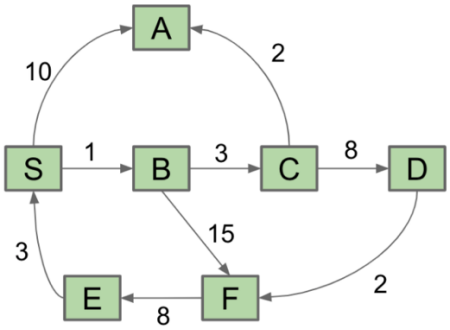
Answer the following questions regarding **BSTs**, **Heaps**, and WeightedQuickUnion. Remember the "level" of a node is defined as the number of edges between it and the root, and the "height" of some tree is defined as the number of edges from the root to the furthest away leaf. For example, for the tree below, the root is at level 0, its children are at level 1, and the height of the tree is 1.



a) (50 Points) Any binary tree of comparable items can be transformed into a valid **BST** using only rotation operations.	○ True ○ False
b) (40 Points) How many possible unique **BSTs** of height 2 with the elements 1, 2, 3, 4, 5, 6, 7 exist?	Answer: _____1_____
c) (30 Points) How many nodes are at **level 3** in a **heap** with 8 elements inserted? The root is at level 0.	Answer: _____1_____
d) (30 Points) How many nodes are at **level 3** in a **heap** with 12,329 elements inserted? The root is at level 0.	Answer: _____8_____
e) (50 Points) For any series of connect and isConnected operations, both a WeightedQuickUnion and a HeightedQuickUnion (connect the root of the **shorter** tree to the root of the **taller** tree) will result in the same final tree structure (if you were to draw out the trees represented by the underlying arrays), as long as you use the same tiebreaking strategy.	○ True ○ False
f) (50 Points) What is the maximum height of a fully connected WeightedQuickUnion with 8 elements?	Answer: _____3_____

4. Finding the Shortest Path! (125 Points)

Consider the following directed graph with weighted edges:



For all parts of this question, assume we break ties alphabetically. For Dijkstra's algorithm, assume all vertices except the starting vertex are inserted into the priority queue with infinite priority in the beginning.

a) (75 Points)
If we run Dijkstra's algorithm on the above graph starting from "S", after we visit vertex C (right after all of C's outgoing edges are relaxed, and right before we are going to pop the next vertex off of the fringe), what are the priorities of each vertex? If a vertex has infinite priority, input **"inf"** into the corresponding box. If a vertex is not in the priority queue, input **"none"** into the corresponding box.

Priority of S: _____none_____

Priority of A: _____6_____

Priority of B: _____none_____

Priority of C: _____none_____

Priority of D: _____12_____

Priority of E: _____inf_____

Priority of F: _____16_____

b) (50 Points)

If we run Dijkstra's algorithm **to completion** on the above graph starting from "S", in what order will the vertices be visited (popped off of the fringe)? Include S in your answer. Leave your answer as a **comma separated** list of vertex names **with no spaces**, for example "B,C,S,F,E,D,A". Make sure to double check that your list is exactly 7 vertices long.

Order: __S, B, C, A, D, F, E_____

5. Subsequences (240 Points)

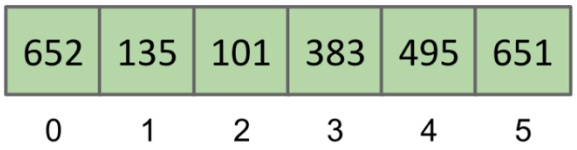
In this problem, we will analyze the behavior of different sorting algorithms on the array [652, 135, 101, 383, 495, 651]. For each part below, we will give a partial state of the array, e.g. [1, 2, 3, ...] where the ... represents the rest of the array, and you will select **all** sorting algorithms that could have the given array state **at any point** during the sorting algorithm's execution (i.e., even in the middle of an operation). If none of the above apply, select **None**.

Hint: You do not need to manually carry out all four sorting algorithms in their entirety to find the answer! You can do this, of course, but manually carrying out every sort (especially heapsort) will be tedious and much slower than thinking more carefully.

For this part, the possible sorting algorithms are:

- Insertion sort
- Selection Sort
- Quicksort using leftmost item as pivot, no shuffling, and with Hoare partitioning
- Heapsort using a max heap and bottom up heapification

Here is a diagram of the array that is being sorted for your convenience:



a) (50 Points) [652, 495, 651, ...]

☐ Insertion sort ☐ Selection sort ☐ Quicksort ☐ Heapsort

b) (50 Points) [101, 135, 652, ...]

☐ Insertion sort ☐ Selection sort ☐ Quicksort ☐ Heapsort

c) (50 Points) [135, 101, 652, ...]

☐ Insertion sort ☐ Selection sort ☐ Quicksort ☐ Heapsort

2. Quicksort and Heap Convergence

a) (50 Points) The array [383, 135, 101, 495, 651, 652] occurs during execution for both Quicksort and Heapsort when applied to the array from part 1. Assume Quicksort and Heapsort are as described in part 1. After how many completed partition operations does this array occur for Quicksort?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7

b) (50 Points) The array [383, 135, 101, 495, 651, 652] occurs during execution for both Quicksort and Heapsort when applied to the array from part 1. Assume Quicksort and Heapsort are as described in part 1. After how many completed deletion operations does this array occur for Heapsort?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7

6. MapDeque. (240 points)

Suppose we have the Deque interface defined as follows:

```
public interface Deque<T> {
    void addFirst(T item); // adds an item to the front of the deque
    void addLast(T item);  // adds an item to the end of the deque
    T removeFirst();       // gets the front item
    T removeLast();        // gets the ending item
    int size();            // number of items in the deque
}
```

Fill in the MapDeque class below. The class implements the Deque interface, and should behave exactly as ArrayDeque or LinkedListDeque from Project 1. If you didn't do Project 1, a Deque is just a list where you can only remove or add from the front and back, as given in the API above.

As a refresher, all of the following test cases must pass as a result of your implementation:

```
Deque<Integer> md = new MapDeque<>();
assertEquals(0, md.size());

md.addFirst(0); // after this operation, deque is: [0]
md.addLast(1);  // after this operation, deque is: [0, 1]
assertEquals(2, md.size());

assertEquals(0, md.removeFirst()); // after this op, deque is: [1]
assertEquals(1, md.removeLast());  // after this op, deque is empty
assertEquals(0, md.size());
```

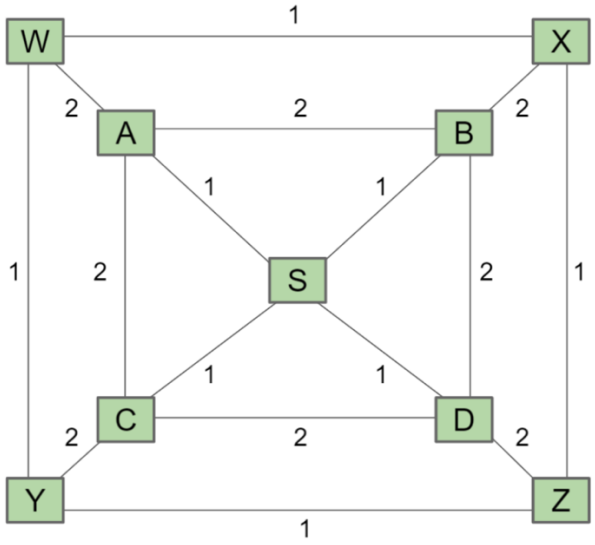
You **MAY NOT** add extra lines or write any additional methods. Also, you **MAY NOT** use any imports other than those already imported. You may leave blank lines blank. Failure to follow these guidelines will result in loss of points. For full credit, you should avoid **loitering**, i.e. do not store references to unnecessary objects. You may find the Map interface on the reference sheet useful.

```
import java.util.HashMap;
import java.util.Map;
import java.util.NoSuchElementException;

public class MapDeque<T> implements Deque<T> {
    private Map<Integer, T> items;
    private int first; // YOU MAY NOT USE ANY LIST DATA STRUCTURES
    private int last; // i.e. no using java.util.List, etc.

    public MapDeque() {
        items = new HashMap<>();
        first = 0;
        last = 1;
    }

    public int size() {
        return items.size();
    }
```

```
    public void addLast(T item) {
        items.put(last, item);
        last += 1;
    }

    public void addFirst(T item) {
        items.put(first, item);
        first -= 1;
    }

    public T removeFirst() {
        if (size() <= 0) {
            throw new NoSuchElementException("Size is <= 0");
        }
        First += 1;
        T toRet = items.remove(first);
        return toTet;
    }

    public T removeLast() {
        if (size() <= 0) {
            throw new NoSuchElementException("Size is <= 0");
        }
        last -= 1;
        T toRet = items.remove(last);
        return toRet;
    }
}
```

7. MSTs (180 points).

Consider the graph below:



a) (60 Points)

If we run Kruskal's algorithm, which of the following edges could be the last added to the MST? Select all that apply. Assume any ties are broken randomly.

☐ SA ☐ SB ☐ SC ☐ SD ☐ AB ☐ AC ☐ BD ☐ CD
☐ WA ☐ XB ☐ YC ☐ ZD ☐ WX ☐ XZ ☐ WY ☐ YZ

b) (60 Points)

If we run Prim's algorithm from S, which of the following edges could be the last added to the MST? Select all that apply. Assume any ties are broken randomly.

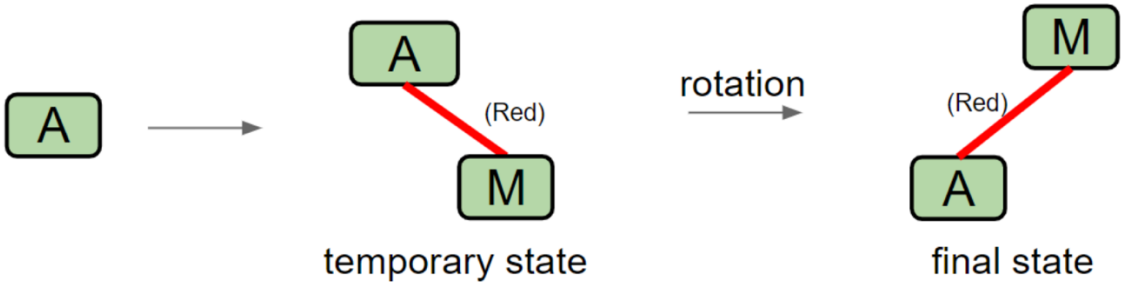
☐ SA ☐ SB ☐ SC ☐ SD ☐ AB ☐ AC ☐ BD ☐ CD
☐ WA ☐ XB ☐ YC ☐ ZD ☐ WX ☐ XZ ☐ WY ☐ YZ

c) (60 Points) How many valid MSTs are there for this graph?

Answer: _____16_____

8. Left Leaning Red Black Trees (160 Points).

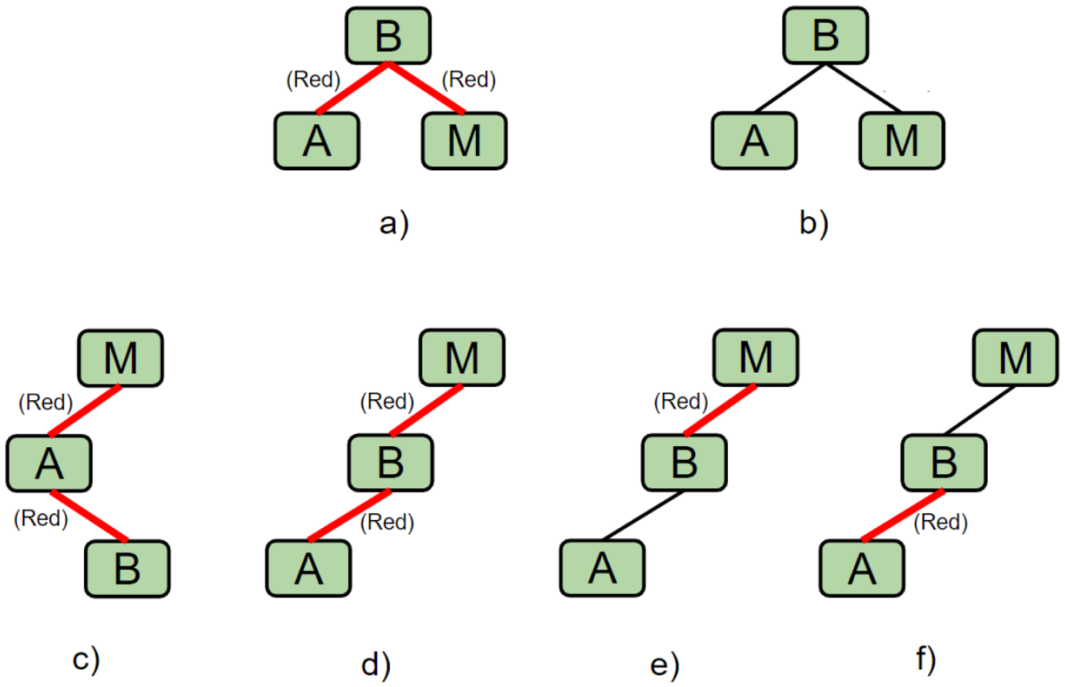
a) (25 points) If we have an LLRB (left leaning red black) tree with a single node A, then we add M, the M is temporarily added to the right hand size of A before a rotation is performed to make the tree obey the LLRB tree invariants, yielding the tree shown below on the right.



What rotation is performed to complete the operation shown above, i.e. which rotation occurs between the temporary state and the final state?

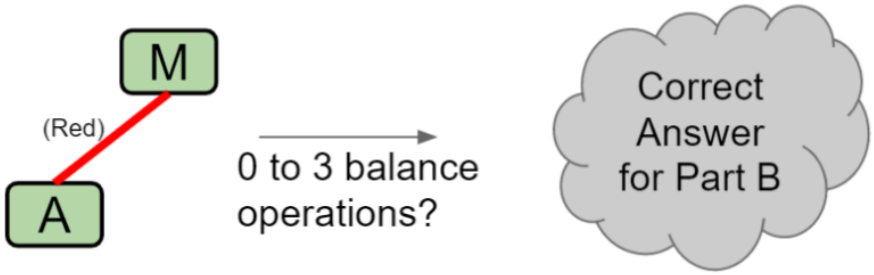
- ○ rotateLeft(A)
- ○ rotateRight(A)
- ○ rotateLeft(M)
- ○ rotateRight(M)

b) (60 points) Now suppose that we add B to the LLRB tree above (after the rotation is complete). Which of the following will be the **final** resulting LLRB? (Hint if you're stuck: What will the corresponding 2-3 tree look like?)



○ a ○ b ○ c ○ d ○ e ○ f

c) (75 points) What rotations and color flips are necessary after the addition of B? In other words, what rotations and color flips do we need to get from the final state given in part a of this problem to the correct final state for part b of this problem, as shown below:



If only two operations are necessary, pick "None" for the third operation. If only one operation is necessary, pick "None" for the second and third operation. If no operations are necessary, pick "None" for all three operations.

	Operation applied	Node to apply on
1st operation	◯ rotateLeft() ◯ rotateRight() ◯ colorFlip() ◯ None	◯ A ◯ B ◯ M ◯ None
2nd operation	◯ rotateLeft() ◯ rotateRight() ◯ colorFlip() ◯ None	◯ A ◯ B ◯ M ◯ None
3rd operation	◯ rotateLeft() ◯ rotateRight() ◯ colorFlip() ◯ None	◯ A ◯ B ◯ M ◯ None

d) PNH (0 points). In 2006, a bug was found in Java's binary search and mergesort implementation which caused them to fail castrophically on arrays of over approximately one billion items. How long had this bug been present?

Duration: _____

9. Palindromic Tries (330 Points).

A **palindrome** is defined as a sequence of characters which reads the same forward and backward, such as "ada" and "huh".

A **substring** is defined as a contiguous sequence of characters within a string.

- For example, "bob" is a substring of "aboba". Note that "aoa" is not a substring of "aboba" as the characters are not contiguous (next to each other in sequence).

A **centered subpalindrome** of an odd-length string s is defined as an odd-length substring that is also a palindrome and which shares its middle character with s.

- For example, "omzmo" is a centered subpalindrome of "momzmom" while "mom" is not because it does not share its middle character with momzmom.

Given a collection of strings, let numCenteredSubPalindromes(String subStr) be the number of strings in the collection which have subStr as a centered subpalindrome. For example, for the list ["madam", "ada", "wow", "dadad", "huh", "adamada"], numCenteredSubPalindromes("ada") would evaluate to 3.

a) (50 Points) Which strings in this list have "ada" as a centered subpalindrome? Note that the numCenteredSubPalindromes("ada") method evaluates to 3, so you should check 3 boxes below.

☐ "madam" ☐ "ada" ☐ "wow" ☐ "dadad" ☐ "huh" ☐ "adamada" ☐ None

If all of the strings in the collection are palindromes, it turns out we can implement the numCenteredSubPalindromes(String subStr) operation extremely efficiently. The resulting data structure is a special version of a Trie, which we'll call a PalindromicTrie with the methods (and runtimes) given below:

void insert(String str):

- Method description: It inserts a palindrome str into the PalindromicTrie.
- Runtime: $\theta(L)$, where L is the length of the string we are inserting.

int numCenteredSubPalindromes(String subStr):

- Method description: It returns how many strings in the PalindromicTrie have subStr as a *centered subpalindrome*.
- Runtime: $\theta(K)$, where K is the length of the parameter subStr and constant with respect to the total number of strings that have subStr as a *centered subpalindrome*.

Other assumptions:

- We will only insert **odd-length palindromes** into the PalindromicTrie (you do not need to check for this).
- All strings are in lower case.

For the rest of this problem you'll design the internals of the PalindromicTrie. In part b, you'll determine an additional instance variable for each node, and in part c, you'll write the numCenteredSubPalindromes method. Recommended but not required: Start by trying to create a design without writing any code, i.e. by drawing out the state of the PalindromicTrie after inserting ["madam", "ada", "wow", "dadad", "huh"]. Note that your design cannot simply be a Trie as designed in class (where the root has children m, a, w, d, and h), as this will not meet the runtime requirements above. If you're stuck, see parts b and c to help you think through some subtasks you'll need to resolve.

Example:

```
PalindromicTrie pt = new PalindromicTrie();
pt.insert("madam");
pt.insert("ada");
pt.insert("wow");
pt.insert("dadad");
pt.insert("huh");
pt.numCenteredSubPalindromes("d");     // returns 3
pt.numCenteredSubPalindromes("ada");   // returns 3
pt.numCenteredSubPalindromes("odo");   // returns 0
pt.numCenteredSubPalindromes("madam"); // returns 1
pt.numCenteredSubPalindromes("o");     // returns 1
pt.numCenteredSubPalindromes("wow");   // returns 1
pt.numCenteredSubPalindromes("x");     // returns 0
```

b) (60 points) Keeping in mind the requirements for insert(String str) and numCenteredSubPalindromes(String subStr), select one additional instance variable for the TrieNode class and declare it below. You may not remove or alter the instance variable we've already provided.

```
import java.util.*;

public class PalindromicTrie {
    private class TrieNode {
      private Map<Character, TrieNode> children; // <-- do not change
      private int numWordsBelow;
    }
    ...
}
```

c) (220 points) Now let's implement numCenteredSubPalindromes(String subStr). Remember that numCenteredSubPalindromes(String subStr) should run in , where is the length of the parameter subStr, and should run in constant time with respect to the total number of strings that have subStr as a *centered subpalindrome*. You can assume insert(String str) has been implemented for you and will set your instance variables correctly. **You might not need all lines**, but you **may not** add additional lines.

Hint: Map<K, V> has a method V getOrDefault(K key, V default) which returns the value corresponding to key if it is in the Map, and the value default if key is not in the Map. Consider using this method on node.children somewhere in this question, with a default value of null.

```
public class PalindromicTrie {
    /*
       ... Not shown: insert, constructor, definition of TrieNode
       ...            Assume TrieNode is as you defined in part b.
    */

    /** Returns true if s is an odd length string that is a palindrome. */
    private boolean isOddLengthPalindrome(String s) {
        // code not shown. DO NOT WRITE.
    }

    private TrieNode root;

    public int numCenteredSubPalindromes(String subStr) {
        if (!isOddLengthPalindrome(subStr)) {
            throw new IllegalArgumentException("Must be an odd length palindrome.");
        }
        return numCenteredSubPalindromesHelper(root, subStr, subStr.length()/2);
    } // YOU MAY NOT MODIFY THE SKELETON CODE
      // YOU MAY NOT ADD ADDITIONAL LINES
      // YOU MAY LEAVE LINES BLANK THAT YOU DON'T WANT TO USE

    private int numCenteredSubPalindromesHelper(TrieNode node, String subStr, int
index) {
        if (node == null) {
            return 0;
        }
        if (index == subStr.length()) {
            return node.numWordsBelow;
        }
        TrieNode toCall = node.children.getOrDefault(subStr.charAt(index), null);
        return numCenteredSubPalindromesHelper(toCall, subStr, index + 1);
    } // YOU MAY NOT MODIFY THE SKELETON CODE
      // YOU MAY NOT ADD ADDITIONAL LINES
      // YOU MAY LEAVE LINES BLANK THAT YOU DON'T WANT TO USE
}
```

10. Graphs (350 Points).

a) (130 Points)

Consider the following algorithm:

```
public MinPQ<Vertex> getVertices(Vertex s) {
    MinPQ&<Vertex>; pq = new MinPQ<>();
    pq.insert(s);
    TreeSet<Vertex>; seen = new TreeSet<>();
    collect(s, pq, seen);
    return pq;
}

public void collect(Vertex v, MinPQ<Vertex> pq, TreeSet<Vertex> seen) {
   for (Vertex w : v.neighbors()) {
      if (!seen.contains(w)) {
        pq.insert(w);
        seen.add(w);
        collect(w, pq);
      }
   }
}
```

For each of the following operations, give the number of operations in *O* notation needed to complete execution on the graph. Also, give the runtime of the operation in *O* notation. Assume comparing two vertices takes constant time. Assume the TreeSet is a balanced search tree (specifically, it's a red black tree). Assume the MinPQ is a heap based PQ. Give your answer in terms of *E* and *V*, the number of total edges and vertices in the graph respectively. Give the tightest possible *O* bound.

Please follow the following format when writing your answers on this problem to avoid having to request a regrade:

- Omit the multiply sign *.
- Don't include parentheses after logarithms (e.g. write logV rather than log(V)).
- Use ^ to indicate exponentiation.
- Do not explicitly write big O, i.e. assume that the O is already written for you.
- As you should always do, if any term includes log E or V^k, write as log V . This is because O(logE) and O(V^k) are the same thing as O(log V) (you can prove this, we won't do so here!).
- For example, if you thought the runtime for the algorithm in part c was O(EV^4logV) , you could write EV^4logV in the corresponding blank.

contains calls: _____E_____ runtime per contains: _____logV_____
pq insert calls: _____V_____ runtime per pq insert: _____logV_____
seen add calls: _____V_____ runtime per seen add: _____logV_____

b) (90 Points)

Give the overall runtime of the algorithm in terms of *E* and *V*.

overall runtime: _____ElogV_____

c) (130 Points)

Now suppose we take away the TreeSet so that the algorithm becomes:

```
public MinPQ<Vertex> getVertices(Vertex s) {
    MinPQ<Vertex> pq = new MinPQ<>();
    pq.insert(s);
    collect(s, pq, seen);
    return pq;
}

public void collect(Vertex v, MinPQ<Vertex> pq) {
   for (Vertex w : v.neighbors()) {
      if (!pq.contains(w)) {
        pq.insert(w);
        collect(w, pq);
      }
   }
}
```

Assume that the runtime for the MinPQ contains operation is O(N) for a MinPQ with N elements in it. What is the overall runtime of the algorithm in terms of E and V.

overall runtime: _____EV_____

11. Asymptotics (190 Points).

Answer the following questions regarding runtime analysis.

Consider the code below.

```
public static int f1(int n) {
    int[] arr = new int[n + 1];
    arr[0] = 1;
    arr[1] = 1;
    arr[2] = 1;
    for (int i = 3; i &lt;= n; i += 1) {
        arr[i] = arr[i - 1] * arr[i - 2] * arr[i - 3];
    }
    return arr[n];
}
```

a) (30 Points) What is the runtime for f1(N)?

○ θ(1) ○ θ(log (log N)) ○ θ((log N)^2) ○ θ(log N) ○ θ(N) ○ θ(N log N) ○ θ(N^2)

○ θ(N^2 log N) ○ θ(N^3) ○ θ(N^3 log N) ○ θ(N^4) ○ θ(N^4 log N) ○ Worse than θ(N^4 log N)

○ Never terminates (infinite loop)

Consider the code below.

```
public static void z1(int n) {
    if (n &lt;= 10) {
        return;
    } else {
        z1(n / 2);
        z1(n / 2);
    }
}
```

b) (50 Points) What is the worst case runtime for z1(N)?

○ θ(1) ○ θ(log (log N)) ○ θ((log N)^2) ○ θ(log N) ○ θ(N) ○ θ(N log N) ○ θ(N^2)

○ θ(N^2 log N) ○ θ(N^3) ○ θ(N^3 log N) ○ θ(N^4) ○ θ(N^4 log N) ○ Worse than θ(N^4 log N)

○ Never terminates (infinite loop)

Consider the code below.

```
public static void z2(int n) {
    for (int i = 0; i < n; i += 1) {
        System.out.println("hi");
    }
    if (n &lt;= 10) {
        return;
    } else {
        z2(n / 2);
        z2(n / 2);
    }
}
```

c) (50 Points) What is the worst case runtime for z2(N)?

○ θ(1) ○ θ(log (log N)) ○ θ((log N)^2) ○ θ(log N) ○ θ(N) ○ θ(N log N) ○ θ(N^2)

○ θ(N^2 log N) ○ θ(N^3) ○ θ(N^3 log N) ○ θ(N^4) ○ θ(N^4 log N) ○ Worse than θ(N^4 log N)

○ Never terminates (infinite loop)

Consider the code below.

```
public static void z3(int n) {
    if (n &lt;= 10) {
        return;
    } else if (n % 2 == 0) {
        z3(n / 2);
        z3(n / 2);
    } else {
        z3(n + 1);
    }
}
```

d) (30 Points) What is the **best case** runtime for z3(N)?

○ θ(1) ○ θ(log (log N)) ○ θ((log N)^2) ○ θ(log N) ○ θ(N) ○ θ(N log N) ○ θ(N^2)

○ θ(N^2 log N) ○ θ(N^3) ○ θ(N^3 log N) ○ θ(N^4) ○ θ(N^4 log N) ○ Worse than θ(N^4 log N)

○ Never terminates (infinite loop)

e) (30 Points) What is the **worst case** runtime for z3(N)?

○ θ(1) ○ θ(log (log N)) ○ θ((log N)^2) ○ θ(log N) ○ θ(N) ○ θ(N log N) ○ θ(N^2)

○ θ(N^2 log N) ○ θ(N^3) ○ θ(N^3 log N) ○ θ(N^4) ○ θ(N^4 log N) ○ Worse than θ(N^4 log N)

○ Never terminates (infinite loop)

12. Sorting Lists (620 Points).

Suppose we have the PosIntegerList class, a list data structure that can only hold positive integers:

```
/** List class containing positive integers. */
public class PosIntegerList {
    public PosIntegerList();    // constructor for an empty PosIntegerList

     public int size();          // returns the number of elements in the PosIntegerList
    public int get(int index);  // returns the int at position "index" in the PosIntegerList
    public int add(int item);   // adds the int "item" to the back of the PosIntegerList

    public static PosIntegerList of(int ... items);
    // returns a PosIntegerList containing items, "int ... items" just
    // means you can pass any number of arguments
    // e.g. PosIntegerList.of(1, 2, 3); will return an PosIntegerList
    // containing 1, then 2, then 3.
}
```

Suppose we want to sort a collection of PosIntegerLists. We compare PosIntegerLists by comparing their digits, treating the leftmost digit as the most significant digit. For example, the list [5, 3, 1, 104] would be considered greater than [5, 2, 91, 150], because 3 is greater than 2. If two lists have the same numbers but one list is longer, then the longer list is greater. For example, [3, 9, 15] is greater than [3, 9], but less than [3, 9, 15, 20].

The easiest way to sort a list of PosIntegerLists is to write a Comparator for PosIntegerList and use Collections.sort, e.g. if we have ArrayList<PosIntegerList> lil, then Collections.sort(lil, new PosIntegerListComparator()) will mutate lil so that it is sorted. A full example is shown below.

```
PosIntegerList l1 = PosIntegerList.of(1, 2, 3, 4);
PosIntegerList l2 = PosIntegerList.of(3, 3, 2, 105);
PosIntegerList l3 = PosIntegerList.of(3, 1, 3, 3);
PosIntegerList l4 = PosIntegerList.of(9, 6, 3, 1);
PosIntegerList l5 = PosIntegerList.of(9, 6, 3);
PosIntegerList l6 = PosIntegerList.of(1, 1, 1);
List<PosIntegerList> lil = new ArrayList<>();
lil.add(l1);
lil.add(l2);
lil.add(l3);
lil.add(l4);
lil.add(l5);
lil.add(l6);

Collections.sort(lil, new PosIntegerListComparator());
System.out.println(lil);
/* correct output should be:
[[1,  1,  1],
 [1,  2,  3,   4],
 [3,  1,  3,   3],
 [3,  3,  2, 105],
 [9,  6,  3],
 [9,  6,  3,   1]] */
```

a) (250 Points) Fill in the PosIntegerList comparator below. **The closing brace must be on or before line 19.** You don't have to use our blanks, though it is recommended. You may not import anything, and your code may not modify either list passed in.

```
public static class PosIntegerListComparator implements Comparator<PosIntegerList> {
    public int compare(PosIntegerList a, PosIntegerList b) {
        int lim = Math.min(a.size(), b.size());
        for (int i = 0; i < lim; i++) {
            if (a.get(i) != b.get(i)) {
                return a.get(i) - b.get(i);
            }
        }
        return a.size() - b.size();
    }
} // if necessary you may add more lines. This closing brace
  // must be on line 19 or earlier
  // YOUR CODE SHOULD NOT MODIFY EITHER LIST!
```

b) (60 Points) Assuming we have N lists of length M, and that Collections.sort uses Mergesort, what is the **worst case** runtime of the call to Collections.sort? Assume that all PosIntegerList instance methods take constant time. Assume that the compare(PosIntegerList a, PosIntegerList b) operation is implemented as efficiently as possible. You may do this problem even if you did not complete part a.

○ θ(1) ○ θ(N) ○ θ(M) ○ θ(N+M) ○ θ(NM) ○ θ(N log M) ○ θ(M log N)

○ θ(N log N) ○ θ(NM log NM) ○ θ(NM log N) ○ θ(NM log M) ○ θ(N log N + M log M)

c) (250 Points) There are, of course, an infinite number of ways to sort a List of PosIntegerLists. In this part, we will use LSD Radix Sort with Heapsort as the subroutine to sort the List of PosIntegerLists.

Fill in the code below to implement the PQNode class and List<PosIntegerList> heapSortLists(List<PosIntegerList> slls) method. You may have up to 3 instance variables in your PQNode. You may not use additional imports. You may not add additional lines. You may not modify the skeleton code. Hint: For full credit, you'll need to make sure that each pass of Heapsort is stable somehow (this is tricky!!).

Note that the logic for the LSD sort portion is already completed for you. Your job is just to make each Heapsort pass works correctly.

Your code will receive partial credit if it only works for equal sized lists. Your code will also receive partial credit if the HeapSort is not stable. Our recommended approach is to first write the code so that it only works for equal sized lists and does not do anything special to ensure stability. Then try to adapt your code to work for unequal sized lists. Then try to adapt your code so that the HeapSort is stable.

```
import edu.princeton.cs.algs4.MinPQ;
import java.util.ArrayList;
import java.util.List;

public class HeapPosIntegerListSort {
    /** Returns the length of the longest PosIntegerList
        contained in ils. YOU WILL NOT NEED TO USE THIS
        FUNCTION YOURSELF! */
    public static int maxLength(List<PosIntegerList> ils) {
        int maxSize = 0;
        for (int i = 0; i < ils.size(); i += 1) {
            PosIntegerList s = ils.get(i);
            if (s.size() > maxSize) {
                maxSize = s.size();
            }
        }
        return maxSize;
    }

    private static class PQNode implements Comparable<PQNode> {
        PosIntegerList il;       // you may add up to 2 additional instance variables
        val = v; // instance variable 1
        tiebreak = t; // instance variable 2

        PQNode(PosIntegerList i, int v, int t) {
            il = i;
            val = v; // set instance variable (if applicable)
            tiebreak = t; // set instance variable (if applicable)
       }

       public int compareTo(PQNode o) { // you may not need all lines
            if (val != o.val) {
                  return val - o.val;
            } else {
                  return tiebreak - o.tiebreak;
            }
       } // this comment must be on line 41 or earlier (ours is on line 38)
    }

    public static List<PosIntegerList> heapSortLists(List<PosIntegerList> ils) {
        for (int i = maxLength(ils) - 1; i >= 0; i --) {
            ils = heapSortLists(ils, i);
        }
        return ils;
    }

    /** Stably heapsorts the given list of PosIntegerLists
        using only the ith entry of each list. */
    public static List<PosIntegerList> heapSortLists(List<PosIntegerList> ils, int i) {

        MinPQ<PQNode> pq = new MinPQ<>();
        int count = 0; // you may not need this line


        /* YOU MAY NOT MODIFY ANY SKELETON CODE!!! */
        /* All editable lines in this function are marked with // comments */
        for (PosIntegerList currentIl : ils) { // you may not modify skeleton code!
```

```
            PQNode p;
            if (i >= currentIl.size()) {// if you don't need two conditions just put true
here
                p = new PQNode(currentIl, 0, count); // fill in
            } else {
                p = new PQNode(currentIl, currentIl.get(i), count); // you may not need
this line
            }
            count += 1; // you may not need this line
            pq.insert(p);
        }

        List<PosIntegerList> returnList = new ArrayList<>();
        while (pq.size() != 0) {
            PQNode sll = pq.delMin();
            returnList.add(sll.il);
        }

        return returnList;
    }
}
```

d) (60 Points) Suppose we have PosIntegerLists of length . What is the **worst case** runtime to complete an LSD-heapsort on this list of lists? You may complete this problem even if you did not complete part c. Assume that all PosIntegerList instance methods take constant time.

○ θ(N) ○ θ(M) ○ θ(N+M) ○ θ(NM) ○ θ(N log M) ○ θ(M log N)

○ θ(N log N) ○ θ(NM log NM) ○ θ(NM log N) ○ θ(NM log M) ○ θ(N log N + M log M)