Practice Final

This test has 13 questions worth a total of 200 points, and is to be completed in 180 minutes. The exam is closed book, except that you are allowed to use one double sided 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. You may do this before the exam begins.

#	Points	#	Points
1	16	8	0
2	12	9	10
3	9	10	15
4	25	11	20
5	8	12	17
6	25	13	30
7	9		
		Total	200

1. Starting this Test on a Positive Note (15 pts).
a) (4 pts) Give an example where Dijkstra's would fail because of a negative edge weight

b) (4 pts) Give an example where Dijkstra's **does not** fail because of a negative edge weight. State assumption that you must make in order for this to work. If you have some method to make it work, explain the method.

c) (4 pts) What is the least amount of negative edges (>0) that a graph can have and still have Dijkstra's work? What is the most amount of negative edges that we can have? State assumptions that must be made

d) (4 pts) What is the smallest weight we can have and guarantee that Dijkstra's will always work?

2. Sorting and Fun (12 pts). We worked with a few various sorts. For each sort, provide the runtime and give a brief description of what it does (how it sorts). Each questions is worth 4 points

	private static void dwarfSort(int[] ar) {
	  int i = 1;
	  int N = ar.length;
	  while (i < N) {
	  if (i == 0 || ar[i - 1] <= ar[i]) {
	  i++;
	  } else {
	  int tmp = ar[i];
	  ar[i] = ar[i - 1];
	  ar[i--] = tmp;
	  }
	  }
	}

	private static int[] waffle(int[] arr){
	  int[] syrup = arr;
	  for(int i = 1; i < arr.length; i++){
	  if(arr[i] < arr[i-1]){
	  arr[i] = arr[i] + 1;
	  }
	  }
	  return syrup;
	}

	//find the runtime of flapJackSort. Note that the flip function is defined as follows flip(array, index) and reverses the entire array up until that index.
	public static int[] flapJackSort(int arr[], int n){
	  for (int curr_size = n; curr_size > 1; curr_size--)
	  {
	  for(int choco = 0, chip =0; chip < curr_size; chip++) {
	  if (arr[chip] > arr[choco]) {
	  choco = chip;
	  }
	  if (choco != curr_size - 1) {
	  flip(arr, choco);
	  flip(arr, curr_size - 1);
	  }
	  }
	  }
	  return arr;}

3. The Fruits of Your Labor (9 pts).

a) (3 pts) You own a tree that can hold many fruits; however, you want to save space so that your neighbor doesn't cut off your branches. Find how you make one modification (change, delete, or add a letter) to one of the following 3 words "Apple", "Peach, "Pear". What modification can you do to save the most space? Draw the resulting trie after your modification.

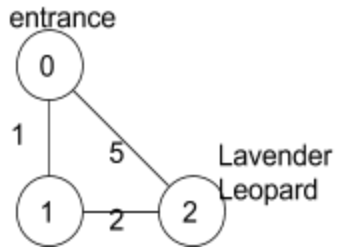
b) (3 pts) A wealthy CEO wants to encrypt all his secrets; however, he decided that he wants to store all of his secrets within a flashy hashmap. But he doesn't want a plain old hashmap, after all, that could get hacked. To be different, he decides that, for any given day, a secret will hash to different location than it would have the previous day. Is his idea valid? Provide your reasoning.

c) (3 pts) 2-3 Trees are known to be fast because they are self balancing and they have a nice low level? Why don't we just use 2-3-4 trees or 2-3-4-5-6 trees?

4. The Lavender Leopard (20 points).

You are a jewel thief starting at the entrance of a museum where all rooms are connected to each other- that is a room is connected to every room besides itself; however, hall lengths may differ. You are in search of the very expensive gem, the "Lavender Leopard", the only problem is that you have no idea what room it is in.
Your idea is to go to the room closest to the one you are in (excluding ones that you have already visited) and then as soon as you find the gem, you will immediately exit the building, going through the same rooms you did earlier; however, if there was a faster way to a room that is closer to the exit, you will go that way.

a) (20 pts) Fill in the Museum class and following code to find the gem and then escape as quickly as possible. You are guaranteed that the jewel will be in the building.



In the above example, you would travel to 1, then 2, get the gem, and go back to room 1 then the entrance.

```
public class Hall {
  public final Room room1; public final Room room2;
  public final double length;
  public Hall(Room r1, Room r2, double length){
     this.room1 = r1; this.room2 = r2; this.length = length;
  }
  public Room r1(){return room1;}
  public Room r2(){return room2;}
  public double howFar(){return this.length;}
```

```
public class Room {
  public LinkedList<__________> adjacentHall;
  boolean beenTo = false;
  public boolean hasGem;
  public int order; //provides a definite ordering for the rooms, does not tell you the
location, but can be used as a reference point.
  public Room(int order, boolean gem, int rooms) {
     this.order = order;
     this.hasGem = gem;
     adjacentHall = new LinkedList<Hall>();
     for(int i = 0; i < rooms; i++){
        adjacentHall.add(null);
}}}
```

```
import java.util.ArrayList; //feel free to use these imports, if not it's okay as well
import java.util.LinkedList;

public class Museum {
public int totalRooms;
public int totalHalls;
public LinkedList<_____> adj;
public Room entrance;

//Constructor
public Museum(Room[] rooms) {
  this.totalRooms = rooms.length;
  this.totalHalls = 0;
  adj = new LinkedList();
  for (_____) {

    _____

  }
  entrance = _____
}

//method for stealing from the museum
public void stealing() {
  ArrayList<Room> visited = new ArrayList<>();

  _____

  _____

}
```

```
//You are sneaking into the museum, attempting to find the jewel, this is the
hardest method of the class just because of sheer length.
public Room goIn(_____ visited) {
   Room r = entrance;
   boolean stolegem = false;

   while (_____) {

      _____

      _____

      if (r.hasGem) {

         _____

         break;
      }
      Hall minHall = _____

      for (int i = 0; _____; i++) {

         Hall curr = _____

         if (minHall _____) {

            if (curr != null &&

            _____ )) {
               minHall = curr;}
         } else if (curr != null &&

            _____ &&

            _____) {
            minHall = curr;}}

      if (_____) {
         r = minHall.to();
      } else {
         r = minHall.from();}}
   return r;}
```

```
//You have the gem now you are leaving. You can do this part without having done
the earlier method (there are no references to it; however, you should understand
what it is doing)
public void goOut(Room r, ArrayList<Room> visited) {

    int curr_lim = _____);

    while (_____) {
        Room closestroom = null;
        double dist = Double.MAX_VALUE;

        int currentlimit = _____


        for (int i = 0; i < _____; i++) {
            Room curr = visited.get(i);
            if (adj.get(r.order).size() < dist) {
                closestroom = curr;

                _____

                _____
            }
        }

     _____
    }
  }
    System.out.println("We outtie");
}
```

b) (5 pts) What is the worst case runtime of stealing in terms of rooms (r) and halls (h)? Only keep necessary terms.

5. Trick or Tree-t (12 pts). Mark true or false for the following problems, a brief justification is required. Each question is worth 3 points

a) The fastest time for any comparison based can ever be is **Ω**(Nlog(N)) regardless of input.
☐True ☐False

b) When looking through a binary search tree, if we are looking for a number of items between 2 elements, the best case runtime is Θ(N) because we will need to go through every single element to see if it fits into our range?
☐True ☐False

c) All valid left leaning Red Black Trees can become a valid Binary Search Tree by replacing all red nodes with black ones.
☐True ☐False

d) 2-3 Trees are faster than Red-Black Trees for most major operations because of the height of the 2-3 tree is smaller than the height of a Red-Black Tree.
☐True ☐False

6. The Lavender Leopard Strikes Again (20 pts). You were examining your gem after successfully stealing it only to realize that you had stolen a fake gem, and that the real Lavender Leopard is in a much more secure museum. In this museum, not every room is connected. To avoid ridiculous amounts of backtracking, you have decided to place teleporters in every room you visit. At any point, if there is any path that is closer to the entrance than the one you are taking, you will teleport to the room before and then travel to it. Once you find the gem, you need to get out as soon as possible. To get out, you will teleport in the reverse order that you arrived so you can collect all your teleporters (they aren't that cheap!). Note that Hall and Room have the same definition that they did for #4. Additionally, you **do not** need to redefine class level variables and the constructor (you can if you want to), as they are the same as the Museum class's . You have 3 pages to complete this.

If you cannot write the code, but can write the idea behind the problem, you can get up to % of the points.

Hint 1: Use old algorithms that we've gone over and see how you can modify them to match this problem

Hint 2: Look at other data structures.

```
//Optional Node class
class Node {

  _____

  _____

  public Node(_____) {

    _____

    _____

  }
}

//Method to teleport between rooms then teleport out of the museum.
 public void teleportInNOut() {

  _____

  _____

  _____

  }
```

```
//Method to steal the jewel from the museum

public void teleportSteal(_____) {

   Hall[] edgeTo = _____

   double[] distTo = _____

   PriorityQueue pq = _____

   for (_____) {

      _____

   }
   distTo[entrance.order] = 0.0;

   pq.insert(_____);

   while (_____) {

      _____

      _____

      _____

      if (_____) {
         break;
      }

      for (_____) {

         if (_____) {

            done(_____);
         }
      }
   }
}
```

```
\\Method header is not complete
public void done(Hall h, Room temp,

_____) {
      Room r;
      if (h.from().equals(temp)) {

          _____
      } else {

          _____
      }

      if (_____) {

          distTo[r.order] = distTo[temp.order] + h.length;

          edgeTo[r.order] = h;

          if (_____) {

         _____

          } else {

              _____
          }
      }
  }
}


//Method for leaving the museum

public void teleportLeave(_____) {

  while (_____) {

        _____
  }
  System.out.println("Out again");
}
```

7. D-Arranged Lab (9 pts). You are viewing a research group, unfortunately, this research group is full of some bumbling buffoons who happens to make a lot of mistakes. Each part is worth 3 points.

a) While putting his chemicals into test tubes, one of the scientists realized that he accidentally made a duplicate of one of his n chemicals, he just doesn't know which one. However, he does know that the duplicate element will be, at most, 10 test tubes away. What sort should he use to quickly find the duplicate? Assume that all the chemicals are in their natural order, except for the duplicate.

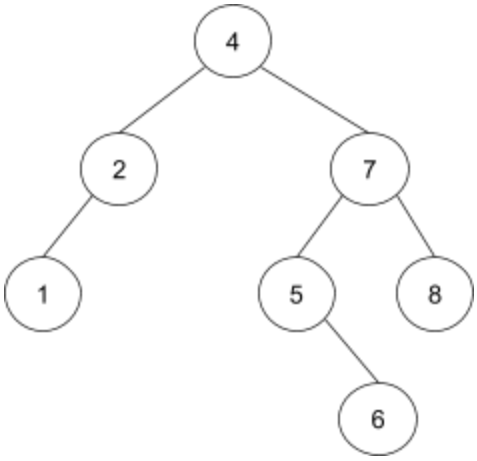
b) You have separated each of your n test tubes into k sections (so test tube 1 is now in seperated into k test tubes and so on). The only problem is that while they were all getting analyzed, you mixed them up again. Luckily, you have the name of the chemical and the time a chemical was put into a test tube. You want to sort your test tubes by chemical and then by amount of time it has been oxidizing. What sort can you use to ensure this is done as quickly as possible?

c) Now the scientists are going to input the name of their chemicals into a spreadsheet to see if they are dealing with your list of hazardous materials. The list is in alphabetical order so to easily cross reference what chemicals are dangerous, the scientists want to put the name of their chemicals in alphabetical order, what sort should they use?

8) Riddle me this:
Why is an Orange like a Bell?

9. Rotations and Aberrations (10 pts).

a) (5 pts) Create the 2-3 tree that corresponds to the below Binary Search Tree.



b) (5 pts) Create a Red-Black Tree that corresponds to the 2-3 tree/Binary Search Tree in part a.

10. JJJJ UNIT (15 pts).
You are starting your rapping career as a part of the prestigious rapping group J Unit. You want to recruit new members so you can have a hip possi. To prove their worth to your group, you have decided that you will test their vocabulary. To do this, you will make a TrieMap. This will make a Trie, but for each word, you also have a note to see how many times it has been used. If any word has been used more than 5 times, the person will not be able to join your group.
You are given the TrieNode class defined as follows:

```
public class TrieNode {
   char c;
   boolean isWord;
   String word;
   TrieNode[] children = new TrieNode[26];
   int count;}
```

Additionally, you are given that the TrieMap has a root class level variable and an **addWord** and **contains** method.
*The LinkedList has an addAll method which adds all items from a collection to the end of the list (e.g. LinkedList A = {A,B,C}, LinkedList B = {D, E,F}. A.addAll(B), A = {A,B,C,D,E,F}

Fill in the below methods

```
public LinkedList getAllWords(TrieNode start) {
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

```
public static boolean testCount(TrieMap t){

_____

_____

_____

_____

_____

_____

_____

_____

_____
}
```

11. Median Heap (20 pts).

We have earlier dealt with Min Heaps and Max Heaps, now we are going to explore a new type of heap, the **Median Heap**. The Median Heap will have 3 methods that you will need to fill in, **median**, which finds the median element, **insert**, which inserts an element into the Median Heap, and **balanceHeap**, which makes the heap balanced. You may need to use some of these methods inside of others. Feel free to use any data structure that we have gone over.

```
public class MedianHeap {

   private _____ top = new _____

   private _____ bottom = new _____

   //find the median element, if there are an even amount of elements, take the
average of the 2 elements closest to the middle
   public int median() {

      int minSize = _____

      int maxSize = _____

      if (_____) {

        _____
      }

      if (_____) {

         _____

      }
      if (_____) {

         _____

      }
      return _____

   }
```

```
//used to insert elements into the MedianHeap
public void insert(int element) {
```

_____

_____

_____

_____

_____

_____

_____

_____

```
balanceHeap();

}
```

```
 //Balance the heap
private void balanceHeap() {
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

```
}
```

12. Faster than a Speeding Runtime (17 pts)
Fill in the runtimes for each of the following operations. If there is no tight bound, provide a lower and upper bound. Provide a brief explanation as to why the runtime is what it is.

a) (5 pts) How much time does it take to create a heapify an array of N elements?

b) (3 pts) How much time does it take to find the minimum element in a Binary Search Tree

c) (3 pts) How much time does it take to place all the contents of a given MinHeap into an array that is ordered from least to greatest.

d) (3 pts) How much time does it take to find the minimum value in a MaxHeap.

e) (3 pts) Find the runtime of getting all the words from a trie and then using MSD sort on them.

13) Future-Roaming (30 points).
You are a croissant delivery boy in the 31st century. You want to deliver your croissants as quickly as possible; however, because you lost your client's addresses and they live all across the galaxy, so you don't want to go door to door. You have the phone numbers of all the people who ordered croissants from you, so you will attempt to use this to figure out in where you should go. Each phone number is about 10000000000000000000000 digits long and you have a total of N clients. You will use the following information about telephone numbers to help:

1. The sector code (the leftmost 10 digits) will allow you to figure out which sector a client is in. On average, there are a total of $\sqrt{N}$ of households you need to deliver to in the sectors.
2. In the 31st century, telephone numbers have a cool feature where the difference between the last 4 digits of any 2 telephone numbers in the same sector is equal to the distance between their two corresponding households.
3. You know the address of 1 of the houses you need to go in each sector.

You want to go to sectors based off how many of your clients are in them and how close it is . If a sector has 5 clients and is 2 Megadistances away, it is more appealing than a sector that has 1 client and is 1 Megadistance away. If a sector has 6 clients and is 2 Megadistances away, it is less appealing than a sector with 9 clients and 3 Megadistances. If a sector has 4 clients and it is 2 Megadistances away, it is considered equal to a sector that has 2 clients that is 1 MegaDistance away, at this point, you just pick which one to go to randomly . You will deliver to all the households in that sector before going to the next sector. You also know how many megadistances are between each sector, and you can assume that you will start at a location that is equidistant from all the sectors.

Provide an implementation where figuring out which households are in which sector in **Θ(N)** time, figuring out the distance between a house and all the other houses in a sector, must take, on average,**O(** $\sqrt{N}$ **)** time, and delivering croissants to all the households on average, **O(** $N^{\frac{3}{2}} log\sqrt{N}$) time.

Assume that the time it takes you to walk 1 Minidistance (inside of a sector) is the same amount of time it takes your spaceship to travel one Megadistance (between sectors). Justify why your design works in the given time.