UC Berkeley – Computer Science
CS61B: Data Structures

Final Exam, Spring 2017.

This test has 13 questions worth a total of 200 points, and is to be completed in 165 minutes. The exam is closed book, except that you are allowed to use three double sided written cheat sheets (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.**

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

Signature: ______________________________

#	Points	#	Points
0	0.5	7	23
1	12	8	0
2	28	9	16
3	14	10	16
4	12	11	12
5	16	12	12
6	12	13	26.5
		TOTAL	200

Name: ______________________________

SID: ______________________________

Three-letter Login ID: __________

Left SID: ________________________

My left neighbor has ID: ________

Right SID: _______________________

My right neighbor has ID: _______

Exam Room: _______________________

Tips:

- 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.
- 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.
- **See the coding reference sheet on the last page for potentially useful data structures.**
- Unless otherwise stated, all given code on this exam should compile. 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. Unless we specifically give you the option, the correct answer is not 'does not compile.'
- ○ indicates that only one circle should be filled in.
- ☐ indicates that more than one box may be filled in.
- For answers which involve filling in a ○ or ☐, **please fill in the shape completely.**

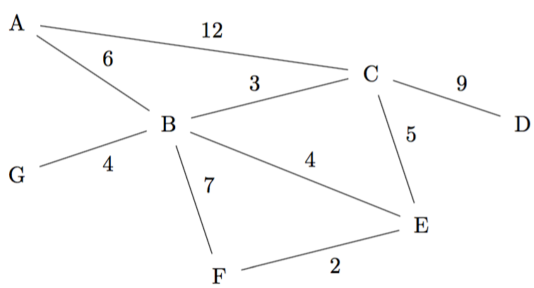
Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

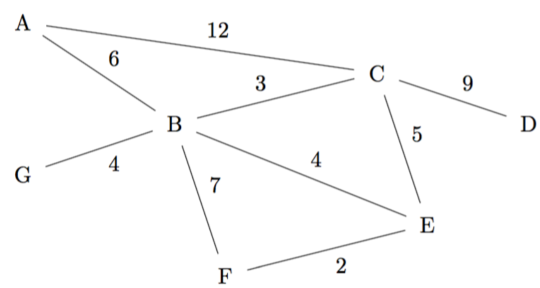
Before exam: [☹________________☺].
After exam: [☹________________☺].

0. So it begins (0.5 points). Write your name and ID on the front page. Write the exam room. Check the IDs of your neighbors. Write the given statement. Sign. Write your login in the corner of every page. Enjoy your free 0.5 points ☺.

1. Mystery Spanning Tree 3000 (12 points).

a) (4 pts) For the graph below, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts from vertex A. Assume ties are broken in alphabetical order (i.e. the edge $\overline{AB}$ would be considered before $\overline{AC}$). Denote each edge with alphabetical overbar notation $\overline{AB}$, which represents the edge from A to B. You may not need all blanks. For your convenience, the graph is printed twice (to make running algorithms easier).





Prim's algorithm order: _____ _____ _____ _____ _____ _____ _____ _____ _____

Kruskal's algorithm order: _____ _____ _____ _____ _____ _____ _____ _____ _____

b) (2 pts) Is there any vertex for which the shortest paths tree (SPT) is the same as your Prim MST above?

○ Yes, and it's _________ ○ No

c) (6 pts) For the following propositions, fill in true or false <u>completely</u> **and provide a brief explanation**. For a proposition that is false, a counter-example suffices. Assume all edge weights are unique.

○ True / ○ False: Adding 1 to the smallest edge across any cut of a graph G must change the total weight of its minimum spanning tree.

○ True / ○ False: The shortest path from vertex A to vertex B in a graph G is the same as the shortest path from A to B using only edges in T, where T is the MST of G.

○ True / ○ False: Given any cut, the maximum-weight crossing edge is in the maximum spanning tree.

2. Sorting (28 points).

a) (2 pts) How many inversions are in the list [A, D, B, W, K] assuming we want the list sorted alphabetically?

b) (3 pts) Suppose we want to sort the array [5, 6, 10, 17, 14, 12, 13] using in-place heapsort. Give the array after heapification and a single remove-max operation.

c) (14 pts) Suppose we have N items we want to sort. For each scenario below, pick the "best" sort to get them into sorted order. Assume for all scenarios that N is very large. **There may be multiple correct answers,** and the correct answer may even be ambiguous. Give the running time **for the absolute worst case** in the right-most column as a function of N. Running time may not be the only consideration for "best". In all cases, assume we're using Java.

Choose from among 1: Insertion sort, 2: Merge sort, 3: Quicksort (with Hoare partitioning), and 4: LSD radix sort. You may not need to use all four answers. Assume that we want stability when potentially useful. **Give your answer to "Best Sort" as a number.**

Scenario (i.e. What You're Sorting)	Best Sort	Running Time
Array of N integers whose max value k is a constant. Do not include k in your runtime.		O( )
Array of N `BigIntegers`[1] whose max value is N^3. Assume comparison takes `log(N)` time.		O( )
Array of N objects that implement `Comparable`, assuming comparison takes constant time.		O( )
Doubly linked list of N objects that implement `Comparable`, assuming constant time comparison and all variables `public`.		O( )
Array of 10 `Strings` of length W. Give runtime in terms of W.		O( )
Array of N objects that implement `Comparable` with $\Theta(\sqrt{N})$ inversions, assuming compare takes constant time.		O( )
Array of N objects that implement `Comparable` with $\Theta(N^2)$ inversions, assuming compare takes constant time.		O( )

d) (9 pts) We call a sort **monotically improving** if the number of inversions never increases as the sort is executed. Which sorts from the list below are monotically improving? Assume that all sorts are as presented during lecture on arrays. Assume insertion sort and selection sort are in-place. Assume heapsort is in-place and that the array acts as a max heap. Assume that Quicksort is non-randomized, uses the leftmost item as pivot, and uses the Hoare partitioning strategy (i.e. using "smaller than" and "bigger than" pointers) from lecture.

☐ Insertion sort ☐ Selection sort ☐ Heapsort ☐ Quicksort ☐ LSD Sort ☐ MSD Sort

[1] A `BigInteger` is an "immutable arbitrary precision integer." It can represent any integer, not just those that fit into 32 bits.

3. Traversals (14 points). Suppose we have an `NAryIntTree`, defined as shown below. Any node may have any number of children. If a node is a leaf, `children` is `null`. Assume that `children[i]` is never `null` for any `i`.

a) (6 pts) Fill in the `printTreePostOrder` method below, which prints the values of the tree in postorder, with one `val` per line. Your solution **must be recursive** and take linear time in the number of nodes.

```
public class NAryIntTree {
    private Node root;
    public class Node {
        public Node[] children;
        public int val;
    }
    public void printTreePostOrder() {
        ______________________________________________
    }

    public ________________________________________________ {
        ________________________________________________
        ________________________________________________
        ________________________________________________
        ________________________________________________
        ________________________________________________
        ________________________________________________
     }
   /* ... */
}
```

b) **(8 pts)** Fill in the code below which prints out the values of the tree in level order with one `val` per line. Your solution **must be iterative** and take linear time in the number of nodes for any tree.

```
    private void printTreeLevelOrder() { // is a method of NAryIntTree
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
        _________________________________________
}
```

4. Algorithms and Data Structures (12 points).

a) (4 pts) In class we primarily considered two graph representations: the adjacency list and the adjacency matrix. Antares suggests that we can improve the performance of Dijkstra's algorithm with a third graph representation he calls an "adjacency heap". For each vertex `v`, `v`'s adjacency heap stores all of `v`'s neighbors in a heap ordered by edge weight, so that the smallest edge adjacent to `v` is at the root of its heap. Naturally, Antares stores these heaps as arrays. Antares reasons that by considering small edges first, Dijkstra's will be able to complete faster.

Will using an adjacency heap result in better, equivalent, or worse **asymptotic** runtime performance for Dijkstra's algorithm than using a regular adjacency list? Assume that we only care about worst case asymptotic performance. Briefly justify your answer.

○ Adjacency heap is better ○ Performance is the same ○ Adjacency heap is worse

Justification: ____

b) (4 pts) Suppose Antares has conjured up the **Gulgate Priority Queue (GPQ)**. Given a GPQ containing N elements, the worst-case running time for insertion, deletion, and change-priority are given as follows: Insertion: $\Theta(N)$, Deletion: $\Theta(N)$, Change-Priority: $\Theta(1)$.

Suppose we run the implementation of Dijkstra's algorithm provided in class (where every vertex is initially inserted into the PQ with infinite priority) using a GPQ on a graph with V vertices and E edges. What is the worst case runtime of Dijkstra's? **Give your answer in big O notation in terms of <u>V and E</u>**. Assume that E >> V (this means E is much greater than V).

Runtime: O(________________________________)

c) (4 pts) Suppose Antares has also created a **Xelha Quick Union (XQU)** to check if two vertices are connected while running Kruskal's. Given that there are N items in an XQU, the running time for XQU operations is as follows: Constructor: $\Theta(N)$, Union: $\Theta(N \log N)$, Is-Connected: $\Theta(\log N)$

Suppose we run the implementation of Kruskal's algorithm as presented in class using a XQU and a heap-based priority queue. Recall that in our version of Kruskal's from class, all edges are initially inserted into a regular heap-based priority queue and removed one by one, and added to the MST so long as there are no cycles. What is the worst case runtime of Kruskal's algorithm? **Give your answer in big O notation in terms of <u>V and E</u>**. Assume that E >> V.

Runtime: O(________________________________)

5. Potpourri (16 points).

a) (6 pts) We learned in lecture and in lab that we can use an array to compactly store a min/max heap, with formulas to calculate the parent, left child, and right child given a node. Now suppose we want to store a ternary heap, where every node has 0, 1, 2, or 3 children. Would the compact array representation work? If your answer is **<u>yes</u>**, give formulas on the **<u>left</u>** to calculate the parent, left child, middle child, and right child. If your answer is **<u>no</u>**, explain why on the **<u>right</u>**.

`Assuming root is at index: __________`
`public int parent(int k) { return ________;}`
`public int left(int k)   { return ________;}`
`public int middle(int k) { return ________;}`
`public int right(int k)  { return ________;}`
	Impossible, because:

b) (2 pts) In class, we said that anytime you override `equals`, you must also override `hashCode`. Suppose the `Yarg` class overrides the `equals` method, but does not override `hashCode`. Suppose that `yargSet` is a `HashSet<Yarg>`. What are the potential direct consequences of not overriding `hashCode`?

○ True / ○ False: `yargSet.contains()` may return an incorrect result.
○ True / ○ False: `yargSet.contains()` runs a much higher risk of taking linear time.

c) (6 pts) If we wanted to build a generic `TrieSet` that could hold many different types, we'd need to require all such types to implement some interface, much like items in a `TreeSet` must implement the `Comparable` interface (shown below). Give a declaration of an appropriate interface and describe any methods with comments. Provide useful names for your methods and interface (not silly ones, sorry). You may not need all blanks.

```
public interface Comparable<Item> {
    // Returns negative int if this < x, positive if this > x, 0 if equal.
    int compareTo(Item x);
}

public interface __________________________________________ {
    ________________________________________________________________________
    ________________________________________________________________________
    ________________________________________________________________________
    ________________________________________________________________________
}
```

d) (2 pts) Suppose the creator of a new `DogPicture` class is deciding whether or not to implement interface X, where X is the interface from part c. What is the primary consideration of the creator? "Will somebody ever want to build a `Trie` of `DogPictures`" is not enough of an answer.

6. **Stocks (12 pts).** Define the price of a stock as the price of its most recent trade. Suppose we want to track the highest priced stocks at the end of the day using the code below. Assume the `MaxPQ` is heap based and uses the same approach as detailed in lecture (where insertion and deletion operations take worst case logarithmic time). Assume all stock names are unique. Assume `STOCK_LIST` is a list of `Stock` objects with prices equal to yesterday's final price. A stock may be traded multiple times in one day.

```
public static List<Stock> getMostExpensiveStocks(int k) {
      MaxPQ<Stock> rankedStocks = new MaxPQ<>();
      HashMap<String, Stock> nameToStock = new HashMap<>();
      addAllStocks(STOCK_LIST, nameToStock);  //assume nice hashcode spread
      addAllStocks(STOCK_LIST, rankedStocks); //uses bottom up heapification
      while (marketIsStillOpenToday()) {      //market closes at 5 PM
          Trade t = getNextTrade();           //waits if no trade available
          Stock s = nameToStock.get(t.name);  //assume key always in map
          s.price = t.price;                 //may be higher or lower
      }
      ArrayList<Stock> returnStocks = new ArrayList<Stock>();
      for (int i = 0; i < k; i += 1) {        //assume k <= s
           returnStocks.add(rankedStocks.delMax());
      }
      return returnStocks;
}
```

Where the `compareTo` method of `Stock` is defined as `public double compareTo(Stock s) { return this.price - s.price; }` where `price` is an integer.

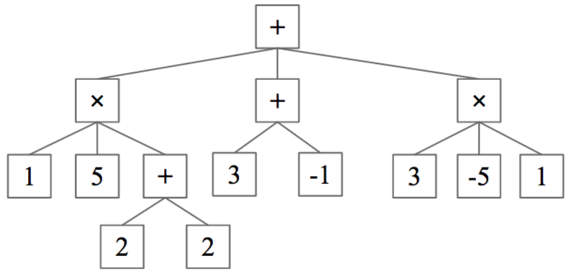
a) Which of the correctness or compilation issues listed below are present in this code? Check all that apply. If you believe there are compilation errors, consider the other boxes assuming the compilation errors are fixed. Assume k is smaller than the number of stocks.

☐ Compilation: The method is supposed to return a `List`, but returns an `ArrayList`.
☐ Compilation: The `HashMap` cannot point to items inside of the `MaxPQ` because the `MaxPQ`'s instance variables are private.
☐ Correctness: The algorithm actually returns the k cheapest stocks.
☐ Correctness: The algorithm may return a list with duplicates if the same stock is traded multiple times.
☐ Other (explain): ____

b) What is the worst case runtime and space complexity of the code above, assuming we fix only any compilation errors you identified in part a? Give your answer in O notation. Your bounds should be as tight as possible with no unnecessary lower order terms or constant factors. Give your answers in terms of S, T, and k, where S is the number of stocks, T is the number of trades, and k is the given argument.

Runtime complexity: O(______________________________)
Space (a.k.a. memory) complexity: O(______________________________)

7. Arithmetic Tree (23 pts). An arithmetic tree is a tree that stores an arithmetic expression. For this problem, assume all nodes are either multiplication (represented with $\times$), addition (represented with +), or a number (represented as a written integer). For example, the following tree represents $(1 \times 5 \times (2 + 2)) + (3 + -1) + (3 \times -5 \times 1)$, which would evaluate to 7.



Your job is to fill out the code below such that `evaluateTree(Node tree)` evaluates the arithmetic tree rooted at `tree` to its correct value. For example, if `evaluateTree` were applied to the $\times$ node at the top left of the figure above, it would return 20. Multiplication and addition operator nodes can have any number of children. You do not need to check for bad inputs (e.g. null children). You may find it easier to work your way from the end of the problem back to the front. You may not need all blanks.

```
public abstract class Node {
    public List<Node> children;
    public abstract void processNode(Stack<Integer> stk);
}

public class ArithmeticTreeEvaluator {
    public static int evaluateTree(Node tree) {
        Stack<Integer> stk = new Stack<>();
        __________________________________________________________________;
        __________________________________________________________________;
    }


    private static void evaluateTreeHelper(Node tree, Stack<Integer> stk) {
        for (__________________________________________________________) {
            evaluateTreeHelper(_____________________);
        }
        __________________________________________________________________;
    }
}
```

If you're stuck on this problem, come back later!

```
public abstract class OperatorNode extends Node {
    public int numArgs() { /* Returns number of args for this node. */ }
    public abstract int apply(int arg1, int arg2);
    public void processNode(Stack<Integer> stk) {
        int res = stk.pop();
        for(____________________________________________________________) {
            res = _______________________________________________________;
        }
        stk.push(res);
    }
}

public class ArgNode extends Node {
    public int value;
    @Override
    public void processNode(Stack<Integer> stk) {
        ______________________________________________________________;
    }
}

/* Don't overthink this! */
public class MultiplicationNode extends OperatorNode {
    @Override
    public int apply(int arg1, int arg2) {
        return ________________________________________________________;
    }
}

public class AdditionNode extends OperatorNode {
    @Override
    public int apply(int arg1, int arg2) {
        return ________________________________________________________;
    }
}
```

8. PNH (0 points). This United States President won office with the smallest fraction of the popular vote in the history of United States presidential elections.

9. Asymptotics (16 points). For each of the code snippets below, give the **best** and *worst case* runtimes in terms of N. Give the **best runtimes** in the column to the **left**, and the *worst* in the column to *the right*.

Best: Worst:

_Θ_____ _Θ_____

```
public static void f1(int N) {
    if (N == 0) { return; }
    f1(N / 2);
    f1(N / 2);
    g(N); // runs in Θ(N²) time
 }
```

_Θ_____ _Θ_____

```
public static int f2(String[] x, int i) {
    int N = x.length;
    int total = 0;
    try {
        while (i < N) {
            total += x[i].length();
            i += 1;
        }
    } catch(NullPointerException e) {
        x[i] = "null";
        total += f2(x, i);
    }
    return total;
}
```

_Θ_____ _Θ_____

Assume `t` is a binary `IntTree` with N nodes:

```
public static void f3(IntTree t) {
    t.value = t.value * 2;
    if (t.left != null)  {  f3(t.left);   }
    if (t.right != null) {  f3(t.right);  }
}
```

_Θ_____ _Θ_____

Assume `t` is a binary `IntTree` with N nodes:

```
public static void f4(IntTree t) {
    t.value = t.value * 2;
    if (t.left != null)  {  f4(t.left);   }
    t.right = t.left;
    if (t.right != null) {  f4(t.right);  }
    t.left = t.right;
}
```

10. Return of the XelhaTree (16 points). Write a function `validXelhaTree` which takes an `IntTree` and a `List` and returns true if the `IntTree` is a `XelhaTree` for the list. You may not need all lines. A `XelhaTree` is valid if it obeys the min heap property, and if an in-order traversal of the `XelhaTree` yields the list of items passed to `createXelhaTree` (in the same order). One line if statements with `{}` on the same line are fine. You may not need all the blanks. Assume there are no duplicates.

```
public class XelhaTreeTest {
    public static class IntTree {
        public int item;
        public IntTree left, right;
    }
    public static IntTree createXelhaTree(List<Integer> x) { ... }
    /** If x is null, returns largest possible integer 2147483647 */
    private static int getItem(IntTree x) {
        if (x == null) { return Integer.MAX_VALUE; }
        return x.item;
    }
    public static boolean isAHeap(IntTree xt) {
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________    }
    public static void getTreeValues(IntTree xt, List<Integer> treeValues){
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________
        ______________________________________________________    }
    public static boolean validXelhaTree(IntTree xt, List<Integer> vals) {
        List<Integer> treeValues = new ArrayList<Integer>();
        /* getTreeValues adds all values in xt to treeValues */
        getTreeValues(xt, treeValues);
        ______________________________________________________
        ______________________________________________________
        return isAHeap(xt) && __________________________________    }
```

11. MaxPQ (12 points). Complete the implementation of MaxPQ using data structures from the reference sheet on the last page of the exam. You may not need all blanks. Write at most one statement per line.

```
public class MaxPQ<Item extends Comparable<Item>> _________________ {
    __________________________________

    __________________________________

    public MaxPQ() {
        ____________________________________________

        ____________________________________________
    }

    public class ____________________________________________ {
        ____________________________________________

        ____________________________________________

        ____________________________________________

        ____________________________________________
    }

    public Item delMax() {
        ____________________________________________

        ____________________________________________
    }

    public void insert(Item x) {
        ____________________________________________

        ____________________________________________
    }
}
```

Welcome to the Chill Out Zone.

12. Danger and Optimization (12 points).

a) (5 pts) Suppose you provide a computing service where users can upload lists of integers and receive back the numbers in sorted order. Which sorts below would be appropriate to choose for this task, assuming you want to prevent users from submitting inputs that either result in terrible[2] runtime or cause an exception? **Assume you are using Java**.

○ Appropriate / ○ Inappropriate : Merge Sort
○ Appropriate / ○ Inappropriate : Insertion Sort
○ Appropriate / ○ Inappropriate : Quicksort using Hoare partitioning and that starts by shuffling
○ Appropriate / ○ Inappropriate : LSD
○ Appropriate / ○ Inappropriate : Recursive MSD

b) (5 pts) Suppose you provide a service where users can upload a list of names (stored as Java Strings) and it will return the list of all unique Strings. Which set implementations below would be appropriate to choose for this task, assuming you want to prevent users from submitting inputs that either result in terrible runtime or cause an exception?

○ Appropriate / ○ Inappropriate : 2-3 Tree based Set
○ Appropriate / ○ Inappropriate : LLRB based Set
○ Appropriate / ○ Inappropriate : Hash based Set
○ Appropriate / ○ Inappropriate : Trie based Set
○ Appropriate / ○ Inappropriate : TST based Set

c) (2 pts) Suppose you provide a service where users can upload their own custom graphs and a start vertex, and you will find the list of all vertices reachable from the start. Which graph search algorithms would be appropriate to choose for this task, assuming you want to prevent users from submitting inputs that either result in terrible runtime or cause an exception?

○ Appropriate / ○ Inappropriate : Recursive DFS
○ Appropriate / ○ Inappropriate : BFS

d) (0 pts) What should Josh name his future kid (assume female if you want a gender specific name)?

[2] By terrible we mean: Imagine you are demoing your website and want to impress someone with its speed. Does it run so slow that you are embarrassed? If so, that is terrible.

13. Reductions (26.5 points). Often in computer science, problems are just other problems in disguise. Complete each problem below according to the directions given. Many of these problems are very challenging.

a) (3 pts) Describe an algorithm to find a **maximum** spanning tree. Your algorithm must use Kruskal's as a "black box," that is, without any modifications. Your answer should be brief.

b) (5 pts) Suppose you want to find the SPT of a graph, but where you redefine the total cost of a path as follows. Let `cost(List<Edge>)` be the sum of the weights of the edges, plus the number of edges. In other words, we want to run Dijkstra's taking into account not just the weights of the edges, but also the number of edges. Describe an algorithm to find this shortest paths tree. Your algorithm must use Dijkstra's as a "black box". Your answer should be brief.

c) (5 pts) Dijkstra's algorithm sometimes fails on graphs with negative edges. Suppose we have a graph G with a single negative edge with weight -Q, and we want to find the shortest path. Suppose we construct a new graph G' where every edge has Q added to its weight. If we run Dijkstra's on G', is the resulting shortest paths tree always a correct shortest paths tree for G? If yes, explain why. If no, provide a counter-example.

○ Yes, because:	○ No, counter-example:

d) (6 pts) Suppose that we're using a programming language Zulg where instead of comparison we have a *zelch* operation. Suppose that we prove that "puppy, cat, dog"[3] requires $\Omega(N \log\log N)$ *zelch* operations. Assume that zelch takes constant time. For each of the following statements, determine whether the answer is false, true, or the answer depends on whether P = NP.

○ True / ○ False / ○ P=NP?: A *zelch* based sort requires **at least** $\Omega(N \log\log N)$ *zelch* operations.
○ True / ○ False / ○ P=NP?: Sorting an array in Zulg requires $\Theta(N \log\log N)$ time in the worst case.
○ True / ○ False / ○ P=NP?: The optimal sorting algorithm in Zulg requires $O(N \log\log N)$ time in the worst case.
○ True / ○ False / ○ P=NP?: All sorting algorithms in Zulg require $O(N \log\log N)$ time in the worst case.

[3] Recall that "puppy, cat, dog" is a game from lecture where we have N boxes, each containing a unique object (e.g. a puppy, a cat, and a dog) of known size, and our job is to determine which box contains which object.

e) (6 pts) Suppose we have the abstract data type `MinimumPQ`, defined as an interface in Java as shown below:

```
public interface MinimumPQ<Item extends<Comparable<Item>> {
   public void add(Item x);
   public Item removeMin();
   public Item min();
}
```

For each statement below, state whether it is true, false, or "depends on whether P = NP". Assume that all implementations are correct.

○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which add requires $\Theta(\log\log N)$ time in the worst case.
○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which removeMin requires $\Theta(\log\log N)$ time in the worst case.
○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which add and removeMin require $\Theta(\log\log N)$ time in the worst case.
○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which add requires $\Theta(1)$ time in the worst case.
○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which removeMin requires $\Theta(1)$ time in the worst case.
○ True / ○ False / ○ P=NP?: There exists a possible MinPQ implementation for which add and removeMin require $\Theta(1)$ time in the worst case.

f) (1.5 pts) Consider the `LongestPath` problem, i.e. given a graph, does there exist a path with total weight k or greater? Suppose that we prove that `LongestPath` cracks 3SAT (i.e. 3SAT reduces to longest path). For each of the following statements, determine whether the answer is false, true, or the answer depends on whether P = NP. Let N be the number of edges.

○ True / ○ False / ○ P=NP?: There exists an algorithm to solve `LongestPath` in $O(N^k)$ time.
○ True / ○ False / ○ P=NP?: There exists an algorithm to check a supposed solution to `LongestPath` in $O(N^k)$ time.
○ True / ○ False / ○ P=NP?: There exists an algorithm to calculate the length in bytes of the shortest Java program that solves `LongestPath`.

… and that's it! 

Nothing written on this page will be graded.

Data Structures Reference:

<table>
<tr>
<td><code>HashSet<Key> {</code>
<code> void add(Key k)</code>
<code> boolean contains(Key k)</code>
<code>}</code>

<code>HashSet</code> is the same except <code>Key</code> must implement <code>Comparable<Key></code></td>
<td colspan="2"><code>MinPQ<Item extends Comparable<Item>> {</code>
<code>   MinPQ(Comparator<Item> c)</code>
<code>   void insert(Item x)</code>
<code>   Item min()</code>
<code>   Item delMin()</code>
<code>}</code>
Uses natural order unless comparator given during construction.</td>
</tr>
<tr>
<td><code>HashMap<Key, Value> {</code>
<code> void put(Key k, Value v)</code>
<code> boolean containsKey(Key k)</code>
<code> Value get(k)</code>
<code>}</code>
<code>TreeMap</code> is the same except <code>Key</code> must implement <code>Comparable<Key></code></td>
<td><code>Stack<Item> {</code>
<code> void push(Item x)</code>
<code> Item pop()</code>
<code>}</code></td>
<td><code>Queue<Item> {</code>
<code> void enqueue(Item x)</code>
<code> Item dequeue()</code>
<code>}</code></td>
</tr>
<tr>
<td colspan="3">Assume all of these classes implement <code>Iterable</code> and have a <code>size()</code> method.</td>
</tr>
</table>