CMPT225, Spring 2021

Final Exam

Name________________________

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. You should write your solutions directly in this word file, and submit it to Coursys. Submitting a pdf is also ok.
2. Submit your solutions to Coursys before April 20, 23:59. No late submissions, no exceptions
3. Write your name and SFU ID on the top of this page.
4. This is an open book exam. You may use textbooks, calculators, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in your solutions.
5. Discussions with other students are not allowed. Posting questions online asking for solutions is not allowed.
6. The exam consists of four (4) problems. Each problem is worth 25 points.
7. Write your answers in the provided space.
8. You may use all classes in standard Java, and everything we have learned.
9. Explain all your answers.
10. **Really, explain all your answers.**

Good luck!

Problem 1 [25 points]

A. (15 points) In this question you need to design a data structure that supports PriorityQueue with deletions. Specifically, you need to write the class PriorityQueueWithDeletions. As a motivation you may think of a priority queue for a printer that allows adding a document, removing a document in some order (e.g. shorter documents are printed first), or a user can cancel the job. Specifically the class needs to support the following operations:
The running time of each operation must be O(log(size of the queue)).

```
public class PriorityQueueWithDeletions<T extends Comparable<T>> {

        public PriorityQueueWithDeletions() - creates an empty priority queue

        public Ticket add(T item) - adds a new element to the queue.
        Returns a ticket that can be used to remove your item from the queue.

        public T removeHighestPriority() - removes the element with the
                     highest priority from the priority queue and returns it.

        public removeByTicket(Ticket t) - removes an element by ticket,
                                          and returns the removed element.

        public int size() - returns the number of elements in the queue

        public boolean isEmpty() - checks if the queue is empty

}
```

The running time of each operation must be O(log(n)), where n is the size of the priority queue.

Explain your answer in detail. Define the class Ticket and explain how you use it.
Make sure Ticket does not allow the user to modify the queue adversarially.

B. (10 pts) Write a data structure that supports all operations of a stack, and in addition supports getMax. Specifically, you need to write the class StackWithMax.
The running time of each operation must be O(1).

```
public class StackWithMax<T extends Comparable<T>> {

        public StackWithMax() - creates an empty stack

        public void push(T item) - adds a new element to the stack.

        public T pop() - removes the element from top and returns it.

        public T getMax() - returns the maximum in the stack (without
        modifying the stack).

        public int size() - returns the number of elements in the stack

        public boolean isEmpty() - checks if the stack is empty

}
```

The running time of each operation must be O(1).

Explain your answer in detail.

Problem 2 [25 points]

In this problem use the following definition of Binary Tree. You may assume the classes have the standard getters/setters.

```java
public class BTNode<T> {
    private T data;
    private BTNode<T> leftChild;
    private BTNode<T> rightChild;
    private BTNode<T> parent;

}

public class BinaryTree<T> {
    private BTNode<T> root;

}
```

A. (7 pts) Write a method that gets a binary search tree and checks if it is an AVL tree. You may assume that the given tree is a binary search tree.
Explain your answer.

```java
public boolean isAVLTree(BinaryTree<T> bst) {

}
```

B. (18 pts) For each of the following statements decide if they are true or false. Explain your answers. A correct T/F answer without a correct explanation will give 1/5 points.

1. (2 pts) Insertion into an AVL tree always increases the number of leaves.

2. (4 pts) An AVL tree of height 3 has at least 7 nodes and at most 15 nodes

3. (6 pts) If inserting a node into an AVL tree requires rebalancing, then the height of the entire tree does not change.

4. (6 pts) When removing a node from an AVL tree with n nodes, then at most 3 vertices need to be rebalanced.

Problem 3 [25 points]

A. (20 pts) Write a method that gets a 2d grid given as an array of ints with 0s and 1s. The goal is to find a path from the top left corner of the matrix to the bottom right using the minimal number of steps when you are only allowed to step on the 1s of the array. Write an algorithm that solves this problem and prints the path. In the end of the algorithm the input needs to be in the same state as it was in the beginning.

For example, for the input below, the path from grid[0][0] to grid[5][4] is marked in red, and consists of 10 ones. Note that you need to find the shortest path. If there are several shortest paths, your algorithm needs to print only one of them.

```
int[][] board = {
       {1,1,1,1,1},
       {1,0,0,0,1},
       {1,0,1,1,1},
       {1,1,1,0,0},
       {1,1,1,1,1},
       {0,0,1,0,1}
};
```

Before writing the algorithm, explain your answer.

```
public static void printPath(int[][] grid) {





}
```

B. (5 pts) What is the running time of your algorithm when the input is an nxn array? Explain your answer.

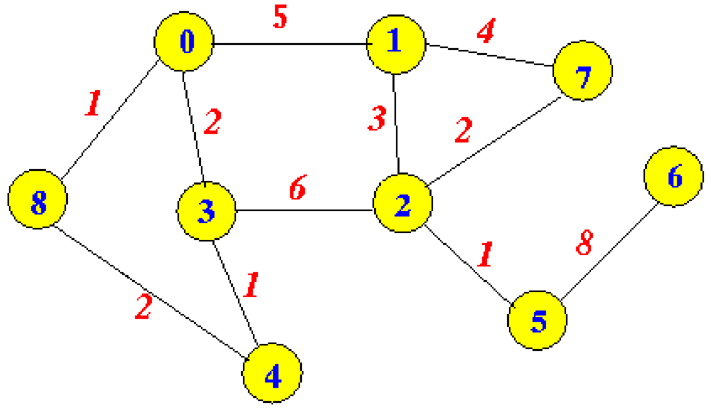
Problem 4 [25 points]

A. (8 pts) Draw the representation of union-find data structure after the following sequence of operations. Draw some intermediate steps.

- For i=1...8

  makeset(i)
- union(1,2)
- union(3,4)
- union(3,5)
- union(1,6)
- union(1,7)
- union(8,1)

B. (7 pts) Show the execution of Kruskal's algorithm on the following graph.



C. (10 pts) Write a function that gets an array A of length n of int, and an integer k, such that $0 \leq k \leq 2n/\log_2(n)$. The function needs to permute the elements of A so that the first k elements of A are the smallest numbers sorted in the non-decreasing order. The **running time** must be **O(n)** and **extra space used** should be **O(1)**.

For example, on input A =[3,1,8,2,6,11,4,12,5,7,10,9] and k=6 the resulting array A needs to have [**1,2,3,4,5,6**] as its first six elements, and the rest can be any permutation of the remaining elements. For example [**1,2,3,4,5,6**,12,7,9,8,10,11] The order of the elements after in the last n-k positions is not important.

Explain your idea before writing the code.