CMPT 225 D100, Spring 2023

Midterm Exam - Solutions
March 10, 2023

Name___________________________

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 100 minutes.
2. Write your full name and SFU ID **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems, each worth 25 points
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. Explain all your answers.
8. Really, explain all your answers.

Good luck!

Problem 1 (Java syntax, Big-O notation) [25 points]

A. (5 points) Explain the difference between the .equal() method and == operator in Java. Provide an example.

Answer: For objects
== compares the addresses/pointers, i.e. checks if we have exactly the same object
.equal() method can be overridden, and is typically used to compare the content.

For example, suppose we have a class

```
public class Book {
  String title;
  String author;
  boolean equals(Object other) {
     check if other is a book with the same title, author
  }
}
```

If we have

```
Book b1, b2;
if (b1==b2)  compares the pointers
if (b1.equals(b2))  uses the overridden method
```

B. (5 points) Explain the concept of Generics in Java. Provide an example.

Answer: It allows us to use similar methods or similar containers for different types without repeating the same code. For example
- LinkedList<Integer> - gives us a list of integers
- LinkedList<Student> - gives us a list of students
The logic of LinkedList is independent of the exact type of objects it contains.

C. (5 points) Is the following statement True or False?
Let $T(n) = T(n/2) + 5n^3$ for all $n>1$, and $T(1) = O(1)$. Then $T=\Omega(n^3)$.
Write a brief explanation.

Answer: $T(n) = T(n/2) + 5n^3 > 5n^3$.
$T(n) > 5n^3$ implies that $T(n) = \Omega(n^3)$

D. (5 points) What is the running time of bar() on an array of length n? **Explain your answer.**

```
public static void bar(int array[]) {
        barRec(array, 0, array.length-1);
}

public static void barRec(int a[], int start, int end) {
        if (start < end) {
                for (int i = start; i <= end; i++)
                        a[i] += 1;
                int mid = (start + end) / 2; // if (start+end) is odd, division rounds down
                barRec(a, mid+1, end);
        }
}
```

Answer: We can write the running time as
$T(n) = T(n/2) + O(n)$
By Master Theorem, this gives $T(n) = O(n)$.

E. (5 points) Prove that the running time of the function foo(n) below is $O(n^2)$.

```
public static int foo(int n) {
        if (n>=5)
                return foo(n/2)+foo(n/3)+foo(n/4)+foo(n/5);
        return 1;
}
```

Answer: We can write the running time as
$T(n) = T(n/2) + T(n/3) + T(n/4) + T(n/5) + O(1)$
$< T(n/2) + T(n/2) + T(n/2) + T(n/2) + O(1)$
$< 4T(n/2) + O(1)$

By Master Theorem, we get $T(n) = O(n^2)$.

Problem 2 (Stack, Queues, Linked Lists) [25 points]

A. Write a generic class StackWithMin that supports the following operations, with running time O(1) for each operation.

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

        public StackWithMin() - a constructor, creates an empty stack

        public void push(T item) - adds an element to the stack

        public T pop() - removes an element from the stack

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

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

        public T min() - returns the minimal element currently in the stack.
        The method does not modify the stack.
        Two elements are compared using the compareTo() method.
        Given a and b, we say a<b if a.compareTo(b)<0.
}
```

Implementing correctly all methods except for min() is worth up to 15 points.

Before writing code, explain your answer.

Answer: One way to do it is to store two stacks (or linked lists) internally: one stack will hold the values, and the other stack will hold the minimum in the stack.
When adding a new element, we do the following

- add it to the first stack
- compare the new item to the previous minimum (which is at the top of the second stack), and add min(new item, previous minimum) to the second stack

```
public class StackWithMin<T extends Comparable<T>> {
    private LinkedList<T> stack;
    private LinkedList<T> min;

    public StackWithMin() {
        stack = new LinkedList<T>();
        min = new LinkedList<T>();
    }

    public void push(T item) {
        stack.addLast(item);
        T currentMin = min.getLast();
        if (item.compareTo(currentMin) < 0)
            min.addLast(item);
        else
            min.addLast(currentMin);
    }

    public T pop() {
        min.removeLast();
        return stack.removeLast();
    }

    public int size() {
        return stack.size();
    }

    public boolean isEmpty() {
        return size()==0;
    }


    public T min() {
        return min.getLast();
    }

}
```

Problem 3 (Binary Trees) [25 points]

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

```
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;

}
```

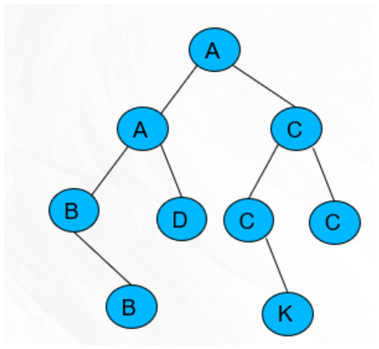
A. (10 pts) Write the method countNodesWithOneChild() for the class Binary Tree. The method returns the number of vertices in the tree with exactly one child.

Explain the idea of the algorithm and its running time.

```
public int countNodesWithOneChild() {
  return countA(getRoot()); // call a recursive helper function
}
```

```
private int countA(BTNode<T> root) {
     if (root == null || root.isLeaf())
          return 0;

     int recResults = countA(root.getLeftChild())
               + countA(root.getRightChild());

     // two children
     if (root.getLeftChild()!= null
               && root.getRightChild()!=null)
          return recResults;
     // one child
else
          return recResults+1;
}
```

B. (10 pts) Write a method for the class Binary Tree that returns the number of vertices v in the tree such that v.data is equal to v.parent.data. Compare the data using equals() method. For example below, the function needs to return 4 (A,B and C x2)



Explain the idea of the algorithm and its running time.

```
public int countChildrenEqualToParent() {
   return countB(getRoot()); // call a recursive helper function
}
```

```
    private int countB(BTNode<T> node) {
          if (node == null)
                return 0;

          int recResults = countB(node.getLeftChild())
                     + countB(node.getRightChild());
          BTNode<T> parent = node.getParent;
          if (parent != null &&
                node.getData()==parent.getData())
                return recResults+1;
          else
                return recResults;
    }
```

C. (5 pts) Draw a Binary Tree whose inOrder sequence is [1,2,3,4,5,6,7,8] and postOrder is [2,1,4,6,5,3,8,7].

Answer: Looking at the postOrder list the root is 7.
Looking at both lists, the postOrder of
the left subtree is [2,1,4,6,5,3],
and the right subtree is only [8].
In the left subtree the root is 3,
and the subtrees of 3 are: left- [1,2] and right-[4,5,6]
And so on, we continue using this logic.

Problem 4 (Binary Search Trees) [25 points]

A. (10 points) Consider the class BinarySearchTree, defined as

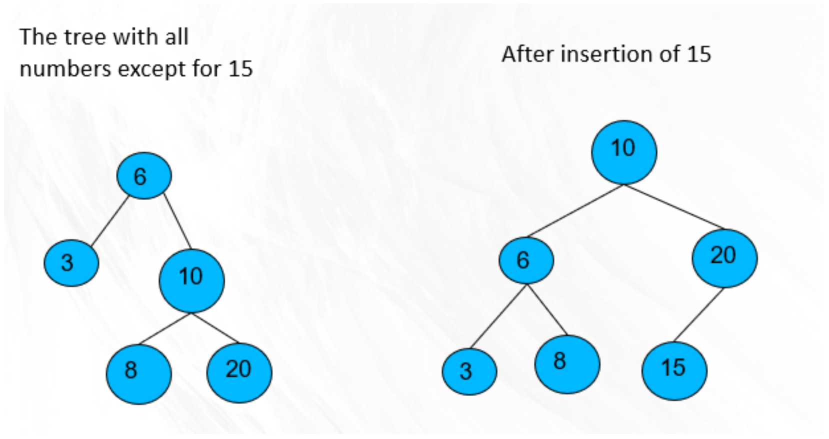
```
public class BinarySearchTree<T extends Comparable<T>> {

        private BTNode<T> root;

}
```

Use the definition of BTNode from Problem 3.

Write the method insert() for the class BinarySearchTree. It gets an item, adds it to the Binary Search Tree as a leaf, and returns the pointer to the new node.

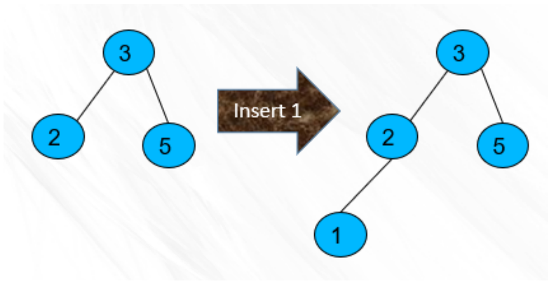
```
public BTNode<T> insert(T elt) {

        BTNode<T> parent = null;
        BTNode<T> cur = getRoot();
        while (cur != null) {
            parent = cur;
            if (cur.getData().compareTo(item) > 0)
                cur = cur.getLeftChild();
            else
                cur = cur.getRightChild();
        }
        BTNode<T> newNode = new BTNode<T>(item);
        if (parent == null) // first node inserted
            root = newNode;
        else {
            if (parent.getData().compareTo(item) >
0)
                parent.setLeftChild(newNode);
            else
                parent.setRightChild(newNode);
        }
        return newNode;

}
```

B. (5 points) Draw the AVL tree obtained by inserting the sequence [8, 3, 6, 10, 20, 15].

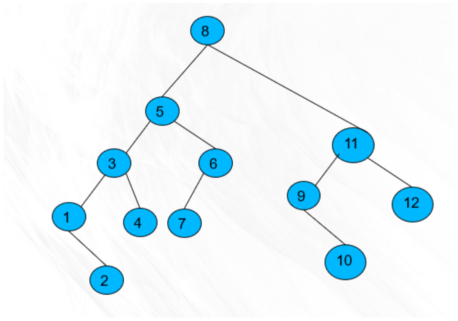


C. (5 points each question) For each of the following statements, decide if it is True or False. **Prove your answer**.

1. Insertion into an AVL tree always increases the number of leaves.
   Answer: False. For example, below we have 2 leaves before and after.



2. There exists an AVL tree of height 4 with exactly 5 leaves.
   Answer: True. For example



Extra page

Master Theorem

Let $T(n) = a\,T(n/b) + f(n)$, $T(1) = O(1)$

Define $c = \log_b(a)$

- If $f(n) = O(n^d)$ for $d<c$, then $T(n) = \Theta(n^c)$
- If $f(n) = \Omega(n^d)$ for $d>c$, then $T(n) = \Theta(n^d)$
- If $f(n) = \Theta(n^c)$, then $T(n) = \Theta(n^c \log(n))$