



**1. Matching [ 5 pts ]**

Choose the **best** definition for each of the words below.

1. \_\_\_\_\_ A LIFO data structure that can be implemented with a Linked List.
2. \_\_\_\_\_ A data structure where every parent node is smaller than or equal to the values of either of its child nodes.
3. \_\_\_\_\_ An exception that the compiler does not require the programmer to keep track of it.
4. \_\_\_\_\_ An abstraction for a sequence of bytes which data can be read or to which data can be written.
5. \_\_\_\_\_ Provides layers of functionality to I/O classes in Java.
  - A. Stack
  - B. Full Binary Tree
  - C. Adornation Class
  - D. Min-Heap
  - E. Queue
  - F. Unchecked Exception
  - G. Checked Exception
  - H. Error Exception
  - I. Max-Heap
  - J. Priority Node
  - K. Stream
  - L. Decorator Class
  - M. Complete Binary Tree

## 2. Tracing [ 15 pts ]

What is the output of following code when the main method is run?

```
public class TracerC{

    public static void main(String[] args){
        int x=2;
        int y=3;
        String temp="?";
        try{
            System.out.println("Before the while");
while(x>=0){
                System.out.println(y/x);
                temp+="?";
                x--;
            }

            System.out.println("After the while");
        }
        catch(ArithmeticException ae){
            System.out.println("I can't do that.");
        }//end catch(ArithmeticException)
        catch(Exception e){
            System.out.println(temp+"!");
            temp="?!?";
        }//end catch(Exception)
        finally{
            System.out.println("Am I leaving already"+temp);
        }//end finally
    }//end main(String[])
}//end TracerC class
```

---

### 3. Exceptions [ 15 pts ]

- 5 (a) Write a custom checked Exception called **TooSmallException**. You do not have to write any methods for this class.

- 10 (b) Write a method called **public void evaluateSize(int a)** inside the **Calculator** class that takes in an int and throws a **TooSmallException** if the number passed in is less than 10, or otherwise prints the number. Remember to write the method header.

```
public class Calculator{
```

```
}
```

**4. File I/O [ 15 pts ]**

Given the following class `MyIO` write the method **`public void readAndPrint`**, which reads all the lines from `System.in` until the end of file (this works the same as reading from any file) and prints them out to `System.out`:

```
public class MyIO{  
  
public void readAndPrint(){
```

```
    }  
} //end class MyIO
```

Hint: You might want to use the class `BufferedReader` and the method `readLine()` in the `BufferedReader` class.

## 5. Queues [ 10 pts ]

Assume you have the following **LinkedList** class. You may only assume that you have the following methods in the **LinkedList** class:

```
public class LinkedList{
    private LLNode head; // Beginning reference of LL
    public void addToFront(Object o){/*Code omitted for brevity */ }
    public void addToBack(Object o){/*Code omitted for brevity */ }
    public Object removeFromFront(){/*Code omitted for brevity */ }
    public Object removeFromBack(){/*Code omitted for brevity */ }
}
```

Write the following methods for the **Queue** class. The methods should maintain the integrity of the **Queue** data structure.

5 (a) `public void enqueue(Object o){`

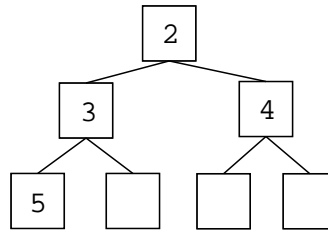
`}`

5 (b) `public Object dequeue(){`

`}`  
`}//end class Queue`

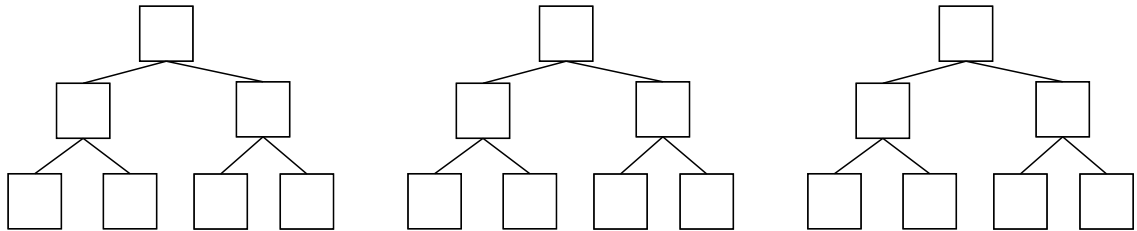
6. **Heaps [ 15 pts ]**

Given the following Heap data structure:

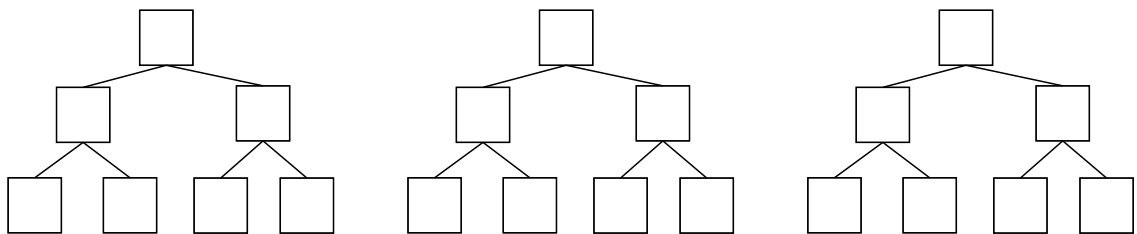


Show the tree data structure at each step of the process when:

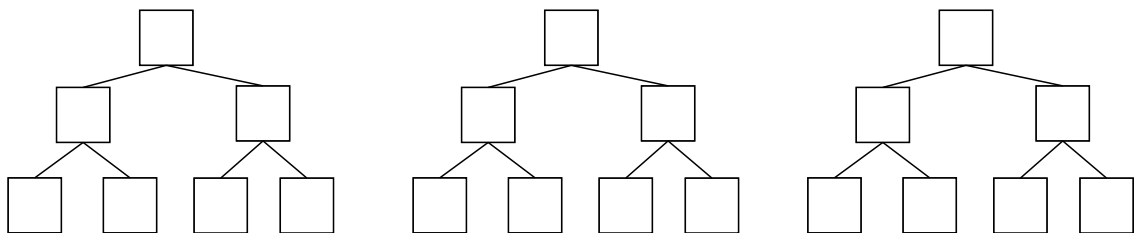
- 5 (a) The number 1 is added to the **original Heap**.



- 5 (b) The number 7 is added to the **original Heap**.



- 5 (c) The number 2 is removed from the **original Heap**.



## 7. **LinkedList** [ 10 pts ]

Assume you have a fully functioning Linked List Node class called `LLNode`. With the following methods and constructor:

- `public void setData(Object o)`
- `public Object getData()`
- `public void setNext(LLNode next)`
- `public LLNode getNext()`
- `public LLNode(Object o) //set the data of the LLNode to o`

Write a method called **`public void removeAllButOne(Object o)`** which leaves only one occurrence of `Object o` in the `LinkedList`, and removes the rest. You may use any helper methods you wish to write.

```
public class LinkedList{
    private LLNode head;

    public void removeAllButOne(Object o){
```

```
    }
} //end LinkedList class
```

## 8. Binary Search Trees [ 12 pts ]

Given a fully functioning Binary Search Tree node class called **BSTNode** with the following constructor and methods:

- `public BSTNode(Comparable c) //sets the data of the node to c`
- `public void setData(Comparable c)`
- `public Comparable getData()`
- `public void setRight(BSTNode rightChild)`
- `public BSTNode getRight()`
- `public void setLeft(BSTNode leftChild)`
- `public BSTNode getLeft()`

Write the method **Comparable findSmallestButNotSmallerThan(Comparable c)** which takes in a Comparable, and finds and returns a Comparable in the BST which is the smallest value inside the tree that is greater than the data passed in. If there is no data that satisfies the criteria, return null.. There should be no duplicated data inside the tree. You can only assume that you have the listed methods in the BSTNode to use. You may write any helper methods that you see fit.

```
public class BST {
    private BSTNode root;

    public Comparable findSmallestButNotSmallerThan(Comparable c) {
```

```
/* More space on the next page */
```

/\* Extra space for previous questions \*/

```
    } //end Comparable findSmallestButNotSmallerThan(Comparable c)  
} //end BST class
```