

1. Matching [5 pts]

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

1. _____ A FIFO data structure that can be implemented with a Linked List.
2. _____ A data structure where every parent node is greater than or equal to the values of either of its child nodes.
3. _____ An exception that describes a problem that is likely to occur regardless of the programmers carefulness.
4. _____ An abstraction for a sequence of bytes which data can be read or to which data can be written.
5. _____ A heap has the property of this type of binary tree.
 - 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 **TooBigException**. 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 **TooBigException** if the number passed in is greater 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.

```
public class Queue {
    private LinkedList ll;
```

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

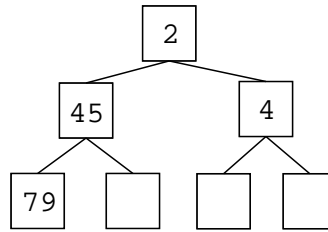
```
    }
```

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

```
    }
} //end class Queue
```

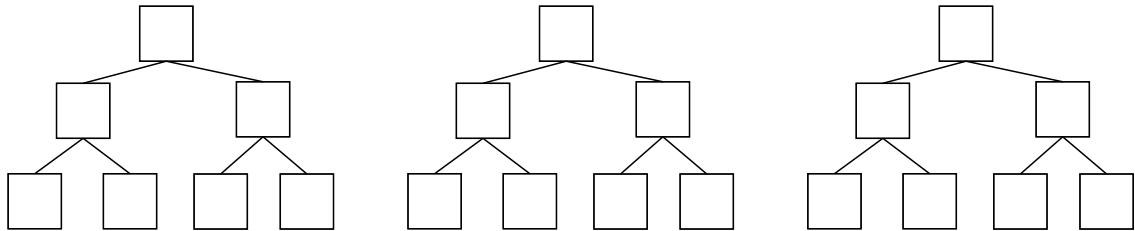
6. **Heaps [15 pts]**

Given the following Heap data structure:

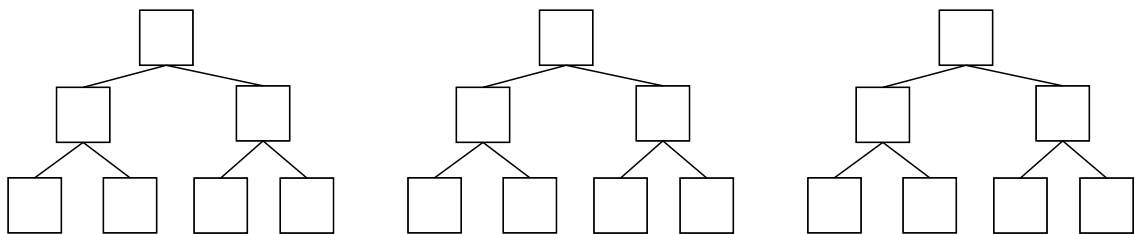


Show the tree data structure at each step of the process when:

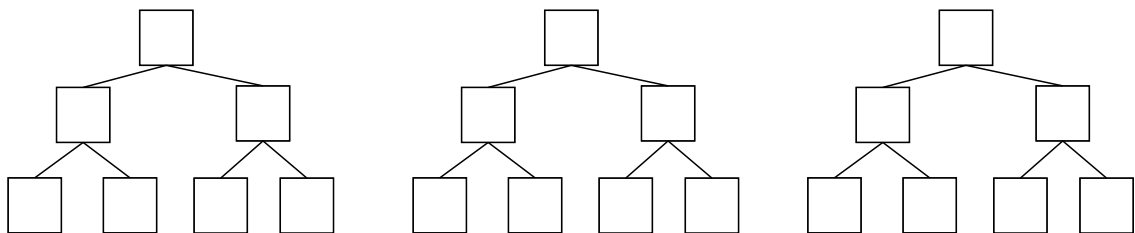
- 5 (a) The number 1 is added to the **original Heap**.



- 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 boolean removeAll(Object o)`** which tries to remove all occurrences of `Object o` inside the `LinkedList`. You may use any helper methods you wish to write.

```
public class LinkedList{
    private LLNode head;

    public boolean removeAll(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 **void add(Comparable c)** which takes in a Comparable, and adds the Comparable c to the BST). There should be no duplicated data inside the tree. If a duplicate data element is being added to the tree, throw an `IllegalArgumentException`. 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 void add(Comparable c) {
```

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

```
/* Extra space for previous questions */
```

```
    } //end void add(Comparable c)  
} //end BST class
```