

1. Matching [4 pts]

Select the **best** definition for each of the following words:

1. _____ Divide and Conquer
2. _____ Component
3. _____ JButton
4. _____ UI Delegate
 - A. An interface that requires the method **public int compareTo(Object o)**
 - B. A paradigm of GUI design in which the Model and Controller are combined, but are separate from the View
 - C. A GUI construct that can be added to a Container
 - D. In Swing, a top level window
 - E. A GUI construct that specifies how other components are arranged within a container
 - F. A data structure which while conceptually can be thought of as a complete tree, is implemented as an array. Gives fast access to the smallest item.
 - G. A Swing component that simply displays text
 - H. A paradigm of GUI design in which the Model, View, and Controller are separated
 - I. A method which is used in a heap during deletion to restore the heap ordering
 - J. Graphical User Interface
 - K. A Swing component that the user can click on to accomplish some action
 - L. A category of algorithms in which the data is split into smaller pieces (for example halves), each piece is recursively processed, and the results are then put back together in some way
 - M. A style of laying out java files where the Methods, Variables, and Constructors are grouped together
 - N. A paradigm of GUI design in which the View and Controller are combined, but are separate from the Model
 - O. A GUI construct that holds other pieces of the GUI

2. GUIs [9 pts]

What is displayed on the screen when the following GUI program is run?

```
import java.awt.*;
import javax.swing.*;
public class GUI2 {
    public static void main(String[] args) {
        JFrame jf=new JFrame("My GUI");
        jf.setSize(200,200);

        JPanel jp=new JPanel();
        jp.setLayout(new BorderLayout());
        jf.getContentPane().add(jp);

        JButton jb1=new JButton("W");
        JButton jb2=new JButton("X");
        JButton jb3=new JButton("Y");
        JButton jb4=new JButton("Z");

        jp.add(jb1,BorderLayout.WEST);
        jp.add(jb2,BorderLayout.NORTH);
        jp.add(jb3,BorderLayout.SOUTH);
        jp.add(jb4,BorderLayout.EAST);

        JPanel jp2=new JPanel();
        jp2.setLayout(new GridLayout(2,2));

        jp.add(jp2,BorderLayout.CENTER);

        for(int i=0;i<4;i++) {
            jp2.add(new JLabel("thing "+i));
        }

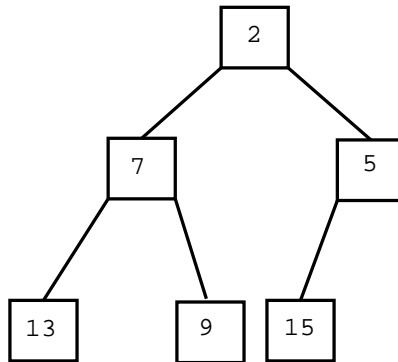
        jf.setVisible(true);

    } //main
} //class GUI2
```

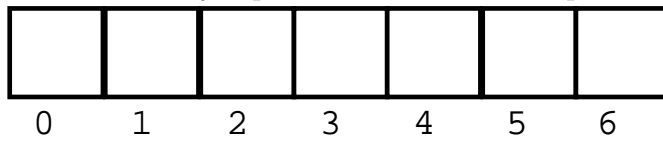
(Hint: You will be drawing a picture to answer this question)

3. Heaps [15 pts]

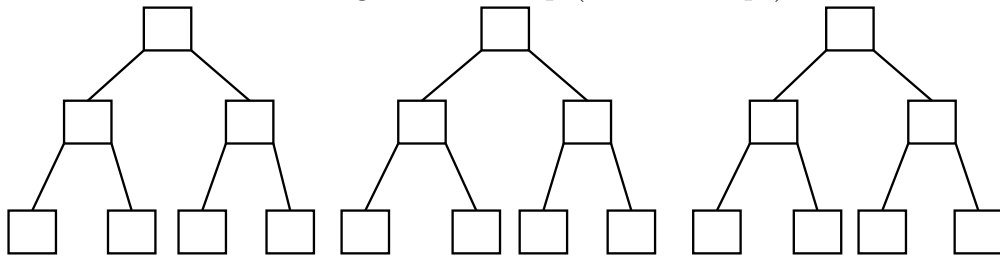
Given the following heap:



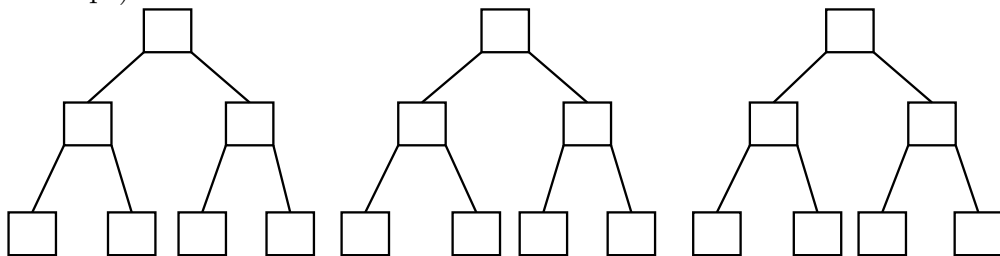
5 (a) Draw the array representation of the heap:



5 (b) Show the results of adding 1 to the heap (show all steps):



5 (c) Show the results of removing the minimum element **from the original heap** (show all steps):



4. **Sorting [13 pts]**

Given the following code:

```
public class Sort2 {
    public static void printArray(int[] data) {
        System.out.print("[");
        for(int i=0;i<data.length;i++)
            System.out.print(" "+data[i]);
        System.out.println("]");
    }
    public static void mysterySort(int[] data) {
        boolean done;
        do{
            done=true;
            for(int i=0;i<data.length-1;i++){
                if(data[i]>data[i+1])
                {
                    done=false;
                    int temp=data[i];
                    data[i]=data[i+1];
                    data[i+1]=temp;
                }
            }
        } while(!done);
    }
    public static void main(String[] args) {
        int[] data={4,2,6,8,0};
        mysterySort(data);
    }
} //main
} //Sort2
```

3

(a) What is the name of the sorting algorithm implemented above?

10

(b) What is the output when the **main** method in Sort2 is run? (Note: You may wish to use your knowledge of his sort to help you trace the code.)

5. BSTs [15 pts]

Assuming that you are given a standard BSTNode class, as follows:

```
public class BSTNode{
    private Comparable data;
    private BSTNode left;
    private BSTNode right;
    public BSTNode(Comparable d) {
        data=d;
        left=null; right=null;
    }
    public void setData(Comparable d) { data = d; }
    public Comparable getData() { return data; }
    public void setLeft(BSTNode l) { left = l; }
    public BSTNode getLeft() { return left; }
    public void setRight(BSTNode r) { right = r; }
    public BSTNode getRight() { return right; }
} //class BSTNode
```

Write the method **public void inorderTraverse()** in the class **BST** below. This method should do an inorder traversal of the tree, and print the data from each node appropriately. You may make any helper methods that you wish.

```
public class BST {
    private BSTNode root;
    //Other methods and constructors are omitted here,
    //but do not assume anything about what methods exist
    public void inorderTraverse() {
        //YOUR CODE GOES HERE
```

```
    }
}
```

6. Priority Queues [16 pts]

You have been assigned the task of making a priority queue that will hold generic Objects. When each object is added to the Queue, a priority will be passed to the **enqueue** method along with the Object. Since you need to associate the priority with the Object, we have given you a Holder class which will hold the Object and its priority (see below). When dequeuing from your priority queue, the item with the lowest numbered priority should be returned first. Assume that you are given a properly implemented min-heap with the following methods:

- public void add(Comparable toAdd)
- public Comparable removeMin()

as well as the following class which you may find useful:

```
public class Holder implements Comparable {
    Comparable comparator;
    Object data;
    public Holder(Object d, Comparable c) {
        comparator=c;
        data=d;
    } //Holder(Object, Comparable)
    public int compareTo(Object o) {
        return comparator.compareTo(((Holder)o).getComparator());
    }
    public Comparable getComparator() { return comparator; }
    public Object getData() { return data; }
} //class Holder
```

Complete the class **PriorityQueue** by implementing the **enqueue** and the **dequeue** methods below. You must use the Heap **myHeap** that is an instance variable of the **PriorityQueue** class to accomplish this task.

```
public class PriorityQueue {
    Heap myHeap=new Heap();
    public void enqueue(Object toAdd,int priority) {
        //YOUR CODE GOES HERE

    }

    public Object dequeue() {
        //YOUR CODE GOES HERE

    }
}
```

7. String Recursion [13 pts]

Write the method `public int countOccurrencesOf(String s, char c)`. This method should return a count of the number of times that the character `c` appears in `s`. **You MUST use recursion only** for this method. If you do not use recursion, or if you use any iteration, you will receive no credit for this method. You may write any helper methods that you wish to. You may make any variables inside the method or any helper methods, but you may not make any instance or static variables. You may assume that `s` is not *null*. Below is an exclusive list of the methods from the String class that you may use. This means that you may use these methods from String, but no other methods from the String class.

1. `public char charAt(int index)`
2. `public String substring(int beginIndex)`
3. `public boolean equals(Object o)`

Here are some examples of what should be returned:

```
countOccurrencesOf("", 'b') = -1
```

```
countOccurrencesOf("abceabaa", 'a') = 4
```

```
countOccurrencesOf("abcdefg", 'q') = 0
```

```
public int countOccurrencesOf(String s, char c) {
```

```
}
```

8. Heaps [15 pts]

Assuming that your Heap class already has a properly working **heapify** method, write the method **public Comparable removeMin()** which removes and returns the smallest element in the Heap. (The Heap is implemented as a Min-Heap). You may write any helper methods that you desire.

```
import java.util.*;
public class Heap {
    private Vector myData;
    //Other methods and constructors are omitted here,
    //but do not assume anything about what methods exist
    private void heapify(int index) {
        //Implementation omitted
        //assume a standard, working heapify method
    }
    public Comparable removeMin() {
        //YOUR CODE GOES HERE
    }
}
```