Name……………………………id ………….………section……….CR58………

1. (10 marks) Given a hash table for integer data of size 11. Let *hash(x) = x %TableSize* and *$hash_2(x)$ = 7-(x%7)*. The hash table already has 2 and 5 inside. Show and explain (step by step) what happens when 13, 24, 16, 27, 9 are inserted into the table in order. The table at each step is drawn for you.

		2			5					

Insert 13

		2			5					

Insert 24

		2			5					

Insert 16

		2			5					

Insert 27

		2			5					

Insert 9

		2			5					

Name…………………………id ………..………section……….CR58………

2. (10 marks) A heap is used to build a Huffman tree for English alphabets. Each data element stored in the heap is stored as an object of class HuffmanNode.

```
public class HuffmanNode{
        public String element;
        // an English alphabet. At start, each HuffmanNode in the heap represents an //
        alphabet and its frequency. But as the Huffman tree is built, a HuffmanNode //
        does not need an alphabet in its element if it is not a leaf node.

        public int frequency; //this can be added up during the Huffman tree
                        //construction.

        public HuffmanNode left;
        public HuffmanNode right;

}
```

Let the class Heap have the following methods:

- *public boolean isEmpty()*: returns true if the heap is empty, false otherwise.
- *public Object removeMin()*: removes a HuffmanNode object with the lowest frequency from the heap. It returns the removed object.
- *public void add(Object x)*: adds a HuffmanNode object to the heap. The object is automatically put into its correct position within the heap according to its frequency value.

Write the following method of class Heap:

public HuffmanNode buildHuffmanTree(Heap h) that builds a Huffman tree from any given heap. The method returns the root of the finished Huffman tree as its result.

More space next page

Name…………………………id …………..………section……….CR58………

Name…………………………id ………..………section……….CR58………

3. (10 marks in total. **Read the whole question first.**) In this question, we are trying to add queue-like operations to heap, while still keeping all heap operations and structure intact. Let class Heap be as follows:

```
public class Heap{
        Object[] thearray;
        public Heap(){
                // a working constructor that initializes everything correctly.
                // Do not write code for this.
        }

        public boolean isEmpty(){
                // returns true if the heap is empty, false otherwise.
                // Do not write the code for this method.
        }

        public Object removeMin(){
                // removes an object with the lowest value from the heap.
                // It returns the removed object. Do not write code for this.
        }

        public void add(Object x){
                //adds new object to heap. The object is put in correctly according to its
                // value. Do not write code for this method.
        }
}
```

a) (2 marks) Explain how each piece of data stored in a heap should change so that the ordering of a heap and a queue can both be preserved.

Name………………………id ………..………section……….CR58………

b) (1 mark) Write a class (please name it HeapInfo) and constructor for your proposed piece of data from a).

For question c) and the rest, assume the following:

- class HeapInfo from question b) has method *public int compareTo(Object x)* that it can use to compare itself with another HeapInfo object, x. The method compareTo returns 0 if the two objects are equal in value, returns -1 if *this* has smaller value than x, and returns 1 if *this* has larger value than x.
- To remove anything from the heap array, you must use only the provided *removeMin()* as a basis. You cannot modify the array directly.
- To add anything to the heap array, you must use only the provided *add()* as a basis. You cannot modify the array directly.
- You can create another Heap (this new heap must only call the provided Heap methods).
- You **must not** create another array, list, stack, queue, tree.
- You will be writing methods of class QueueHeap, which extends from Heap.
    - o You can add new variables and methods to class QueueHeap, as long as it does not violate earlier assumptions.

c) (2 marks)Write method:
public void enqueue(Object x) : This method puts a new object into QueueHeap.

Next question is on the next page.

Name…………………………id ………..………section……….CR58………

d) (1 mark) Give an example of how a dequeue could be done.

e) (4 marks) Write method:
public Object dequeue(): This method removes an object from QueueHeap. The removed object is the oldest object put into the QueueHeap with the *enqueue()* method. The method returns the removed object as its result.

More space next page.

Name…………………………id ………….………section……….CR58………

Name…………………………id …………..………section……….CR58………

4. (5 marks) If we want to insert 16, 15, 8, 9, 4, 5, 3, 11, 10 into an empty heap. What will the finished heap look like?

5. (5 marks) Draw a Huffman tree from the following data:

a has frequency = 700
b has frequency =400
c has frequency =100
d has frequency =300
e has frequency =500

6. (6 marks) write the following method for class heap:

```
Heap mergeHeap(Heap thesecondHeap){
```

This method combines this heap with a given heap. What is the big O of this method?

Name…………………………id ………..………section……….CR58………

This method (in class Heap) tries to test if the array in class Heap (given in lectures) preserves heap properties. That is, element in a parent is always less than element in its child.

8. (10 marks) Assume we have a heap that stores Student objects. A student class is:

```
public class Student {
        String name; //name
        int mark; // score

        public Student(String n, int m){
                name = n; mark = m;
        }
        …..
        // assume this class has all get and set methods.
        …..
}
```

Assume that our heap already has the following methods:

- compare(Student first, Student second) returns negative number when first has less marks than second, positive number when first has more marks than second. (assume no one has the same score as any other person.)
- add(Object item) as in lectures
- percolateUp() as in lectures
- percolateDown(int startindex) as in lectures

write method void changemark(String name, int newmark) that we can call to change the mark of any one student. Our student heap must still preserve its heap properties.

Name…………………………id ………….………section……….CR58………