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

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, First Semester, Final Examination Date 16 May 2019 Time 8.30-11.30 (3 hours)

Important

1. This exam paper has 10 questions. There are 6 pages in total including this page. The total mark is 48 (will be rounded to 40).
2. **All questions asked you to write Java code. If you write in some pseudo code and it is understandable, your score will be deducted appropriately.**
3. **Your answer must only be written in a given answer book.**
4. Write your name and ID on the first page of this exam paper.
5. When the exam finishes, students must stop writing and remain in their seats until all question sheets and answer books are collected and the examiners allow students to leave the exam room.
6. A student must sit at his/her desk for at least 45 minutes.
7. A student who wants to leave the exam room early (must follow (5)) must raise his/her hand and wait for the examiner to collect his/her papers. The student must do this in a quiet manner.
8. No books, lecture notes or written notes of any kinds are allowed in the exam room.
9. No calculators are allowed.
10. A student must not borrow any item from another student in the exam room. If you want to borrow an item, ask the examiner to do it for you.
11. Do not take any part of the question sheet and answer books out of the exam room. All papers are properties of the government of Thailand. Violators of this rule will be prosecuted in a criminal court.
12. A student who violates rules will punished by the following rule:
    - **Suspected cheaters will get an F in the subject they are suspected to cheat, and will not be able to enroll in the next semester.**
    - **Cheaters will get an F in the subject they are caught cheating, and will not be able to enroll in the next 2 semesters.**

I understand and agree to the given instructions.

Signature(…………………………)

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

1. (4 marks) Given a quadratic probing hash table (the table has size 11) for integer. Let *hash(x) = x%TableSize*.

The array for this hash table looks like

		2	3			6				

We sequentially do the following actions:

- Add 13
- Add 24
- Delete 3
- Add 14

Using DELETED object to mark deleted slot and replacing DELETED object whenever possible, **draw and explain** (step by step) what happens to the array.

2. (5 marks) The code for double hashing hash table is given as follows:

```
1: public class OpenAddressing{
2:      static int DEFAULT_SIZE = 101;
3:      static final Object DELETED = new Object();
4:      static int MAXFACTOR = 0.5;
5:      int currentSize =0;
6:      Object[] a;
7:
8:      public OpenAddressing(){
9:         this(DEFAULT_SIZE);
10:      }
11:
12:      public OpenAddressing(int size){
13:          int nextPrimeSize = Utility.nextPrime(size);
14:          array = new Object[nextPrimeSize];
15:     }
16:   } // end of class OpenAddressing.
```

```
1: class DoubleHashing extends OpenAddressing
2:     static int MAXFACTOR = 0.75;
3:     int occupiedSlots = 0;
4:
5:     public DoubleHashing(){
6:        this(DEFAULT_SIZE);
7:     }
8:
9:     public DoubleHashing(int size){
10:           super(size);
11:     }
12:
13:     public int hash(Object data){
14:       int hashValue = data.hashCode();
15:       int abs = Math.abs(hashValue);
16:       return abs%array.length;
17:     }
18:
19:     public int hash2(Object data){ //DO NOT IMPLEMENT THIS METHOD
20:       //Return a value calculated from data.
```

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

```
        //It uses a different calculation from hash(Object data).
     }
} //end of class DoubleHashing
```

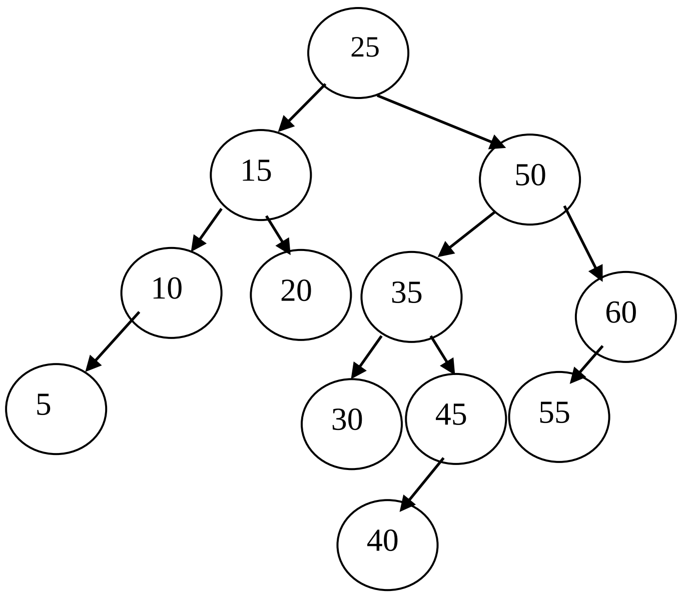
Write code for method ***public void add(Object data)*** of class DoubleHashing.

This method does nothing if ***data*** is in our hash table. But if the data is not in the hash table, it adds the data to the table. The added data replaces DELETED object if possible.

*To compare ***this*** object with object ***o***, use method ***boolean equals(Object o)*** which returns true if ***this*** is equal to ***o***, and returns false otherwise.

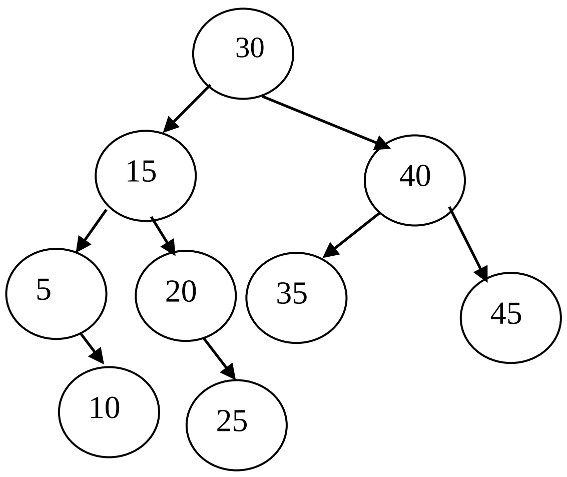
*If any other methods or variables not given in this question are needed, you must write their code by yourself.

3. (4 marks) An AVL Tree looks like:



Draw pictures (if there is a rotation, show the tree before and after each rotation), step by step, what happens when we delete 50. (**a removed node is replaced by the minimum node in its right subtree**)

4. (2 marks) An AVL Tree looks like:



Draw pictures (if there is a rotation, show the tree before and after each rotation), step by step, what happens when we add 27 to the tree.

5. (5 marks) An AVL tree has the following code:

```
public class AVLNode {
 int data;
  AVLNode left, right, parent;
  int height;

  public AVLNode(int data) {
    this.data = data;
    left = null;
    right = null;
       parent = null;
       height = 0;
   }

   public AVLNode(int data, AVLNode left, AVLNode
   right, AVLNode parent, int height) {
       this.data = data;
       this.left = left;
       this.right = right;
       this.parent = parent;
       this.height = height;
       }

   public static int getHeight(AVLNode n) {
       return (n == null ? -1 : n.height);
   }

   public static void updateHeight(AVLNode n) {
       if (n == null)
        return;
       int leftHeight = getHeight(n.left);
       int rightHeight = getHeight(n.right);
      n.height = 1 + (leftHeight < rightHeight ?
       rightHeight : leftHeight);
   }

```

```
    public static int tiltDegree(AVLNode n) {
      if (n == null)
        return 0;
      return getHeight(n.left)- getHeight(n.right);
    }
  } //end class AVLNode
```

```
public class AVLTree {
    AVLNode root;
    int size;

    public AVLNode rebalance(AVLNode n) {
     if (n == null)
        return n;
     int balance = AVLNode.tiltDegree(n);
     if (balance >= 2) {
          if (AVLNode.tiltDegree(n.left) <= -1)
             //3rd case
             n.left = rotateRightChild(n.left);
          n = rotateLeftChild(n); //1st case
      } else if (balance <= -2) {
        if (AVLNode.tiltDegree(n.right) >= 1)
           //4th case
           n.right = rotateLeftChild(n.right);
        n = rotateRightChild(n); //2nd case
      }
      AVLNode.updateHeight(n);
      return n;
    }
  } // end class AVLTree
```

Assume all methods called in method rebalance has been implemented for you. Write code for method ***public AVLNode rebalanceAll(AVLNode n).***
This method changes any binary search tree (built using AVLNode) (starting from node n downward) into an AVL tree. It returns the new root of the changed tree.

6. (5 marks) Explain (with drawing) how to do a merge sort on array.

7. (4 marks) write code for method
   public int[] makeHeap(int[] a)

This method takes an array of int and transform it into an array that is used for min-heap (small values are more important than large values).

8. (7 marks) A heap of integer that regards small values to be important starts with only one value, 15, inside it. What will the heap look like (in tree form) if we sequentially:
   - add 10,
   - add 7,
   - add 12,
   - add 3,
   - remove the most important value,
   - add 20,
   - remove the most important value

   Draw each step of the change.

9. (5 marks) A text file has the number of characters as follows:
   a: 100
   b: 160
   c: 250
   d: 300
   e: 320

Draw each step when you create a huffman tree from this data. What are the final representations for each character?

10. (7 marks) Class Heap (a min-heap that stores integers) has the following variables and methods (but the class is not complete):

```
public class Heap{
        int[] a;
        int size; //number of elements in the heap

        public Heap(){
                // a working constructor that initializes everything correctly.
                // Do not write code for this. You can call this constructor.
        }

        public void add (int element) {
                if (++size == a.length) {
                        int[ ] newHeap = new int [2 * a.length];
                        System.arraycopy (a, 0, newHeap, 0, size);
                        a = newHeap;
                }
                a [size - 1] = element;
                percolateUp(); //Can be called. DO NOT WRITE CODE FOR THIS METHOD.
        }

        public int pop( ) {
                if (size==0)
                        throw new NoSuchElementException("Priority queue empty.");
                int minElem = a [0];
                a [0] = a [size – 1];
                a [--size] = minElem;
                percolateDown (0);
                return minElem;
        }

        // percolate up from specified position.
        //You can call this method. DO NOT WRITE CODE FOR THIS METHOD.
        public void percolateUp(int index) {
                …
        }
 }
```

Add the following methods to class Heap:

- ***public void percolateDown(int index)***: Percolate down from a given position.
- ***public void changeData(int index, int value)*****:** This method changed data at position *index* in the array to *value*. The array after the change must still have the quality of min-heap.