Name……id ……..……section…….CR58……

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Final Examination Date 11 May 2017 Time 08.30-11.30 (3 hours)

Important

1. This exam paper has 7 questions. There are 6 pages in total including this page. The total mark is 40.
2. Write your answers in the provided answer book.
3. Write your name and ID on every page of this exam paper.
4. When the exam finishes, students must stop writing and remain in their seats until all question sheets are collected and the examiners allow students to leave the exam room.
5. A student must sit at his/her desk for at least 45 minutes.
6. 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.
7. No books, lecture notes or written notes of any kinds are allowed in the exam room.
8. No calculators are allowed.
9. 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.
10. 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.
11. A student who violates rules will be considered as a cheater and will be 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. (10 marks) Given a double hashing hash table for integer data of size 13. Let *hash(x) = x%TableSize* and $hash_2(x) = 7-(x\%7)$. The position for the data, x, after colliding for the i^{th} time, $h_i(x)$, is $hash(x) + i*hash_2(x)$.

The array looks like

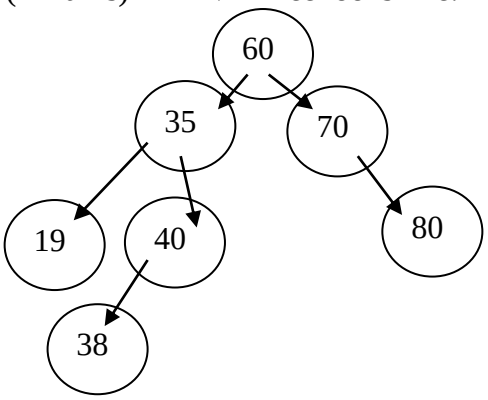
					5					10		

What will happen if we sequentially do the following actions:

- Add 18
- Add 19
- Add 31
- Delete 31
- Add 32

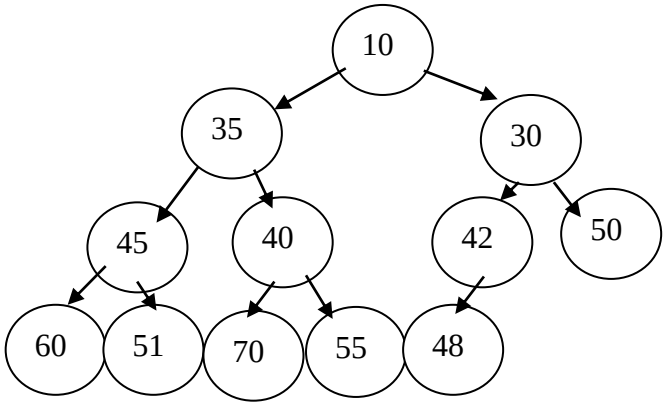
Using DELETED object and replacing DELETED object whenever possible, **show and explain** (step by step) what happens.

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



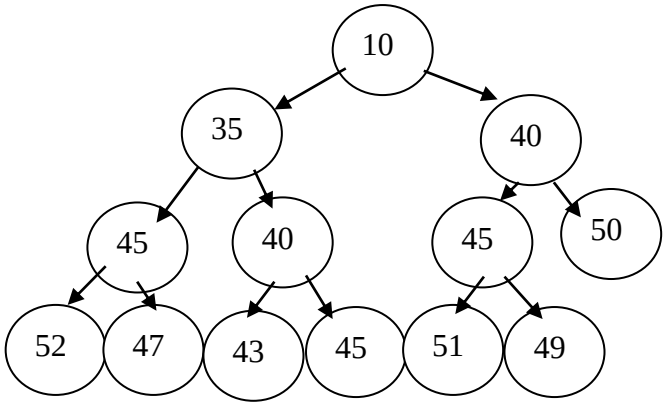
Draw pictures and explain, step by step, what happens when we delete 19.

3. (2 marks) A heap of integer that regards small values to be important looks like:



Draw the final form that the heap becomes after adding value 5.

4. (2 marks) A heap of integer that regards small values to be important looks like:



Draw the final form that the heap becomes after removing the most important data.

5. (10 marks) A Binary Search tree storing integer values is constructed using *BSTNode*, which has the following code:

```
public class BSTNode{
     public int value;
     public BNode left;
     public BNode right;
     public BNode parent;

}
```

Write code for method:

a) (4 marks) *public static int height(BSTNode n)*:
   This method receives *n*, a node that roots the subtree we are looking at (*n* can be *null*). It calculates the height of the subtree. If *n* is *null*, the method returns -1.

b) (6 marks) *public static CDLinkedList nonAVLNodes(BSTNode n):*
   This method receives *n*, a node that roots the subtree we are searching (*n* can be *null*). It returns a doubly linked-list containing all nodes from the subtree that do not satisfy AVL tree constraint. (Yes, we are finding all the nodes of a binary search tree that fail AVL rule).

*You can create a *CDLinkedList* (a doubly-linked list) that stores *BSTNode* by using its default constructor, *CDLinkedList()*.

*The following methods can be called for both a) and b):

- *static CDLinkedList append(CDLinkedList list1, CDLinkedList list2)*:
  - This method receives 2 linked lists of *BSTNode*, *list1* and *list2*. It returns a new linked list that has all contents from *list1* and *list2*.
- *static CDLinkedList insertToFront(CDLinkedList list1, BSTNode v)*:
  - This method receives a linked list, *list1*, and a data, *v*. It returns a new linked list that has all contents of *list1* (in the order stored on *list1*), with *v* as an additional data at the front of the list.

6. (6 marks) In this question, we have a heap of integer (small values are more important than large values) constructed using nodes of a binary tree (*class HeapNode*):

```
public class HeapNode{
     int data; //the value that the node stores.
     HeapNode left;
     HeapNode right;
     HeapNode parent; //pointers to other nodes within
                      //the same tree.
}

public class Heap{
     HeapNode root;
}
```

Write code for the following method of class Heap:

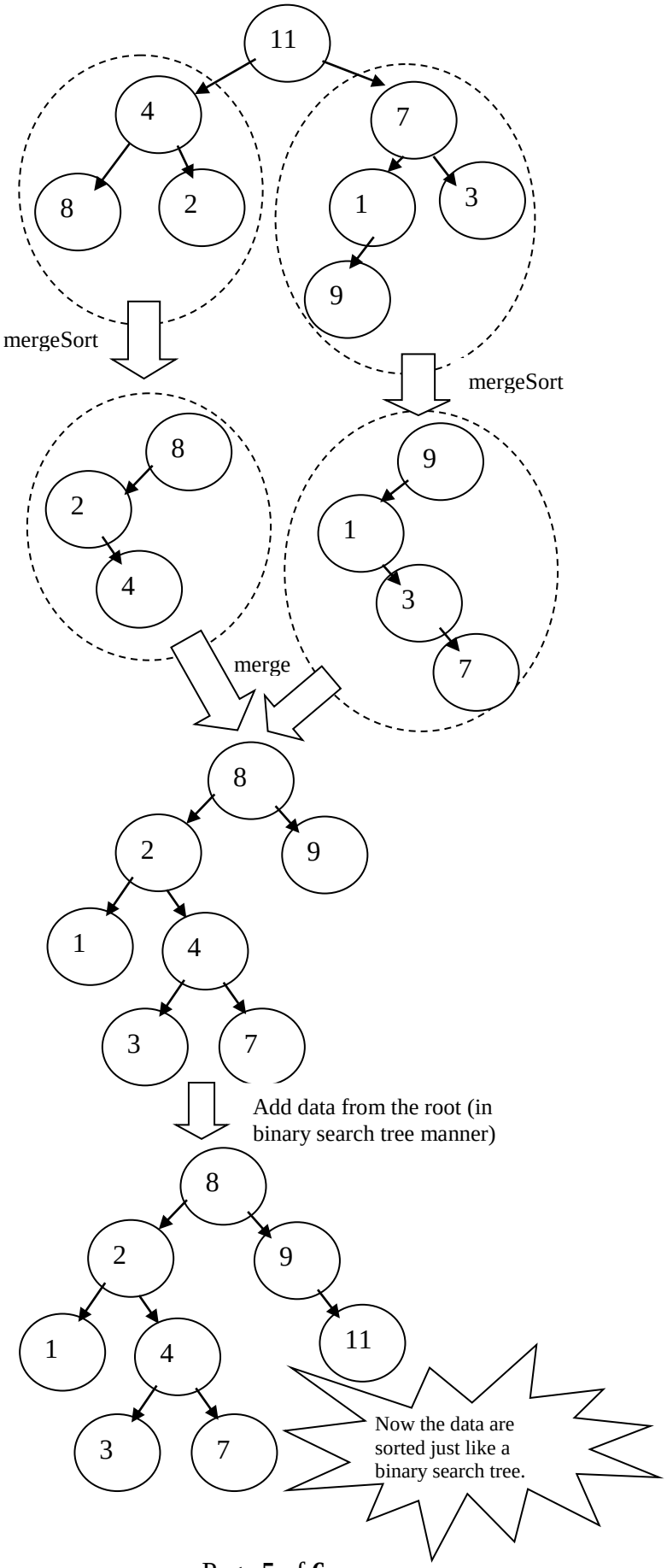
public void percolateDown(): This method tries to move data (stores in the root of this heap) down the heap, such that the heap can become correct again after the root's data is replaced by a new value (but we do not perform such replacement in this question).

7. (6 marks) A binary tree of integer is constructed by connecting nodes together. A node is defined as:

```
public class BNode{
   int data; //the value that the node stores.
   BNode left;
   BNode right;
  BNode parent;
}
```

Data in binary tree do not have to be sorted.

A mergesort on binary tree is performed as follows:


11
4
7
8
2
1
3
9
mergeSort
mergeSort
8
2
4
9
1
3
7
merge
8
2
9
1
4
3
7
Add data from the root (in binary search tree manner)
8
2
9
1
4
11
3
7
Now the data are sorted just like a binary search tree.

Write code for method:

a) (3 marks) *public BNode merge(BNode n1, BNode n2)*:
This method receives *n1* and *n2* (each of them can be *null*). They are roots of two sorted binary trees (or subtrees). The method returns a root of a sorted tree (or subtree) that has all values from *n1* and *n2*. *n1* and *n2* are allowed to change.

b) (3 marks) *public BNode mergeSort(BNode n):*
This method receives *n*, a node that roots the tree (or subtree) we are trying to sort (*n* can be *null*). The method performs mergesort for the tree (or subtree) and returns a node that roots the finished sorted tree (or subtree).

*The following methods can be called for both a) and b):

- *static BNode remove(BNode n, int v)*:
    - This method removes *v* from a sorted tree (or subtree) that has *n* as its root. It returns the root of the changed tree (or subtree). The changed tree (or subtree) remains sorted.
- *static BNode add(BNode n, int v)*:
    - This method adds *v* to a sorted tree (or subtree) that has *n* as its root. It returns the root of the changed tree (or subtree). The changed tree (or subtree) remains sorted.