KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
Information and Computer Science Department
ICS202: Data Structures
HOMEWORK 3 (Term 062)
Due Date: Monday, April 23rd, 2007

Question#1 [20 points: 4 + 10 + 6]

Consider the following two methods which do the same thing in different ways:

```
public static long power1 (int x, int n) {
   long product = 1;
   for (int i = 1; i <= n; i++)
      product *= x;
   return product;
}

public static long power2 (int x, int n) {
   if (n == 1) return x;
   else if (n == 0)return 1;
   else {
      long t = power2(x , n / 2);
      if ((n % 2) == 0) return t * t;
      else return x * t * t;
   }
}
```

(a) Write the recurrence relation for the running time **T(n)** of the method **power2(x, n)**.

T(n) = a **for n = 1 …… (1 point)**
T(n) = b **for n = 0 …… (1 point)**
T(n) = T(n/2) + c **for n > 1 …… (2 points)**

(b) Solve the recurrence relation in (a) by the method of ***unrolling and summing*** to find the big-O complexity of **power2(x, n)**. Show all details.

T(n) = T(n/2) + c **…… (1 point)**
= T(n/4) + 2c **…… (1 point)**
= T(n/8) + 3c **…… (1 point)**
= …..
T(n) = T(n/2k) + kc **…… (1 point)**

Now consider n/2k = 1 **…… (1 point)**
n = 2k **…… (1 point)**
k = Log2 n **…… (1 point)**
By putting the value of k in the above equation:
T(n) = T(1) + c * Log n **…… (1 point)**
= a + c * Log n **…… (1 point)**
Therefore, the complexity of power2 is O(Log n) **…… (1 point)**

(c) For large value of n, which of the above two methods is more efficient in terms of: i) time and, ii) memory space. Give reason for your answer.

i. **The time efficiency for power1 is O(n) and for power2 is O(Log n). Therefore power2 is more time efficient.**
ii. **power2 needs more stack space because of recursion, but power1 does not need it. Therefore power1 is more space efficient.**

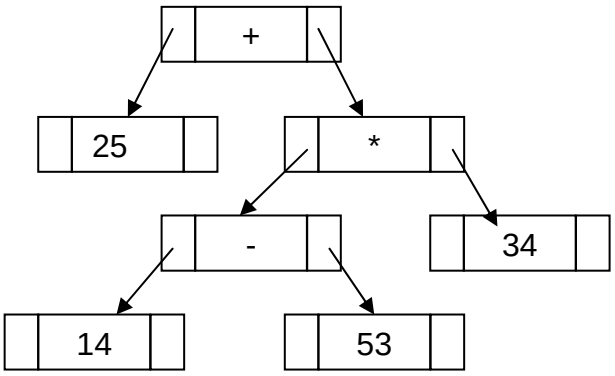
Question#2 [25 points: 20 + 5]

(a) Implement a Java class **`ExpressionTree`** having the following two methods:

```
public static double evaluate (BinaryTree b)
public static void main (String[] args)
```

The **`BinaryTree`** type parameter of the **`evaluate`** method represents an arithmetic expression, where each leaf node contains an operand (a **`Double`** object), and each internal node contains an operator (a **`Character`** object) as its key.

As an example, the expression: **25 + (14 – 53) * 34** is represented as the following binary tree:



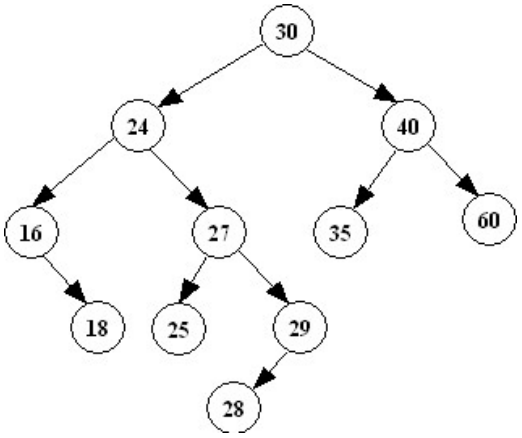
The **`evaluate`** method should recursively find the value of the expression represented by its parameter. You only consider **+, -, /** and ***** as valid operators. If the parameter is an empty tree, the method must throw an appropriate exception.

The **`main`** method should test the **`evaluate`** method. It should initialize a **`BinaryTree`** object to represent the following expression and then use the **`evaluate`** method to print its value.

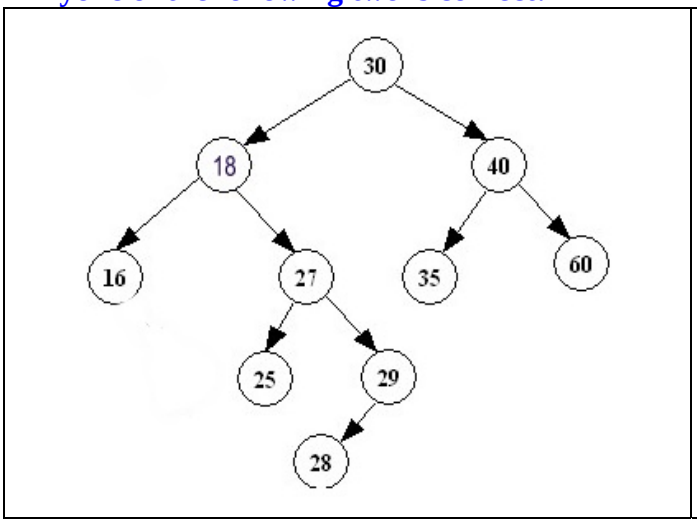
3 * (5 + 7) – 54 / (9 * 2)

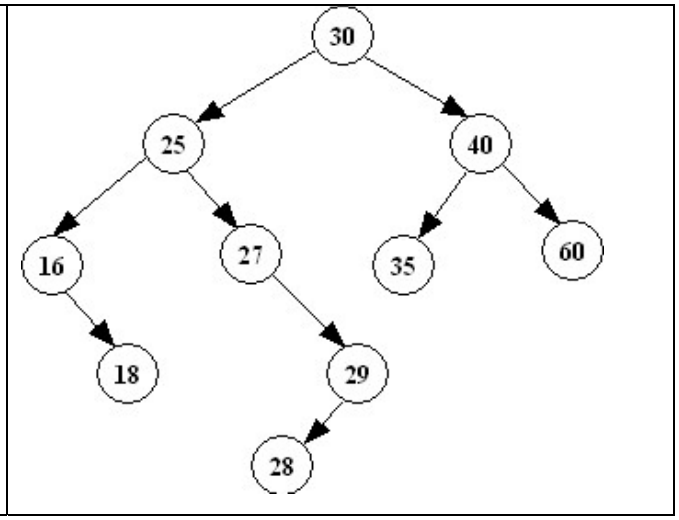
```java
package ics202.hw03;
import ics202.*; // .....................................import has 1 point
public class ExpressionTree {

   public static double evaluate (BinaryTree b){
      // ..................................each line has 1 point = 13 points
      if (b.isEmpty())
         throw new InvalidOperationException("The tree is empty");
      if (b.isLeaf())
         return ((Double)b.getKey()).doubleValue();

      char operator = ((Character)b.getKey()).charValue();
      double left = evaluate(b.getLeft());
      double right = evaluate(b.getRight());
      switch (operator){
      case '+': return left + right;
      case '-': return left - right;
      case '*': return left * right;
      case '/': return left / right;
      default: throw new InvalidOperationException(
                        "Operator not supported: "+operator);
      }
   }

   public static void main(String[] args){
      // ....................................each line has 1/2 point = 6 points
      //3 * ( 5 + 7 ) – 54 / ( 9 * 2)
      BinaryTree v5 = new BinaryTree(new Double(5));
      BinaryTree v7 = new BinaryTree(new Double(7));
      BinaryTree v3 = new BinaryTree(new Double(3));
      BinaryTree v9 = new BinaryTree(new Double(9));
      BinaryTree v2 = new BinaryTree(new Double(2));
      BinaryTree v54 = new BinaryTree(new Double(54));

      BinaryTree t1 = new BinaryTree(new Character('+'),v5,v7);
      BinaryTree t2 = new BinaryTree(new Character('*'),v9,v2);
      t1 = new BinaryTree(new Character('*'),v3,t1);
      t2 = new BinaryTree(new Character('/'),v54,t2);
      BinaryTree t = new BinaryTree(new Character('-'),t1,t2);
      System.out.println("3 * ( 5 + 7 ) – 54 / ( 9 * 2) = " + evaluate(t));
   }
}
```

(b) Draw the resulting Binary search tree after deleting 24 by **copying method** from the following BST:



Anyone of the following two is correct:





Question#3 [30 points: 5 + 5 + 5 + 15]

(a) Show the result of converting the array given below into a **min-heap** by the **bottom-up** approach:

15	10	13	9	6	7	8	17	8	5

5	**6**	**7**	**8**	**10**	**13**	**8**	**17**	**9**	**5**

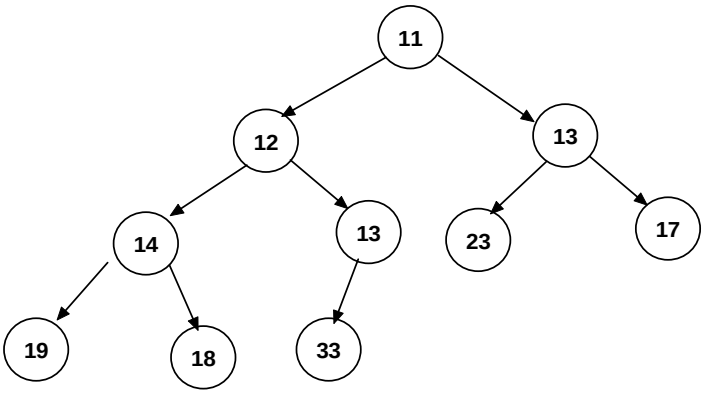
(b) Show the resulting heap after the **dequeueMin** operation is performed on the following heap.

15	17	19	20	35	20	25	27	21	37	36

17	**20**	**19**	**21**	**35**	**20**	**25**	**27**	**36**	**37**

(c) Draw the tree that results in building a **max-heap**, **top-down**, when the following keys are inserted one at a time in an initially empty **max-heap**.

`11, 19, 23, 12, 13, 17, 13, 14, 18, 33`



(d) Write a method in the **`BinaryHeap`** class with the following signature:

```
public void changeKey(Comparable oldKey, Comparable newKey)
```

The method will replace the first occurrence of **`oldKey`** with the **`newKey`**, and restore the **Min-Heap** property after the change. If the **`oldKey`** does not exist in the heap, the method prints an appropriate message and returns without changing the heap. Suppose our binary heap object (**`bh`**) has the following keys:

5	10	9	11	15	10	12	13	14	16	18

Then the method call: **`bh.changeKey (new Integer(10), new Integer(3))`** should change the keys to:

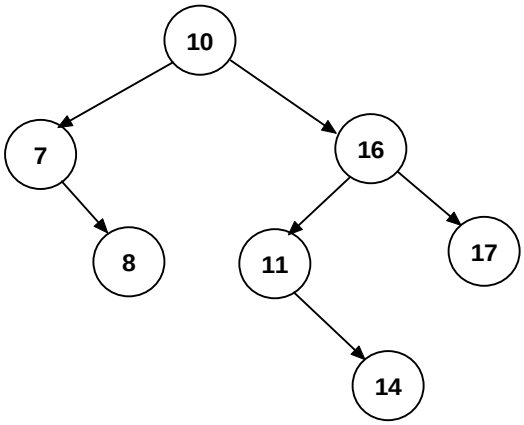
3	5	9	11	15	10	12	13	14	16	18

Also write test code to create a binary heap and use the changeKey method on it.

```
// in class BinaryHeap (each of body line has 1 point = 12 points)

    public void changeKey(Comparable oldKey, Comparable newKey){
      int i=1;
      while (i < count+1){
            if (array[i].equals(oldKey))
                  break;
            i++;
      }
      if (i == count+1){
            System.out.println(oldKey + " is not present in the heap"); return;
      }
      array[i] = newKey; // change the key
      if (i>1 && array[i].compareTo(array[i/2])<0)
            percolateUp(i);
      else
            percolateDown(i);
    }


// inside the main method of BinaryHeapTest class (= 3 points)

      case 7:
        System.out.print("Enter the key to replace: ");
        int oldk = stdin.nextInt(); // ............................. (1 point)
        System.out.print("Enter the new key: ");
        int newk = stdin.nextInt();// ............................. (1 point)
        binaryheap.changeKey(new Integer(oldk), new Integer(newk));// (1 point)
        break;
```

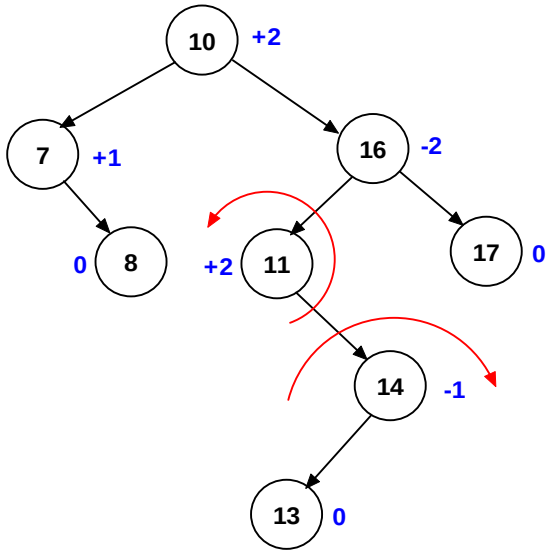
Question#4 [25 points]

Draw the **intermediate** trees, the **final** tree, and **mention the rotations performed**, if any, when the keys **13** and **12** are inserted, in the given order, in the AVL tree given below. (**Note**: The second insertion is done on the AVL tree modified by the first insertion).

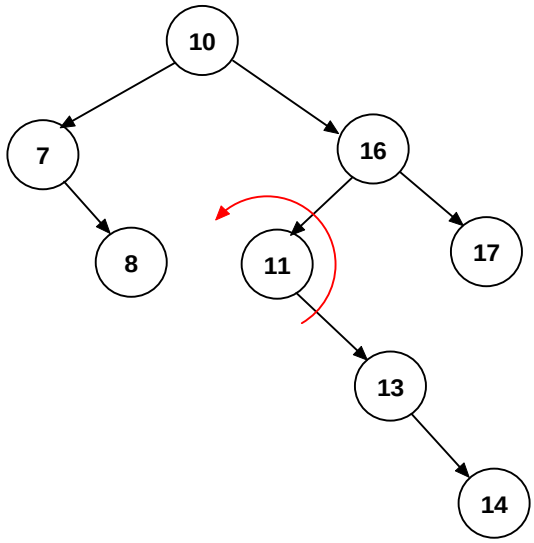


Grading Policy:
3 points: First diagram (after inserting 13)
1 point: Stating: double right-left rotation or right rotation followed by left rotation
6 points: Second diagram (after the first rotation of the double rotation)
6 points: Third diagram (after the second of the double rotation)
2 points: Fourth diagram (after inserting 12)
1 point: Stating: single right rotation
6 points: Fifth diagram (final tree)
Note: DO NOT PENALIZE STUDENTS FOR NOT SHOWING THE BALANCE FACTORS

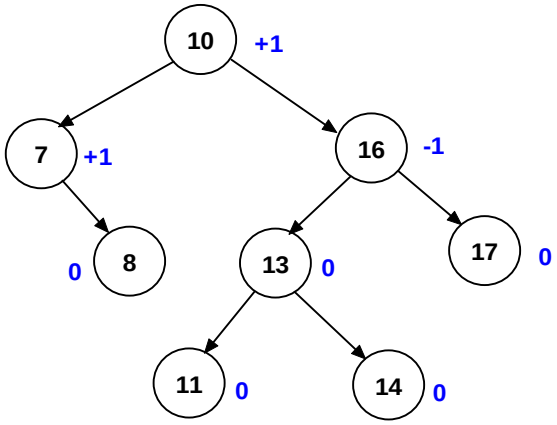
SOLUTION:
After inserting 13, the tree will become the following. It needs a double Right-Left Rotation on Node (11)



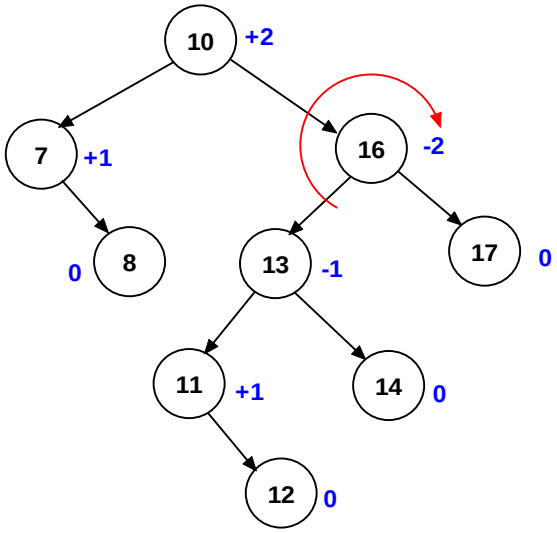
For the double rotation, first we need to do a Right rotation on 14. This will change the tree to:



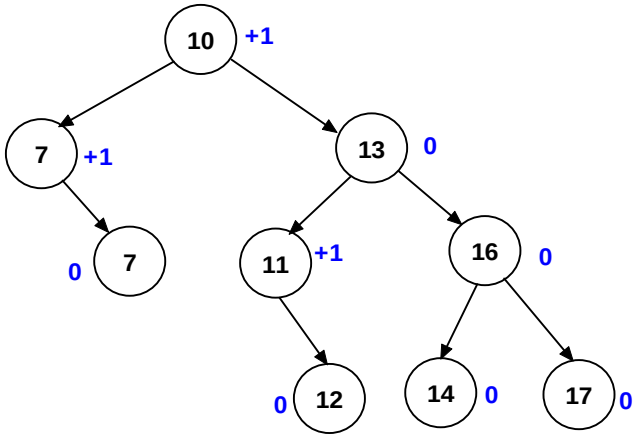
The Left rotation on 11 will change the tree to:



Now after inserting 12, the tree will become the following. It needs a single Right rotation on 16.



After the right rotation on 16, the tree will become:



Important Notes:

- Your report for this homework must be **word-processed** and must follow the **homework submission template** format, which you can get in the downloadables section of the WebCT.
- Diagrams can be drawn using a tool (Visio, SmartDraw, etc) or by hand.
- All the classes for this homework must be stored in a package **`ics202.hw03`.**
- You must import the necessary packages needed for your program.
- You need to submit two things:
    1. A printed copy of your report at the beginning of your class on the due date.
    2. Submit your entire **`ics202`** package into the webCT under the Assignments option.