**1.** Write a program in Java to implement a binary search tree (BST). The program will insert five names into the BST. All names must be read from the console. After insertion, the program will show five names in ascending order. Then it will ask for any name from the user. The name will be searched in the BST, and if found, the name will be deleted from the tree. Otherwise, an appropriate message will be shown. (**Do not use any built-in tree class.**) The sample inputs/ outputs are given below. **(25 points)**

**Sample inputs and outputs:**

(User’s inputs are shown in **bold**) **Sample execution 1:**

Enter 5 names: **Bob Edwin Cory Alice Mark**

The names in the BST: Alice Bob Cory Edwin Mark

Enter the name to be removed: **Edwin**

The name is removed. The elements in the BST: Alice Bob Cory Mark

**Sample execution 2:**

Enter 5 names: **Bob Edwin Cory Alice Mark**

The names in the BST: Alice Bob Cory Edwin Mark Enter the name to be removed: **Travis** The name is not found.

1. Insert the names of 12 months in an AVL tree in the regular order from January to December. **Treat the month names as dictionary words, NOT as the month’s position.** For instance, January as a month comes before February, but the word ‘February’ comes before ‘January’ in the English dictionary. Show each step of insertion and rotation. **(25 points)**



January



January

0



January



February



February

1

0



January



February



March

March

0

0

0



January



February



April

March

1

1

0

April

0



January



February



May

March

0

1

1

April

0

May

0



January



February



June

March

1

1

0

April

0

May

0

June

0



January



February



July

March

0

1

1

April

0

May

0

June

1

July

0



January



February



August

March

1

2

1

April

1

May

0

June

1

July

0

March

August

0



January



August



September

March

0

0

0

April

0

May

1

June

1

July

0

March

0

February

September

0



January



August



August

March

0

0

1

April

0

May

0

June

1

July

0

March

0

February



January



August



October

March

1

0

1

April

0

October

0

June

1

July

0

March

0

February

September

0

May

0



January



August



October

March

0

0

0

April

0

May

2

June

1

July

0

March

0

February

September

1

October

0



January



August



December

March

1

1

0

April

0

October

1

June

1

July

0

March

1

February

September

0

May

1

November

0

December

0



January



August



November

March

0

0

0

April

0

October

1

June

1

July

0

March

0

February

September

0

May

1

November

0

**Bonus question:**

1. Represent the following two code snippets as appropriate functions by calculating primitive operations of each statement and hence find the Big-Oh running time

complexity: **(20 points)**

a) public static int code1(int[ ] data)

{

int low = 0, high = data.length − 1; while

(low < high)

{

int temp = data[low]; data[low++] = data[high];

data[high−−] = temp;

} int i = 0; while(i < data.length)

System.out.println(data[i++]);

}

b) public static int code2(int[ ] first, int[ ] second)

{

int n = first.length, count = 0;

for(int i=0; i < n; i++)

{ int total = 0;

for(int j=0; j < n; j++) for(int k=0; k <= j; k++) total += first[k];

if(second[i] == total)

count++;

}

return count;

}

**I will indicate answer as O function often used to present** complexity in programming engineering.

**Complexity of a) = O(n)**

**Complexity of b) = O(n3)**