ASSIGNMENT 3

NAME : RADHIKA
ER. NO.: 18114060

Problem 1:
Given the set of integers, write a C++ program to create a binary search tree (BST) and print all possible paths for it. You are not allowed to use subarray to print the paths.

Convert the obtained BST into the corresponding AVL tree for the same input. AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.

Convert the obtained BST into the corresponding red-black tree for the same input. Red-Black Tree is a self-balancing Binary Search Tree (BST) where every node follows following rules.
1) Every node has a color either red or black.
2) Root of tree is always black.
3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).
4) Every path from a node (including root) to any of its descendant NULL node has the same number of black nodes.

Write a menu driven program as follows:
1. To insert a node in the BST and in the red-black tree
2. To create AVL tree from the inorder traversal of the BST
3. To print the inorder traversal of the BST/AVL/red-black tree
4. To display all the paths in the BST/AVL tree/red-black tree
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation (print color for red-black tree)
6. Exit

Example:

Input:
10 20 30 40 50 25

Output:

BST:
```
10 [4]
        20 [3]
                25
                30 [2]
                        40 [1]
                                50
```
Inorder traversal (This output is for BST Tree): 10 20 25 30 40 50

AVL Tree:
```
30 [0]
        20 [0]
                10
                25
        40 [1]
                50
```
Inorder traversal (This output is for AVL Tree): 10 20 25 30 40 50

Red Black Tree:
```
20 [2] [BLACK]
        10 [BLACK]
        40 [1] [RED]
```

```
                30 [1] [BLACK]
                    25 [RED]
                50 [BLACK]
```
Inorder traversal (This output is for Red Black Tree): 10 20 25 30 40 50

Paths (This output is for AVL tree):

```
30->20->10
20->10
10
30->20->25
20->25
25
30->40->50
40->50
50
```

Data Structure :-

- ➔ For Binary Search Tree and AVL Tree a class Node is used with data members val(int), right , left(node pointers) i.e. right child, left child respectively.
- ➔ For Red-Black Tree, class Red-Black Node is used with parameters val(int), colour, parent, right child, left child.
- ➔ Colour Red and black are defined as true and false respectively.

Algorithm :-

- ➔ Recursive algorithms for multiple functions like getHeight and printing paths
- ➔ Use of vectors to store the paths
- ➔ Using rotate functions to re-establish the property of Red-Black trees.
- ➔ Using rotate functions to establish the property of self-balancing and create an avl tree using the inorder traversal of the BST.

```
^C
[radhika@localhost L3]$ g++ Q1.cpp
[radhika@localhost L3]$ ./a.out

1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
10
Time: 51 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
20
Time: 58 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
30
Time: 51 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
40
Time: 47 microsec
```

```
1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
50
Time: 83 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
1
Enter the number to insert
25
Time: 51 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
2
Time: 13 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
3

The inorder traversal for the BST is
10 20 25 30 40 50
The inorder traversal for the AVL tree is
10 20 25 30 40 50
The inorder traversal for the Red-Black Tree is
10 20 25 30 40 50
Time: 78 microsec
```

```
1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
4
For BST :
10->20->30->25
10->20->30->40->50
20->30->25
20->30->40->50
30->25
30->40->50
25
40->50
50
For AVL Tree :
30->20->10
30->20->25
30->40->50
20->10
20->25
10
25
40->50
50
For Red-Black Tree :
20->10
20->40->30->25
20->40->50
10
40->30->25
40->50
30->25
25
50
Time: 201 microsec
```

```
1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
5
For BST :
10[4]
                20[3]
                                30[1]
                                                25[0]
                                                40[1]
                                                                50[0]
For AVL Tree :
30[0]
                20[0]
                                10[0]
                                25[0]
                40[1]
                                50[0]
For Red-Black Tree :
20[2][BLACK]
                10[0][BLACK]
                40[1][RED]
                                30[1][BLACK]
                                                25[0][RED]
                                50[0][BLACK]
Time: 121 microsec



1.Insert a node in Both tree
2.Create AVL from BST
3.Print Inorder traversal of Both tree
4.Display all paths in Both tree
5.Print Both Tree using level-wise Indentation
6.Exit this program
6
[radhika@localhost L3]$
```

Problem 2:
For a given sequence of positive integers $A_1, A_2, \ldots, A_N$ in decimal, find the triples (i, j, k), such that $1 \le i < j \le k \le N$ and $A_i \oplus A_{i+1} \oplus \ldots \oplus A_{j-1} = A_j \oplus A_{j+1} \oplus \ldots \oplus A_k$, where $\oplus$ denotes bitwise XOR. This problem should be solved using dynamic programming approach and linked list data structures.

Input:
(a) Number of positive integers N.
(b) N space-separated integers $A_1, A_2, \ldots, A_N$.

Output:
Print the number (count) of triples and list all the triplets in lexicographic order (each triplet in a new line).

Example:
Input:
N = 3
5 2 7

Output:
2
(1, 2, 3)
(1, 3, 3)

3

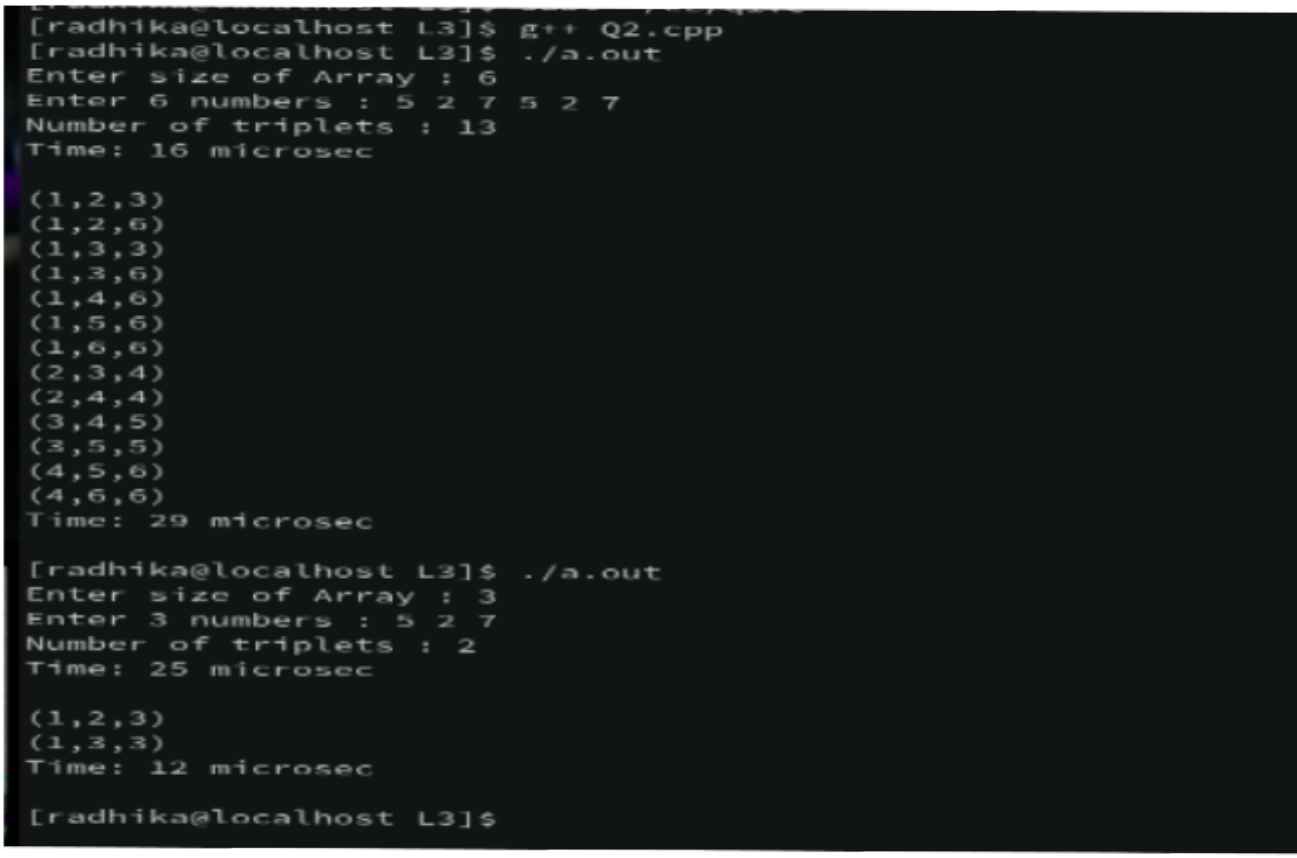
Data Structure:

1. 2-way Linkedlist
   Vertical List -> all nodes are in increasing order and they stores the xor value of array
   Horizontal list -> this list is respective to all nodes of vertical list.

2. Array to store node of vertical list nodes.

Algorithm:

1. Insert 0 as node in vertical list and -1 as index in list respective to 0, because totalxor before 0th index is 0.
2. Iterate on array and calculate xor till that index and store that index in horizontal list of node value with totalxor.
3. If node is not present with totalxor value, then insert new node with totalxor in vertical list in increasing order.
4. Calculate number of triplets with dynamical programming using linked list.
5. Basically Logic id we need to find ith and kth index (k>i) s.t. A(i) xor A(i+1) xor...xor A(k) = 0. then we conclude A(0) xor ... xorA(i-1) = A(0) xor ... xor A(k).
6. In last we print all triplets using snodes array and respective horizontal list.


[radhika@localhost L3]$ g++ Q2.cpp
[radhika@localhost L3]$ ./a.out
Enter size of Array : 6
Enter 6 numbers : 5 2 7 5 2 7
Number of triplets : 13
Time: 16 microsec
(1,2,3)
(1,2,6)
(1,3,3)
(1,3,6)
(1,4,6)
(1,5,6)
(1,6,6)
(2,3,4)
(2,4,4)
(3,4,5)
(3,5,5)
(4,5,6)
(4,6,6)
Time: 29 microsec
[radhika@localhost L3]$ ./a.out
Enter size of Array : 3
Enter 3 numbers : 5 2 7
Number of triplets : 2
Time: 25 microsec
(1,2,3)
(1,3,3)
Time: 12 microsec
[radhika@localhost L3]$