1. Write a program to declare this "int MostFrequentCharaceter(char str[], int n);" function. The function accepts a string ‘str’ of length ‘n’, that contains alphabets. Implement the function to find and return the frequency of the letter, which has the highest frequency.

Assumption: String only contains lower case letters.

Note, Return -1 if str is null.

Example:

Input:

abcdaabcaba

Output:

5

2. **Maximum profit obtained by buying and selling stocks**

You have been given stock prices for next N days. Find out the maximum profit that can be obtained by buying and selling the stocks. Conditions: Stock must be sold any day after the buying date You can buy and sell multiple times, but cannot hold more than 1 stock at a time. You can only perform a single transaction on a day i.e., can only either buy or sell on a single day. For example: Share price in thousands 5 1 4 6 7 8 4 3 7 9 Max benefit:

Buy share on day 2 at the cost of 1

Sell share on day 6 when the price rises to 8

Buy a share again on day 8 at the cost of 3

Sell share on day 10 when the price rises to 9 Total: (8-1)+(9-3)=13

Sample Input:

10

5 1 4 6 7 8 4 3 7 9

Sample Output:

13

3. **Binary Search Tree - Lowest Common Ancestor**

Write a program to find a common ancestor of a given two numbers in the tree. Let T be the root of a tree. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T. Which has both n1 and n2 as descendants (where we allow a node to be a descendant of itself). The LCA of n1 and n2 in T is the shared ancestor of n1 and n2, which is located farthest from the root. Computation of the Lowest common ancestor is useful. For instance, as part of a procedure for determining the distance between the pairs of a node in a tree. The distance from n1 to n2 can be computed as a distance from a root to n1. Plus, the

distance from a root to n2. Later, subtract two times the distance from a root to their lowest common ancestor.

Sample Input:

6

3

1

4

2

-1

3 6

Sample Output:

6

4. **Balanced parenthesis**

Write a program to check whether the given parenthesis is balanced or not.

Sample Input:

{()}[]

Sample Output:

Balanced

5. **Detect cycle in an undirected graph**

Given an Undirected Graph. Check whether it contains a cycle or not.

Sample Input:

3

2 1

0 1

4 3

0 1 1 2 2 3

5 4

0 1 2 3 3 4 4 2

Output:

0

0

1

Explanation:

Testcase 1: There is a graph with 2 vertices and 1 edge from 0 to 1. So, there is no cycle.

Testcase 2: There is a graph with 3 vertices and 3 edges from 0 to 1, 1 to 2, and 2 to 3.

Testcase 3: There is a cycle in the given graph formed by vertices 2, 3, and 4.