TASK 1

AIM:To find the contiguous subarray with the largest sum using Kadane's Algorithm.

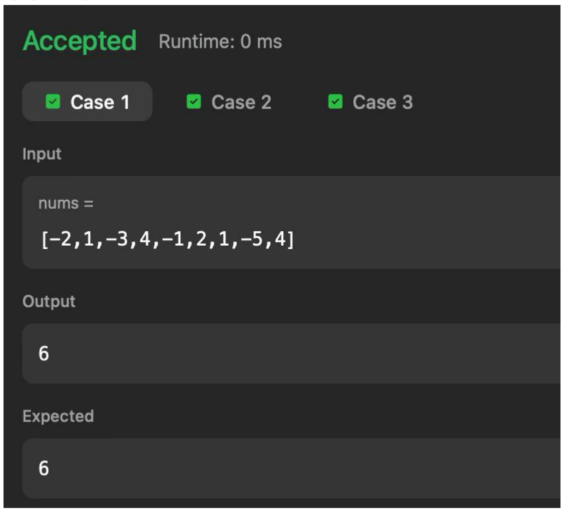
ALGORITHM:

1. Initialize maxSum and currentSum with first element.
2. Traverse array from index 1.
3. Update currentSum = max(arr[i], currentSum + arr[i]).
4. Update maxSum = max(maxSum, currentSum).
5. Return maxSum.

PROGRAM

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```

OUTPUT



Input

```
nums =
[-2,1,-3,4,-1,2,1,-5,4]
```

Output

```
6
```

Expected

```
6
```

RESULT:Thus the program for Maximum Subarray was executed successfully.

TASK 2

AIM:To find the number of contiguous segments whose sum equals given value d.

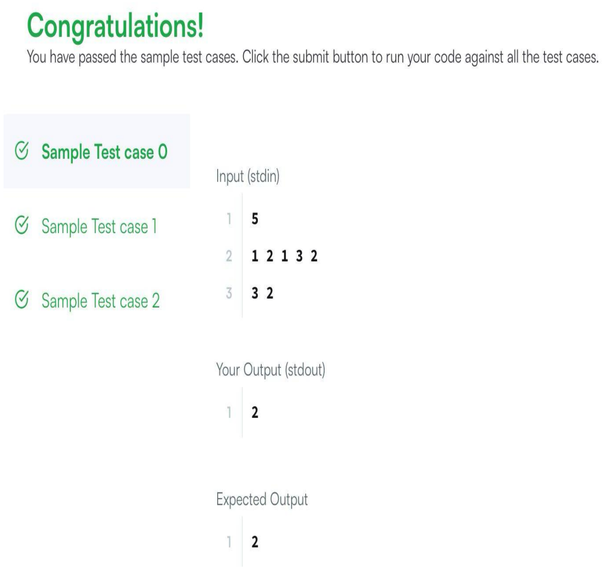
ALGORITHM:

1. Initialize count = 0.
2. Traverse from index 0 to n-m.
3. Find sum of next m elements.
4. If sum == d, increment count.
5. Return count.

PROGRAM

```
import java.io.*;
import java.util.*;

public class Solution {

    public static int birthday(List<Integer> s, int d, int m) {
        int count = 0;

        for (int i = 0; i <= s.size() - m; i++) {
            int sum = 0;
            for (int j = i; j < i + m; j++) {
                sum += s.get(j);
            }

            if (sum == d) {
                count++;
            }
        }

        return count;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        List<Integer> s = new ArrayList<Integer>();

        for (int i = 0; i < n; i++) {
            s.add(sc.nextInt());
        }

        int d = sc.nextInt();
        int m = sc.nextInt();

        int result = birthday(s, d, m);
        System.out.println(result);

        sc.close();
    }
}
```

OUTPUT:



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
5
1 2 1 3 2
3 2
```

Your Output (stdout)

```
2
```

Expected Output

```
2
```

RESULT:Thus the program for Birthday Bar was executed successfully

TASK 3

AIM:To find maximum contiguous and non-contiguous subarray sum.

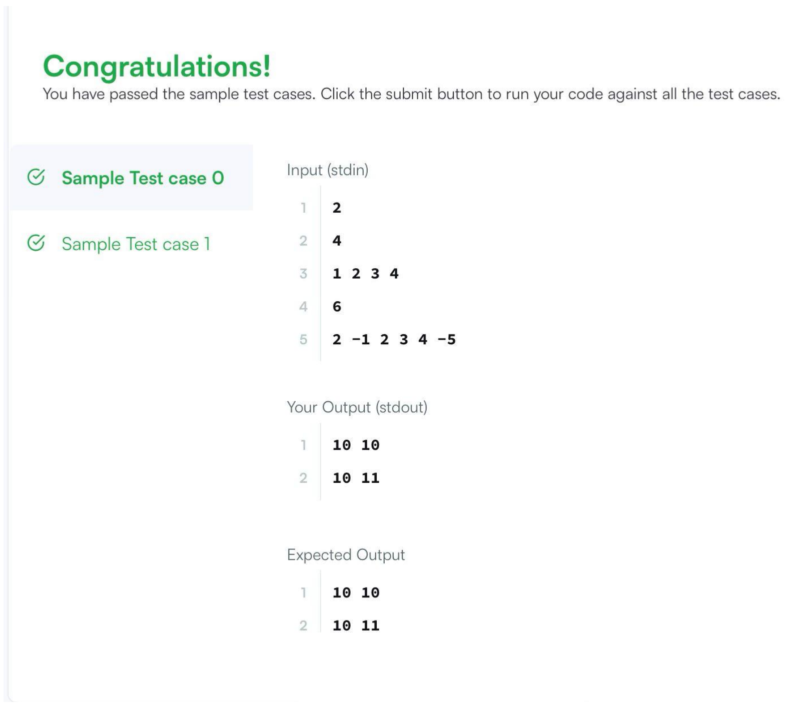
ALGORITHM

1. Start from first element.
2. Keep adding elements.
3. If sum becomes smaller than current element → start new sum.
4. Keep track of maximum value.

PROGRAM

```
import java.io.*;
import java.util.*;

public class Solution {

    public static int[] maxSubarray(int[] arr) {
        int currentMax = arr[0];
        int globalMax = arr[0];

        for (int i = 1; i < arr.length; i++) {
            currentMax = Math.max(arr[i], currentMax + arr[i]);
            globalMax = Math.max(globalMax, currentMax);
        }
        int subsequenceSum = 0;
        int maxElement = arr[0];

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > 0) {
                subsequenceSum += arr[i];
            }
            maxElement = Math.max(maxElement, arr[i]);
        }
        if (subsequenceSum == 0) {
            subsequenceSum = maxElement;
        }

        return new int[]{globalMax, subsequenceSum};
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();   // number of test cases

        while (t-- > 0) {

            int n = sc.nextInt();
            int[] arr = new int[n];

            for (int i = 0; i < n; i++) {
                arr[i] = sc.nextInt();
            }

            int[] result = maxSubarray(arr);
            System.out.println(result[0] + " " + result[1]);
        }

        sc.close();
    }
}.
```

OUTPUT



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
2
4
1 2 3 4
6
2 -1 2 3 4 -5
```

Your Output (stdout)

```
10 10
10 11
```

Expected Output

```
10 10
10 11
```

RESULT:Thus the program for Maximum Subarray (Contiguous & Non-Contiguous) was executed successfully.

TASK 4

AIM:To find maximum sum of circular subarray.

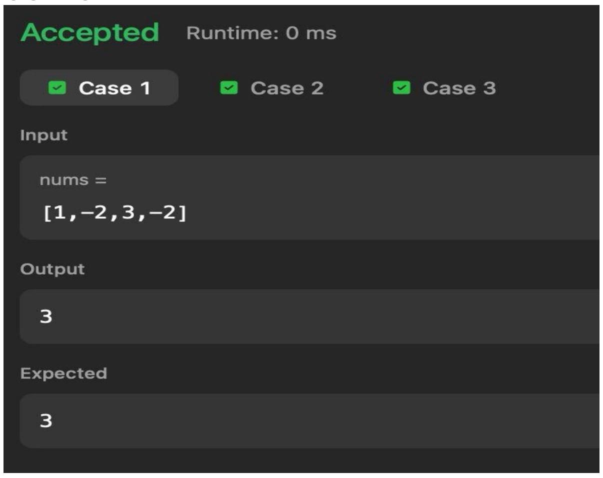
ALGORITHM:

```
1. Find normal max subarray sum using Kadane.
2. Find total array sum.
3. Find minimum subarray sum using Kadane.
4. Circular sum = total - minSum.
5. Return max(normalSum, circularSum).
```

PROGRAM:

```
class Solution {
    public int maxSubarraySumCircular(int[] nums) {

        int totalSum = 0;
        int currentMax = 0;
        int maxSum = nums[0];
        int currentMin = 0;
        int minSum = nums[0];
        for (int num : nums) {

            currentMax = Math.max(num, currentMax + num);
            maxSum = Math.max(maxSum, currentMax);
            currentMin = Math.min(num, currentMin + num);
            minSum = Math.min(minSum, currentMin);
            totalSum += num;
        }

        if (maxSum < 0) {
            return maxSum;
        }

        return Math.max(maxSum, totalSum - minSum);}}
```

OUTPUT:



```
Accepted    Runtime: 0 ms

✓ Case 1    ✓ Case 2    ✓ Case 3

Input
nums =
[1,-2,3,-2]

Output
3

Expected
3
```

RESULT:Thus the program for Maximum Sum Circular Subarray was executed successfully.

TASK 5

AIM:To convert a string into a 32-bit signed integer.

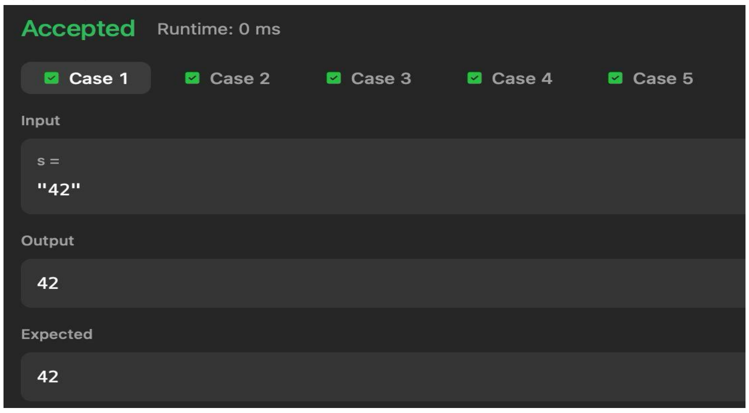
ALGORITHM:

1. Remove leading spaces.
2. Check sign.
3. Convert digits to number.
4. Handle overflow.
5. Return result.

PROGRAM

```
class Solution {
    public int myAtoi(String s) {
        int i = 0, n = s.length();
        while (i < n && s.charAt(i) == ' ') i++;
        int sign = 1;
        if (i < n && (s.charAt(i) == '+' || s.charAt(i) == '-')) {
            if (s.charAt(i) == '-') sign = -1;
            i++;
        }
        long result = 0;
        while (i < n && Character.isDigit(s.charAt(i))) {
            result = result * 10 + (s.charAt(i) - '0');
            if (sign == 1 && result > Integer.MAX_VALUE)
                return Integer.MAX_VALUE;
            if (sign == -1 && -result < Integer.MIN_VALUE)
                return Integer.MIN_VALUE;
            i++;
        }
        return (int)(sign * result);
    }
}
```

OUTPUT:



```
Accepted    Runtime: 0 ms

Case 1   Case 2   Case 3   Case 4   Case 5

Input
s =
"42"

Output
42

Expected
42
```

RESULT:Thus the program for String to Integer (atoi) was executed successfully.

TASK 6

AIM:To find minimum deletions required so that no two adjacent characters are same.

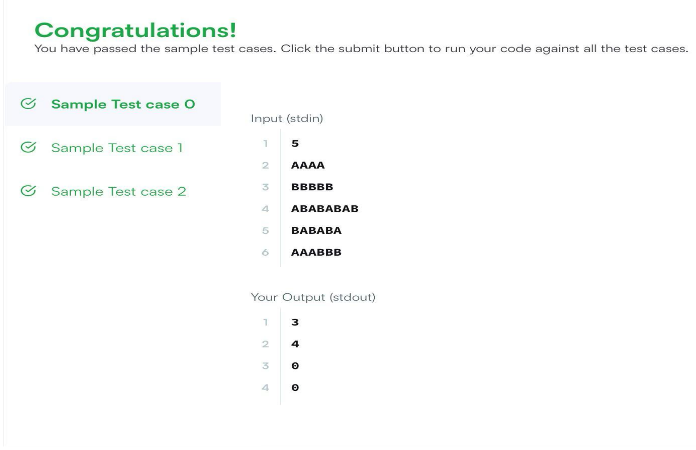
ALGORITHM:

1. Initialize count = 0.
2. Traverse string from index 1.
3. If current character equals previous, increment count.
4. Return count.

PROGRAM:

```
import java.io.*;
import java.util.*;
public class Solution {
    public static int alternatingCharacters(String s) {
        int deletions = 0;
       for (int i = 1; i < s.length(); i++) {
            if (s.charAt(i) == s.charAt(i - 1)) {
                deletions++;
            }
        }
         return deletions;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int q = sc.nextInt();
        sc.nextLine();
        while (q-- > 0) {
            String s = sc.nextLine();
            System.out.println(alternatingCharacters(s));
        }
          sc.close();
    }
}
```

OUTPUT:



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
5
AAAA
BBBBB
ABABABAB
BABABA
AAABBB
```

Your Output (stdout)

```
3
4
0
0
```

RESULT:Thus the program for Alternating Characters was executed successfully.