**DSA Problems:**

**(15-11-2024)** **JANET SHINY V(22CZ021)**

**1)Remove Duplicates Sorted array :**

Given a sorted array arr. Return the size of the modified array which contains only distinct elements.

*Note:*

1. Don't use set or HashMap to solve the problem.

2. You must return the modified array size only where distinct elements are present and modify the original array such that all the distinct elements come at the beginning of the original array.

Examples :

Input: arr = [2, 2, 2, 2, 2]
Output: [2]
Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should return 1 after modifying the array, the driver code will print the modified array elements.

Input: arr = [1, 2, 4]
Output: [1, 2, 4]

Explation: As the array does not contain any duplicates so you should return 3.

**Program :**

```
class Duplicate {


   static int removeDuplicates(int[] arr) {

      int n = arr.length;
```

```
        if (n <= 1)
            return n;

        int idx = 1; // Start from the second element
        for (int i = 1; i < n; i++) {
            if (arr[i] != arr[i - 1]) {
                arr[idx++] = arr[i];
            }
        }
        return idx;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 2, 3, 4, 4, 4, 5, 5};
        int newSize = removeDuplicates(arr);

        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

**Output :**

**1 2 3 4 5**

**Time Complexity : O(n)**
**Space Complexity : O(1)**

**2) Stock Buy and Sell:**
The cost of stock on each day is given in an array A[] of size N. Find all the segments of days on which you buy and sell the stock such that the sum of difference between sell and buy prices is maximized. Each segment consists of indexes of two elements, first is index of day on which you buy stock and second is index of day on which you sell stock.
Note: Since there can be multiple solutions, the driver code will print 1 if your answer is correct, otherwise, it will return 0. In case there's no profit the driver code will print the string "No Profit" for a correct solution.

**Example 1:**
**Input:**
N = 7
A[] = {100,180,260,310,40,535,695}
Output:
1
**Explanation:**
One possible solution is (0 3) (4 6)
We can buy stock on day 0,
and sell it on 3rd day, which will
give us maximum profit. Now, we buy
stock on day 4 and sell it on day 6.

**Example 2:**

**Input:**
N = 5
A[] = {4,2,2,2,4}
Output:
1
**Explanation:**
There are multiple possible solutions.
one of them is (3 4)

We can buy stock on day 3,
and sell it on 4th day, which will
give us maximum profit.

**Your Task:**

The task is to complete the function stockBuySell() which takes an array of A[] and N as input parameters and finds the days of buying and selling stock. The function must return a 2D list of integers containing all the buy-sell pairs i.e. the first value of the pair will represent the day on which you buy the stock and the second value represent the day on which you sell that stock. If there is No Profit, return an empty list.

**Program :**

```
import java.util.Arrays;


class Buy{


    static int maximumProfit(int[] prices) {
        int res = 0;


        for (int i = 1; i < prices.length; i++) {
            if (prices[i] > prices[i - 1])
                res += prices[i] - prices[i - 1];
        }
```

```
        return res;
    }

    public static void main(String[] args) {
        int[] prices = { 100,200,13,24,2334,45656,23 };
        System.out.println(maximumProfit(prices));
    }
}
```

**Output** :

```
D:\>javac buy.java

D:\>java Buy
45743
```

Time Complexity : O(n)

Space Complexity : O(1)

**3) Coin Change (Count Ways) :**

Given an integer array coins[ ] representing different denominations of currency and an integer sum, find the number of ways you can make sum by using different combinations from coins[ ].

Note: Assume that you have an infinite supply of each type of coin. And you can use any coin as many times as you want.

Answers are guaranteed to fit into a 32-bit integer.

Examples:

Input: coins[] = [1, 2, 3], sum = 4

Output: 4

Explanation: Four Possible ways are: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3].

Input: coins[] = [2, 5, 3, 6], sum = 10

Output: 5

Explanation: Five Possible ways are: [2, 2, 2, 2, 2], [2, 2, 3, 3], [2, 2, 6], [2, 3, 5] and [5, 5].

Input: coins[] = [5, 10], sum = 3

Output: 0

Explanation: Since all coin denominations are greater than sum, no combination can make the target sum.

**Program :**

```
import java.util.Arrays;

class CoinChange {

    static long count(int coins[], int n, int sum)

{

    int a[] = new int[sum + 1];
```

```
    a[0] = 1;

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

        for (int j = coins[i]; j <= sum; j++)

            a[j] += a[j - coins[i]];



    return a[sum];

}



    public static void main(String args[])

    {

        int coins[] = {1,2,3,4,4,2,6 };

        int n = coins.length;

        int sum = 5;

        System.out.println(count(coins, n, sum));

    }

}
```

**Output :**

```
D:\>javac CoinChange.java

D:\>java CoinChange
11
```

**Time Complexity : O(n)**

**Space Complexity : O(1)**

**4) First and Last Occurence :**

Given a sorted array arr with possibly some duplicates, the task is to find the first and last occurrences of an element x in the given array.
Note: If the number x is not found in the array then return both the indices as -1.

Examples:

Input: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Output: [2, 5]

Explanation: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

Input: arr[] = [1, 3, 5, 5, 5, 5, 7, 123, 125], x = 7

Output: [6, 6]

Explanation: First and last occurrence of 7 is at index 6

Input: arr[] = [1, 2, 3], x = 4

Output: [-1, -1]

Explanation: No occurrence of 4 in the array, so, output is [-1, -1]

**Program** :

Solution:

```
import java.util.*;

public class tUf {

    public static int[] firstAndLastPosition(ArrayList<Integer> arr, int n, int k) {

        int first = -1, last = -1;

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

            if (arr.get(i) == k) {

                if (first == -1) first = i;

                last = i;

            }

        }

        return new int[] {first, last};
```

```
    }

    public static void main(String[] args) {

        ArrayList<Integer> arr = new ArrayList<>(Arrays.asList(new Integer[] {2,
4, 6, 8, 8, 8, 11, 13}));

        int n = 8, k = 8;

        int[] ans = firstAndLastPosition(arr, n, k);

        System.out.println("The first and last positions are: "

                        + ans[0] + " " + ans[1]);

    }

}
```

**Output :**

**The first and last positions are: 3 5**

**Time Complexity : O(logn)**

**Space Complexity : O1(1)**

**5)First Repeating Element :**

Given an array arr[], find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.

Note:- The position you return should be according to 1-based indexing.

Examples:

Input: arr[] = [1, 5, 3, 4, 3, 5, 6]

Output: 2

Explanation: 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.

Input: arr[] = [1, 2, 3, 4]

Output: -1

Explanation: All elements appear only once so answer is -1.

Constraints:

1 <= arr.size <= 106

0 <= arr[i]<= 106

**Program :**

```
class FR {
    public static int firstRepeated(int[] arr) {
        HashMap<Integer,Integer> m=new HashMap<>();
        for(int i=0;i<arr.length;i++){
            m.put(arr[i],m.getOrDefault(arr[i],0)+1);
        }
        for(int i=0;i<arr.length;i++){
            if(m.get(arr[i])>1) {return i+1;}
        }
        return -1;
```

```
    }
    public static void main(String args[]){
     int[] a={10,20,10,30,50};
     System.out.println(firstRepeated(a));

}
}
```

**Output :**

**10**

**Time Complexity : O(n)**

**Space Complexity : O(n)**

**6)Find Transition Point**

Given a sorted array, arr[] containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed. If arr does not have any 1, return -1. If array does not have any 0, return 0.

Examples:

Input: arr[] = [0, 0, 0, 1, 1]

Output: 3

Explanation: index 3 is the transition point where 1 begins.

Input: arr[] = [0, 0, 0, 0]

Output: -1

Explanation: Since, there is no "1", the answer is -1.

Input: arr[] = [1, 1, 1]

Output: 0

Explanation: There are no 0s in the array, so the transition point is 0, indicating that the first index (which contains 1) is also the first position of the array.

Input: arr[] = [0, 1, 1]

Output: 1

Explanation: Index 1 is the transition point where 1 starts, and before it, only 0 was observed.

**Program :**

```
class Tp {
    static int transitionPoint(int arr[]) {
        for(int i=0;i<arr.length;i++){
            if(arr[i]==1){ return i;}
        }
        return -1;
    }
   public static void main(String args[]){
        int[] a={0,0,1,1,1,0};
        System.out.println(transitionPoint(a));
}
}
```

**Output :**

**2**

**Time Complexity : O(logn)**

**Space Complexity : O(1)**

**7)Maximum Index**

Given an array arr of positive integers. The task is to return the maximum of j - i subjected to the constraint of arr[i] ≤ arr[j] and i ≤ j.

Examples:

Input: arr[] = [1, 10]

Output: 1

Explanation: arr[0] ≤ arr[1] so (j-i) is 1-0 = 1.

Input: arr[] = [34, 8, 10, 3, 2, 80, 30, 33, 1]

Output: 6

Explanation: In the given array arr[1] < arr[7] satisfying the required condition(arr[i] ≤ arr[j]) thus giving the maximum difference of j - i which is 6(7-1).

Expected Time Complexity: O(n)

Expected Auxiliary Space: O(n)

**Program :**

```
public class MaxDifferenceOptimized {

    public static int maxDifference(int[] arr) {

        int n = arr.length;
```

```
    int maxDiff = -1;

    // Two pointers approach
    int i = 0, j = 0;

    while (i < n && j < n) {
        if (arr[i] < arr[j]) {
            maxDiff = Math.max(maxDiff, j - i);
            j++;  // Move the second pointer forward
        } else {
            i++;  // Move the first pointer forward
        }
    }

    return maxDiff;
}
```

```
    public static void main(String[] args) {
        int[] arr1 = {1, 10};
        System.out.println("Max Difference for arr1: " + maxDifference(arr1));

        int[] arr2 = {34, 8, 10, 3, 2, 80, 30, 33, 1};
        System.out.println("Max Difference for arr2: " + maxDifference(arr2));
    }
}
```

**Output :**

**Max Difference for arr1: 1**

**Max Difference for arr2:6**

**Time Complexity: O(n)**

**Space Complexity : O(n)**

**8)Wave Array**

Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....

If there are multiple solutions, find the lexicographically smallest one.

Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array.

Examples:

Input: arr[] = [1, 2, 3, 4, 5]

Output: [2, 1, 4, 3, 5]

Explanation: Array elements after sorting it in the waveform are 2, 1, 4, 3, 5.

Input: arr[] = [2, 4, 7, 8, 9, 10]

Output: [4, 2, 8, 7, 10, 9]

Explanation: Array elements after sorting it in the waveform are 4, 2, 8, 7, 10, 9.

Input: arr[] = [1]

Output: [1]

**Program :**

```
public class SortWave
{
    void swap(int arr[], int a, int b)
    {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }

    void sortInWave(int arr[], int n)
    {
```

```
        for(int i = 0; i < n; i+=2){
            if(i > 0 && arr[i - 1] > arr[i])
              swap(arr, i, i-1);
              if(i < n-1 && arr[i + 1] > arr[i])
              swap(arr, i, i+1);
        }
    }

    public static void main(String args[])
    {
        SortWave ob = new SortWave();
        int arr[] = {10, 90, 49, 2, 1, 5, 23};
        int n = arr.length;
        ob.sortInWave(arr, n);
        for (int i : arr)
            System.out.print(i+" ");
    }
};
```

**Output :**

**90 10 49 1 5 2 23**

**Time Complexity : O(n)**

**Space Complexity : O(1)**