DSA Practice 5 (14-11-2024)

Name: Atchayaa T

Department: B.Tech CSBS

Register No.: 22CB004

1. **Stock Buy and Sell – Max one Transaction Allowed**

Given an array prices[] of length N, representing the prices of the stocks on different days, the task is to find the maximum profit possible by buying and selling the stocks on different days when at most one transaction is allowed. Here one transaction means 1 buy + 1 Sell.

Note: Stock must be bought before being sold.

Input: *prices[] = {7, 10, 1, 3, 6, 9, 2}*
Output: *8*
Explanation: *Buy for price 1 and sell for price 9.*
Input: *prices[] = {7, 6, 4, 3, 1}*
Output: *0*
Explanation: *Since the array is sorted in decreasing order, 0 profit can be made without making any transaction.*

Input: *prices[] = {1, 3, 6, 9, 11}*
Output: *10*
Explanation: *Since the array is sorted in increasing order, we can make maximum profit by buying at price[0] and selling at price[n-1]*

Code:

```java
public class Stock_Buy_And_Sell {
    static int solution(int[] p) {
        int m = p[0];
        int res = 0;
        for (int i = 1; i < p.length; i++) {
            m = Math.min(m, p[i]);

            res = Math.max(res, p[i] - m);
        }
        return res;
    }

    public static void main(String[] args) {
        int[] p = {7, 10, 1, 3, 6, 9, 2};
        System.out.println(solution(p));
    }
}
```

Output: 8
Time Complexity: O(n)

2. **Coin Change – Count Ways to Make Sum**

   Given an integer array of coins[ ] of size N representing different types of denominations and an integer sum, the task is to count all combinations of coins to make a given value sum.

   Note: Assume that you have an infinite supply of each type of coin.

   Input: *sum = 4, coins[] = {1,2,3}*
   Output: *4*
   Explanation: *there are four solutions: {1, 1, 1, 1}, {1, 1, 2}, {2, 2} and {1, 3}*
   Input: *sum = 10, coins[] = {2, 5, 3, 6}*
   Output: *5*
   Explanation: *There are five solutions:*
   {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}
   Input: *sum = 10, coins[] = {10}*
   Output: *1*
   Explanation: *The only is to pick 1 coin of value 10.*
   Input: *sum = 5, coins[] = {4}*
   Output: *0*

   Code:

```java
import java.util.*;

public class CoinChangeWays {
    static int count(List<Integer> coins, int n, int sum) {
        int[][] dp = new int[n + 1][sum + 1];
        dp[0][0] = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j <= sum; j++) {
                dp[i][j] += dp[i - 1][j];

                if ((j - coins.get(i - 1)) >= 0) {
                    dp[i][j] += dp[i][j - coins.get(i - 1)];
                }
            }
        }
        return dp[n][sum];
    }

    public static void main(String[] args) {
        List<Integer> coins = Arrays.asList(1, 2, 3);
        int n = 3;
        int sum = 5;
        System.out.println(count(coins, n, sum));
    }
}
```

   Output: 5
   Time Complexity: O(N*Sum)

3. **Find first and last positions of an element in a sorted array**

Given a sorted array arr[] with possibly duplicate elements, the task is to find indexes of the first and last occurrences of an element x in the given array.

Input : *arr[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}, x = 5*
Output *: First Occurrence = 2*
Last Occurrence = 5
Input : *arr[] = {1, 3, 5, 5, 5, 5, 7, 123, 125 }, x = 7*
Output : *First Occurrence = 6*
Last Occurrence = 6

Code:

```
import java.util.*;
class FirstAndLAstPosition {
    static int search(int[] nums, int target,
                      boolean findStartIndex)
    {
        int ans = -1;
        int start = 0;
        int end = nums.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;

            if (target < nums[mid]) {
                end = mid - 1;
            }
            else if (target > nums[mid]) {
                start = mid + 1;
            }
            else {
                ans = mid;
                if (findStartIndex) {
                    end = mid - 1;
                }
                else {
                    start = mid + 1;
                }
            }
        }

        return ans;
    }

    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };
        int n = arr.length;
        int x = 2;
        int[] ans = { -1, -1 };
        ans[0] = search(arr,x, true);
        if (ans[0] != -1) {
            ans[1] = search(arr, x, false);
        }
        System.out.println("First :" + ans[0]);
```

```
        System.out.println("Last :" + ans[1]);
    }
}
```

Output:
First :1
Last :4
Time Complexity: O(log n)

4. **Find the transition point in a binary array**

Given a sorted array containing only numbers 0 and 1, the task is to find the transition point efficiently. The transition point is the point where "0" ends and "1" begins.

Input: 0 0 0 1 1

Output: 3

Explanation: Index of first 1 is 3

Input: 0 0 0 0 1 1 1 1

Output: 4

Explanation: Index of first 1 is 4

Code:

```
import java.util.*;

class TransitionPoint
{
static int solution(int arr[], int n)
{
    for(int i = 0; i < n ; i++)
    if(arr[i] == 1)
        return i;
    return -1;
}
public static void main (String[] args)
{
    int arr[] = {0, 0, 0, 0, 1, 1};
    int n = arr.length;

    int point = solution(arr, n);

    if (point >= 0)
        System.out.print(point);
    else
        System.out.print("There is no transition point");
}
}
```

Output: 4
Time Complexity: O(log n)

5. **Find the first repeating element in an array of integers**
   Given an array of integers arr[], The task is to find the index of first repeating element in it i.e. the element that occurs more than once and whose index of the first occurrence is the smallest.

   Input: *arr[] = {10, 5, 3, 4, 3, 5, 6}*
   Output: *5*
   Explanation: *5 is the first element that repeats*
   Input: *arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}*
   Output: *6*
   Explanation: *6 is the first element that repeats*

   Code:

```
import java.util.*;

class FirstRepeatingElement {
    static void solution(int arr[])
    {
        int min = -1;
        HashSet<Integer> h = new HashSet<>();
        for (int i = arr.length - 1; i >= 0; i--) {
            if (h.contains(arr[i]))
                min = i;

            else
                h.add(arr[i]);
        }
        if (min != -1)
            System.out.println("1st repeating element: "+ arr[min]);
        else
            System.out.println("No repeating elements");}
    public static void main(String[] args)
        throws java.lang.Exception
    {
        int arr[] ={6, 10, 5, 4, 9, 120, 4, 6, 10};
        solution(arr);
    }
}
```

   Output: 1st repeating element: 6
   Time Complexity: O(n)

6. **Remove duplicates from sorted array**
   Given a sorted array arr[] of size N, the task is to remove the duplicate elements from the array. We need keep order of the remaining distinct elements as it was in the original array.

Input: *arr[] = {2, 2, 2, 2, 2}*
Output: *arr[] = {2}*
Explanation: *All the elements are 2, So only keep one instance of 2.*
Input: *arr[] = {1, 2, 2, 3, 4, 4, 4, 5, 5}*
Output: *arr[] = {1, 2, 3, 4, 5}*

Input*: arr[] = {1, 2, 3}*
Output *: arr[] = {1, 2, 3}*
Explanation *: No change as all elements are distinct*

Code:

```
class RemoveDuplicates {

    static int solution(int[] arr) {
        int n = arr.length;
        if (n <= 1)
            return n;

        int idx = 1;
        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 = solution(arr);

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

Output: 1 2 3 4 5
Time Complexity: O(n)

7. **Maximum Index**
   Given an array arr[] of N positive integers. The task is to find the maximum of j – i subjected to the constraint of arr[i] <= arr[j].

   Input: {34, 8, 10, 3, 2, 80, 30, 33, 1}
   Output: 6 (j = 7, i = 1)

   Input: {9, 2, 3, 4, 5, 6, 7, 8, 18, 0}
   Output: 8 (j = 8, i = 0)

   Input: {1, 2, 3, 4, 5, 6}
   Output: 5 (j = 5, i = 0)

Input: {6, 5, 4, 3, 2, 1}
Output: 0

Code:

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

class MaximumIndex{
static int solution(ArrayList<Integer> arr, int n)
{
    Map<Integer,
    ArrayList<Integer>> hashmap = new HashMap<Integer, ArrayList<Integer>>();
    for(int i = 0; i < n; i++)
    {
        if(hashmap.containsKey(arr.get(i)))
        {
            hashmap.get(arr.get(i)).add(i);
        }
        else
        {
            hashmap.put(arr.get(i), new ArrayList<Integer>());
            hashmap.get(arr.get(i)).add(i);
        }
    }

    Collections.sort(arr);
    int maxDiff = Integer.MIN_VALUE;
    int temp = n;
    for(int i = 0; i < n; i++)
    {
        if (temp > hashmap.get(arr.get(i)).get(0))
        {
            temp = hashmap.get(arr.get(i)).get(0);
        }
        maxDiff = Math.max(maxDiff,
        hashmap.get(arr.get(i)).get(
            hashmap.get(arr.get(i)).size() - 1) - temp);
    }
    return maxDiff;
}
public static void main(String[] args)
{
    int n = 9;
    ArrayList<Integer> arr = new ArrayList<Integer>(
        Arrays.asList(34, 8, 10, 3, 2, 80, 30, 33, 1));
    int ans = solution(arr, n);

    System.out.println(ans);
}
}
```

Output: 6
Time Complexity: O(n logn)

8. **Sort an array in wave form**
   Given an unsorted array of integers, sort the array into a wave array. An array arr[0..n-1] is sorted in wave form if:
   arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= …..

   Input: *arr[] = {10, 5, 6, 3, 2, 20, 100, 80}*
   Output: *arr[] = {10, 5, 6, 2, 20, 3, 100, 80}*
   Explanation:
   here you can see {10, 5, 6, 2, 20, 3, 100, 80} first element is larger than the second and the same thing is repeated again and again. large element – small element-large element -small element and so on .it can be small element-larger element – small element-large element -small element too. all you need to maintain is the up-down fashion which represents a wave. there can be multiple answers.
   Input: *arr[] = {20, 10, 8, 6, 4, 2}*
   Output: *arr[] = {20, 8, 10, 4, 6, 2}*

   Code:

```java
public class SortInWaveForm {
    void swap(int arr[], int a, int b) {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }
    void solution(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[]) {
        int arr[] = {10, 90, 49, 2, 1, 5, 23};
        int n = arr.length;
        SortInWaveForm waveForm = new SortInWaveForm();
        waveForm.solution(arr, n);

        for (int i : arr)
            System.out.print(i + " ");
    }
}
```

Output: 90 10 49 1 5 2 23
Time Complexity: O(n)