**SANJAY S,22IT093  
DSA PRACTICE SET-5**

**1.Stock buy and sell**

import java.util.ArrayList;

class Solution {

ArrayList<ArrayList<Integer>> stockBuySell(int A[], int n) {

int minima = A[0];

int maxima = A[0];

int i = 0;

ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

while (i < n - 1) {

while (i < n - 1 && A[i] >= A[i + 1]) {

i++;

}

if (i == n - 1) break;

minima = i;

while (i < n - 1 && A[i] <= A[i + 1]) {

i++;

}

maxima = i;

ArrayList<Integer> crr = new ArrayList<>();

crr.add(minima);

crr.add(maxima);

ans.add(crr);

}

return ans;

}

public static void main(String[] args) {

Solution obj = new Solution();

int A[] = {100, 180, 260, 310, 40, 535, 695};

int n = A.length;

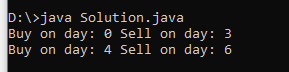
ArrayList<ArrayList<Integer>> result = obj.stockBuySell(A, n);

for (ArrayList<Integer> pair : result) {

System.out.println("Buy on day: " + pair.get(0) + " Sell on day: " + pair.get(1));

}

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**2. Coin Change (Count Ways)**

import java.util.Arrays;

class Solution {

public int fun(int[] coins, int ind, int sum, int[][] dp) {

if (sum == 0) return 1;

if (ind < 0) return 0;

if (dp[ind][sum] != -1) return dp[ind][sum];

int take = 0;

if (coins[ind] <= sum) {

take = fun(coins, ind, sum - coins[ind], dp);

}

int notTake = fun(coins, ind - 1, sum, dp);

return dp[ind][sum] = take + notTake;

}

public int count(int coins[], int sum) {

int n = coins.length;

int[][] dp = new int[n][sum + 1];

for (int[] row : dp) {

Arrays.fill(row, -1);

}

return fun(coins, n - 1, sum, dp);

}

public static void main(String[] args) {

Solution obj = new Solution();

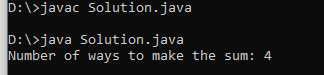
int coins[] = {1, 2, 3};

int sum = 4;

System.out.println("Number of ways to make the sum: " + obj.count(coins, sum));

}

}



Time Complexity: O(n \* sum)

Space Complexity: O(n \* sum)

**3.First and Last Occurrences**

import java.util.ArrayList;

class GFG {

ArrayList<Integer> find(int arr[], int x) {

int n = arr.length;

int min = Integer.MAX\_VALUE;

int max = Integer.MIN\_VALUE;

ArrayList<Integer> ls = new ArrayList<>();

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

if (arr[i] == x) {

min = Math.min(min, i);

max = Math.max(max, i);

}

}

if (min == Integer.MAX\_VALUE) {

min = -1;

max = -1;

}

ls.add(min);

ls.add(max);

return ls;

}

public static void main(String[] args) {

GFG obj = new GFG();

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

int x = 2;

ArrayList<Integer> result = obj.find(arr, x);

System.out.println(result);

}

}

****

Time Complexity: O(n)

Space Complexity: O(1)

**4. Find Transition Point**

class Solution {

int transitionPoint(int arr[]) {

int n = arr.length;

int ans = -1;

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

if (arr[i] == 1) {

ans = i;

break;

}

}

return ans;

}

public static void main(String[] args) {

Solution obj = new Solution();

int arr[] = {0, 0, 1, 1, 1};

System.out.println(obj.transitionPoint(arr));

}

}



Time Complexity: O(n)

Space Complexity: O(1)

**5. First Repeating Element**

import java.util.Arrays;

class Solution {

public static int firstRepeated(int[] arr) {

int n = arr.length;

int[] hash = new int[(int) Math.pow(10, 6) + 1];

Arrays.fill(hash, -1);

int min = Integer.MAX\_VALUE;

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

if (hash[arr[i]] != -1) {

min = Math.min(min, hash[arr[i]]);

} else {

hash[arr[i]] = i;

}

}

return min == Integer.MAX\_VALUE ? -1 : min + 1;

}

public static void main(String[] args) {

int arr[] = {1, 5, 3, 4, 3, 5, 6};

System.out.println(firstRepeated(arr));

}

}

****

Time Complexity: O(n)

Space Complexity: O(m)

**6.Wave Array**

class Solution {

public static void convertToWave(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i += 2) {

int temp = arr[i];

arr[i] = arr[i + 1];

arr[i + 1] = temp;

}

}

public static void main(String[] args) {

int arr[] = {1, 2, 3, 4, 5, 6};

convertToWave(arr);

for (int i : arr) {

System.out.print(i + " ");

}

}

}****

Time Complexity: O(n)

Space Complexity: O(1)