**ASSIGNMENT DSA-2**

**Que(1)** Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2),..., (an, bn) such that the sum of min(ai, bi) for all i is maximized. Return the maximized sum.

**Example :** Input: nums = [1,4,3,2] Output: 4

**Ans.**

class Solution {

public int arrayPairSum(int[] nums) {

Arrays.sort(nums);

int len = nums.length;

int result = 0;

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

result += nums[i];

}

return result;

}

}

**Que(2)** Alice has n candies, where the ith candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor. The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice. Given the integer array candyType of length n, return the maximum number of different types of candies she can eat if she only eats n / 2 of them.

**Example :** Input: candyType = [1,1,2,2,3,3] Output: 3

**Ans.**

class Solution {

public int distributeCandies(int[] candyType) {

Set<Integer> uniqueCandiesSet = new HashSet<>();

for (int candy: candyType) {

uniqueCandiesSet.add(candy);

}

return Math.min(uniqueCandiesSet.size(), candyType.length / 2);

}

}

**Que(3)** We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1. Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences. A subsequence of an array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

**Example**: Input: nums = [1,3,2,2,5,2,3,7] Output: 5

**Ans.**

public class Solution {

public int findLHS(int[] nums) {

HashMap < Integer, Integer > map = new HashMap < > ();

int res = 0;

for (int num: nums) {

map.put(num, map.getOrDefault(num, 0) + 1);

}

for (int key: map.keySet()) {

if (map.containsKey(key + 1))

res = Math.max(res, map.get(key) + map.get(key + 1));

}

return res;

}

}

**Que(4)**  You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots. Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

**Example**: Input: flowerbed = [1,0,0,0,1], n = 1 Output: true

**Ans.**

class Solution {

public:

bool canPlaceFlowers(vector<int>& flowerbed, int n) {

int count = 0;

for (int i = 0; i < flowerbed.size(); i++) {

// Check if the current plot is empty.

if (flowerbed[i] == 0) {

// Check if the left and right plots are empty.

bool emptyLeftPlot = (i == 0) || (flowerbed[i - 1] == 0);

bool emptyRightPlot = (i == flowerbed.size() - 1) || (flowerbed[i + 1] == 0);

// If both plots are empty, we can plant a flower here.

if (emptyLeftPlot && emptyRightPlot) {

flowerbed[i] = 1;

count++;

}

}

}

return count >= n;

}

}

**Que(5)** Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

**Example** : Input: nums = [1,2,3] Output: 6

**Ans**.

public class Solution {

public int maximumProduct(int[] nums) {

int min1 = Integer.MAX\_VALUE, min2 = Integer.MAX\_VALUE;

int max1 = Integer.MIN\_VALUE, max2 = Integer.MIN\_VALUE, max3 =

Integer.MIN\_VALUE;

for (int n: nums) {

if (n <= min1) {

min2 = min1;

min1 = n;

} else if (n <= min2) { // n lies between min1 and min2

min2 = n;

}

if (n >= max1) { // n is greater than max1, max2 and max3

max3 = max2;

max2 = max1;

max1 = n;

} else if (n >= max2) { // n lies between max1 and max2

max3 = max2;

max2 = n;

} else if (n >= max3) { // n lies between max2 and max3

max3 = n;

}

}

return Math.max(min1 \* min2 \* max1, max1 \* max2 \* max3);

}

}

**Que(6)** Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1. You must write an algorithm with O(log n) runtime complexity.

Input: nums = [-1,0,3,5,9,12], target = 9 Output: 4

**Ans** .

class Solution {

public int search(int[] nums, int target) {

int left = 0;

int right = nums.length - 1;

while (left <= right){

int mid = left + (right - left) / 2;

if (nums[mid] < target) {

left = mid + 1;

}

else if (nums[mid] > target){

right = mid - 1; // easy to make mistake

}

else if (nums[mid] == target){

return mid;

}

}

return -1;

}

}

**Que(7)** An array is monotonic if it is either monotone increasing or monotone decreasing. An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j]. An array nums is monotone decreasing if for all i <= j, nums[i] >= nums[j]. Given an integer array nums, return true if the given array is monotonic, or false otherwise.

**Example** : Input: nums = [1,2,2,3] Output: true

**Ans.**

class Solution {

public boolean isMonotonic(int[] A) {

boolean increasing = true;

boolean decreasing = true;

for (int i = 0; i < A.length - 1; ++i) {

if (A[i] > A[i+1])

increasing = false;

if (A[i] < A[i+1])

decreasing = false;

}

return increasing || decreasing;

}

}

**Que(8)** You are given an integer array nums and an integer k. In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i. The score of nums is the difference between the maximum and minimum elements in nums. Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

**Example**: Input: nums = [1], k = 0 Output: 0

**Ans.**

class Solution {

public:

int smallestRangeI(vector<int>& nums, int k) {

int n = nums.size();

int maxx = nums[0], minn = nums[0];

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

maxx = max(maxx, nums[i]);

minn = min(minn, nums[i]);

}

int diff = maxx - minn;

if (diff - k <= k) return 0;

return diff-k-k;

}

}