Q1.import java.util.Arrays;

public class ArrayPairSum {

public int arrayPairSum(int[] nums) {

// Sort the array in ascending order

Arrays.sort(nums);

int sum = 0;

// Pair adjacent elements and sum up the minimum value in each pair

for (int i = 0; i < nums.length; i += 2) {

sum += nums[i];

}

return sum;

}

public static void main(String[] args) {

ArrayPairSum solution = new ArrayPairSum();

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

int maxSum = solution.arrayPairSum(nums);

System.out.println("Maximized sum: " + maxSum);

}

}

Q2.import java.util.HashSet;

public class MaxCandies {

public int maxCandies(int[] candyType) {

int maxCandies = candyType.length / 2;

HashSet<Integer> uniqueCandies = new HashSet<>();

for (int candy : candyType) {

uniqueCandies.add(candy);

// Stop adding candies once Alice reaches the maximum limit

if (uniqueCandies.size() == maxCandies) {

break;

}

}

return uniqueCandies.size();

}

public static void main(String[] args) {

MaxCandies solution = new MaxCandies();

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

int maxTypes = solution.maxCandies(candyType);

System.out.println("Max number of different types of candies: " + maxTypes);

}

}

Q3.import java.util.HashMap;

public class LongestHarmoniousSubsequence {

public int findLHS(int[] nums) {

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

// Count the frequency of each number in the array

for (int num : nums) {

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

}

int longestSubsequenceLength = 0;

// Iterate over the keys of the frequency map

for (int num : frequencyMap.keySet()) {

if (frequencyMap.containsKey(num + 1)) {

int currentLength = frequencyMap.get(num) + frequencyMap.get(num + 1);

longestSubsequenceLength = Math.max(longestSubsequenceLength, currentLength);

}

}

return longestSubsequenceLength;

}

public static void main(String[] args) {

LongestHarmoniousSubsequence solution = new LongestHarmoniousSubsequence();

int[] nums = {1, 3, 2, 2, 5, 2, 3, 7};

int longestLength = solution.findLHS(nums);

System.out.println("Longest harmonious subsequence length: " + longestLength);

}

}

Q4.public class CanPlaceFlowers {

public boolean canPlaceFlowers(int[] flowerbed, int n) {

int count = 0;

int length = flowerbed.length;

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

if (flowerbed[i] == 0) {

// Check if the current plot and its adjacent plots are empty

boolean prevEmpty = (i == 0 || flowerbed[i - 1] == 0);

boolean nextEmpty = (i == length - 1 || flowerbed[i + 1] == 0);

if (prevEmpty && nextEmpty) {

flowerbed[i] = 1;

count++;

}

}

}

return count == n;

}

public static void main(String[] args) {

CanPlaceFlowers solution = new CanPlaceFlowers();

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

int n = 1;

boolean canPlace = solution.canPlaceFlowers(flowerbed, n);

System.out.println("Can place flowers: " + canPlace);

}

}

Q5.import java.util.Arrays;

public class MaximumProduct {

public int maximumProduct(int[] nums) {

Arrays.sort(nums);

int n = nums.length;

int product1 = nums[n - 1] \* nums[n - 2] \* nums[n - 3]; // Case 1: Product of three largest positive numbers

int product2 = nums[0] \* nums[1] \* nums[n - 1]; // Case 2: Product of two smallest (negative) numbers and the largest positive number

return Math.max(product1, product2);

}

public static void main(String[] args) {

MaximumProduct solution = new MaximumProduct();

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

int maxProduct = solution.maximumProduct(nums);

System.out.println("Maximum product: " + maxProduct);

}

}

Q6.public class BinarySearch {

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) {

return mid;

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

BinarySearch solution = new BinarySearch();

int[] nums = {-1, 0, 3, 5, 9, 12};

int target = 9;

int index = solution.search(nums, target);

System.out.println("Index of target: " + index);

}

}

Q7.public class MonotonicArray {

public boolean isMonotonic(int[] nums) {

boolean increasing = true;

boolean decreasing = true;

for (int i = 1; i < nums.length; i++) {

if (nums[i] < nums[i - 1]) {

increasing = false;

}

if (nums[i] > nums[i - 1]) {

decreasing = false;

}

}

return increasing || decreasing;

}

public static void main(String[] args) {

MonotonicArray solution = new MonotonicArray();

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

boolean isMonotonic = solution.isMonotonic(nums);

System.out.println("Is monotonic: " + isMonotonic);

}

}

Q8.public class MinScoreAfterOperation {

public int minScore(int[] nums, int k) {

int minScore = Integer.MAX\_VALUE;

int minValue = Integer.MAX\_VALUE;

int maxValue = Integer.MIN\_VALUE;

for (int num : nums) {

minValue = Math.min(minValue, num);

maxValue = Math.max(maxValue, num);

}

if (minValue == maxValue) {

return 0; // All elements in the array are the same, so the score is 0.

}

for (int num : nums) {

int maxPossible = Math.max(maxValue - k, num + k);

int minPossible = Math.min(minValue + k, num - k);

minScore = Math.min(minScore, maxPossible - minPossible);

}

return minScore;

}

public static void main(String[] args) {

MinScoreAfterOperation solution = new MinScoreAfterOperation();

int[] nums = {1, 3, 6};

int k = 3;

int minScore = solution.minScore(nums, k);

System.out.println("Minimum score: " + minScore);

}

}