**SANJAY\_M – CSE – DSA – PRACTICE – 1**

**Q1**. **Maximum Subarray Sum** – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3}

Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

CODE :

public class MaxSub {

public static int maxSubarraySum(int[] arr) {

int maxCurrent = arr[0];

int maxTotal = arr[0];

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

maxCurrent = Math.max(arr[i], maxCurrent + arr[i]);

maxTotal = Math.max(maxTotal, maxCurrent);

}

return maxTotal;

}

public static void main(String[] args) {

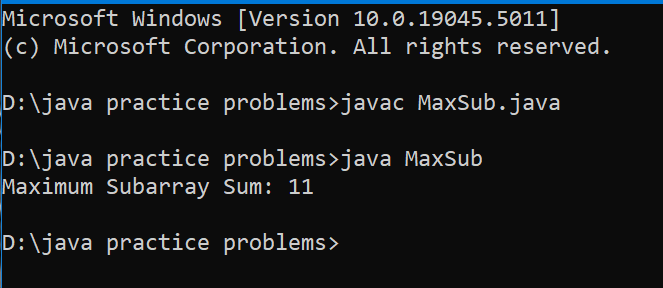
int[] arr = {2, 3, -8, 7, -1, 2, 3};

System.out.println("Maximum Subarray Sum: " + maxSubarraySum(arr));

}

}

OUTPUT :



Time Complexity : O(n)

**Q2**. **Maximum Product Subarray** Given an integer array, the task is to find the maximum product of any subarray.

Input: arr[] = {-2, 6, -3, -10, 0, 2}

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10) = 180

CODE :

class Solution {

public int maxProduct(int[] nums) {

int maxProduct = nums[0];

int maxCurrent = nums[0];

int minCurrent = nums[0];

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

if(nums[i]<0){

int temp = maxCurrent;

maxCurrent = minCurrent;

minCurrent = temp;

}

maxCurrent = Math.max(nums[i], maxCurrent \* nums[i]);

minCurrent = Math.max(nums[i], minCurrent \* nums[i]);

maxProduct = Math.max(maxProduct, maxCurrent);

}

return maxProduct;

}

public static void main(String[] args){

Solution solution = new Solution();

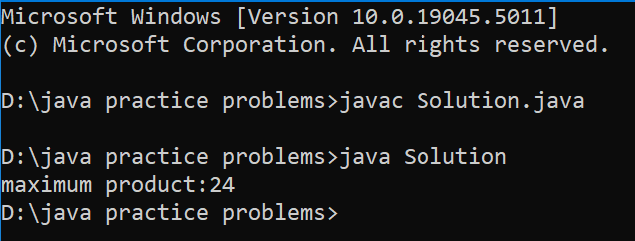
int[] nums = {-1,4,6,-2,8,1};

System.out.print("maximum product:" + solution.maxProduct(nums));

}

}

OUTPUT :



Time Complexity : O(n)

**Q3**. **Search in a sorted and rotated Array**

Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0

Output : 4

CODE :

public class Solution {

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

int left = 0, right = nums.length - 1;

while (left <= right) {

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

if (nums[mid] == target) {

return mid;

}

if (nums[left] <= nums[mid]) {

if (nums[left] <= target && target < nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

}

else {

if (nums[mid] < target && target <= nums[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

Solution solution = new Solution();

int[] nums = {4, 5, 6, 7, 0, 1, 2};

int target = 0;

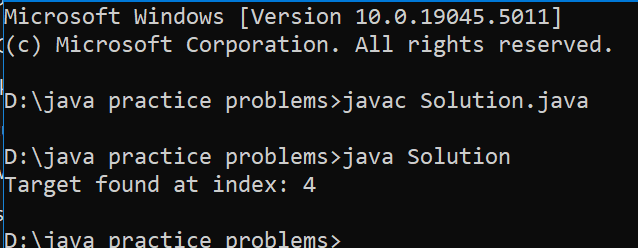
int result = solution.search(nums, target);

System.out.println("Target found at index: " + result);

}

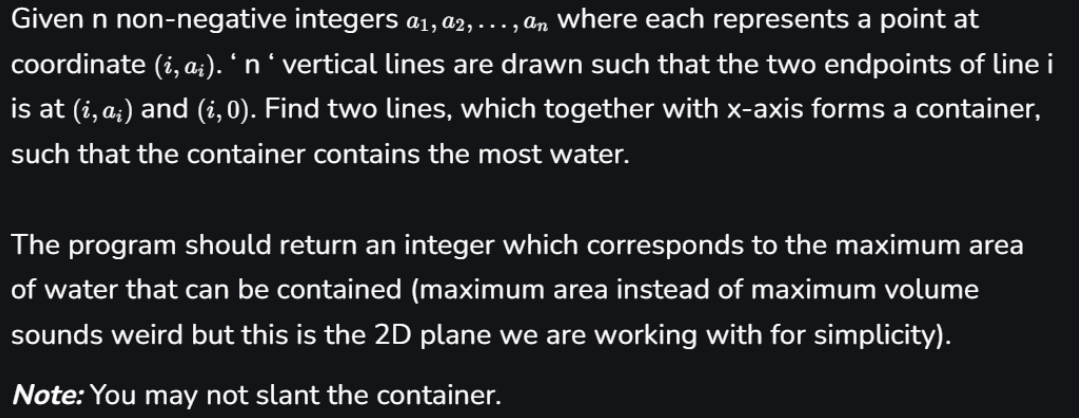
}

OUTPUT :



Time Complexity : O(logn)

**Q4. Container with Most Water**

****

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

Output: 6

Explanation: 5 and 3 are distance 2 apart.

So the size of the base = 2.

Height of container = min(5, 3) = 3.

So total area = 3 \* 2 = 6

**CODE :**

public class Solution {

public int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int maxArea = 0;

while (left < right) {

int width = right - left;

int minHeight = Math.min(height[left], height[right]);

int area = width \* minHeight;

maxArea = Math.max(maxArea, area);

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

Solution solution = new Solution();

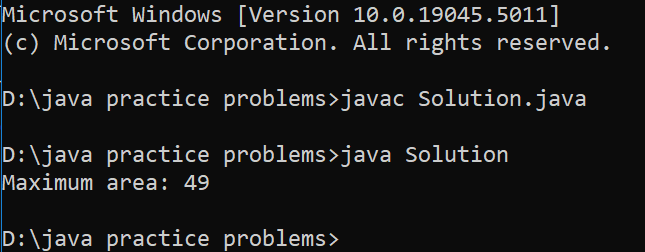
int[] height = {1, 8, 6, 2, 5, 4, 8, 3, 7};

System.out.println("Maximum area: " + solution.maxArea(height));

}

}

OUTPUT :



Time Complexity : O(n)

**Q5. Find the Factorial of a large number**

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

CODE :

import java.math.BigInteger;

public class Factorial {

public static void main(String[] args) {

int number = 100;

BigInteger factorial = factorial(number);

System.out.println(factorial);

}

public static BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

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

result = result.multiply(BigInteger.valueOf(i));

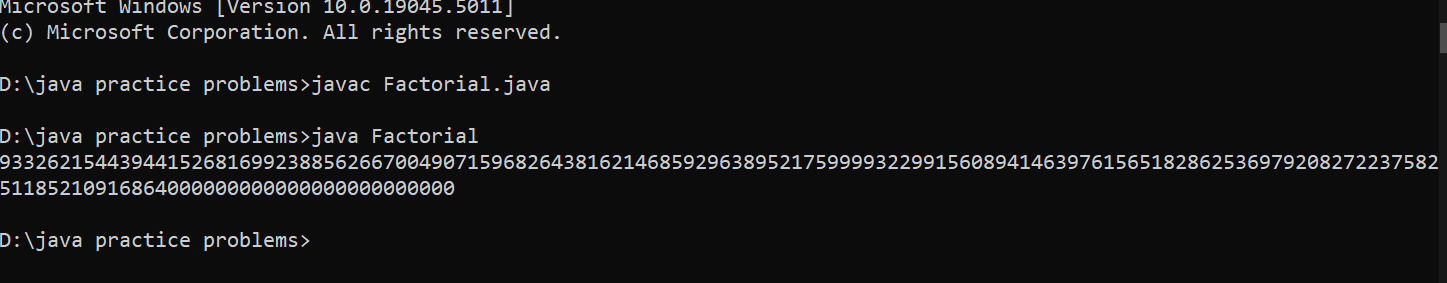
}

return result;

}

}

OUTPUT :



Time Complexity :

**Q6. Trapping Rainwater Problem** states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image

CODE :

public class TrappingRainWater {

public int trap(int[] height) {

if (height == null || height.length == 0) {

return 0;

}

int left = 0;

int right = height.length - 1;

int leftMax = 0;

int rightMax = 0;

int totalWater = 0;

while (left < right) {

if (height[left] < height[right]) {

if (height[left] >= leftMax) {

leftMax = height[left];

} else {

totalWater += leftMax - height[left];

}

left++;

} else {

if (height[right] >= rightMax) {

rightMax = height[right];

} else {

totalWater += rightMax - height[right];

}

right--;

}

}

return totalWater;

}

public static void main(String[] args) {

TrappingRainWater solution = new TrappingRainWater();

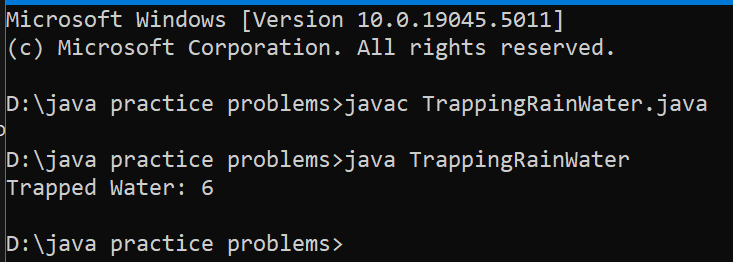
int[] height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};

System.out.println("Trapped Water: " + solution.trap(height));

}

}

OUTPUT :



Time Complexity : O(n)

**Q7. Chocolate Distribution Problem**

Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized. Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

CODE :

import java.util.Arrays;

class ChocolateDistribution {

public static int findMinDifference(int[] packets, int students) {

int totalPackets = packets.length;

if (students == 0 || totalPackets == 0 || students > totalPackets) {

return 0;

}

Arrays.sort(packets);

int minDifference = Integer.MAX\_VALUE;

for (int i = 0; i <= totalPackets - students; i++) {

int currentDifference = packets[i + students - 1] - packets[i];

minDifference = Math.min(minDifference, currentDifference);

}

return minDifference;

}

public static void main(String[] args) {

int[] packets = {7, 3, 2, 4, 9, 12, 56};

int students = 3;

System.out.println("Minimum difference: " + findMinDifference(packets, students));

}

}

OUTPUT :