**Experiment 10.1**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 6th **Date of Performance:** 05/05/2023

**Subject Name:** CC-2 Lab **Subject Code:** 20CSP-351

1. **Aim/Overview of the practical:**

Scramble string

We can scramble a string s to get a string t using the following algorithm:

1. If the length of the string is 1, stop.
2. If the length of the string is > 1, do the following:

* Split the string into two non-empty substrings at a random index, i.e., if the string is s, divide it to x and y where s = x + y.
* **Randomly** decide to swap the two substrings or to keep them in the same order. i.e., after this step, s may become s = x + y or s = y + x.
* Apply step 1 recursively on each of the two substrings x and y.

Given two strings s1 and s2 of **the same length**, return true if s2 is a scrambled string of s1, otherwise, return false.

<https://leetcode.com/problems/scramble-string/>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of Dynamic Programming.

Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as optimal substructure property.

1. **Code:**

class Solution {

 public:

  bool isScramble(string s1, string s2) {

    if (s1 == s2)

      return true;

    const string hashKey = s1 + '+' + s2;

    if (const auto it = memo.find(hashKey); it != cend(memo))

      return it->second;

    vector<int> count(128);

    for (int i = 0; i < s1.length(); ++i) {

      ++count[s1[i]];

      --count[s2[i]];

    }

    if (any\_of(begin(count), end(count), [](int c) { return c != 0; }))

      return memo[hashKey] = false;

    for (int i = 1; i < s1.length(); ++i) {

      if (isScramble(s1.substr(0, i), s2.substr(0, i)) &&

          isScramble(s1.substr(i), s2.substr(i)))

        return memo[hashKey] = true;

      if (isScramble(s1.substr(0, i), s2.substr(s2.length() - i)) &&

          isScramble(s1.substr(i), s2.substr(0, s2.length() - i)))

        return memo[hashKey] = true;

    }

    return memo[hashKey] = false;

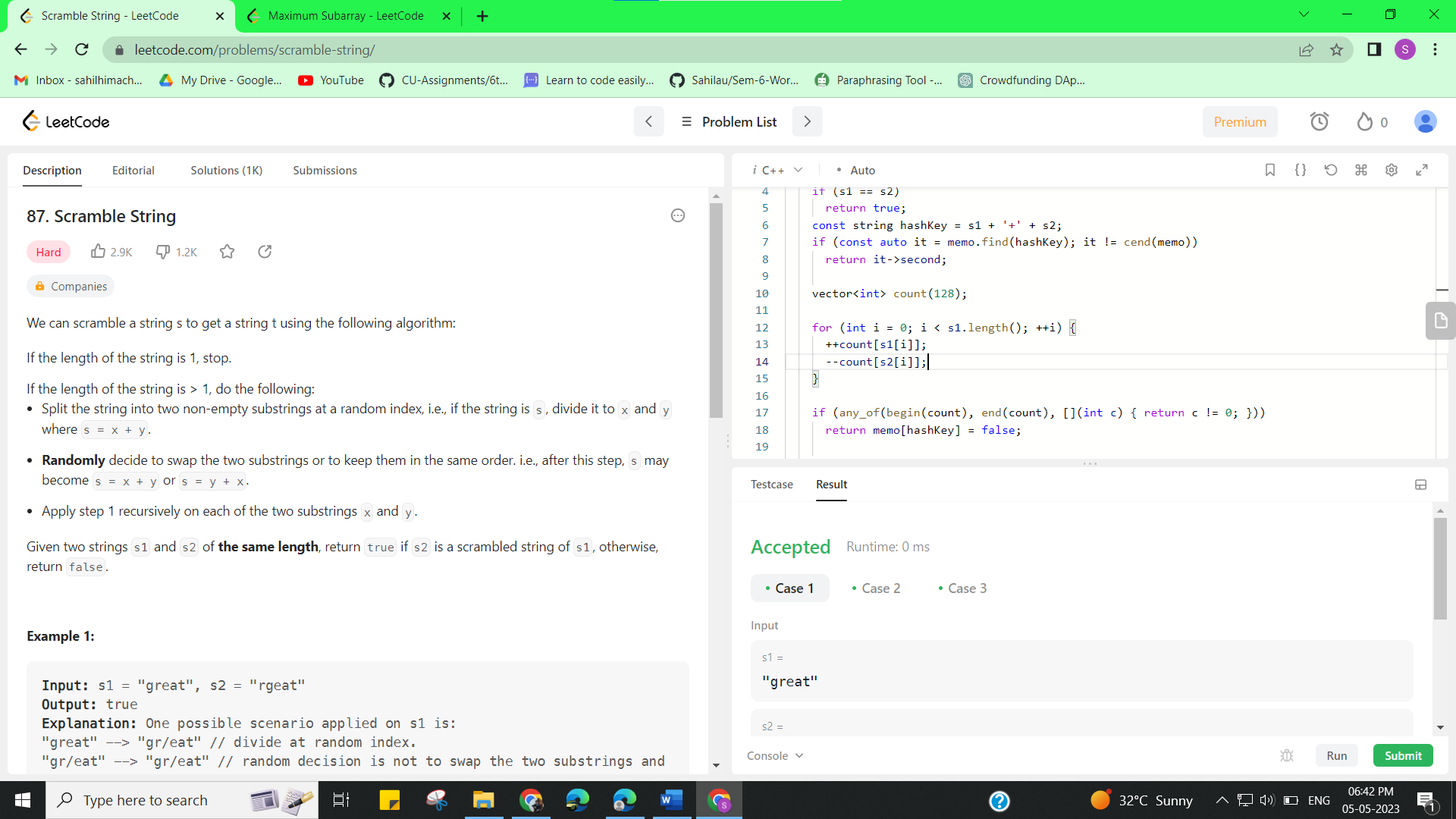
  }

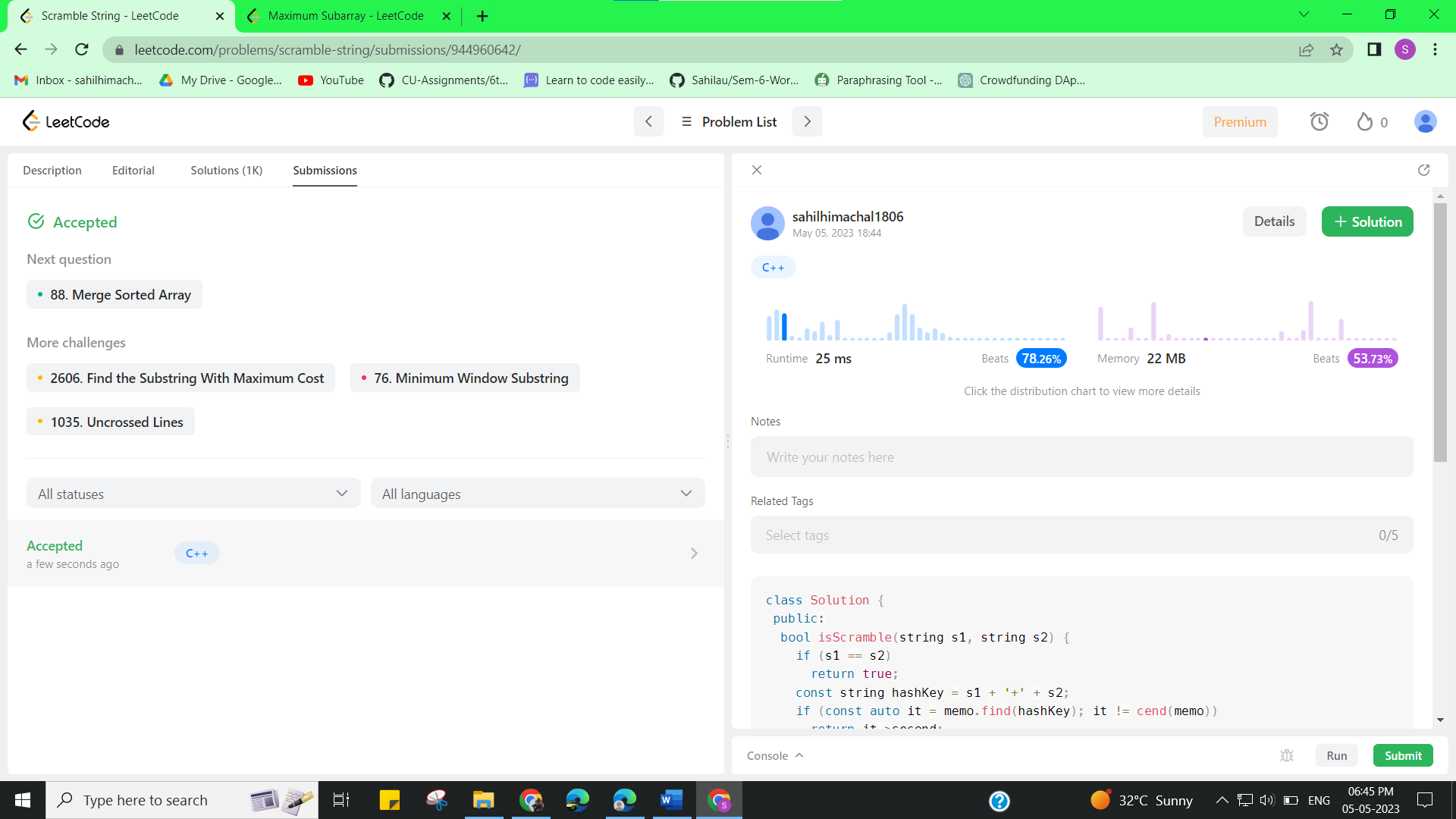
 private:

  unordered\_map<string, bool> memo;

};

**4. Result/Output/Writing Summary:**





**Experiment 10.2**

1. **Aim/Overview of the practical:**

Maximum Subarray

Given an integer array nums, find the Subarray  with the largest sum, and return *its sum*.

<https://leetcode.com/problems/maximum-subarray/>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of Dynamic Programming.

Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as optimal substructure property.

1. **Code:**

class Solution {

 public:

  int maxSubArray(vector<int>& nums) {

    // dp[i] := max sum subarray ending w/ i

    vector<int> dp(nums.size());

    dp[0] = nums[0];

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

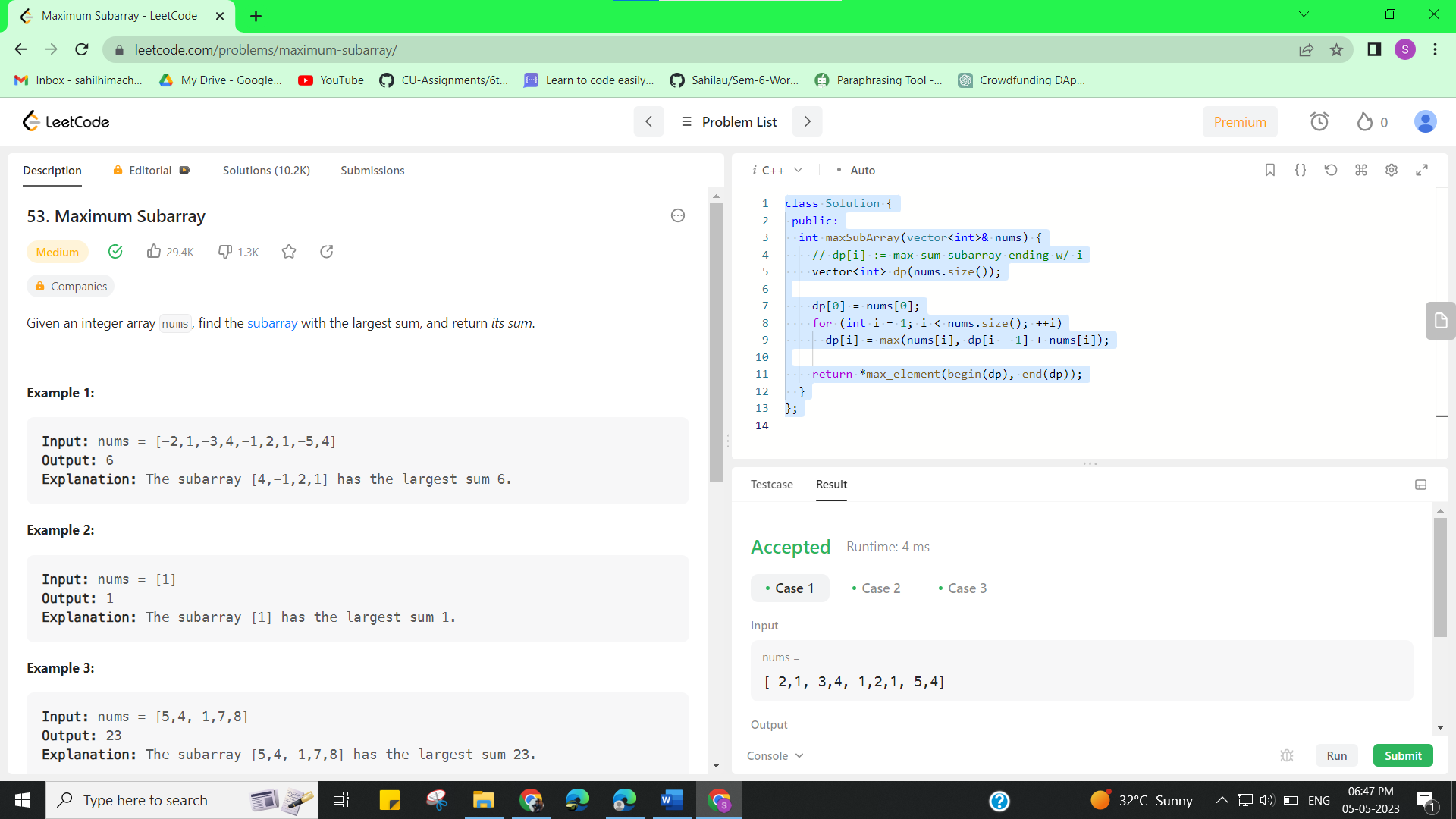
      dp[i] = max(nums[i], dp[i - 1] + nums[i]);

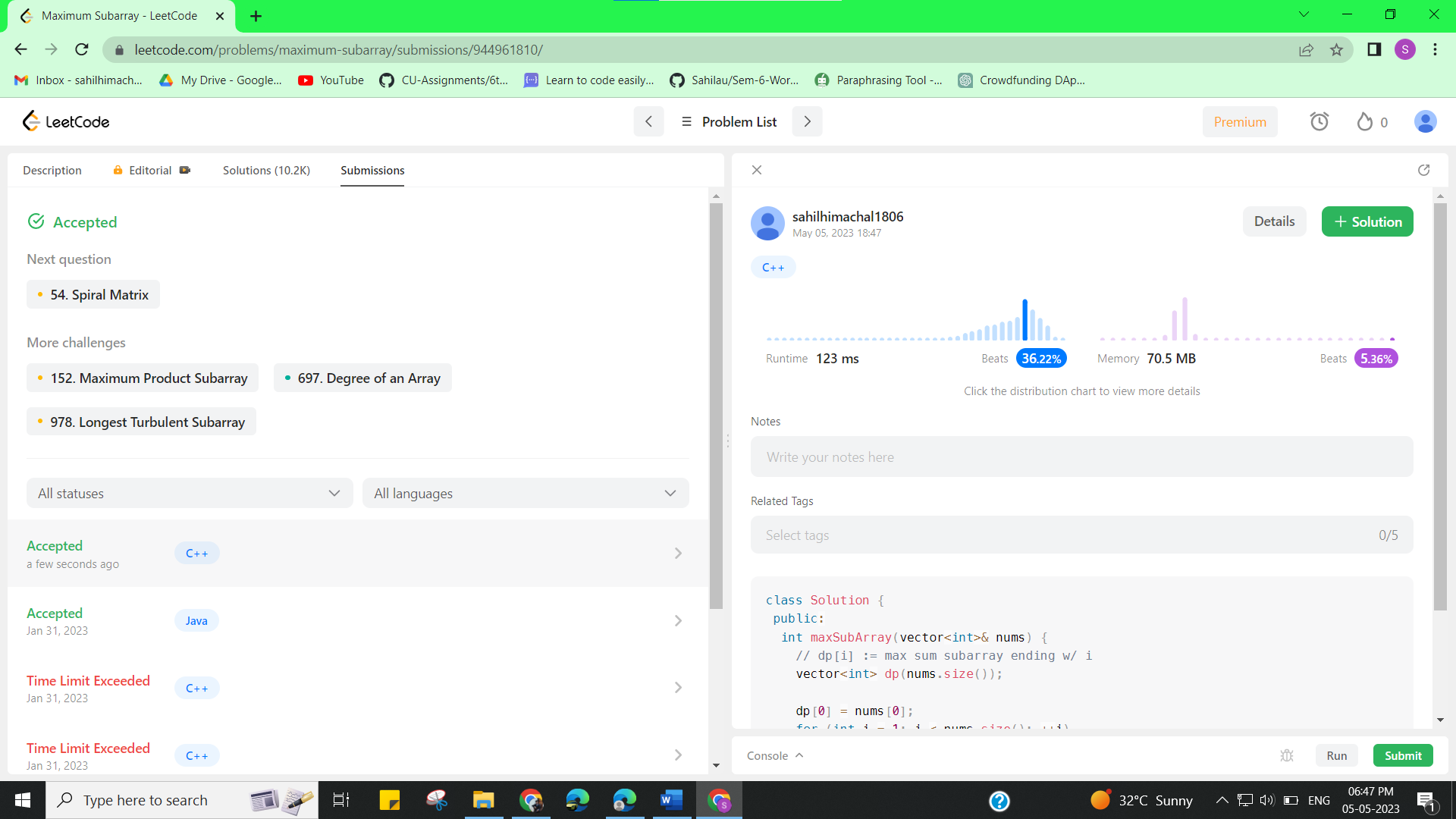
    return \*max\_element(begin(dp), end(dp));

  }

};

1. **Result/Output/Writing Summary:**

****

****

**Learning outcomes (What I have learnt):**

* Learned the concept of Dynamic Programming.