

**UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**Subject Name:** Competitive Coding

**Subject Code:** 20CSP-314

**Submitted to: Submitted by:**

Er. Mamta Punia Name: Sahil Kaundal

UID: 21BCS8197

Section: 616

Group: A

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ex. No** | **List of Experiments** | **Conduct (MM: 12)** | **Viva**  **(MM: 10)** | **Record (MM: 8)** | **Total**  **(MM: 30)** | **Remarks/Signature** |
| 1 | To demonstrate the concept of Array. |  |  |  |  |  |
| 2 | To demonstrate the concept of Stack and Queue. |  |  |  |  |  |
| 3 | To demonstrate the concept of Linked List. |  |  |  |  |  |
| 4 | Sorting and Searching: Implement the concept of Searching and Sorting techniques. |  |  |  |  |  |
| 5 | To implement the concept of Graphs. |  |  |  |  |  |
| 6. | To demonstrate the concept of Tree Data Structure |  |  |  |  |  |
| 7. | To Demonstrate the concept of String Data Structure |  |  |  |  |  |
| 8. | Dynamic Programming |  |  |  |  |  |
| 9. | Backtracking |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Experiment 9.1**

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

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

**Semester:** 5th **Date of Performance:** 10/11/2022

**Subject Name:** CC Lab **Subject Code:** 20CSP-314

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

Backtracking

You have an N \* M chessboard on which some squares are blocked out. In how many ways can you place one or more queens on the board, such that, no two queens attack each other? Two queens attack each other, if one can reach the other by moving horizontally, vertically, or diagonally without passing over any blocked square. At most one queen can be placed on a square. A queen cannot be placed on a blocked square.

<https://www.hackerrank.com/challenges/queens-on-board/problem>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**
   * To understand the concept of Backtracking.

**4. Code:**

#include <vector>

#include <string>

#include <algorithm>

#include <iostream>

#include <unordered\_map>

#include <cassert>

using namespace std;

struct Solution2

{

    typedef basic\_string<unsigned char> \_\_Board;

    typedef \_\_Board::value\_type         \_\_Row;

    long long solve(const vector<string> & B){

        if (B.empty() || B[0].empty())

            return 0;

        for (size\_t i = 0; i < B.size(); ++i){

            \_\_Row row = 0;

            for (size\_t j = 0; j < B[i].size(); ++j){

                if ('.' == B[i][j])

                    row |= (1 << j);

            }

            row = ~row;

            board.push\_back(row);

            \_\_Board p;

            genPlacements(row, p, B[i].size());

            placements.push\_back(p);

        }

        bmask = (1 << B[0].size()) - 1;

        return help(0, 0, 0, 0);

    }

private:

    static void genPlacements(\_\_Row block, \_\_Board & ret, int M){

        for (int i = 0; i < M; ++i){

            \_\_Row p1 = 1 << i;

            if (0 != (p1 & block))

                continue;

            ret.push\_back(p1);

            for (int j = i + 2; j < M; ++j){

                \_\_Row p2 = p1 | (1 << j);

                if (0 != (p2 & block))

                    continue;

                \_\_Row m2 = (1 << j) - (1 << (i + 1));

                if (0 == (m2 & block))

                    continue;

                ret.push\_back(p2);

                for (int k = j + 2; k < M; ++k){

                    \_\_Row p3 = p2 | (1 << k);

                    if (0 != (p3 & block))

                        continue;

                    \_\_Row m3 = (1 << k) - (1 << (j + 1));

                    if (0 == (m3 & block))

                        continue;   //there is not enough blocks between 3 Qs

                    ret.push\_back(p3);

                }

            }

        }

    }

    \_\_Row calcMask(\_\_Row mask, \_\_Row blocks){

        \_\_Row b = mask & blocks;

        mask &= ~b;

        return (mask & bmask);

    }

    static int hash(size\_t row, \_\_Row lmask, \_\_Row dmask, \_\_Row rmask){

        int r = row;

        r <<= 8;

        r += lmask;

        r <<= 8;

        r += dmask;

        r <<= 8;

        r += rmask;

        return r;

    }

    long long help(size\_t row, \_\_Row lmask, \_\_Row dmask, \_\_Row rmask){

        if (row >= board.size())

            return 0;

        const int h = hash(row, lmask, dmask, rmask);

        unordered\_map<int, long long>::const\_iterator wh = save.find(h);

        if (wh != save.end())

            return wh->second;

        const \_\_Row blocks = board[row];

        const \_\_Row mask = lmask | dmask | rmask | blocks;

        long long ret = 0;

        lmask = calcMask(lmask, blocks);

        dmask = calcMask(dmask, blocks);

        rmask = calcMask(rmask, blocks);

        if (\_\_Row(-1) != mask){

            const \_\_Board & ps = placements[row];

            for (size\_t i = 0; i < ps.size(); ++i){

                const \_\_Row p = ps[i];

                if (0 != (mask & p))

                    continue;

                ++ret;

                ret += help(row + 1, (lmask | p) << 1, dmask | p, (rmask | p) >> 1);

            }

        }

        ret += help(row + 1, lmask << 1, dmask, rmask >> 1);

        return (save[h] = ret % 1000000007);

    }

    \_\_Board board;

    vector<\_\_Board> placements;

    unordered\_map<int, long long> save;

    \_\_Row bmask;

};

typedef Solution2 Solution;

int main()

{

    int t;

    cin >> t;

    while (t--){

        int n, m;

        cin >> n >> m;

        vector<string> b;

        for (int i = 0; i < n; ++i){

            string line;

            cin >> line;

            b.push\_back(line);

        }

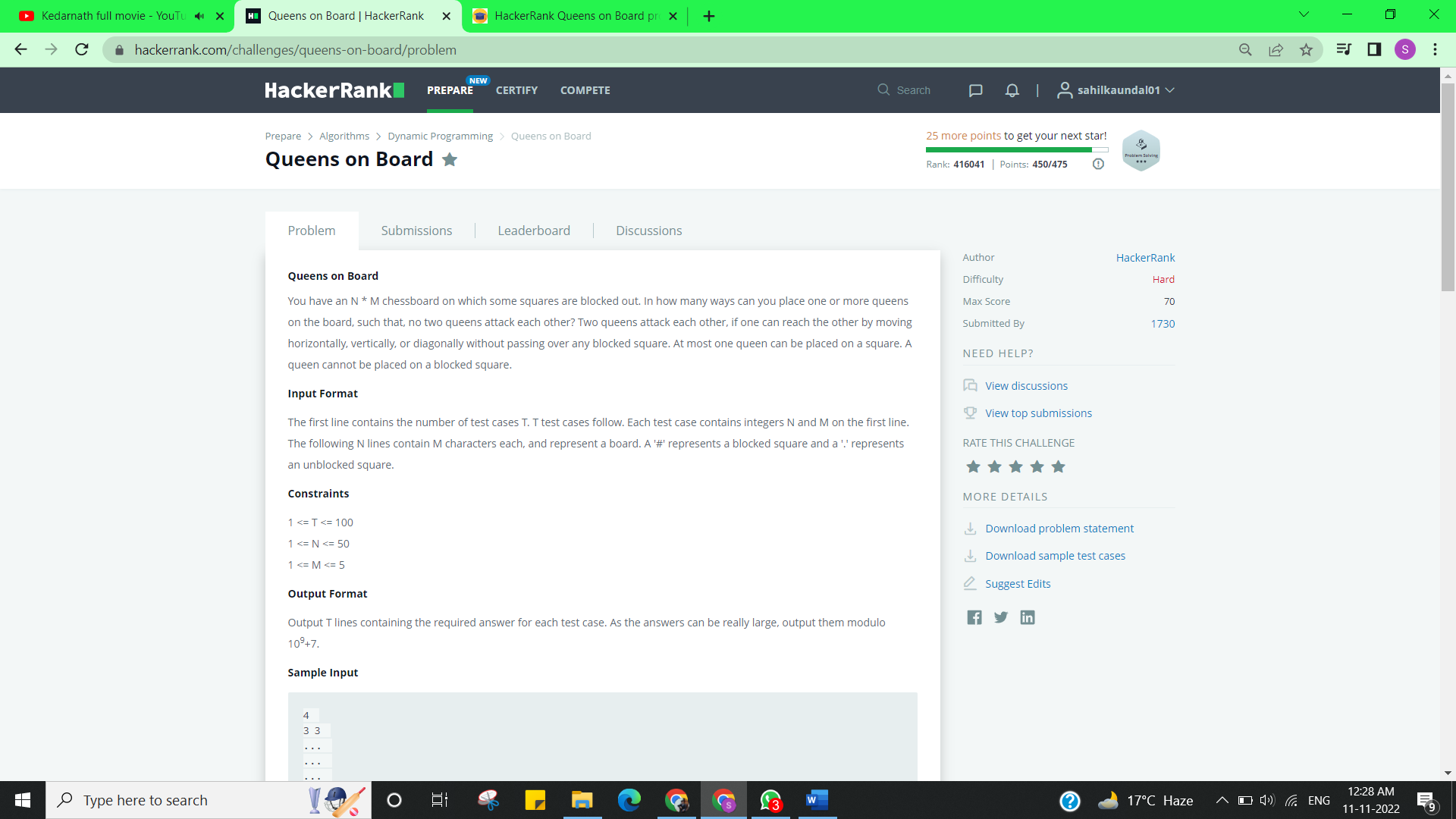
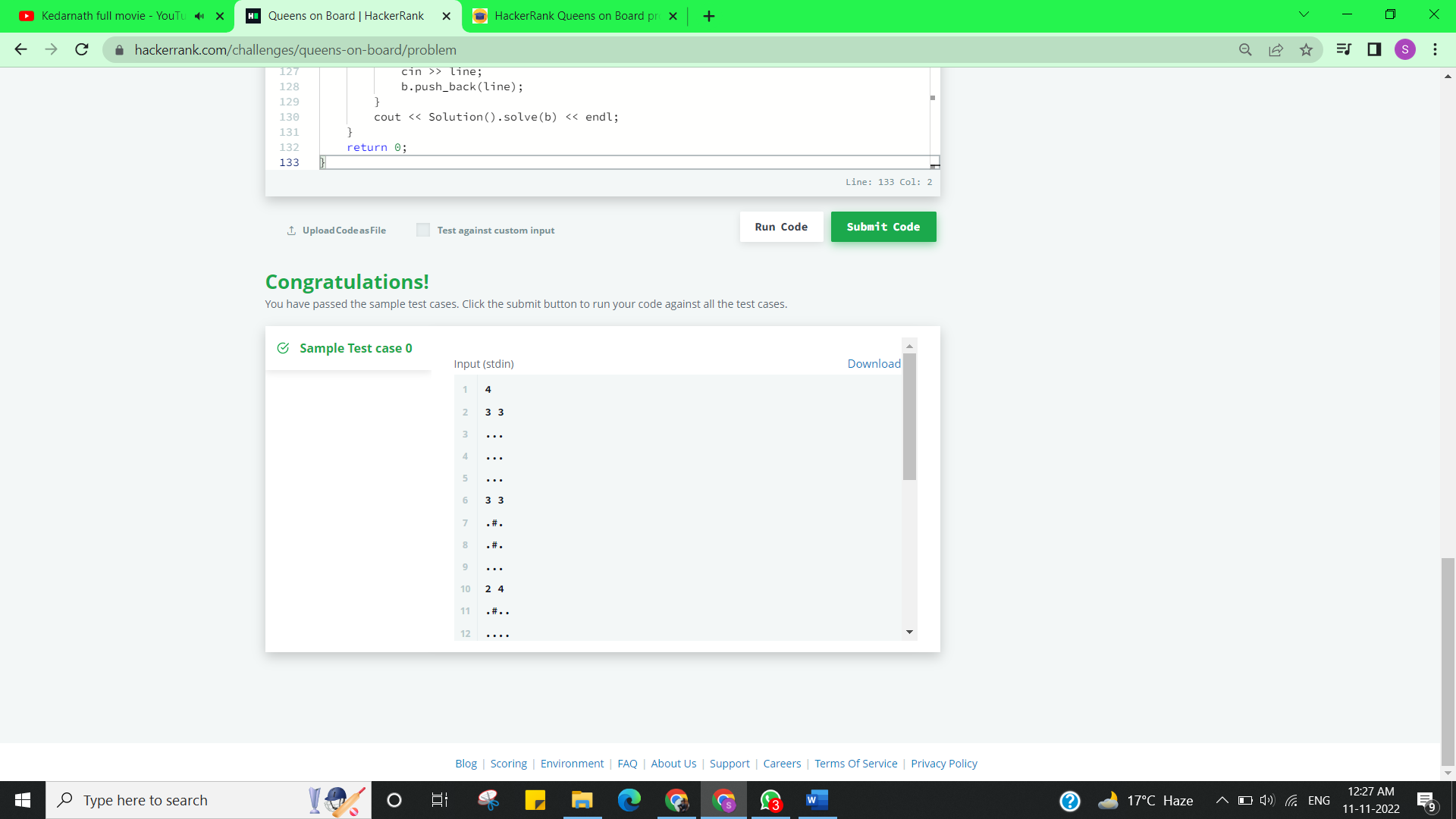
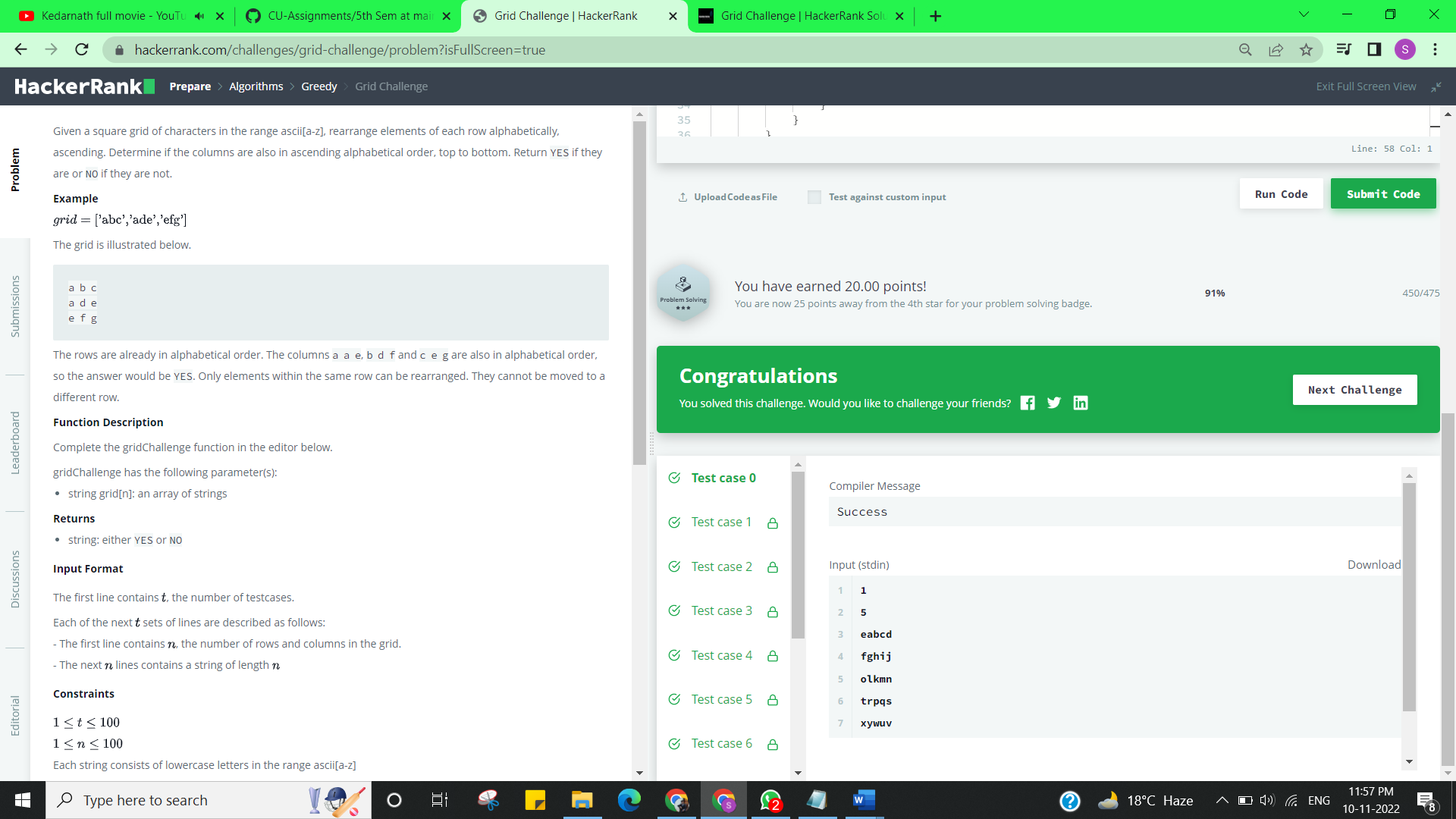
        cout << Solution().solve(b) << endl;

    }

    return 0;

}

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



**Experiment 9.2**

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

Backtracking

You are given a list of N positive integers, A = {a[1], a[2], ..., a[N]} and another integer S. You have to find whether there exists a non-empty subset of A whose sum is greater than or equal to S.

You have to print the size of minimal subset whose sum is greater than or equal to S. If there exists no such subset then print -1 instead.

<https://www.hackerrank.com/challenges/subset-sum/problem>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**
   * To understand the concept of Backtracking.
2. **Code:**

import java.util.Scanner

import scala.collection.Searching.\_

object Solution {

  def main(args: Array[String]): Unit = {

    val sc = new Scanner(System.in)

    sc.nextLine

    val a = sc.nextLine.split(' ').map(\_.toLong).sortBy(-\_)

    var sum = 0L

    val sums = a.map(v => {

      sum += v

      sum

    })

    val t = sc.nextInt

    (0 until t).foreach(\_ => {

      val s = sc.nextLong

      val count = (sums.search(s) match {

        case InsertionPoint(i) => i

        case Found(i) => i

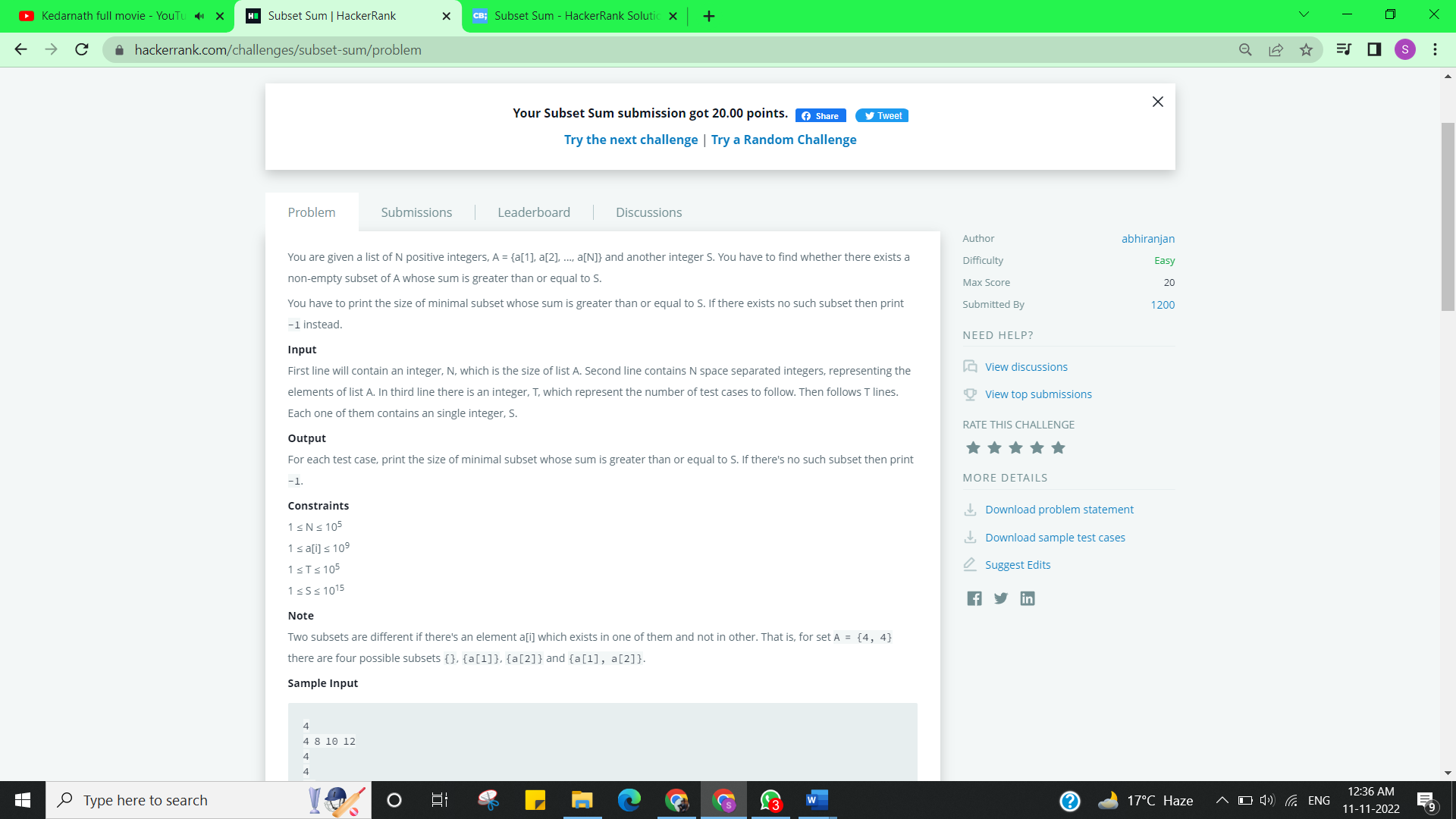
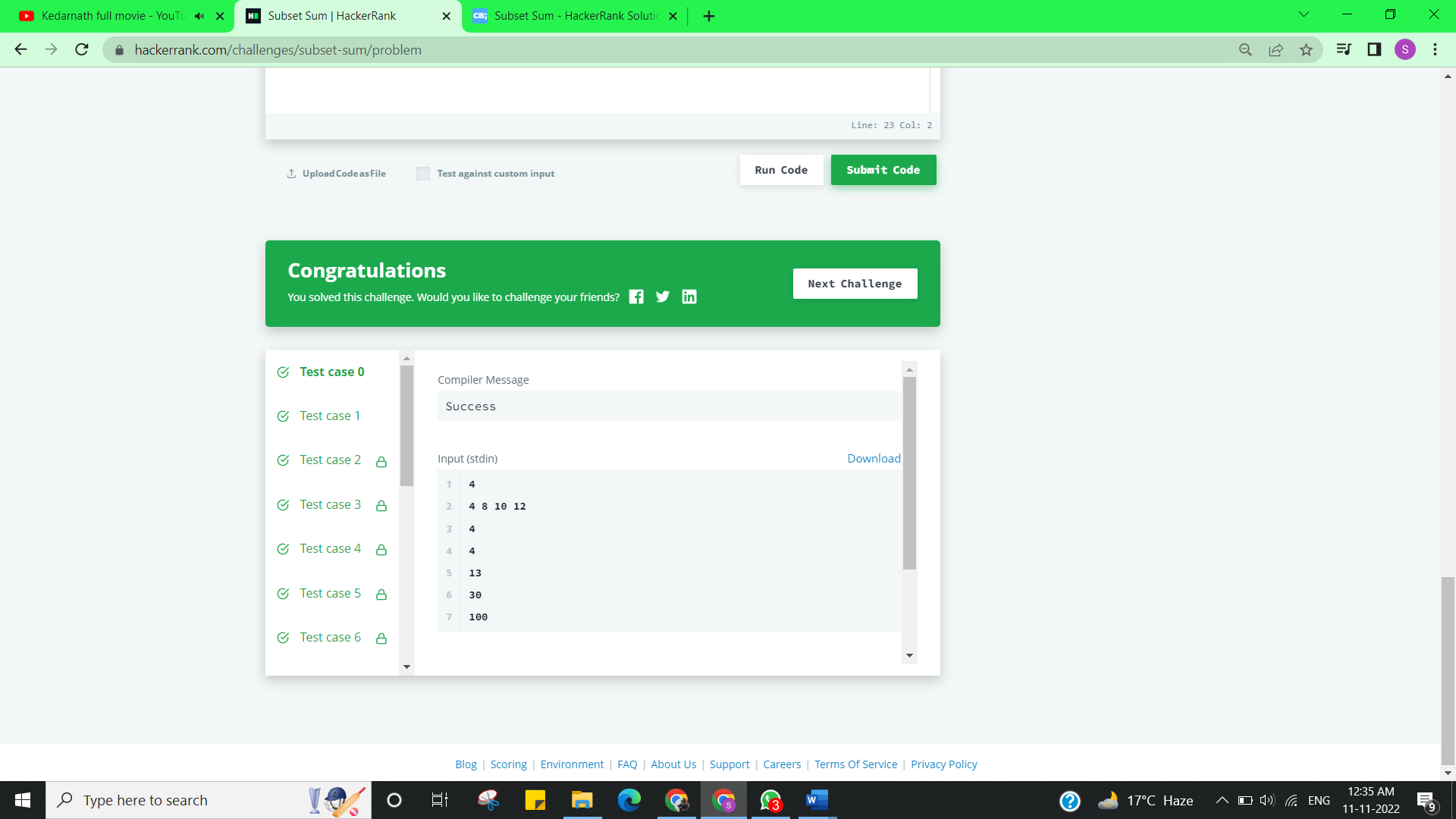
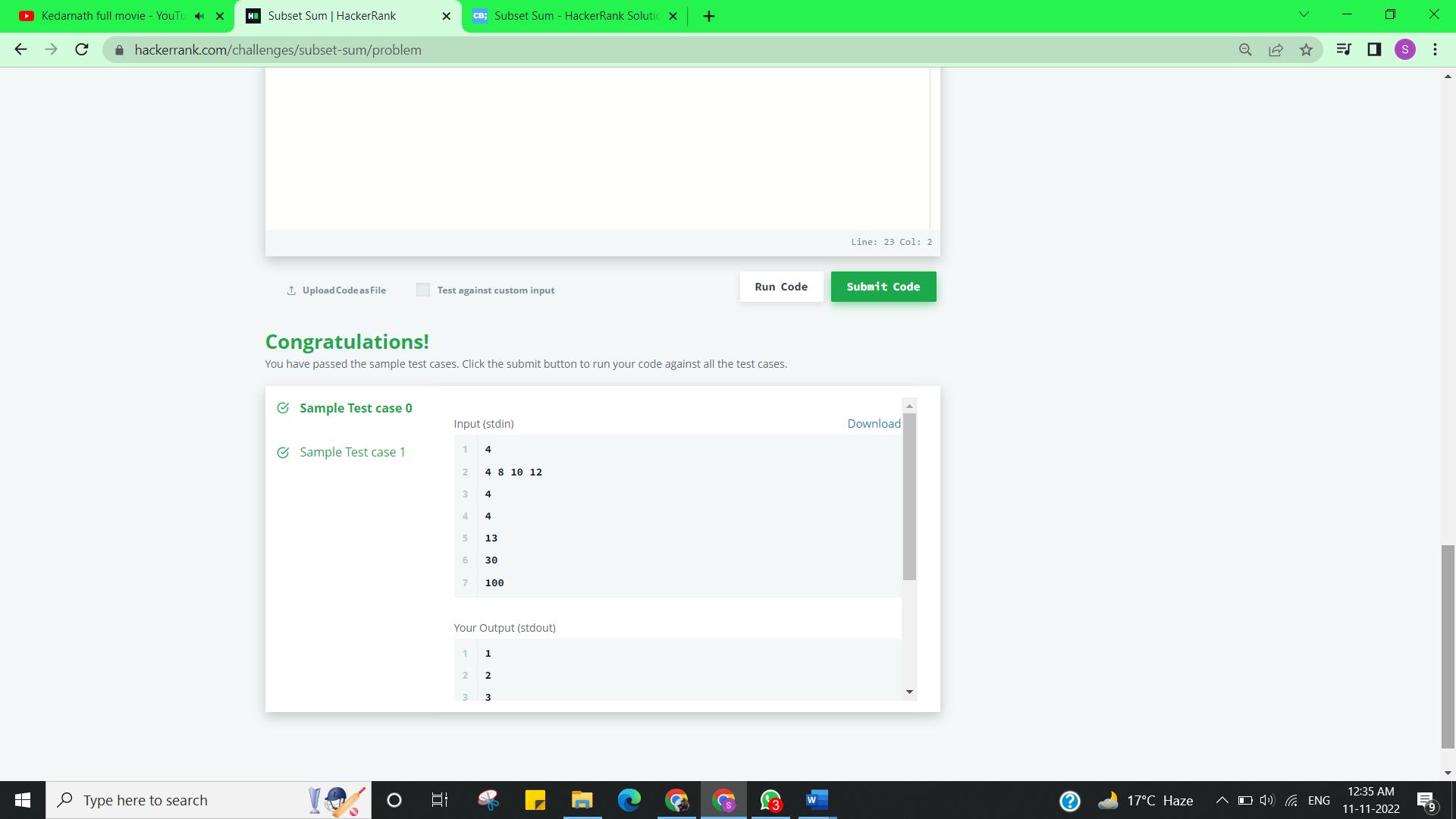
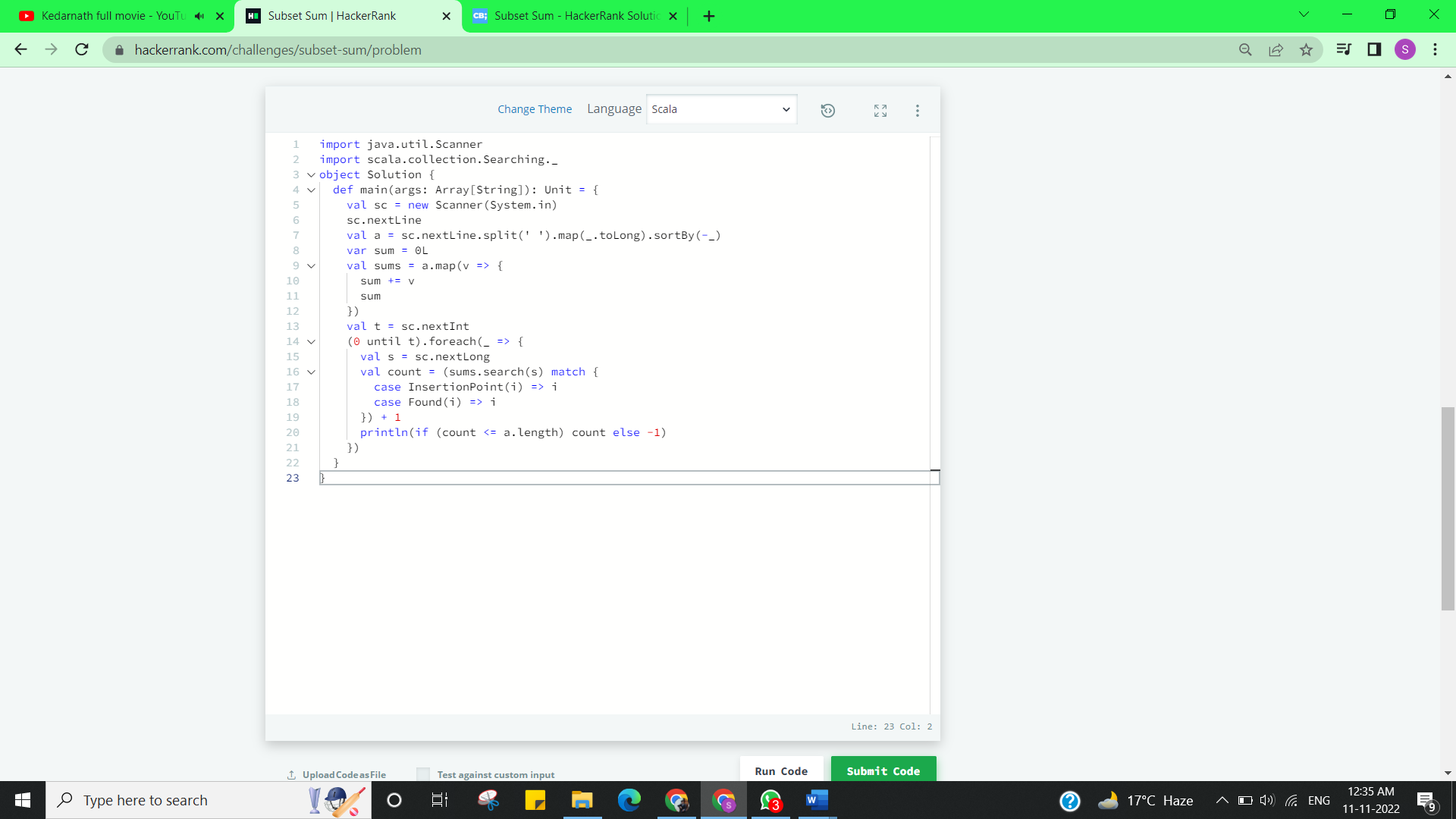
      }) + 1

      println(if (count <= a.length) count else -1)

    })

  }

}

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

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

* + Learned the concept of Backtracking.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |