**Worksheet-1.2**

**Student Name:-** Pushpraj Roy **UID:-** 20BCS9866

**Branch:-** BE- CSE **Section/Group:-** WM\_617 “A”

**Subjetct Code:-** 20CSP-314 **Semester:-** 5th

**Subject Name:-** Competitive Coding Lab

# Question 1-

**Aim** – Find difference between sums of two diagonals.

**Input Code-**

// C++ program to find the difference between the sum of diagonal.

#include <bits/stdc++.h> #define MAX 100 using namespace std;

int difference(int arr[][MAX], int n)

{

// Initialize sums of diagonals int d1 = 0, d2 = 0;

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

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

{

// finding sum of primary diagonal if (i == j) d1 += arr[i][j];

// finding sum of secondary diagonal if (i == n - j - 1) d2 += arr[i][j];

}

}

// Absolute difference of the sums across the diagonals return abs(d1 - d2);

} int main() {

int n = 3;

int arr[][MAX] =

{

{12, 4, 8},

{3 , 6, 9},

{13, 5, 1}

};

cout << "The Difference of sum of Diagonals = "<<difference(arr, n); return 0;

}

**Output-**

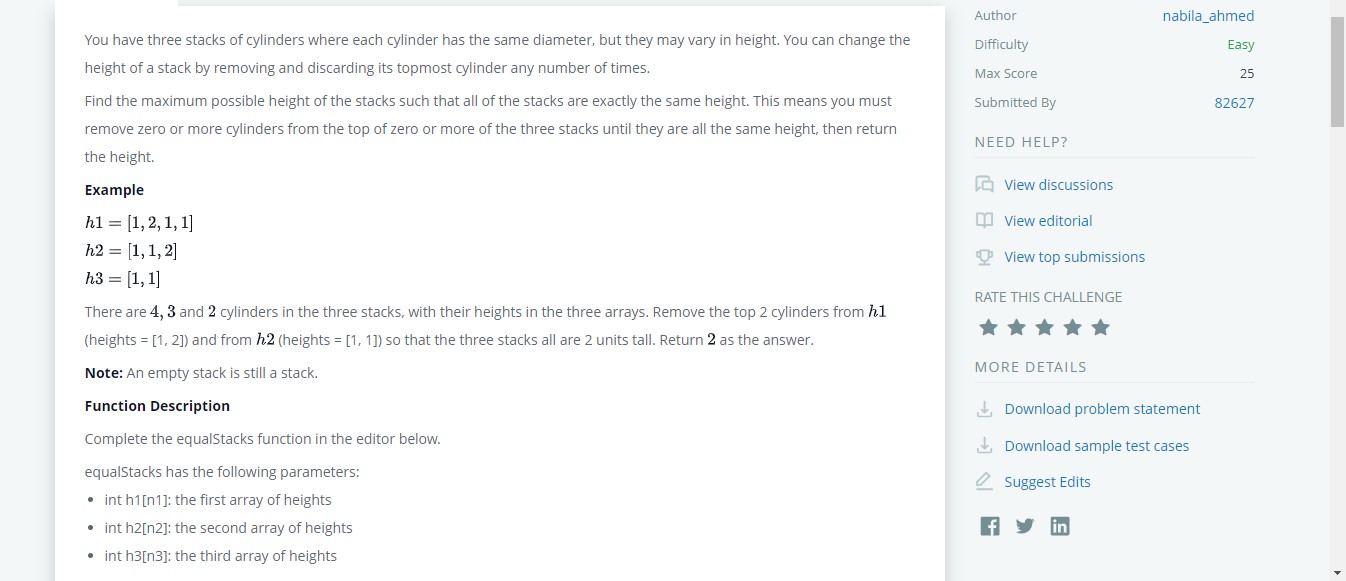


**Time complexity:** O(n\*n)

# Question 2-

**Aim –**You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.

## Screenshot of Question –



**Input Code-**

#include <bits/stdc++.h> using namespace std;

string ltrim(const string &); string rtrim(const string &); vector<string> split(const string &);

/\*

\* Complete the 'equalStacks' function below.

\*/

int equalStacks(vector<int> h1, vector<int> h2, vector<int> h3) { int n=h1.size(); int m=h2.size(); int p=h3.size(); stack<int>a,b,c;

int sum1=0,sum2=0,sum3=0; for(int i=n-1;i>=0;i--){ sum1+=h1[i];

a.push(sum1); }

for(int i=m-1;i>=0;i--){ sum2+=h2[i];

b.push(sum2); } for(int i=p-1;i>=0;i--){ sum3+=h3[i];

c.push(sum3); } while(1){

if(a.empty()||b.empty()||c.empty()){

return 0;

}

if(a.top()==b.top() && a.top()==c.top()){

return a.top();

}

else if(a.top()>=b.top()&&a.top()>=c.top()){ a.pop(); }

else if(b.top()>=a.top()&&b.top()>=c.top()){ b.pop(); } else {

c.pop();

}

}

}

int main()

{

ofstream fout(getenv("OUTPUT\_PATH"));

string first\_multiple\_input\_temp;

getline(cin, first\_multiple\_input\_temp);

vector<string> first\_multiple\_input = split(rtrim(first\_multiple\_input\_temp));

int n1 = stoi(first\_multiple\_input[0]);

int n2 = stoi(first\_multiple\_input[1]);

int n3 = stoi(first\_multiple\_input[2]);

string h1\_temp\_temp;

getline(cin, h1\_temp\_temp);

vector<string> h1\_temp = split(rtrim(h1\_temp\_temp));

vector<int> h1(n1);

for (int i = 0; i < n1; i++) {

int h1\_item = stoi(h1\_temp[i]);

h1[i] = h1\_item;

}

string h2\_temp\_temp; getline(cin, h2\_temp\_temp);

vector<string> h2\_temp = split(rtrim(h2\_temp\_temp));

vector<int> h2(n2);

for (int i = 0; i < n2; i++) {

int h2\_item = stoi(h2\_temp[i]);

h2[i] = h2\_item;

}

string h3\_temp\_temp;

getline(cin, h3\_temp\_temp);

vector<string> h3\_temp = split(rtrim(h3\_temp\_temp));

vector<int> h3(n3);

for (int i = 0; i < n3; i++) { int h3\_item = stoi(h3\_temp[i]);

h3[i] = h3\_item;

}

int result = equalStacks(h1, h2, h3);

fout << result << "\n";

fout.close();

return 0;

}

string ltrim(const string &str) {

string s(str);

s.erase(

s.begin(), find\_if(s.begin(), s.end(), not1(ptr\_fun<int, int>(isspace)))

);

return s;

}

string rtrim(const string &str) {

string s(str);

s.erase( find\_if(s.rbegin(), s.rend(), not1(ptr\_fun<int, int>(isspace))).base(), s.end()

);

return s;

}

vector<string> split(const string &str) { vector<string> tokens;

string::size\_type start = 0; string::size\_type end = 0;

while ((end = str.find(" ", start)) != string::npos) { tokens.push\_back(str.substr(start, end - start));

start = end + 1;

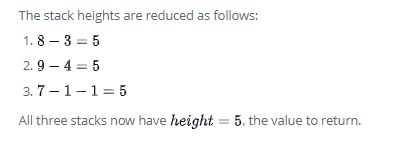
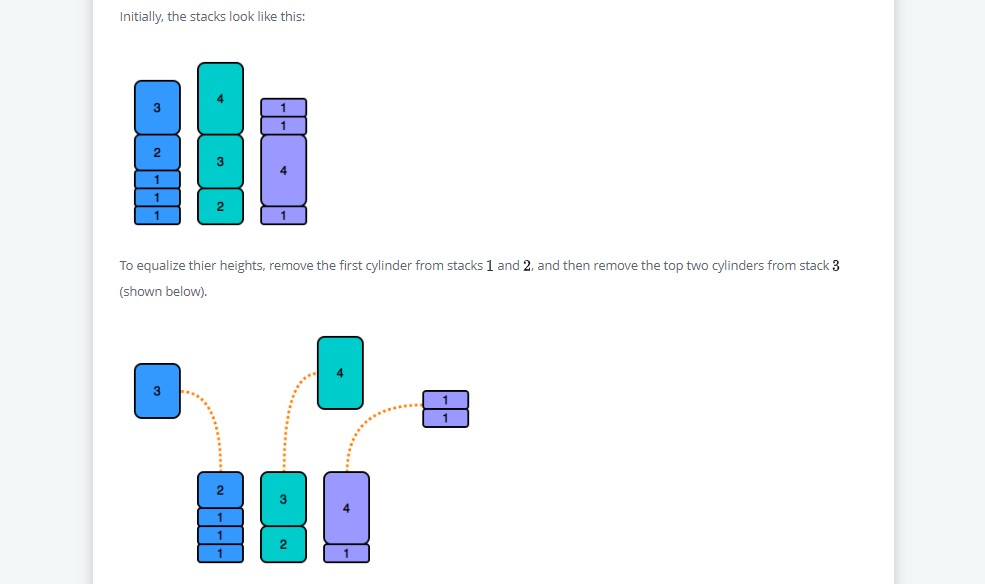
}

tokens.push\_back(str.substr(start));

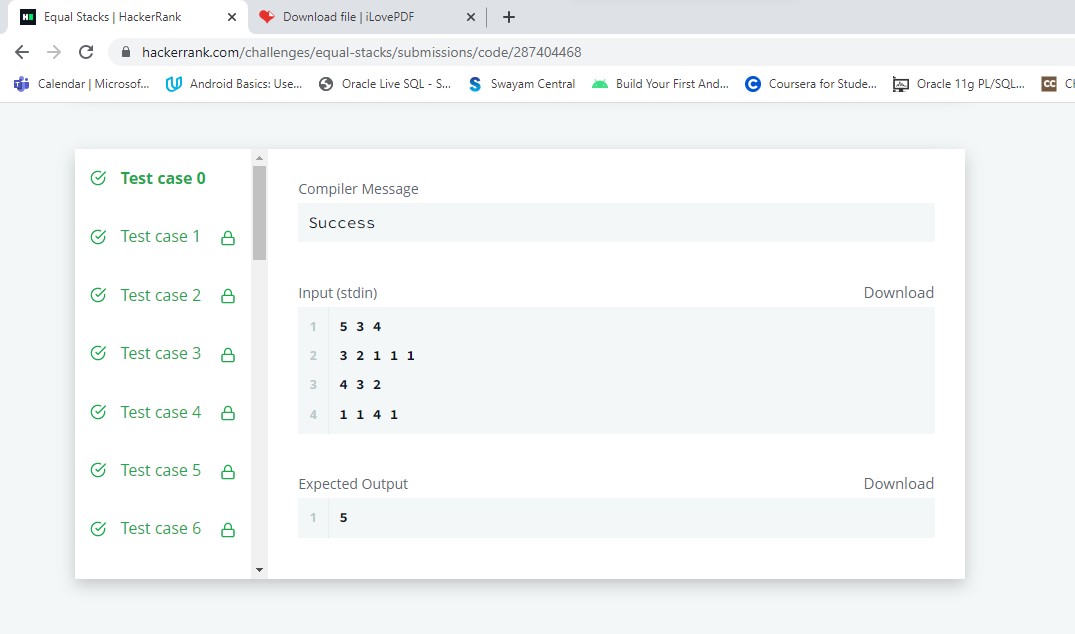
return tokens;

}

**Explanation-**



**Output-**



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

1.Using Stacks to solve complex problems and perform push() , pop() operations.

2.Deeper Understanding of Matrix and Arrays and Use them to solve Problems.

3.To implement problems based on different algorithm design techniques.

4.To learn the importance of designing an algorithm in an effective way by considering space and time complexity.