****

**Technical Assessment**

**Question no 1:**

Given **N** activities with their start and finish day given in array **start[ ]** and **end[ ]**. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a given day.

**Note:**Duration of the activity includes both starting and ending day.

**Example 1:**

**Input:**

N = 2

start[] = {2, 1}

end[] = {2, 2}

**Output:**

1

**Explanation:**

A person can perform only one of the

given activities.

**Example 2:**

**Input:**

N = 4

start[] = {1, 3, 2, 5}

end[] = {2, 4, 3, 6}

**Output:**

3

**Explanation:**

A person can perform activities 1, 3

and 4.

**Your Task :** You don't need to read input or print anything. Your task is to complete the function ***activityselection()*** which takes array **start[ ],** array **end[ ]** and integer **N** as input parameters and returns the maximum number of activities that can be done.

**Expected Time Complexity** : O(N \* Log(N)) **Expected Auxilliary Space**: O(N)

**Constraints:** 1 ≤ N ≤ 2\*105 1 ≤ start[i] ≤ end[i] ≤ 109

SOLUTION:

#include <iostream>

using namespace std;

// Prints a maximum set of activities that can be done by a single

// person, one at a time.

void activityselection(int s[], int f[], int n)

{

int i, j;

cout << " Selected activities are " << endl;

i = 0;

cout << " " << i;

// Consider rest of the activities

for (j = 1; j < n; j++)

{

if (s[j] >= f[i])

{

cout << " " << j;

i = j;

}

}

}

int main()

{

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

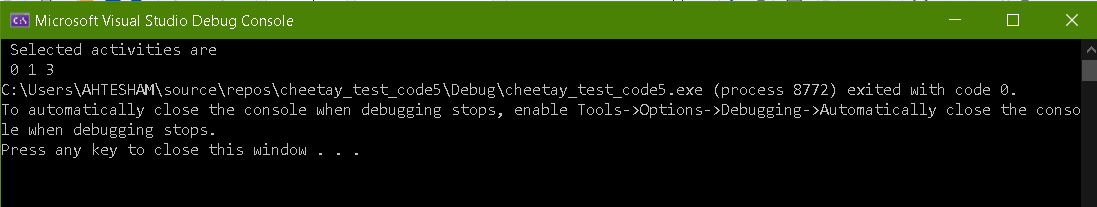
int f[] = { 2, 4, 3, 6,};

int n = sizeof(s) / sizeof(s[0]);

activityselection(s, f, n);

return 0;

}



Question: 2

Given an array A of n positive numbers. The task is to find the first Equilibium Point in the array.  Equilibrium Point in an array is a position such that the sum of elements before it is equal to the sum of elements after it.

**Example 1:**

**Input:**

n = 1

A[] = {1}

**Output:** 1

**Explanation:** Since its the only

element hence its the only equilibrium

point.

**Example 2:**

**Input:**

n = 5

A[] = {1,3,5,2,2}

**Output:** 3

**Explanation:** For second test case

equilibrium point is at position 3

as elements before it (1+3) =

elements after it (2+2).

**Your Task:** The task is to complete the function **equilibriumPoint()** which takes the array and n as input parameters and returns the point of equilibrium. Return -1 if no such point exists.

**Expected Time Complexity:**O(n) **Expected Auxiliary Space:** O(1)

**Constraints:** 1 <= n <= 106 1 <= A[i] <= 108

Solution:

#include <iostream>

using namespace std;

int equilibriumPoint(int arr[], int n)

{

int i, j;

int leftsum, rightsum;

// Check for indexes one by one until

//an equilibrium index is found

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

{

// get left sum

leftsum = 0;

for (j = 0; j < i; j++)

leftsum += arr[j];

// get right sum

rightsum = 0;

for (j = i + 1; j < n; j++)

rightsum += arr[j];

// if leftsum and rightsum

//are same, then we are done

if (leftsum == rightsum)

return i;

}

return -1;

}

int main()

{

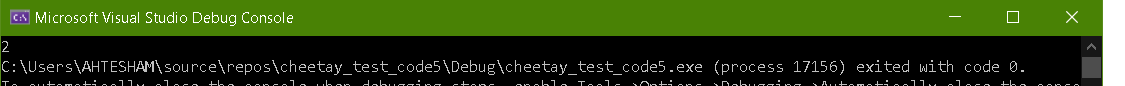
int arr[] = { 1,3,5,2,2 };

int arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << equilibriumPoint(arr, arr\_size);

return 0;

}



Question no 3

Given a string **S**, find length of the longest substring with all distinct characters.

**Example 1:**

**Input:**

S = "geeksforgeeks"

**Output:** 7

**Explanation**: "eksforg" is the longest

substring with all distinct characters.

â€‹**Example 2:**

**Input**:

S = "aaa"

**Output:** 3

**Explanation**: "a" is the longest substring

with all distinct characters.

**Your Task:** You don't need to read input or print anything. Your task is to complete the function **longestSubstrDitinctChars()**which takes the string S as input and returns the length of the longest substring with all distinct characters.

**Expected Time Complexity:**O(|S|). **Expected Auxiliary Space:**O(1).

**Constraints:** 1<=|S|<=105

SOLUTION:

#include <iostream>

#include<vector>

#include<math.h>

#include<algorithm>

using namespace std;

bool areDistinct(string str, int i, int j)

{

vector<bool> visited(26);

for (int k = i; k <= j; k++) {

if (visited[str[k] - 'a'] == true)

return false;

visited[str[k] - 'a'] = true;

}

return true;

}

int longestUniqueSubsttr(string str)

{

int n = str.size();

int res = 0; // result

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

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

if (areDistinct(str, i, j))

res = max(res, j - i + 1);

return res;

}

int main()

{

string str = "geeksforgeeks";

string s = "aaa";

cout << "The input string is " << str << endl;

int len = longestUniqueSubsttr(str);

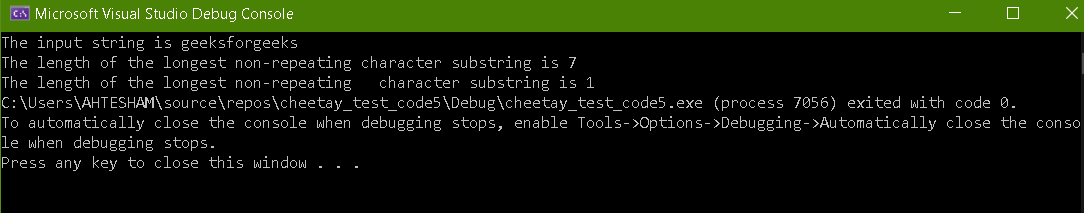
cout << "The length of the longest non-repeating character substring is "<< len << endl;

int len1 = longestUniqueSubsttr(s);

cout << "The length of the longest non-repeating character substring is "<< len1;

return 0;

}



Question no 4

There is **one** meeting room in a firm. There are **N** meetings in the form of **(S[i], F[i])** where **S[i]**is start time of meeting **i**and **F[i]**is finish time of meeting **i.** What is the **maximum** number of meetings that can be accommodated in the meeting room when only one meeting can be held in the meeting room at a particular time? Also note start time of one chosen meeting can't be equal to the end time of the other chosen meeting.

**Example 1:**

**Input:**

N = 6

S[] = {1,3,0,5,8,5}

F[] = {2,4,6,7,9,9}

**Output:**

4

**Explanation:**

Four meetings can be held with

given start and end timings.

**Example 2:**

**Input:**

N = 8

S[] = {75250, 50074, 43659, 8931, 11273,

27545, 50879, 77924}

F[] = {112960, 114515, 81825, 93424, 54316,

35533, 73383, 160252}

**Output:**

3

**Explanation:**

Only three meetings can be held

with given start and end timings.

**Your Task** : You don't need to read inputs or print anything. Complete the function **maxMeetings()**that recieves array **S[ ]**and **F[ ]**along with their size **N** as input parameters and returns the maximum number of meetings that can be held in the meeting room.

**Expected Time Complexity**: O(N\*LogN) **Expected Auxilliary Space** : O(N)

**Constraints:** 1 ≤ N ≤ 105 0 ≤ S[i] < F[i] ≤ 105

**SOLUTION:**

#include <iostream>

#include<vector>

#include<math.h>

#include<algorithm>

using namespace std;

struct meeting {

int start;

int end;

int pos;

};

bool comparator(struct meeting m1, meeting m2)

{

return (m1.end < m2.end);

}

void maxMeeting(int s[], int f[], int n)

{

meeting meet[6];

for (int i = 0; i < 6; i++)

{

// Starting time of meeting i.

meet[i].start = s[i];

// Finishing time of meeting i

meet[i].end = f[i];

// Original position/index of meeting

meet[i].pos = i + 1;

}

sort(meet, meet + 6, comparator);

vector<int> m;

m.push\_back(meet[0].pos);

int time\_limit = meet[0].end;

// Check for all meeting whether it

// can be selected or not.

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

if (meet[i].start >= time\_limit)

{

// Push selected meeting to vector

m.push\_back(meet[i].pos);

// Update time limit.

time\_limit = meet[i].end;

}

}

for (int i = 0; i < m.size(); i++) {

cout << m[i] << " ";

}

}

int main()

{

// Starting time

int s[] = { 1, 3, 0, 5, 8, 5 };

// Finish time

int f[] = { 2, 4, 6, 7, 9, 9 };

// Number of meetings.

int n = sizeof(s) / sizeof(s[0]);

// Function call

maxMeeting(s, f, n);

return 0;

}

