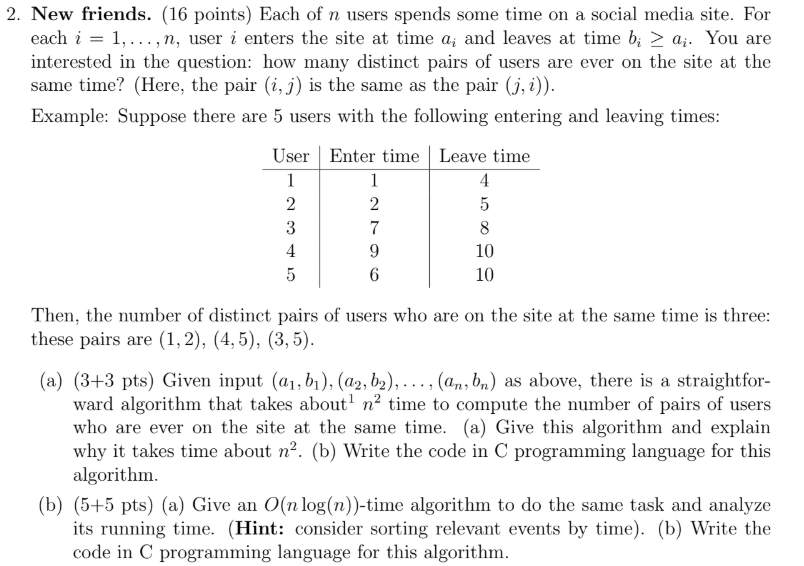


I hope to advanced algorithms and improve my problem solving skills to understand, analyze and solve such problems that usually came in challenges, contests, interviews, etc. This will also help me in future as this course will make us realize the bad coding mistakes people usually make and how to prevent our self from doing the same.



(a). Algorithm

Algorithm **I**: Time Complexity O(n^2)

This Program will check how many unique no of pairs of students will be

available onilne based on thier entry and exit times

The Algorithm Uses Two Nested Loops Which Go over N times

**Input**:

n <= total no of students

people <= [(entry, exit), ...] Array defining a students' entry/exit time

no\_of\_pairs <= 0

**FOR** i from 0 to n - 1:

**FOR** j from (i+1) to n - 1:

**IF** (person1.entry < person2.exit && person2.entry < person1.exit):

Increment no\_of\_pairs by 1

(a). C Program:

#include <stdio.h>

typedef struct person{

int entry;

int exit;

} Person;

int main(int argc, char const \*argv[]){

int noOfPeople;

scanf("%d", &noOfPeople);

Person people[noOfPeople];

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

scanf("%d %d", &people[i].entry, &people[i].exit);

}

int distinctPairs = 0;

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

for (int j = i + 1; j < noOfPeople; j++){

if(people[i].entry < people[j].exit && people[j].entry < people[i].exit){

distinctPairs++;

}

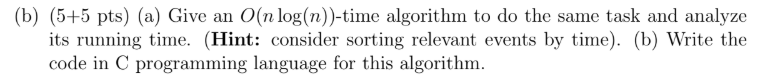
}

}

printf("Possible distinct pairs are %d", distinctPairs);

return 0;

}



1. . Prerequisite Algorithms:

1-Binary Search

Algorithm **BINARYSEARCH**: Time Complexity O(log(n))

This Program will search for the greatest no. less than or equal

to the Value given in the Sorted Array

**BINARYSEARCH**(Array, val)

l <= 0, r <= Array.Length -1

**WHILE** l < r:

**GET** Middle Element and compare with val

mid <= l + (r - l) / 2;

**IF** val is less than Array[mid]:

**SEARCH** for val in right half of sorted Array:

l <= mid + 1

**ELSE**

**SEARCH** for val in left half of sorted Array:

r <= mid

**IF** Array[l] == val:

RETURN l

**ELSE**

RETURN l - 1

2-Merge Sort

Algorithm **MERGESORT**: Time Complexity O(n\*log(n))

**MERGESORT**(arr[], l, r)

**IF** l == r

**RETURN**

**ELSE**

Find the middle point to divide the array into two halves:

middle m <= (l+r)/2

Do **MERGESORT** for first half:

Call **MERGESORT**(arr, l, m)

Do MERGESORT for second half:

Call **MERGESORT**(arr, m+1, r)

MERGE the two halves sorted in step 2 and 3:

Call **MERGE**(arr, l, m, r)

Algorithm MERGE: Time Complexity O(n)

**MERGE**(Array1, Array2, l, m, r):

// Array1.Length is m - l + 1, Array2.Length is r - m

Create a Duplicate array of Total Length:

Temp <= Array of Size [ Array1.Length + Array2.Length ]

index <= 0

Initialize Index for both Array:

i <= 0, j <= 0

**WHILE** index < Temp.Length:

**IF** any Array becomes empty

**IF** i == Array1.Length :

Temp[index++] <= Array2[j++]

**IF** j == Array2.Length :

Temp[index++] <= Array1[i++]

**CONTINUE**

INSERT the lower Element:

**IF** Array1[i] < Array2[j]:

Temp[index++] <= Array1[i++]

**ELSE**

Temp[index++] <= Array2[j++]

(b) Algorithm

Algorithm II: Time Complexity **O(n\*log(n))**

This Program will check how many unique no of pairs of students will be

available online based on their entry and exit times

The First Part of the Problem User MERGESORT which takes **O(n\*log(n))** Time

In Second Part we BINARYSEARCH the Value in Array over a loop so

net Complexity will be **O(n) \* O(log(n))** => **O(n\*log(n))**

Input:

n <= Total no of students

Array defining a student structure with entry/exit time

people <= [(entry, exit), ...]

no\_of\_pairs <= 0

Apply MERGESORT on people Array

**FOR** i from 0 to n - 1:

GET Value When the ith person Leaves

val <= people[i].exit

Find How many people were logged in before ith person left

found\_index <= **BINARYSEARCH**(people, val)

Add No. of Pairs of the respective person

**Increment** no\_of\_pairs by found\_index - i

(b) C Program

#include <stdio.h>

typedef struct person {

int entry;

int exit;

} Person;

int binarySearchPersonEntry(Person Array[], int val, int start, int end)

{

while (start <= end) {

int m = start + (end - start) / 2;

if (Array[m].entry == val)

return m;

if (Array[m].entry < val)

start = m + 1;

else

end = m - 1;

}

return start - 1;

}

int compare(Person p1, Person p2){

if (p1.entry < p2.entry) {

return 1;

}

else {

if (p1.exit < p2.exit) {

return 1;

}

else {

return 0;

}

}

}

void mergeSort(Person Array[], int start, int end)

{

if (start == end) {

return;

}

int mid = (start + end - 1) / 2;

mergeSort(Array, start, mid);

mergeSort(Array, mid + 1, end);

// Merging Arrayay

int length = end - start + 1;

Person Temp[length];

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

Temp[i] = Array[i + start];

}

int i = 0, j = 0, m = mid - start + 1;

for (int c = start; c <= end; c++){

if(i >= m){

Array[c] = Temp[m + j++];

}

else if (j + mid >= end)

{

Array[c] = Temp[i++];

}

else if (compare(Temp[i], Temp[m + j]))

{

Array[c] = Temp[i++];

}

else

{

Array[c] = Temp[m + j++];

}

}

}

int main(int argc, char const \*argv[])

{

int noOfPeople;

scanf("%d", &noOfPeople);

Person people[noOfPeople];

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

scanf("%d %d", &people[i].entry, &people[i].exit);

}

mergeSort(people, 0, noOfPeople - 1);

int distinctPairs = 0;

int begin = 0, end = 0, current\_online = 0;

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

int k = binarySearchPersonEntry(people, people[i].exit - 1, i+1, noOfPeople-1) - i;

distinctPairs = k;

}

printf("No. of distinct pairs are: %d.\n", distinctPairs);

}

