CIS 675 Homework 2

Design and Analysis of Algorithms

Saket Kiran Thombre

SU ID: 899913802

NetID: sthombre

Email: sthombre@syr.edu

Collaborators for Q1, Q2:

1. Saurav Shashank Narvekar

   NetID: sanarvek

2. Chandan Pothumarthi

   NetID: cpothuma

Problem 1: Finding Opposite Pairs (Point 5)

Let A [0 . . . n1] be an array of n distinct real numbers. A pair (A[i], A[j]) is said to be an opposite if these numbers are out of order, i.e., i < j but A[i] > A[j]. Design an O(nlogn) algorithm for counting the number of opposites. Hint: You can Modify Mergesort to solve it.

Solution:

```
Now we will write a function to use merge sort for counting inversions.
All the algorithm lines are numbered.

1. Inversion Function (A, i, j)
2.     Inversion Counter = 0
3.     if i < j
        Now splitting the array into two equal parts
4.          mid = (i + j)/2
            We use this to calculate inversion in the left split of the array
5.          Inversion Counter += Inversion Function (A, i, mid)
            We use this to calculate inversion in the right split of the array
            Inversion Counter += Inversion Function (A, mid + 1, j)
            Inversion Counter += Merge Sort (A, i, mid, j)
6.      return Inversion Counter


Now we will merge both the sub arrays together.
7. Merge Sort(A,i,mid,j)
8.    Start1 = I         is the index of first element of left split
9.    Start2 = mid + 1   is the index of first element of right split
      Now we define inversion counter here
10.   Inversion Counter = 0
11.   Result Set = [] We create an empty array that will store the sorted
elements.
      We make sure that Start1 and Start2 should not exceed their limits.
12.   while Start1 <= mid and Start2 <= j
          There cannot be any inversion when A[Start1] >= A[Start2]:
13.        if A[Start1] <= A[Start2] ------------------------------------(1)
(Please see the last line for explanation)
14.              Result Set.push[A[Start1]]
15.              Start1 += 1
16.        Else
                In this step we will count the number of inversions
17.             Result Set.push [A[Start2]]
18.             Inversion Counter += (mid - Start1 + 1) ------------------------
(2) (Please see last line for explanation)
19.             Start2 += 1
20.   return Inversion Counter
```

I have created a Inversion Function which takes data array as input.
Here, I have implemented Merge sort Algorithm. Subsequently I have added a couple of lines of O (1) complexity to merge sort which is O(nLog(n)).
Therefore the final complexity is O (n Log(n)).
1. I have inverted the function A[Start1] >= A[Start2] so that we satisfy the given condition of i < j but A[i] > A[j].
2. I have added (mid - Start1) to account for the total number of inversions in the Split sub array.

Problem 2: Finding Opposite Pairs (Point 5)

Suppose that Hulu, for some reason, is worried about account sharing and comes to you with n total login instances. Suppose also that Hulu provides you with the means (e.g., an api) only to compare the login info of two of the items (i.e., login instances) in the list. By this, we mean that you can select any two logins from the list and pass them into an equivalence tester (i.e., provided api) which tells you, in constant time, if the logins were produced from the same account. You are asked to find out if there exists a set of at least n 2 logins that were from the same account. Design an algorithm that solves this problem in θ(nlog(n)) total invocations of the equivalence tester.

Solution:

```
All the algorithm lines are numbered.

1.HULU ID EQUIVALENCE(Array)
2.    end = Array.len
3.    if Array.len == 1 Return value if array consists of only 1 element.
4.        return Array[0]
Selecting the first element of the two split arrays if they are equal.
5.   if Array.len == 2 and EQUIVALENCE TESTER(Array[0], Array[1])
6.        return Array[0]

now we split the array
7.    mid = Array.len / 2
8.    C_left = Array [0..mid]
9.    C_right = Array[mid+1..end]

Giving recursive calls to both halves.
10.    cl = HULU ID EQUIVALENCE (C_left)
11.    cr = HULU ID EQUIVALENCE (C_right)


12.    if cl != NULL
13.        count1 = 0
In superset of the split arrays, we find total ocurrences of CL
```

```
14.        for c in Array
15.            if EQUIVALENCE TESTER(cl, c)
16.                count1 += 1

17.        if count1 >= Array.len / 2
18.            return cl
19.        else if cr != NULL
20.             count2 = 0
In superset of the split arrays, we find total ocurrences of Cr
21.            for c in Array
22.                if EQUIVALENCE TESTER (cr, c)
23.                     count2 += 1
24.            if count2 >= Array.len / 2
25.                return cr
26.            else
27.                return NULL
```

I have created a HULU equivalence function which takes the accounts array as the input.
We have used the merge sort logic, however instead of merging the arrays, we are returning the count of maximum instances of the elements.