HW assignment 2

Wednesday, September 25, 2019 7:08 PM

Problem 1 :

Please find the sample output of the merge sort algorithm below with 50 elements being randomnly generated :

Output :

>> Please enter number of array elements (1,50] : 50

>> Elements before sorting happens :

>>46,10,27,100,94,40,8,67,27,60,28,50,20,72,57,30,35,62,100,42,100,71,4,44,54,23,63,46,93,79,11,7,71,41,7,41,2,45,64,34,9,55,80,54,16,98,61,27,41,92,

>> Elements post sorting :

>>2,4,7,7,8,9,10,11,16,20,23,27,27,27,28,30,34,35,40,41,41,41,42,44,45,46,46,50,54,54,55,57,60,61,62,63,64,67,71,71,72,79,80,92,93,94,98,100,100,100,

Problem 2 :

Text in red , are the changes that would be removed , and in green are the ones added .

Inorder to remove addition of sentinel at the end of array . We add a check to see if the arrays iterators have gone out of bound based on same we would stop incrementing counter for respective array and copy elements from other array . Please find the additions and deletions to algorithm in detail below :

```
MERGE(A,p,q,r)
N1 = q-p+1
N2 = r-q
Let L[1…N1+1] and R[1…N2+1]
   be new arrays
For i=1 to N1
   L[i] = A[p+i-1]
For j=1 to N2
   R[j] = A[q+j]

L[N1+1] = ( infinity )
R[N2+1] = ( infinity )
```

```
i = 1
j = 1

For k =p to r
   if i >= N1
     memcpy ( &A[k] , &R[j] , (N2-j)*sizeof(int)) -> Copy remaining elements from R array to main array ( A )
     break;   -> break from 'for' loop
  else if j>=N2
     memcpy ( &A[k] , &L[i] , (N1-i)*sizeof(int)) -> Copy remaining elements from L array to main array ( A )
     break; -> break from 'for' loop
  else
    if L[i] <= R[j]
      A[k] = L[i]
       i = i + 1
    else A[K] = R[j]
        j = j + 1
```

Problem 3 :

A . Lets assume we have an array with 6 elements = [9,5,8,4,6,2]
Iteration 1 (with i= 0 , and j at A.length -> 1) :
We would have array as [2,9,5,8,4,6]
Last element has bubbled to first by doing a constant exchange with neighbours as
It is the shortest element ('2')
i=1 , j = A.length -> 2 .

Similarly 4 has bubbled to second position
Array , would be [2,4,9,5,8,6]

i=1 , j = A.length -> 3
Array would be [2,4,5,9,6,8]
Elements with values 5 and 6 have exchanged their positions as shown above

Owing to similar interchange
i=1 , j = A.length -> 4
Array would be [2,4,5,6,9,8]

Subsequent interchanges would lead to array being sorted .
[2,4,5,6,8,9] .

Algorithm in Problem 3 ends up sorting the given input elements of array in ascending order by a mechanism of continuous exchange of elements .

B . Analysis of algorithm :

```
For i=1 to A.length - 1
  for j=A.length to i+1
     if A[j] < A[j-1]
        exchange with A[j] and A[j-1]
```

When i=0 ,
We have algorithm like this :

```
  for j=A.length to 1          ( n-1 )
     if A[j] < A[j-1]            ....  c1
        exchange with A[j] and A[j-1]    ... c2
```

It takes around (c1+c2) (n-1) steps ~ c11(n-1).
For i=1 , It would eventually take ~ c12 (n-2)
..
..
i=n , It would take c1n steps

Please note c11 … c1n are constants
Since it is a continuos loop , running time of entire algorithm would be
C11 (n-1) + c12 (n-2) + … + c1n = k1n^2 - k2(n(n-1)/2)
Based on above equation , worst case running time would be proportional to n^2

Problem 4 .

Please find recursive version of insertion-sort algorithm as below .

```
INSERT(A,start_index , end_index , element_index )
 iter_index = end_index
 while(A[element_index] < A[iter_index])
     interchange ( A[element_index] , A[end_index] )
     element_index = iter_index
     iter_index = iter_index - 1

INSERTION_SORT(A,start_index,end_index)
  if(start_index < end_index)
     INSERTION_SORT(A,start_index, end_index-1)
     INSERT(A,start_index,end_index-1,end_index)
```

From the above algorithm , we can frame below recurrence equation :

T(n) = T(n-1) + c11X(1…n-1)
X is a random variable which can take any value from 1 to n-1 as the insertion loop is constrained based on the exchange of element with those

of other elements and it depends on array characteristics

Similarly
T(n-1) = T(n-2) + c12X(1…n-2)
..
..
T(1) = c
By Induction as shown above only considering **worst case** ,

We get T(n) = C11X(1…n-1) + C12X(1…n-2) + … c
Thereby , T(n) is proportional to n^2 or
In simple asymptotic notation of running time would be O(n^2)