BSE-SA Roll = 21k-3881

Assignment 01 (Algo)

Q1)

9, 19, 14, 29, 13, 4, 23, 10, 37

∴ k = 19

19, 9, 14, 29, 13, 4, 23, 10, 37

∴ k = 14

19, 14, 9, 29, 13, 4, 23, 10, 37

∴ k = 29

29, 19, 14, 9, 13, 4, 23, 10, 37

∴ k = 13

29, 19, 14, 13, 9, 4, 23, 10, 37

∴ k = 4

29, 19, 14, 13, 9, 4, 23, 10, 37

∴ k = 23

29, 23, 19, 14, 13, 9, 4, 10, 37

∴ k = 10

29, 23, 19, 14, 13, 10, 9, 4, 37

∴ k = 37

37, 29, 23, 19, 14, 13, 10, 9, 4

For time Complexity:

~~$T(n) = C_1 n + C_2(n-1) + \dots +$~~

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n} (t_j) + C_6 \sum_{j=2}^{n} (t_j - 1) +$$

21k-3881

$$C_7 \sum_{j=2}^{n} (t_{j-1}) + C_8(n-1)$$

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\left(\frac{n(n+1)}{2} - 1\right) + C_6\left(\frac{n(n-1)}{2}\right) + C_7\left(\frac{n(n-1)}{2}\right) + C_8(n-1)$$

$\Rightarrow an^2 + bn + c$

function of n

$T(n) = O(n^2)$

Loop Invariant:

Initialization:

→ Before the 1st iteration $j=2$ the subarray $A[1 \cdots j-1] = A[1]$ (the element originally in $A[1]$) is sorted.

Maintanance:

The while inner loop moves $A[j-1]$, $A[j-2]$, $A[j-3]$ and so on, by one position to the right position until the proper position for key (which is the value that started out in $A[i]$) is found.

→ At that point the value of key is

21k-3881

laced into this position.

Termination:

The outer for loop end when $j=n+1$

$\Rightarrow j-1=n$.

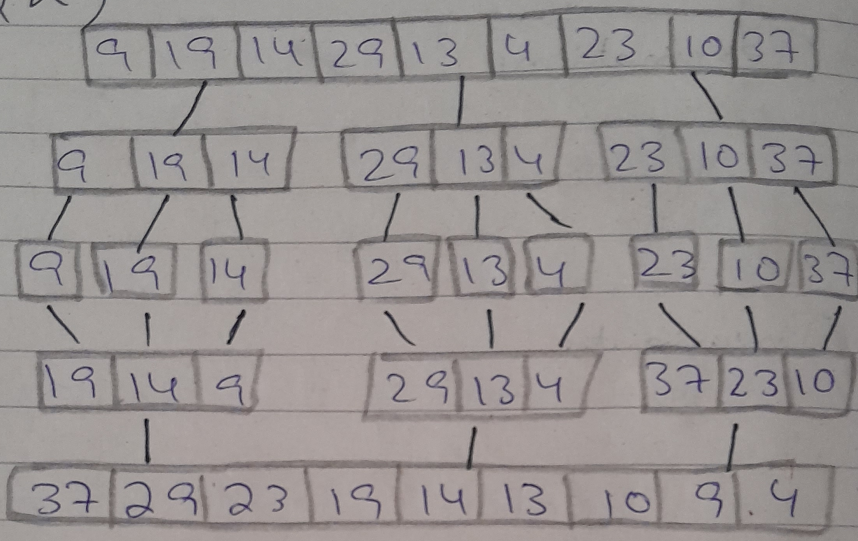
->Replace n with $j-1$ in the loop invariant.

Invariant:

At the start of the for loop the elements in $A[1 \cdots j-1]$ are in sorted order.

21k-3881

Q2)



→ Using Master Theorem to find time complexity.

$T(n) = a\,T(n/b) + Cn$

$a = 3, b = 3, d = 1$

$a > b^d \iff 3 > 3$ ×

$a < b^d \iff 3 < 3$ ×

$a = b^d \iff 3 = 3$ ✓

$\Rightarrow O(n^d \log n)$

$O(n \log n)$

✱

21k-3881

(Q3)

9, 19, 14, 29, 13, 4, 23, 10, 37

Sol)

⇒ 9, 19, 14, 29, 13, 4, 23, 10, 37

Pivot element

⇒ 37, 9, 19, 14, 29, 13, 4, 23, 10

Pivot element

⇒ 37, 19, 14, 29, 13, 23, 10, 9, 4

Pivot element Pivot element

⇒ 37, 29, 23, 19, 14, 13, 10, 9, 4

Loop Invarient:

Initialization:

Before the loop starts all the conditions are satisfied due to the subarrays A[P....i] and A[i+1...j-1] are null.

Maintanance:

during the loop is running; if A[i] ≤ pivot element then ~~Cod~~ A[j] and A[i+1] are swapped and i and j are incremented. If A[j] > pivot element then increment only j.

21K-3881

Termination:

When the loop terminates, j=r all elements in A are partitioned into one of the three cases : $A[p \dots i] \leq$ pivot, $A[i+1 \dots r-1] >$ pivot and $A[r] =$ pivot.

The last two lines of partition move the pivot element from the end of array b/w two subarrays

This is done by swapping pivots and first element of the 2nd subarray by swapping $A[i+1]$ and $A[r]$.

× ×

21k-3881

Q4)

(i) $O(n^2)$:

Sol)

```
Bubble sort (players)
n = length (players)
for i = 0 to n-1 do
for j = 0 to n-i-1 do
if players[j].rating < players[j+1].
    rating then
  swap player[j] with player[j+1]
  end if
  end for
  end for
Bubble sort (players)
  n = length (players)
  team 1 = [ ]
  team 2 = [ ]
 for i in range (2 x n)
    if i x 2 == 0
    if sum(team 1) <= sum (team 2)
     team 1 . append (rating [i])
    else
 team 2 . append (rating [i])
return team 1, team 2
```

(ii) $O(n \log_2 n)$

Sol)

```
players sort(reverse = true)
    n = length(player)/2
    team1 = []
    team2 = []
    team1_rating = 0
    team2_rating = 0
    for i in range(2 x n)
        if i % 2 == 0
        if team1_rating <= team2_rating:
        team1.append(player[i])
        team1_rating += player[i]
    else:
    team2.append(playerss[i])
    team2.rating += playerss[i]

    return team1, team2
```

→ We have a pool of 2n players with numerical ratings

→ Sorted by sorting players in descending order based on their talent ratings.

→ Once the player is sorted, you can divide them into two teams, team1 and team2, while trying to maintain a balance in talent. Initialize both teams.

21k-3881

(Q5)

Sol) In order to minimize the sum, pair up the numbers in an array that largest number would be added to the smallest element of array. Thus we can use Merge Sort to sort elements then selecting an element from odd index (1, 3, 5, 7, n-1) and other from even index (2, 4, 6, 8, n) of sorted array.