Name : Radeep. T

Roll No : 20161145

Assignment - 2 Algorithms

1.

For the given problem,

It is similar to mergesort but instead the problem is divided into 3 parts rather than '2'.

$$\Rightarrow T(n) = K + T(n/3)$$

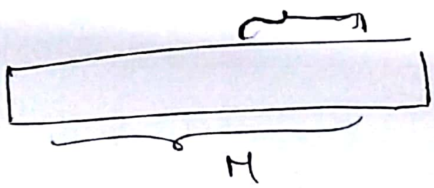
$$a = 1 \quad b = 3$$

$$n^{\log_b a} = 1 = \Theta(1) = n^0$$

$\Rightarrow$ By applying masters theorem,

$$T(n) = \Theta(n^0 \log_b n) = \Theta(\log_3 n)$$

2. The given list is of size M, given that among them 's' elements are not sorted.



let n_i be the number belonging to the set 's' which is arbitarily choosen.

while running the loop the maximum number of element should be checked with n_i during insertion sort = (M-1)
[i.e if n_i is at last in unsorte

array and is least among all elements.]

Since n_i is arbitarily choosen element,

$$T(n) = s \times O(M)$$

$$T(n) = O(sM)$$

3.

1. $T(n) = n! \times O(n)$

Since $f(n)$ is to be calulated everytime while running the for loop.

$n! \leq n \cdot \ldots n$ times

$\Rightarrow n! = O(n^n)$

$\Rightarrow T(n) \leq n! \times kn$

$\Rightarrow T(n) \leq k(n+1)! \leq k\, n^{n!}$

2. Similar to above,

$T(n) = O(n) \cdot n \leq kn^2 = \Theta(n^2)$

3. $T(n) = O(n^2) \cdot n \leq kn^3 = O(n^3)$

4. $T(n) = O(1)$, the loop will not run.

Scanned by CamScanner

4.

1. $T(n) = \sqrt{n}\, T(\sqrt{n}) + 100n$

divide with n

$$\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 100$$

$n = 2^m \Rightarrow m = \log n$

$$\frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 100 \qquad \text{let } f(m) = \frac{T(2^m)}{2^m}$$

$\Rightarrow f(m) = f(m/2) + 100$

$\Rightarrow f(m) = \theta(100 \log_2 m)$ [$\because$ By masters theorem]

$\Rightarrow T(2^m) = \theta(2^m\, 100 \log_2 m)$

$\Rightarrow T(n) = \theta(n \log_2 \log_2 n)$

$\Rightarrow T(n) = \theta(n \log\log n)$

Scanned by CamScanner

3. $T(n) = T(n-2) + \log n$

$T(n) \rightarrow \log n$

$T(n-2) \rightarrow \log(n-2)$

$T(n-4)$ ⋮ } number of terms $= n/2 - 1$

$\log(2)$

$\Rightarrow T(n) = \log n + \log n-2 + \cdots + (n/2 - 1)$ terms

$\Rightarrow T(n) = \log(n \cdot n-2 \cdot n-4 \cdots)$ $(n/2-1)$ terms

we know that

$n \cdot n-2 \cdot n-4 \cdots \le n \cdot n \cdot \cdots$

$\Rightarrow T(n) \le k \log(n \cdots (n/2-1) \text{ times})$

$\Rightarrow T(n) \le k \log(n^{n/2-1})$

$\Rightarrow T(n) \le k_1 n \log n$

$\Rightarrow T(n) = O(n \log n)$ --- (1)

Also,

$n \cdot n-2 \cdot n-4 \cdots \ge k(n/2 \cdots)$ $(n/4 - 1)$ times

Scanned by CamScanner

$$\Rightarrow \quad n \cdot n-2 \cdot n-4 \cdots \geq k\left(n/2\right)^{n/4}$$

$$\Rightarrow \quad n \cdot n-2 \cdots = \Omega\left(\left(n/2\right)^{n/4}\right)$$

Apply log on both sides

$$\Rightarrow \quad \log(n \cdot n-2 \cdots) \geq \log\left(\left(n/2\right)^{n/4}\right)$$

$$\Rightarrow \quad \log(n \cdots) \geq \left(n/4\right)\log n/2$$

$$\Rightarrow \quad T(n) \geq k \, n/4 \log n/2$$

$$\Rightarrow \quad T(n) \geq k_1 \, n \log n$$

$$\Rightarrow \quad T(n) = \Omega(n \log n) \quad \cdots (2)$$

from (1) and (2)

$$T(n) = \Theta(n \log n)$$

5.

1. Sudo code:

from the given it is obvious that 'm' is largest element in the array.

lower Index, larger Index

get-max(A[], l_1, l_2):

$$mid = l_1 + \frac{l_2 - l_1}{2}$$

if A[mid] > A[mid+1] && A[mid] > A[mid-1]:
  return mid

else if A[mid] > A[mid-1]:

get-max(A[], mid+1, h2)

else

get-max (A[], l1, mid-1)

return.

The above algorithm will give the required answer.

2.

Since the above problem is a divide and conquour algorithm,

$T(n) = T(n/2) + \theta(1)$ [∵ only half of the array was checked that of previous one]

$a=1 \quad b=2$

$n^{\log_2 1} = n^0 = 1 = \theta(1)$

By applying masters theorom

$T(n) = \theta(\log n)$

3. <u>Loop Invariant</u>:

In the recursion tree at any stage the maximum element lies between the indices h1 and h2.

In every recursion if the middle element is larger than its neighbourhood elements than that is the required answer.

a) Initialization :

Initially the search indices are from 0 to n-1. It is obvious that the maximum element lies in this range.

$\Rightarrow$ Initialization is true.

b) Maintenance:

Assume maximum element lies b/w ~~[illegible]~~ l_1, l_2

then in this recursive call first,

we check if $A[\frac{l_1+l_2}{2}]$ is greater than ~~the~~ its neibhour elements if it is true then it is answer.

(or)

If right side ~~[illegible]~~ number is more than $A[\frac{l_1+l_2}{2}]$ then check in the array in range from $(\frac{l_1+l_2}{2}, l_2]$.

~~(or~~ else:

we check in the range of $[l_1, \frac{l_1+l_2}{2})$.

Thus even though we didn't find max value in this iteration we know that the maximum value lies in the range called by the caller function.

c) Termination:

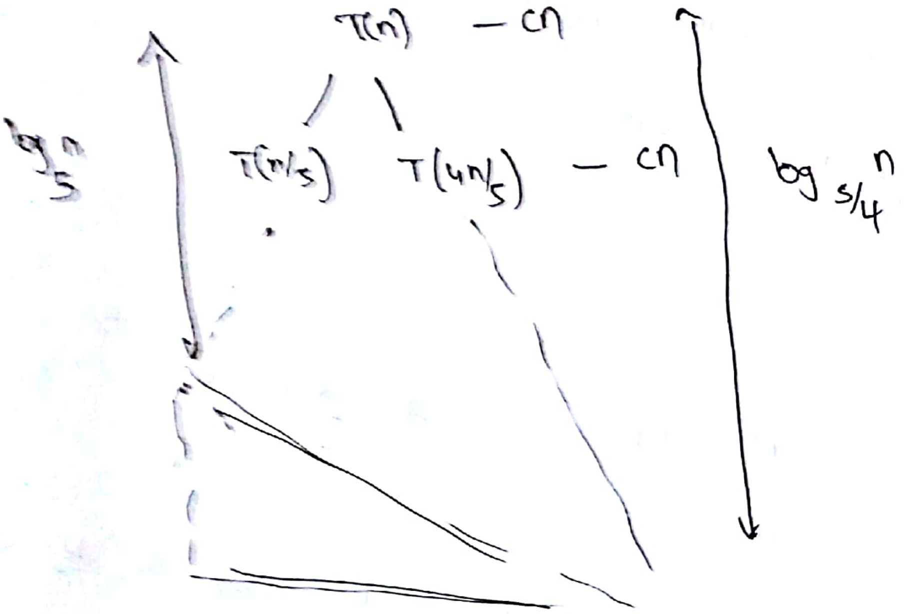
~~[illegible] every happen [illegible] [illegible] ways~~ ~~[illegible]~~ Since we check for the max ~~[illegible]~~ element in array by applying those checks in every call (whether the middle element is larger than both its neighbours) if it returns any number then that number is maximum number.

Hence at termination it gives the required maximum element.

Q.

2. $T(n) = T(n/5) + T(4n/5) + \theta(n)$

solving by using reccurrence tree method



at every level sum is cn .

∴ $T(n) \le cn \times \log_{5/4} n$ [∵ considering height as $\log_{5/4} n$ for Big O]

⇒ $T(n) = O(n \log n)$ — 1

similarly if we take heights as $\log_5 n$

then

$cn \log_5 n \le T(n)$

⇒ $T(n) = \Omega(n \log n)$ — 2

from 1 and 2

$T(n) = \theta(n \log n)$