Junaid Girkar
60004190057

AOA TT1

27/2/2021

Q1 Calculate the time complexity of following recurrence relationship.

a] $T(n) = 2^n T(n/2) + n^n$

Here $a = 2^n$, $b = 2$, $f(n) = n^n$

∵ Here $a = 2^n$ so a is not constant.

∴ We cannot apply master theorem.

b] $T(n) = 3T(n/4) + n \log n$

Here $a = 3$, $b = 4$, $f(n) = n \log n$

$\therefore \log_b a = \log_3 4 \cong 0.793$

Also comparing $f(n)$ with n^k we get $k = 1$

$\because k > \log_b a$

∴ By Master theorem $T(n) = \theta(n^k \log n) = \theta(n \log n)$

c] $T(n) = 0.5\, T(n/2) + 1/n$

On comparing $T(n)$ with $T(n) = a\, T(n/b) + f(n)$

we get, $a = 0.5$, $b = 2$, $f(n) = \frac{1}{n}$

$\because a < 1$; ∴ Master theorem is not applicable.

d] $T(n) = 6T(n/3) + n^2 \log n$

On comparing $T(n)$ with $T(n) = a\, T\left(\frac{n}{b}\right) + f(n)$

we get; $a = 6$, $b = 3$, $f(n) = n^2 \log n$

$\therefore \log_b a = \log_3 6$

$\cong 1.631$ //

Q2 How would you calculate time complexity of the following recurrence relationship using recurrence tree?

$T(n) = 2T(n/3) + n$

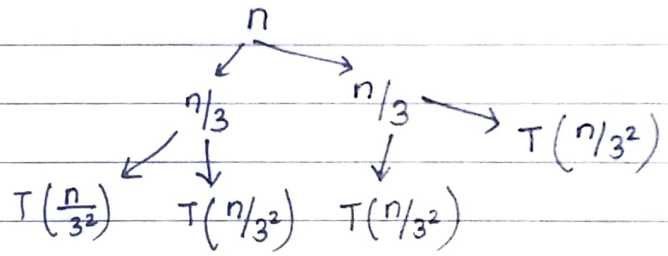
ANS The given recurrance relation is $T(n) = 2T(n/3) + n$

Here initially the problem is of size n. So overall time complexity would be $T(n)$.

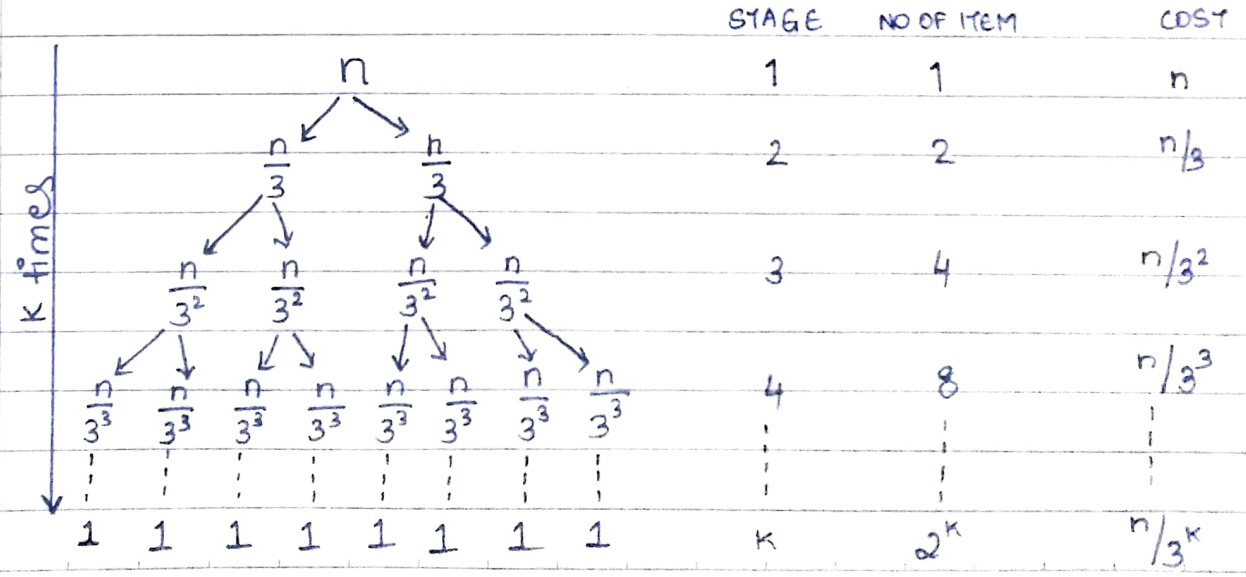
Now in second step $T(n)$ will be divided further as n and $2T(n/3)$

$\therefore T(n) \Rightarrow$ n — $T(n/3)$, $T(n/3)$

In next time $T(n/3)$ will be divided as $T(n/6)$ and $n/3$



Further more $T(n/3^2)$ will be divided as $T(n/3^3)$ and $n/3^2$, this will continue until we get 1.



Junaid Girkar

Now the process will continue upto

$$\frac{n}{3^k} = 1 \Rightarrow n = 3^k$$

$$\therefore k = \log_3 n$$

∴ The time complexity at stage 1 is n, At stage 2 is $2^{n/3}$ and at stage 3 the total complexity is $4^{n/3^2}$, at stage 4 it is $8^{n/3^3}$ and so on.

∴ Total complexity is : $n + \frac{2n}{3} + \frac{4n}{3^2} + \frac{8n}{3^3} + \dots$ (K times)

This is sum of GP with $a = 1$, $r = 2/3$ and $N = K$

$$\therefore n\left(1 + \frac{2}{3} + \frac{4}{3^2} + \frac{8}{3^3} + \dots \text{ K times}\right)$$

$$\therefore n\,\frac{(1 - (2/3)^K)}{1 - 2/3}$$

$$\therefore 3n\left(1 - \left(\frac{2}{3}\right)^{\log_3 n}\right)$$

$$\therefore 3n\,(1 - n^{\log_3(2/3)})$$

$$\therefore 3n\,(1 - n^{\log_3 0.667})$$

$$\therefore 3n\left(1 - \left(\frac{1}{n}\right)^{0.3691}\right)$$

$$\because 1 >> \left(\frac{1}{n}\right)^{0.3691}$$

∴ The overal complexity will be $\approx 3n$ which falls under the class $O(n)$

∴ The time complexity is $O(n)$.

Q3 a]

```
int fun(int n)
{
    int count = 0;
    for (int i=n; i>0; i /= 2)
        for (int j=0; j<i; j++)
            count += 1;
    return count;
}
```

Analyze the time complexity of the above function.

ANS Let us consider the outer for loop i.e. for(int i=n; i>0; i /= 2).

Now for particular value of i.

let us say i=n; the inner for loop runs 1 time.

Now lets consider the inner for loop i.e for(int j=0; j<1; j++)

count += 0

Now since the value of i is 8 so the for loop will run 8 times so n times.

Similarly for $i = n/2$ i.e. $i = 4$ the inner loop will run $n/2$ and so on.

∴

```
int fun(int n) {
    int count = 0;                          ——— O(1)
    for (int i=n; i>0; i /= 2)
        for (int j=0; j<1; j++)
            count += 1;
    return count;                           ——— O(1)
}
```

∴ T(n) = complexity of inner for loop.

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots + 2$$

∵ At each time n is halfed

$$\therefore T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots + 2$$

3.1

For worst case time complexity we consider upto infinity.

$\therefore T(n) = \frac{n}{1-1/2} + 2 = 2n + 2$

$\therefore$ The time complexity is $O(n)$

b] $f_1(n) = n^{0.999\,999} \log n$

$f_2(n) = 1000000\, n$

$f_3(n) = (1.00001)^n$

$f_4(n) = n^2$

Since $f_3(n)$ is exponential, it will be largest amongst all.

Since $f_4(n)$ is quadratic, it will be greater than $f_1(n)$ and $f_2(n)$ but lesser than $f_3(n)$

$\therefore f_3(n) > f_4(n) > \dots$

Now, for $f_1(n)$ and $f_2(n)$

$f_2(n) = 1000000\, n \simeq n$ [constant can be neglected]

$= n^{0.999999} \cdot n^{0.000001}$

Removing common part of $f_1(n)$ and $f_2(n)$

$f_1(n) = \log n$

$f_2(n) = n^{0.000001}$

For large n ; $f_2(n) > f_1(n)$

$\therefore$ Order of time complexities :

$$\boxed{f_3(n) > f_4(n) > f_2(n) > f_1(n)}$$

c] Given four different function use a simple for loop within the for loop, same set of statements are executed. Consider the for loops : A) for (p=0 ; p<n ; p++)

B) for (i=0 ; i<n ; i+=2)

C) for (i=1 ; i<n ; i*=2)

D) for (i=n ; i>-1 ; i/=2)

If n is the size of input (positive) which is most efficient, if task to be performed is not an issue. explain the reason ?

ANS a] for (p=0 ; p<n ; p++)

This loop runs from p=0 to n-1 with increment 1 each time.

∴ n times

∴ time complexity = O(n)

b] for (i=0 ; i<n ; i+=2)

This loop will run from 0 to n and in each step the value of i increment by 2, i take even value i.e. 0, 2, 4, ...

∴ The steps is $n/2$

∴ time complexity = O(n)

c] for (i=1 ; i<n ; i*=2)

This loop will run from 0 to n and in each step i is multiplied by 2, i takes values 1, 2, 4, 8, ..., n-1

∴ $2^K = n-1$, $K = \log_2(n-1)$

∴ Time complexity is $O(\log_2 n)$

d] for (i=n ; i>-1 ; i/=2)

In this loop, i takes values $n, \frac{n}{2}, \frac{n}{4}, \ldots$ but $\frac{n}{2^i}$ is always greater than 0

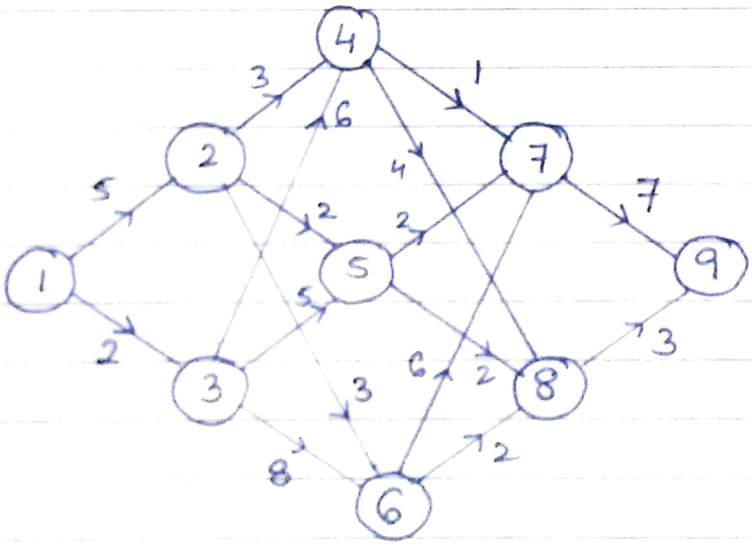
but the condition is $i > -1$

∴ It is an infinite loop.

∴ From all the time-complexity, complexity of $O(\log_2^{(n-1)})$ is most efficient which occurs in case c.

∴ The loop for (i=1 ; i<n; i*=2) is more efficient.

Q4 Using dynamic programming find the minimum distance from source to sink of following multistage graph.



ANS

VERTEX	1	2	3	4	5	6	7	8	9
DISTANCE	12	7	10	7	5	5	7	3	0
d	9	9	9	9	9	9	9	0	9

We know that, $dist(i, p) = \min \{ c(p,q) + dist(i+1, q) \}$

Also $p \in V_i$, $q \in V_{i+1}$, $\langle p, q \rangle \in E$

Here total number of levels is 5.

$\therefore$ dist (5,9) = 0 — LEVEL 5

$\therefore$ dist (4,7) = 7
dist (4,8) = 3 — LEVEL 4

AT LEVEL 3 :

$$dist(3,4) = \min \begin{Bmatrix} c(4,7) + dist(4,7) \\ c(4,8) + dist(4,8) \end{Bmatrix}$$

$$= \min \{ (1,7), (4+3) \}$$

dist (3,4) = 7

We select node 8 from level 4 as source 3. Cost is minimum.

$$dist(3,5) = \min \{ c(5,7) + dist(4,7), c(5,8) + dist(4,8) \}$$

$$= \min \{ (2+7), (3+2) \}$$

dist (3,5) = 5

We select node 8 from level 4 source 5 as cost is minimum.

$$dist(3,6) = \min \{ c(6,7) + dist(4,7), c(6,8) + dist(4,8) \}$$

$$= \min \{ (6+7), (2+3) \}$$

dist (3,6) = 5

We select node 8 from level 4 source 6 as cost is minimum.

AT LEVEL 2

$$dist(2,2) = \min \{ c(2,4) + dist(3,4), c(2,5) + dist(3,5), c(2,6) + dist(3,6) \}$$

$$= \min \{ (3+7), (2+5), (3+5) \}$$

dist (2,2) = 7.

We select node 5 from level 3 and source 2.

$dist(2,3) = \min\{ c(3,4) + dist(3,4),\ c(3,5) + dist(3,5),\ c(3,6) + dist(3,5) \}$

$= \min\{(6+7), (5+5), (8+5)\}$

$dist(2,3) = 10$

We select node 5 from level 3 source 3.

AT LEVEL 1

$dist(1,1) = \min\{ c(1,2) + dist(2,2),\ c(1,3) + dist(2,3) \}$

$= \min\{(5+7), (2+10)\}$

$dist(1,1) = 12.$

We can select node 3 or node 2 from level 2 as cost is minimum for both of them.

∴ The path is $1 \to 3 \to 5 \to 8 \to 9$ or $1 \to 2 \to 5 \to 8 \to 9$

∴ The minimum distance from source to sink is 12.

Q5 Analyse Quick sort and derive its time complexity for best case and worst case.

ANS Quick sort is a sorting algorithm based on idea of divide and conquer algorithm.

In quick sort we find a pivot element then bring pivot to its appropriate position.

After that we quick sort left part and right part.

The above two steps we peryorm recursively.

Here the junction quick soxt to partition.

```
T(n) → int partition ( int array [], int low, int high)
       {
            1 → int pivot = array [high];
            1 → int i = low - 1;
          n+1 → for (int j = low; j < high; j++) {
            n →       if (array [j] <= pivot) {
            n →             i++;
            n →       swap(array [i], array [j]); }}

          1→swap ( array [i+1], array [high])
          1→ return i+1 }
```

∴ The total complexity, $T(n) = 1+1+n+1+n+n+1+1$

$T(n) = 3n + 5$

$T(n) = O(n)$

∴ The time complexity of partition is to n.

Now for the quick sort function.

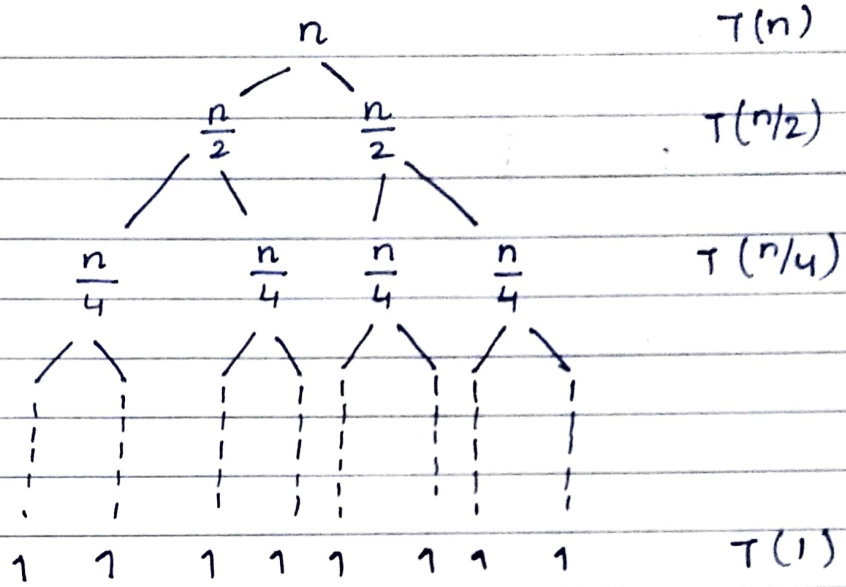
```
T(n) → void quicksort ( int array, int low, int high) {
            if (low < high) {
         n →    int pi = partition (array, low, high);
     T(n/2) →   quicksort (array, low, pi - 1);
     T(n/2) →   quicksort (array, pi+1, high);
            }}
```

As the complexity of partition is n, ∵ we don't know what will be the position of pivot we consider recursively call of quicksort has complexity approximately to $n/2$ for average case.

JUNAID . GIRKAR

∴ For best case.

For best case we consider pivot to be in middle location.



$$\therefore T(n) = 2T(n/2) + n$$

∴ By master theorem

$a = 2$, $b = 2$, $f(n) = n$

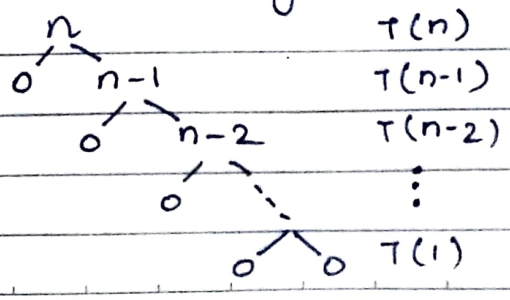
$$\therefore \log_b a = \log_2 2 = 1$$

Also comparing $f(n)$ with n^k we get $k = 1$

$$\therefore T(n) = \Theta(n^{\log_b a} \log n)$$
$$= \Theta(n \log n)$$

∴ For worst case

For worst case we consider array to be divided at n-1 to part as pivot is either first or last end.



$\therefore T(n) = T(n-1) + n-1$

$\therefore$ By substitution method

$T(n) = T(n-2) + 2n-2$

$T(n) = T(n-3) + 3n-3$

$\vdots \quad \vdots$

$T(n) = T(n-K) + Kn-K$

$\therefore n-K = 1$

$K = n-1$

$T(n) = T(1) + (n-1)n - K$

$T(n) = n^2 - n - n + 1$

$= n^2 - 2n + 1$

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

For worst case, complexity is $\Theta(n^2)$.