11/4/24/Thursday.

Page :
Date :

Algorithm:

Finite set of instructions to solve a problem.

Design methods / strategies:

1. Divide & conquer
2. Backtracking
3. Dynamic Programming
4. Branch & Bound

Analysis strategies:

1. Time complexity
2. Space complexity.

Real life Algorithm Usage:

1. Google Map ← Shortest path algorithm (eg: Dijkstra, Bellmann Ford, Floyd warshall)
2. Shopping sites ← Recommendation algorithms (eg: apriori)
3. Youtube ← Recommendation algorithm

Algorithm v/s Program:

Algorithm	Program
1. Design Phase	1. Implementation Phase
2. Domain specific knowledge	2. Programmer
3. Natural language; no specific syntax	3. In programming language, ~~no~~ specific syntax
4. Analysis	4. Testing

Q. What are the main properties of an algorithm?

↳ 1. Finite
2. Unambiguous
3. Effective / Efficient
4. input
5. output

1* **Input:**
For an algorithm, it has 0 or more inputs.

2* **Output:**
Should have atleast 1 output

3* **Unambiguous:**
- Should not have any confusion
- An algorithm should be enclosed within start/begin and stop/end statements.

eg: algorithm for sum of 2 numbers:

Start
Step 1: Read a } or Step 1: Read a, b
Step 2: Read b }
Step 3: sum :- a+b
Step 4: print sum
End

4* **Finite:**
Number of steps are countable.

5* **Effective/Efficient:**
Avoid unnecessary steps.

Q. What is the need of an algorithm?

↳ For time saving. It provides a plan on how to make the program.

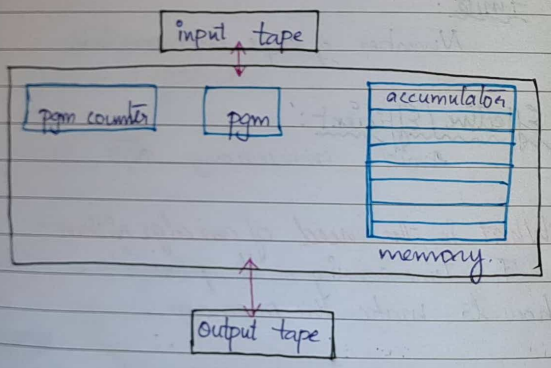
Q. Algorithm for largest number among 3 numbers?

Start
Step 1: Read a, b, c
Step 2: If a > b then max = a else max = b
Step 3: If c > max then max = c
Step 4: print max
End

Analysis Of Algorithms:

1. Factors included in the Analysis → depends on f(n)
2. Types of Analysis → priori, posteriori
3. Asymptotic Analysis → Big Oh, Big Omega, Big Theta

RAM Machine / Model:



We need RAM Machine for finding f(n) (for time complexity)

We are counting the number of primitive operations in the pgm or algorithm. It is basically RAM model.

I/p & O/p tape are groups of cells. →

Based on the ~~map~~ mapping of i/p tape & o/p tape we get ∄ f(n) [the number of primitive operations]

Say,

eqn ⇒ $5n^2 + 6n + 2$

Q. What is the role of n^2, n & 2 in the value?

Let n = 1

Role of n^2 = $\frac{5}{5\times1^2+6\times1+2} \times 100 = \frac{5}{13} \times 100 = 38.46\%$

Role of n = $\frac{6}{13} \times 100 = 46.15\%$

Role of 2 = $\frac{2}{13} \times 100 = 15.38\%$

n	Role of n^2	Role of n	Role of 2
1	38.46	46.15	15.38
100	9881	0.01	0.003
1000	⋮	⋮	⋮
⋮			

From this we can see n^2 has more role. From this we approximate time complexity to $O(n^2)$

2# Type of Analysis:

Priori	Posteriori
• Before execution	After execution
• Independent of hardware ~~memory, processing time~~	Dependent on hardware, memory & processing time.
• Algorithm	Program
• Approximation	Actual

3# Asymtotic Notations.

The approximation is the asymtotic notations.
Big Oh - worst case Big Theta - avg case
Big Omega - best case

Best case:
Takes the least time for execution.
Number of primitive operations is least.

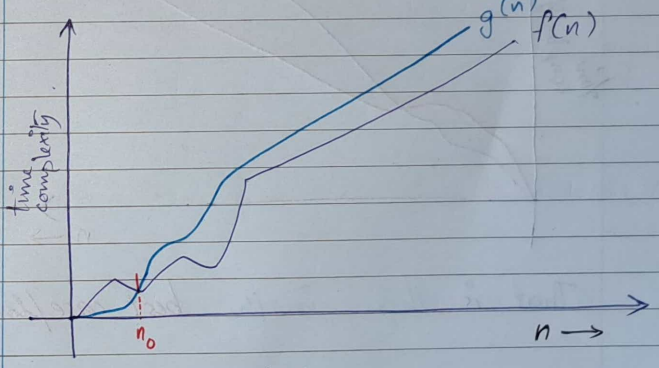
Worst case:
Maximum no: of primitive operations occurs.

Average case:
This is between best & worst case.

Graphical:

1. Big Oh:

$f(n) \leq c\,g(n)$



$n \geq n_0 \geq 1$

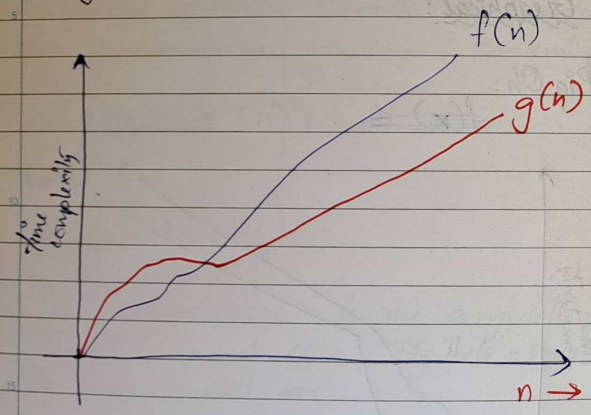
That is - this is max time taken.
That is: Worst case / upper bound.

2. Omega.

- $g(n) \leq cf(n)$

Or

$cg(n) \leq f(n)$



That is this is the best case/lower bound.

3. Theta

Average case.

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

18/4/24/Thursday.

$$f(n) = \Theta(g(n))$$

Q. Suppose you have two functions $f(n) = 4n+3$ and $g(n) = n$. Check whether $f(n) = O(g(n))$ is true or not? If true find c and n_0?

Ans: Two methods to solve:

1. Table method.

n	$f(n) = 4n+3$	$g(n) = n$	$2n$ $5n$ $7n$
1	7	1	2 5 7
2	11	2	4 10 14
3	15	3	6 15 21
4	19	4	8 20 28
5	23	5	10 25 35
6	27	6	12 30 42
7	31	7	14 35 49

Camlin

$f(n) \leq O(g(n))$ → $c \cdot g(n) = c \cdot n$

$4n+3 \leq c \cdot n$

When $c=1$

$4n+3 \leq n$

↳ Not possible. Proof: Hash table.

When $c=2$

$4n+3 \leq 2n$ → From table we can see it is not possible.

⋮

$4n+3 \leq 5n$ for $n \geq 3$

Select min: 'c' value. Hence we take $c=5$.

$\therefore c=5 \quad n_0 = 3$

Hence $f(n) = O(g(n))$

where $c=5$ and $n_0 = 3$

2. Shortcut method.

1. Take highest order ~~of~~ coefficient.

④$n+3$

2. Add 1

$\therefore c = 4+1 = 5$

Will not work for all problems

$4n+3 \leq c \cdot n$

$4n+3 \leq 5n$

$3 \leq 5n-4n$

$n=3 \longrightarrow n_0$

Camlin

$\therefore 4n+3 = O(n)$

where $c=5$

$n_0 = 3$

Q. $f(n) = n+4$, $g(n) = n$.

A:

①$n+4$

↳ $c = 1+1 = 2$

$n+4 \leq 2n$

$4 \leq n_0 \longrightarrow n_0 = 4$

$\therefore n+4 = O(n)$

where $c=2$

$n_0 = 4$

Q. $f(n) = n^2+n+3$, $g(n) = n^2$

A:

①n^2+n+3

↳ $c = 1+1 = 2$

$n^2+n+3 \leq 2n^2$

$n+3 \leq n^2$

$3 \leq n^2-n$

$0 \leq n^2-n-3$

$n = \frac{1+\sqrt{13}}{2}, \frac{1-\sqrt{13}}{2} = -1.30$

$= 2.30$ ✓

Camlin

$n_0 = \lceil 2.3 \rceil$

$= 3$

OR,

$n^2 - n = 3$

$n(n-1) \geq 3$

~~n=3 or n=~~

$n=1$ Not true

$2(2-1) = 2$ Not true

$3(3-1) = 3 \times 2 = 6 \geq 3$ → true

$\therefore n_0 = 3$

$\therefore$ ~~If~~ $n^2+n+3 = O(n^2)$

where $c = 2$

& $n_0 = 3$

Q. ~~Find~~ $f(n) = n^2$ & $g(n) = 2^n$. Check if $f(n) = O(g(n))$. If yes, find c & n_0?

A:

n	$f(n)=n^2$		$g(n)=2^n$
1	1	<	2
2	4	=	4
3	9	>	8
4	16	=	16
5	25	<	32
6	36	<	64

Camlin

at $c=1$ & $n_0 = 4$,

$f(n) = O(g(n))$

Q. $f(n) = 3n+2$ $g(n) = n$

A:

$c = 3+1 = 4$

~~If~~ $3n+2 \not\leq 4n$

~~$2 \leq n$~~

$4n \leq 3n+2$

$n \leq 2$

$n_0 = 2$

$\therefore f(n) = O(g(n))$ at $c = 4$ & $n_0 = 2$

Q. $f(n) = n^2$ $g(n) = n$

A:

n	$f(n)=n^2$	$g(n) = n$				
1	1	1	2	3	4	5
2	4	2	4	6	8	10
3	9	3	6	9	12	15
4	16	4	8	12	16	20
5	25	5	10	15	20	25
6	36	6	12	18	24	30
⋮	⋮	⋮				

This is not possible.

Hence $f(n) \neq O(g(n))$.

Camlin

Q. $f(n) = 3n+2$ and $g(n) = n$. Find Ω ?

A: Put $c = 1$

Then find n_0 value.

So,

$f(n) \geq c \cdot g(n)$

$f(n) \geq g(n)$

$3n+2 \geq n$

$2n \geq -2$

$n \geq -1$

$n_0 = 1$

n_0 has to be greater than 1.

n	$f(n) = 3n+2$		$g(n) = n$
1	5	$\geq$	1
2	8	$>$	2
3	11	$>$	3
4	14	$>$	4

Q. $f(n) = n$ & $g(n) = 3n+2$. Find Ω ?

A: $c = 1$

$f(n) \geq c g(n)$

$n \geq (3n+2)c$

$-2n \geq 2$

$n \geq -1$

So $n_0 = 1$

We can't take $c=1$ as it won't work.

n	$f(n) = n$	$g(n) = 3n+2$, $c = 1/5$
1	1	5
2	2	8
3	3	11
4	4	14
5	5	17
6	6	20

If $c = 1/10$

$n \geq c \cdot g(n)$

$n \geq \frac{1}{10} \cdot (3n+2)$

$10n \geq 3n+2$

$7n \geq 2$

$n \geq 2/7$

$n_0 = \lceil 2/7 \rceil$

$= \lceil 0.28 \rceil$

$= 1$

0.28
7 | 20
14
60
56
4

★ For logarithmic values we use floor ($\lfloor \rfloor$), instead of ciel ($\lceil \rceil$)

Q. $100n\log n = O(\frac{n\log n}{100})$. Is this true or false. If true find c?

A: $100n\log n \leq \frac{n\log n}{100} \cdot c$

$c = 10^4$

Then,

$100n\log n \leq 10^4 \times \frac{n\log n}{100}$

$100n\log n \leq 100\log n$

That is LHS = RHS. (= also works)

Hence $100n\log n = O(\frac{n\log n}{100})$ proved.

Q. $\sqrt{\log n} = O(\log\log n)$. Check whether it is true?

A:

$\sqrt{\log n} \leq c\log\log n$

$\log n^{1/2} \leq \log\log n$

But $\log n > \log\log n$

So $\log n^{1/2} > \log\log n$

Hence $\sqrt{\log n} \neq O(\log\log n)$

Q. If $0 < x < y$, then prove $n^x = O(n^y)$ g.t. is true or false?

A:

$n^x \leq n^y$

log on both sides

$\log n^x \leq \log n^y$

$x \log n \leq y \log n$

$x < y$.

So the condition is always true.

OR take an example of value for x & y & then prove.

Q. $2n \neq O(nk)$. Check whether true or false?

A: $2n \leq O(nk)$

When $k = 2$ $\quad 2n = O(2n)$

So the stmt $2n \neq O(nk)$ is false.

Q. $(n+k)^m = O(n^m)$

A: $m=2$, $k=3$

$(n+3)^2 = O(n^2)$

$n^2 + 6n + 9 = O(n^2)$

↓ highest order

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

LHS = RHS.

Hence proved.

Q. $2^{n+1} = O(2^n)$. True or False?

A:

$2^{n+1} \leq 2^n \cdot c$

$2^n \cdot 2 \leq 2^n \cdot c$

$\therefore c = 2$

Hence stmt is true.

Q. $2^{2n+1} = O(2^n)$? If true, find c value?

A:

$2^{2n+1} \leq c \cdot 2^n$

$2^n \cdot 2^n \cdot 2 \leq c \cdot 2^n$

$c \geq 2^{n+1}$ (c value has to be constant) ∴ Not possible.

Hence stmt is false.

Q. $X = \sum_{i=0}^{n} i^3$: $O(n^4)$, $O(n^5)$, $\Omega(n^3)$, $\Theta(n^5)$. Which of these are true?

A:

$X = 0^3 + 1^3 + 2^3 + 3^3 + \ldots + n^3$

$= \left[\frac{n(n+1)}{2}\right]^2$

$= \left[\frac{n^2+n}{2}\right]^2 = \frac{n^4 + 2n^3 + n^2}{2}$

↓ $O(n^4)$, $\Omega(n^4)$ and $\Theta(n^4)$

$O(n^4)$ is UB, so $O(n^5)$ would also become an UB

So correct options are: $O(n^4)$, $O(n^5)$, $\Omega(n^3)$.

$\Omega(n^4)$ is LB, and $\Omega(n^3)$ comes inside $\Omega(n^4)$ so this is also true.

20/4/24 | Saturday

Page :
Date :

Q1. If $f(n) = O(g(n))$ then,

$2^{f(n)} = O(2^{g(n)})$ ~~Prove~~ True or False?

Q2. If $f(n) = O(g(n))$ then ~~$f(n) \times h(n)$~~

$f(n) * h(n) = O(g(n) \times h(n))$. True or False.

Answers:

1. ~~Let $f(n) = g(n) = 2n$~~

Let $f(n) = 2n$ $\quad g(n) = n$

$f(n) \leq c \cdot g(n)$

$2n \leq c \cdot n$

$c = 2$

Here I got it in one case but check for 3 cases like,

If $f(n) = n$ & $g(n) = 2n$ ($f(n) < g(n)$)

$f(n) = 2n$ & $g(n) = 2n$ ($f(n) = g(n)$)

$f(n) = 2n$ & $g(n) = n$ ($f(n) > g(n)$)

→ do all three cases

Now, To check: $2^{f(n)} \leq c_2 \cdot 2^{g(n)}$

$2^{2n} \leq c_2 \cdot 2^n$

$2^n \cdot 2^n \leq c_2 \cdot 2^n$

$c_2 = 2^n$

but this is not possible as c is constant & n is variable

Hence false.

2. $f(n) = O(g(n))$

ie; $f(n) \leq c_1 \cdot g(n)$

Now to check if:

$f(n) \cdot h(n) = O(g(n) \cdot h(n))$

ie; $f(n) \cdot h(n) \leq c_2 \cdot g(n) \cdot h(n)$

ie; $f(n) \leq c_2 \cdot g(n)$

↓ which is true when $c_1 = c_2$

Hence stmt is true.

Camlin

Page :
Date :

X. Q3: $f(n) = O(f(n))$. Is it true or false?

A: It is true. This is called Reflexive property.

$f(n) \leq c \cdot f(n)$

~~$f(n) < c \cdot f(n)$~~ $\rightarrow c \geq 1$

Q4: $f(n) = O(f(n/2))$

A: To check if $f(n) \leq c \cdot f(n/2)$

Say $f(n) = n^2$.

then $f(n/2) = (n/2)^2$

Now

$n^2 \leq c \cdot \frac{n^2}{4}$

So when $c \geq 4$, it is true.

Hence this is ~~always~~ true. (We are taking complexities so remove constants).

~~Q5: $f(n)$~~

$f(n) = 2^{2n}$

$f(n/2) = 2^{2 \cdot \frac{n}{2}} = 2^n$

$f(n) \leq c \cdot f(n/2)$

$\Rightarrow 2^{2n} \leq c \cdot 2^n$

$c = 2^n$ which is not possible

Hence this is False.

Camlin

Page :
Date :

Qs. $f(n) = 5n^2 + 6n + 8$. find Big Oh notation?

Ans: $O(5n^2 + 6n + 8)$

$= O(5n^2 + 6n)$ Removal of constants & lower powers.

$= O(5n^2)$

$= O(n^2)$

OR.

$O(5n^2 + 6n + 8)$

$\downarrow \quad \downarrow \quad \downarrow$ maximise

$O(5n^2 + 6n^2 + 8n^2)$

$O(19n^2)$

$O(n^2)$

ie; $c = 19$

$n_0 \geq 1$

24/4/24 | Wednesday.

Page :
Date :

Guidelines for Asymptotic Notations :

1. Loop

```
for (i=1 ; i<=n ; i++) {
    //stmt
}
```

Loop executes n times

Time complexity = O(n)

2. Nested Loop

```
for(i=1 ; i<=n ; i++) {
    for(j=1 ; j<=n ; j++) {
        //stmt executes nxn times.
    }
}
```

executes n times

executes n^2 times

TC = $O(n^2)$

3. Consecutive statements

```
int x = 2 ;
int i ;
x = x+1 ;
for(i=1; i<=n; i++) {
    //stmt
}
for(i=1 ; i<=n ; i++){
    for(j=1; j<=n ; j++){
        //stmt
    }
}
```

1

n

n^2

Camlin

TC = O(1+n+n²)
= O(n²)

4. if-then-else stmts

```
scanf("%d", &n);          → 1
if (n==0){                → 1
    // stmt               → O(1)
}                         O(1+1) = O(1)
else {                    OR
    for(i=1; i<=n; i++){  → n   O(1+n) = O(n)
        // stmt;
    }
}
```

scanf("%d", &n) ↳ test case (if(x==0))

TC = O(1+1+n) ↳ else part.

= O(n)

We generally take worst case time complexity & just take greater value.

5. Logarithmic Time Complexity:

```
for(i=1; i<=n){
    // stmt
    i = i*2
}
```

i	stmt repeats
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
⋮	⋮
2^x	$x+1$

$2^x <= n$ → $2^{x-1} \le n$

$x+1 = \log_2 n$ → $x-1 = \log_2 n$

$x = \log_2 n + 1$

$TC = O(\log_2 n)$ ⟷ $O(\log_2 n)$

* if $i = i*3$ ↳ $O(\log_3 n)$

⋮

ie; $i = i*k$ ↳ $O(\log_k n)$

~~Q. Find time comp~~

6. Logarithmic extra

```
for(i=n; i>=1)
{
    // stmt;
    i = i/2;
}
```

i	iteration
n	1
$n/2$	2
$n/2^2$	3
⋮	⋮
$n/2^{x-1}$	x

$$\frac{n}{2^{x-1}} \geq 1$$

$$n \geq 2^{x-1}$$

$$\log n \geq x-1$$

$$\log_2 n + 1 \geq x.$$

$$O(\log_2 n)$$

$$\therefore TC = O(\log_2 n)$$

$\therefore$ $i = i/k$

$\hookrightarrow O(\log_k n)$

Q1. void fun(int n) {

$O(1)$ ← int i, j, k, count = 0; → steps @ $n/2$ so $O(n/2) = O(n)$

$O(n)$ ← for(i = n/2 ; i <= n; i++) {

for(j = 1 ; j + n/2 <= n; j++) {

$O(\log_2 n)$ ← for (k = 1; k <= n; k = k * 2) {

count++;

} } }

↳ logarithmic rule

1st loop: 2nd loop:

j	$j + n/2 \leq n$
1	↓
⋮	$j = n/2$
$n/2$	

$$O(n/2) = O(n).$$

$$1 + (O(n) \times O(n) \times O(\log_2 n))$$

$$= O(n^2 \log_2 n)$$

25/4/24 | Thursday.

Q1. void fun (int n) {
for (i = n/2 ; i <= n ; i++) { → O(n/2) = O(n)
for (j = 1 ; j <= n ; j = j * 2) { → O(log n)
for (k = 1 ; k <= n ; k++) { → O(n)
count ++;
}
}
}

A: O(n × log n × n) = O(n² log n)

Q2. void fun (int n) {
if (n <= 1) → O(1)
return;
int i, j; → O(1)
for (i = 1 ; i <= n ; i++) { → O(n)
for (j = 1 ; j <= n ; j++) { → O(1)
printf ("hello");
break;
}
}

A: O(n)

Q3. void fun (int n) {
int i = 1; → 1
while (i < n) { → log n
int j = n;
while (j > 0) { → log n
j = j / 2;
}
i = i * 2;
}
}

A: Same as waiting

for (i = 1 ; i < n ; i = i * 2) { → O(log n)
for (j = n ; j > 0 ; j = j / 2) { O(log n)
#stmt
}
}

TC = O(log n × log n) = O(log n)²

Q4. void fun (int n) {
int i, j; → 1
for (i = 1 ; i <= n/3 ; i++) { → O(n/3) = O(n)
for (j = 1 ; j <= n ; j += 4) { → O(n)
printf ("hello");
}
}

j	
1	1
1+4	2
1+4×2	3
1+4×3	4
1+4×(k-1) = n	k
k = (n-1)/4	

A: $O(n^2)$

2nd loop: j	iteration
1	1
1+4	2
1+4x2	3
1+4x3	4
⋮	⋮
1+4x(k-1)	k

$1 + 4x(k-1) \leq n$

$1 + 4k - 4 = n$

$4k - 3 = n$

$4k = n+3$

$k = \frac{n+3}{4}$

$\hookrightarrow O(n)$

TC = $1 + (n * n) = 1 + n^2 = O(n^2)$

Qs.
```
A() {
int
```
```
fun() {
    int i, j, k, n;
    for(i=1; i<=n; i++){
    for
```

Qs.
```
fun() {
int i, j, k, n;                          → 1
for(i=1; i<=n; i++) {                    → n
    for(j=1; j<=i; j++){
        for(k=1; k<=100; k++){
            printf("hello");
        }
    }
}
```

A:

i	j
1	1
2	1, 2
3	1, 2, 3
4	1, 2, 3, 4
⋮	⋮
n	1, 2, 3, ..., n

printf "hello" occurrence:

$100 (1+2+3+4+...+n)$

$\downarrow$

$\frac{n(n+1)}{2}$

$O(n^2)$

OR

i=1	i=2	i=3	... i=k
j=1	j=1,2 (2 times)	j=1,2,3 (3 times)	j=1,2,3,...,k
k=100	k=2x100	k=3x100	k=kx100

$TC = 100 + 2\times100 + 3\times100 + 4\times100 + \cdots + k\times100$

How many times printf("Hello") occurs.

$= 100\left(\frac{k\times(k+1)}{2}\right)$ (Because $i=n \Rightarrow$ termination so $k=n$)

$= O\left(\frac{n(n+1)}{2}\right)$

$= O(n^2)$

Q6.
```
fun() {
  int i, j, k, n;
  for(i=1; i<=n; i++){
    for(j=1; j<=i^2; j++){
      for(k=1; k<=n/2; k++){
        printf("Hello");
      }
    }
  }
}
```

Camlin

A.

i=1	i=2	i=3	...
j=1	j=1,2,3,4	j=3^2	
k=n/2 times	k=$4\times\frac{n}{2}$	k=$3^2\times\frac{n}{2}$	

$TC = \frac{n}{2} + 4\frac{n}{2} + 3^2\frac{n}{2} + \cdots + k^2\cdot\frac{n}{2}$

$= \frac{n}{2}\left(1^2+2^2+3^2+\cdots+k^2\right)$ ($k=n$: termination)

$= \frac{n}{2}\left(1^2+2^2+3^2+\cdots+n^2\right)$

$= \frac{n}{2}\left[\frac{n(n+1)(2n+1)}{6}\right]$

$= O(n^4)$

Q7.
```
fun(){
  int i, j, k;                          ---- 1
  for(i=n/2; i<=n; i++){                ---- n
    for(j=1; j<=n; j=2*j){              ---- log n
      for(k=1; k<=n; k=k*2){            ---- log n
        printf("hello");
      }
    }
  }
}
```

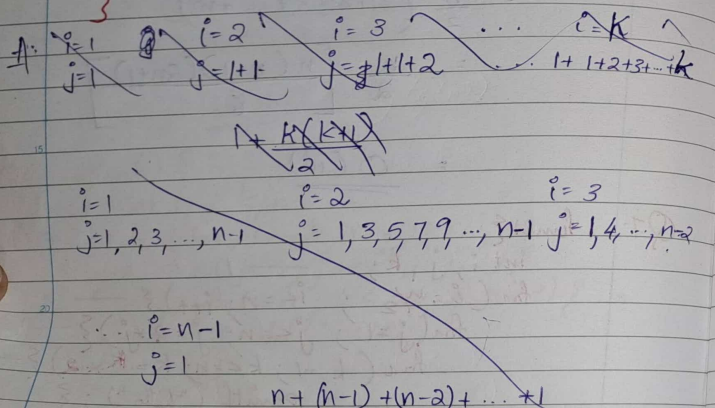
Camlin

A: $O(1 + n\log^2 n)$

$= O(n\log^2 n)$

Q8.
```
fun(){
  for(i=1; i<=n; i++){
    for(j=1; j<=n; j=j+i){
      printf("hello");
    }
  }
}
```

A:



A:

$i=1$	$i=2$	$i=3$	$i=k$
$j=n$	$j=1$ to n	$j=1$ to n	$j=1$ to n
$j=j+1$	$j=j+i$	$j=j+i+3$	$j=j+k$
1, 2, 3, ...	1, 3, 5, 7, ...	1, 4, 7, 10, ...	
n times	$n/2$ times	$n/3$ times	n/k times

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{k}$$

$$n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}\right]$$

$n(\log n)$

~~$\therefore O(n\log n) \cdot O(1)$~~

$\therefore O(n\log n)$

Comparison Of Growth Rate:

1. Log Method.
2. L Hospital Method. → applied in ∞/∞ or $0/0$
   ↳ if $\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty/\infty$ or $0/0$ then apply.

~~n~~

★ $1 \cdot \log n \quad n \quad n\log n \quad n^2 \quad n^3 \quad \cdots \quad n^n \quad 2^n \quad 3^n$

↓ least. → increasing order.

Q1. Compare:
1. $f(n) = n^3 \log n$
$g(n) = (n \log n)^{20}$

A: $\log(f(n)) = \log(n^3 \log n)$
$= \log n^3 + \log \log n$
$= 3 \log n + \log \log n$

$\log(g(n)) = \log(n \log n)^{20}$
$= 20 (\log(n \log n))$
$= 20 \log n + 20 \log \log n$

$\therefore \log(f(n)) < \log(g(n))$

$f(n)$ grows slower than $g(n)$

Q2. $f(n) = 4n^{\sqrt{n}}$
$g(n) = 2^{\sqrt{n} \log_2 n}$

A: $4n^{\sqrt{n}}$ $\quad$ $2^{\sqrt{n} \log n}$

$\log(f(n))$ $\quad$ $\log(g(n))$
$= \log(4n^{\sqrt{n}})$ $\quad$ $\log(2^{\sqrt{n} \log n})$
$= \log 4 + \sqrt{n} \log n$ $>$ $\sqrt{n} \log n \log_2 2^1$

$\therefore f(n) > g(n)$

Q3 $f(n) = \log 2^n$ $\quad$ $g(n) = \log 2^{2n}$

A: $g(n) \geq f(n)$

Q4. $f(n) = n^{\log n}$ $\quad$ $g(n) = 2^{\sqrt{n}}$

A: $\log(f(n))$ $\quad$ $\log(g(n))$
$= \log(n^{\log n})$ $\quad$ $= \log(2^{\sqrt{n}})$
$= \log n \log n$ $\quad$ $= \sqrt{n} \log 2 \to 1$
$= \log^2 n$ $\quad$ $= \sqrt{n}$

Again log.

$\log(\log n)^2$ $\quad$ $\log(n^{1/2})$
$2 \log \log n$ $<$ $\frac{1}{2} \log n$

$\therefore g(n) > f(n)$

Q5. $f(n) = 2^{\log n}$ $\quad$ $g(n) = n^{\sqrt{n}}$

A:

$\log(f(n))$ $\quad$ $\log(n^{\sqrt{n}})$
$= \log(2^{\log n})$ $\quad$ $\sqrt{n} \log n$
$= \log n \cdot \log 2^1$ $<$

$\therefore g(n) > f(n)$

Camlin

L-Hospital Rule:

$$\lim_{x\to 0} \frac{\sin x}{x} = \frac{\sin 0}{0} = \frac{0}{0}$$

$$\lim_{x\to 0} \frac{\frac{d}{dx}(\sin x)}{\frac{d}{dx}(x)} = \lim_{x\to 0} \frac{\cos x}{1}$$

$$= \lim_{x\to 0} \cos x$$

$$= \cos 0 = 1$$

Evaluate,

1. $\lim_{x\to 0} \frac{1-\cos x}{x}$

$$\lim_{x\to 0} \frac{\frac{d}{dx}(1-\cos x)}{\frac{d}{dx}(x)}$$

$$\lim_{x\to 0} \frac{\sin x}{1}$$

$$= \sin 0$$

$$= 0$$

2. $\lim_{x\to 0} \frac{3x+\sin 6x}{4x^2-3\sin x}$

A:

$$\lim_{x\to 0} \frac{\frac{d}{dx}(3x+\sin 6x)}{\frac{d}{dx}(4x^2-\sin x)}$$

$$\lim_{x\to 0} \frac{3+6\cdot\cos 6x}{8x-\cos x}$$

$$\frac{3+6}{-1} = -9$$

3. $\lim_{x\to 3} \frac{x-3}{x^2-x-6}$

A:

$$\lim_{x\to 3} \frac{\frac{d}{dx}(x-3)}{\frac{d}{dx}(x^2-x-6)}$$

$$= \lim_{x\to 3} \frac{1}{2x-1}$$

$$= \frac{1}{6-1} = \frac{1}{5}$$

Camlin

4. $\lim_{x \to a} \frac{\cos x - \cos a}{x - a}$

A: $\lim_{x \to a} \frac{d/dx(\cos x - \cos a)}{d/dx(x - a)}$

$= \lim_{x \to a} \frac{-\sin x - 0}{1}$

$= -\sin a$

Page :
Date :

4. $\lim_{x \to a} \frac{\cos x - \cos a}{x - a}$

A: $\lim_{x \to a} \frac{\frac{d}{dx}(\cos x - \cos a)}{\frac{d}{dx}(x - a)}$

$= \lim_{x \to a} \frac{-\sin x - 0}{1}$

$= -\sin a$

8/5/24/Wednesday.

Q1. $\lim_{x \to 0} \frac{1 - \cos 4x}{x^2}$

A: $= \lim_{x \to 0} \frac{\frac{d}{dx}(1 - \cos 4x)}{\frac{d}{dx}(x^2)}$

$= \lim_{x \to 0} \frac{2 \times \sin 4x}{2x}$

$= \lim_{x \to 0} \frac{2 \sin 4x}{x}$ Again do L hospital rule

(Because it is not 0, ∞, constant)

Page :
Date :

$= \lim_{x \to 0} \frac{\frac{d}{dx} 2 \sin 4x}{\frac{d}{dx} x}$

$= \lim_{x \to 0} 2 \cos 4x \times 4$

$= \lim_{x \to 0} 8 \cos 4x$ → a constant.

$= 8$

Q2: $\lim_{x \to 0} \frac{\cos 2x - 1}{\cos x - 1}$

A: $= \lim_{x \to 0} \frac{\frac{d}{dx}(\cos 2x - 1)}{\frac{d}{dx}(\cos x - 1)} = \lim_{x \to 0} \frac{\sin 2x \times 2}{\sin x}$

$= \lim_{x \to 0} \frac{2 \sin 2x}{\sin x}$

Again apply L hospital rule.

$= \lim_{x \to 0} \frac{4 \cos 2x}{\cos x} = 4$ ↓ constant

If @ end of L hospital rule you get (f is faster ↑)

1. ∞ → Ω (best case - Omega) → f > g
2. 0 → O (worst case - big Oh) → f < g
3. constant → Θ (avg) → f ± g are comparable.

Camlin

Q3. Compare x^m & e^{nx} ; $m>0$ & $n>0$ & m is an Integer.

A. $\lim_{x\to\infty} \frac{x^m}{e^{nx}}$ → in $\frac{\infty}{\infty}$ form so we can apply L hospital rule.

$$\lim_{x\to\infty} \frac{m x^{m-1}}{e^{nx}\cdot n} \to \frac{\infty}{\infty}$$

So apply L hospital rule again

$$\lim_{x\to\infty} \frac{m\cdot(m-1)\, x^{m-2}}{e^{nx}\cdot n^2} \to 2^{nd} \text{ iteration}$$

Again apply L hospital rule.

⋮

m^{th} iteration: $$\lim_{x\to\infty} \frac{m\cdot(m-1)(m-2)\cdots\times 2\times 1\cdot x^{m-m}}{e^{nx}\cdot n^m}$$

$$\lim_{x\to\infty} \frac{m!\times x^0}{e^{nx}\times n^m}$$

$$= \frac{m!}{n^m\cdot\infty} = \frac{m!}{n^m}\cdot\frac{1}{\infty}$$

$$= \frac{m!}{n^m}\cdot 0$$

$$= 0 \to \text{zero, so big Oh.}$$

So f<g. That is; x^m is lower than e^{nx}.

or

$$x^m = O(e^{nx})$$

Q4. Analyse the growth rate of $\ln(x^{20})$ and $\ln(x)$.

A: $\lim_{x\to\infty} \frac{\ln(x^{20})}{\ln(x)} \to \frac{\infty}{\infty}$ so L hospital rule,

$$= \lim_{x\to\infty} \frac{\frac{d}{dx}(\ln(x^{20}))}{\frac{d}{dx}(\ln(x))}$$

$$= \lim_{x\to\infty} \frac{\frac{20x^{19}}{x^{20}\,x}}{\frac{1}{x}} \qquad \lim_{x\to\infty}\frac{20}{x} = \frac{1}{x}$$

$$= \lim_{x\to\infty} 20$$

$$= 20 \to \text{constant}$$

So avg bound.

ie; f(n) & g(n) grow at same rate.

ie; $\ln(x^{20}) = \Theta(\ln(x))$

Q5. Compare e^x & x^2.

A: $\lim_{x\to\infty} \frac{e^x}{x^2} \to \frac{\infty}{\infty}$ so L hospital rule.

$$\lim_{x\to\infty} \frac{\frac{d}{dx}e^x}{\frac{d}{dx}x^2} = \lim_{x\to\infty} \frac{e^x}{2x}$$ Again L hospital rule

$$= \lim_{x\to\infty} \frac{e^x}{2} = \infty/2 = \infty$$

$\infty \rightarrow \Omega \rightarrow f > g$: f is ~~slower~~ faster than g.

ie; $e^x = \Omega(x^2)$

or $e^x > x^2$

e^x has faster growth rate.

Q6. Compare 3^x & 2^x ?

A: $\lim_{x\to\infty} \frac{3^x}{2^x} \rightarrow \frac{\infty}{\infty}$ can apply L hospital rule

$\lim_{x\to\infty} \left(\frac{3}{2}\right)^x$

$= \infty$ so Ω,

That is; $3^x = \Omega(2^x)$ or $3^x > 2^x$.

Q7. Compare x^2 & $\ln(x)$

A: $\lim_{x\to\infty} \frac{x^2}{\ln(x)} \rightarrow \frac{\infty}{\infty}$

$= \lim_{x\to\infty} \frac{2x}{1/x} = \lim_{x\to\infty} 2x^2$

$= 2\infty^2 = \infty$

So Ω.

$x^2 = \Omega(\ln(x))$

Camlin

Q8. Compare $\ln(x)$ & $x^{1/n}$?

A: $\lim_{x\to\infty} \frac{\ln(x)}{x^{1/n}}$

$\lim_{x\to\infty} \frac{1/x}{\frac{1}{n} x^{1/n-1}}$

$= \lim_{x\to\infty} \frac{1}{x} \div \frac{1}{n} \cdot x^{1/n-1}$

$= \lim_{x\to\infty} \frac{1}{x} \cdot \frac{n}{x^{1/n-1}}$

$= \lim_{x\to\infty} \frac{n}{x^{1/n}}$

$= \frac{n}{\infty} = 0$

So Big Oh.

ie; $\ln(x) = O(x^{1/n})$

09/05/24/Thursday.

Q9. Show that $\log x = O(x)$?

A: $f(n) = \log x$

$g(n) = x$.

$\lim_{x\to\infty} \frac{\log x}{x} = \lim_{x\to\infty} \frac{1/x}{1} = \lim_{x\to\infty} \frac{1}{x}$

$= \frac{1}{\infty} = 0$

So big Oh; ie; $\log(x) = O(x)$

Hence proved.

Camlin

Q10. Show that $\sqrt{2}^{\log n} = O(\sqrt{n})$?

A: $f(n) = \sqrt{2}^{\log n}$ $\quad g(n) = \sqrt{n}$

$$\lim_{n\to\infty} \frac{\sqrt{2}^{\log n}}{\sqrt{n}}$$

$a^{\log b} = b^{\log a}$

$$\lim_{n\to\infty} \frac{n^{\log\sqrt{2}}}{\sqrt{n}}$$

[no idea how it got there ...

$$\lim_{n\to\infty} \frac{1}{\sqrt{n}} = \frac{1}{\infty} = 0$$

So $\sqrt{2}^{\log n} = O(\sqrt{n})$

Q11. Compare the growth rate of $\log_2 n$ & $\sqrt{n}$?

A: $$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}}$$

↓ d/dx on num & denom

$$\lim_{n\to\infty} \frac{1/n}{1/2\sqrt{n}}$$

$$= \lim_{n\to\infty} \frac{2\sqrt{n}}{n}$$

$$= \lim_{n\to\infty} \frac{2}{\sqrt{n}}$$

$$= \frac{2}{\infty} = 0 \rightarrow \log_2(n) = O(\sqrt{n})$$

$\log n$ has slower growth rate

Q12. Compare growth rate of $f(n) = n!$ & $g(n) = 2^n$?

A: Here we cannot use L hospital rule unless we do something abt $n!$. So we use Stirling's Approximation ↓

$$n! = \sqrt{2\pi n}\left[\frac{n}{e}\right]^n \left[1 + \Theta\left(\frac{1}{n}\right)\right]$$

Now,

$$\lim_{n\to\infty} \frac{n!}{2^n}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n}\left[\frac{n}{e}\right]^n \left[1 + \Theta \cdot \frac{1}{n}\right]}{2^n}$$

Don't apply derivative. Directly apply $n = \infty$

$$= \frac{\sqrt{2\pi\infty}\left[\frac{\infty}{e}\right]^{\infty}\left[1 + \Theta \cdot \frac{1}{\infty}\right]}{2^{\infty}} \quad (2^{\infty} \rightarrow \text{becomes } \infty)$$

$$= \infty \cdot \left[1 + \Theta \cdot \frac{1}{\infty}\right]$$

$$= \infty \cdot 1$$

$$= \infty$$

$\therefore n! = \Omega(2^n)$

Q13. Prove that $n! = O(n^n)$

A: $n! = (2\pi n)\left[\frac{n}{e}\right]^n [1 + \Theta \cdot \frac{1}{n}]$

$$\lim_{n\to\infty} \frac{n!}{n^n}$$

$$\lim_{n\to\infty} \frac{2\pi n \left[\frac{n}{e}\right]^n [1 + \Theta \cdot \frac{1}{n}]}{n^n}$$

$$= \frac{2\pi\infty \left[\frac{\infty}{e}\right]^\infty [1 + \Theta \cdot \frac{1}{\infty}]}{\infty^\infty}$$

$$= \frac{(1 + \Theta \cdot \frac{1}{\infty})}{\infty} = \frac{1}{\infty} = 0$$

So $n! = O(n^n)$

Hence proved.

Q14. Show that $10n^2 + 9 = O(n^2)$

A:

On hold...

No answer yet...

Q15. Show that $\frac{6n^3}{\log n + 1} = O(n^3)$?

A: $$\lim_{n\to\infty} \frac{\frac{6n^3}{\log n + 1}}{n^3} = \lim_{n\to\infty} \frac{6}{\log n + 1}$$

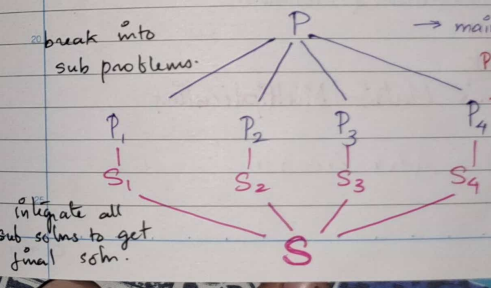
$$= 6/\infty = 0$$

Hence: $\frac{6n^3}{\log n + 1} = O(n^3)$

Recurrence Relation:

- Used for recurring functions.
- use of divide & conquer strategy.
- There are other methods as well. One such method is Dynamic Programming.

Divide & Conquer Strategy: (DAC)



break into sub problems.

→ main problem.

P_1, P_2, P_3, P_4 are → sub problems. They all have to be of the same type. eg: if P is a sorting prob, the P_1, P_2, P_3 and P_4 should be sorting prob.

integrate all sub solns to get final soln.

Guideline to check if Divide & Conquer Strategy can be used:

If a big problem can be divided into subproblems of same type, each having solutions which can be integrated together to form the final soln. Then we can apply DAC.

```
DAC(P):
    if (small(P)){
        S(P);
    }
    else if (big(P)){
        divide P into P1, P2, P3, ..., Pk
        Apply DAC(P1), DAC(P2), ..., DAC(Pk)
    }
    combine S1, S2, S3, ..., Sk to form S.
```

Example problems where we can apply DAC:

- Binary search
- Finding Maximum & Minimum
- Merge Sort
- Quick Sort
- Strassen's Matrix Multiplication

Recurrence Relation

Decreasing Function	Dividing Function	Root Function
$T(n-b)$	$T(n/b)$	$T(\sqrt{n})$
↓	↓	↓
1	1	2

smallest value possible:

Solution approaches:

1. Back Substitution
2. Recurrence tree / Tracing Method
3. Master Method.

15/5/24/Wednesday

Q.

```
void Test(int n)                    ---> T(n)
{
  if (n>0)                          ---> 1
  {
    printf("%d", n);                ---> 1
    Test(n-1);                      ---> T(n-1)
  }
}
```

A: First (This is a decreasing fn).
assume time for Test(n) = T(n)
then remaining as marked ...
Then,

$T(n) = 1 + 1 + T(n-1)$
$= (2) + T(n-1)$ ⟩ asymptotic (approximation)
$= T(n-1) + (1)$

ie; $T(n) = T(n-1) + 1$

Now write this in recurrence relation form:

smallest value for decreasing fn.

$$T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1)+1 & ; n>0 \end{cases}$$

Solving using:

1. Back Substitution:

$T(n) = T(n-1) + 1$ —— ①
$T(n-1) = T(n-2) + 1$ (we get by applying (n-1) in 1)

Now back substitute this in ①,

$T(n) = [T(n-2) + 1] + 1$
$T(n) = T(n-2) + 2$ —— ②

Do such back substitutions for 3 or 4 iterations before generalizing.

Similarly $T(n-2) = T(n-3) + 1$ ⟩ back substitute

$\Rightarrow T(n) = [T(n-3) + 1] + 2 = T(n-3) + 3$

⋮

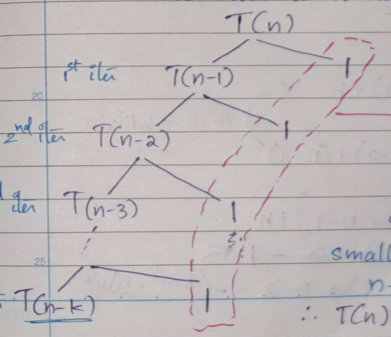
Now in k^{th} iteration,

$T(n) = [T(n-k)] + k$
↳ assume this to be smallest value possible for decreasing fn.
$= 1 + k$
↳ smallest value for decreasing fn.

From the recurrence relation we know that the smallest value of T(n) occurs when n = 0
ie;
$n - k = 0$
$n = k$.
ie; $T(n) = 1 + n$
↓ Asymptotically
$T(n) = n$
ie; $T(n) = O(n)$

2. Tree Method:

We have to draw the recurrence tree.



Now ignore the T branch.
only consider this part.
Add them.
$1 + 1 + 1 \ldots + 1^{k} = 1 \times k = k$
So $T(n) = k$
smallest time: $n = 0$ so,
$n - k = 0 \rightarrow n = k$
$\therefore T(n) = O(n)$

Q2.
```
void Test (int n)                          → T(n)
{
  if (n>0)                                 → 1
  {
    for (int i=0; i<n; i++)                → n
    {
      printf("%d", n);
    }
    Test(n-1);                             → T(n-1)
  }
}
```

A: $T(n) = 1 + n + T(n-1)$

↓ Asymptotically.

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

Recurrence Relation:

$$T(n) = \begin{cases} 1 & ; n = 0 \\ T(n-1) + n & ; n > 0 \end{cases}$$

Back Substitution:

1st — $T(n) = T(n-1) + n$ ← back substitute

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

$T(n) = [T(n-2) + n - 1] + n$

2nd — $T(n) = T(n-2) + 2n - 1$ ← back substitute

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

3rd ← $T(n) = T(n-3) + n - 2 + 2n - 1$

$T(n) = T(n-3) + 3n - 3$ ← back substitute

$T(n-3) = T(n-4) + n - 3$

$T(n) = T(n-4) + n - 3 + 3n - 3$

$T(n) = T(n-4) + 4n - 6$

4th iter ← $T(n) = T(n-4) + 4n - (1+2+3)$

⋮

k^{th} iter: $T(n) = T(n-k) + kn - (1+2+3+\dots+k)$

$$T(n) = T(n-k) + kn - \left[\frac{k(k+1)}{2}\right]$$

$$= T(n-k) + kn - \frac{k^2+k}{2}$$

$T(n) = T(n-3) + n - 2 + 2n - 1$

3rd ← $T(n) = T(n-3) + 3n - 3$

back substitute

$T(n-3) = T(n-4) + n - 3$

$T(n) = T(n-4) + n - 3 + 3n - 3$

$T(n) = T(n-4) + 4n - 6$

4th iteration ← $T(n) = T(n-4) + 4n - (1+2+3)$

⋮

k^{th} iteration: $T(n) = T(n-k) + kn - (1+2+3+\ldots+k)$

$T(n) = T(n-k) + kn - \left[\frac{k(k+1)}{2}\right]$

$= T(n-k) + kn - \frac{k^2+k}{2}$

$n-k=0$

$= T(0) + n^2 - \frac{n^2+n}{2}$

$= 1 + \frac{2n^2-n^2+n}{2}$

$= 1 + \frac{n^2+n}{2}$

$= O(n^2)$

16/05/24/Thu.

OR.

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

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

$T(n) = T(n-3) + (n-2) + (n-1) + n$

So,

k^{th} → $T(n) = T(n-k) + (n-(k-1)) + \ldots + (n-1) + n$

Minimum value → 1 when $n = 0$.

So, $n - k = 0 \rightarrow n = k$.

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

$= 1 + (1) + \ldots + (n-2) + (n-1) + n$

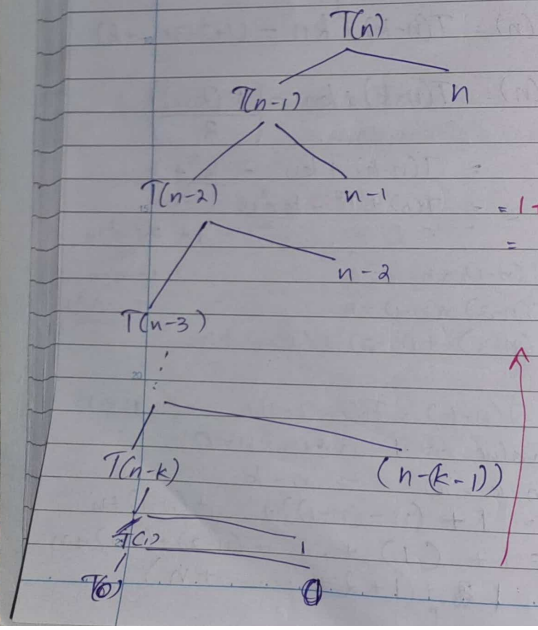
$= 1 + (1 + 2 + \ldots + n)$

$= 1 + \frac{n(n+1)}{2}$

$= 1 + \frac{n^2+n}{2}$

$= O(n^2)$

Trace Method:



$= 1 + 2 + 3 + \dots + n$

$= \frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$

$= O(n^2)$

Q3.
```
void Test(int n)                    ---> T(n)
{
  if(n>0)                           ---> 1
  {
    for(int i=1; i<n; i=(i*2))      }
    {                               } log2n
      printf("%d", n);              }
    }
    Test(n-1);                      ---> T(n-1)
  }
}
```

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

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

$$T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1) + \log n & ; n>0 \end{cases}$$

Back Substitution:

$T(n) = T(n-1) + \log n$ —— ①

$T(n-1) = T(n-2) + \log(n-1)$

$T(n) = T(n-2) + \log(n-1) + \log n$ —— ②

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

$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$ —— ③

$\vdots$

K^{th} ← $T(n) = T(n-k) + \log(n-(k-i)) + \dots + \log(n-1) + \log n$

Minimum at $n=0$ for $T(n)$.

So, $n-k=0$

$n=k$.

$T(n) = T(0) + \log(n-(n-1)) + \dots + \log(n-1) + \log n$

$= 1 + \log(1) + \log 2 + \dots + \log(n-1) + \log n$

$\log ab = \log a + \log b$ → $= 1 + \log(1 \times 2 \times \dots \times (n-1) \times n)$

$= 1 + \log(n!)$

$= \log(n!)$

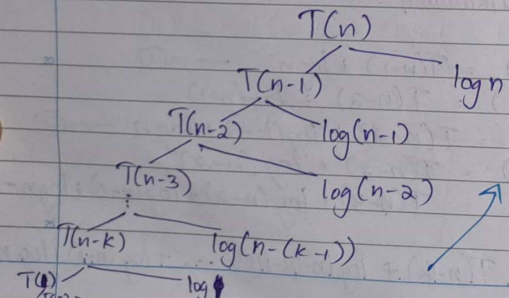
n^n → upper bound.

$= \log(n^n)$

$= n \log n$

$= O(n \log n)$

Tree Method:



Camlin

$\log 1 + \log 2 + \dots + \log(n)$

$= \log(1 \times 2 \times 3 \times \dots \times (n-1) \times n)$

$= \log(n!)$

$= \log(n^n)$

$= n \log n$

$= O(n \log n)$

Q4:

```
void Test(int n)          ---- T(n)
{
if (n>0)                  ---- 1
{
printf("%d", n);         ---- 1
Test(n-1);               ---- T(n-1)
Test(n-1);               ---- T(n-1)
}
}
```

A: $T(n) = 1 + 1 + T(n-1) + T(n-1)$

$= 2 + 2(T(n-1))$ ↓ asymptotically

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

$$T(n) = \begin{cases} 1 & ; n=0 \\ 2T(n-1)+1 & ; n>0 \end{cases}$$

Camlin

$T(n) = 2T(n-1) + 1$ —— ①

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

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

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

$T(n) = 4T(n-2) + 2 + 1$ —— ②

$T(n-2) = 2T(n-3) + 1$

$T(n) = 4(2T(n-3) + 1) + 2 + 1$

$T(n) = 8T(n-3) + 4 + 2 + 1$ —— ③

$T(n-3) = 2T(n-4) + 1$

$T(n) = 8[2T(n-4) + 1] + 4 + 2 + 1$

$T(n) = 16T(n-4) + 8 + 4 + 2 + 1$ —— ④

(with $16 = 2^4$, $8 = 2^3$, $4 = 2^2$, $2 = 2^1$, $1 = 2^0$)

⋮

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0$

Minimum when $n-k=0$; $n=k$.

$T(0) = 1$

$T(n) = 2^n T(n-n) + 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 2^0$

$= 2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 2^0 \rightarrow GP$

GP series sum $= \dfrac{a(r^k - 1)}{r-1}$

(a → 1st term; k → no. of terms; r → common factor)

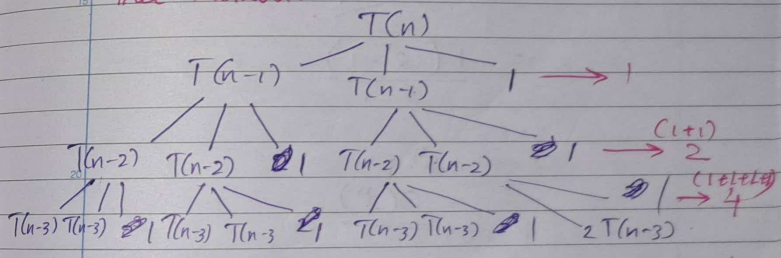
$a=1$; $r=2$; $k=n$

$T(n) = \dfrac{1(2^n - 1)}{2-1}$

$= 2^n - 1$

$= O(2^n)$

Tree Method

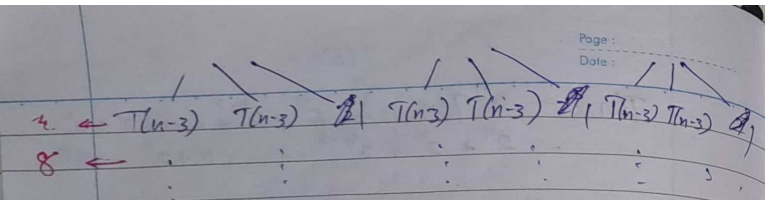


$2^k T(n-k) + 2^{k-1} \ldots$

$2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^3 + 2^2 + 2^1 + 2^0$

$= 2^n - 1 = O(2^n)$

cont in next page



$T(n) = 2^0 + 2^1 + \dots + 2^{k-1}$

$= 2^0 + 2^1 + \dots + 2^{n-1}$

$= \frac{1(1-2^n)}{2-1}$

$= 1 - 2^n$

$= \frac{1(2^n - 1)}{2-1}$

$= 2^n - 1$

$= O(2^n)$

$\frac{a(r^n-1)}{r-1}$

Camlin

Master's Theorem For Decreasing Function :

$T(n) = T(n-1) + 1$	–	$O(n)$
$T(n) = T(n-1) + n$	–	$O(n^2)$
$T(n) = T(n-1) + \log n$	–	$O(n \log n)$
$T(n) = 2T(n-1) + 1$	–	$O(2^n)$
$T(n) = 3T(n-1) + 1$	–	$O(3^n)$
$T(n) = 2T(n-1) + n$	–	$O(n 2^n)$

★ $T(n) = 3T(n-1) + 1$

$T(n-1) = 3T(n-2) + 1$

$T(n) = 3^2 T(n-2) + 3 + 1$

$T(n-2) = 3T(n-3) + 1$

$T(n) = 3^3 T(n-3) + 3^2 + 3 + 1$

$\vdots$

$T(n) = 3^k T(n-k) + 3^{k-1} + \dots + 3^1 + 3^0$

$= 3^n \cdot T(0) + 3^{n-1} + \dots + 3^1 + 3^0$

$= 3^n + 3^{n-1} + \dots + 3^1 + 3^0$

$= \frac{1(3^n - 1)}{2}$

$= O(3^n)$

★ $T(n) = 2T(n-1) + n$ $\quad T(n-1) = 2T(n-2) + n$

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

$T(n) = 2^3 T(n-3) + 2^2 n + 2n + n$

$\vdots$

$T(n) = 2^k T(n-k) + 2^{k-1} n + \dots + 2^1 n + 2^0 n$

Camlin

$T(n) = 2^n + 2^{n-1} + 2^{n-2} + \dots + 2n + n$

$= n(2^n + 2^{n-1} + \dots + 2^0)$

$= n \cdot \frac{2^n - 1}{2-1}$

$= n \cdot 2^n$

$= O(2^n \cdot n)$

Cases:

$T(n) = aT(n-b) + f(n)$

1. Case 1: if $a < 1$; $O(f(n))$
2. Case 2: if $a = 1$; $O(n * f(n))$
3. Case 3: if $a > 1$; $O(f(n) * a^{n/b})$

Recurrence Relations : Dividing Functions :

Q1.
```
void Test(int n)          ---- T(n)
{
  if(n>1)                 ---- 1
  printf("%d", n);        ---- 1
  Test(n/2);              ---- T(n/2)
}
```

$T(n) = 1 + 1 + T(n/2)$

$= 2 + T(n/2)$ } Asymptotically

$= T(n/2) + 1$

$$T(n) = \begin{cases} 1 & ; n = 1 \\ T(n/2) + 1 & ; n > 1 \end{cases}$$

Back Substitution:

$T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/2 \div 2) + 1$

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

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

$T(n) = T(n/2^2) + 2$ — (2)

$T(n/2^2) = T(n/2^3) + 1$

$T(n) = T(n/2^3) + 3$ — (3)

$\vdots$

$T(n) = T(n/2^k) + k$

——→ assume to be min.

$T(n) = 1 + k$

$T(n) = 1 + \log n$

$= \log n$

~~$T(n) = O(n)$~~

↳ @ ~~$k = n-1$~~

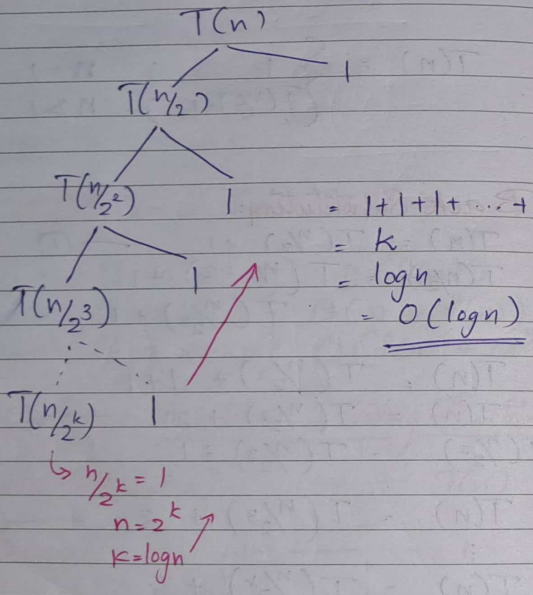
$n/2^k = 1$

$n = 2^k$

$k = \log n$

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

Tree Method:



$= 1+1+1+\dots+1$

$= k$

$= \log n$

$= O(\log n)$

↳ $n/2^k = 1$

$n = 2^k$

$k = \log n$

Q2. $T(n) = \begin{cases} 1 & ; n = 1 \\ T(n/2) + n & ; n > 1 \end{cases}$

1: Back Substitution:

$T(n) = T(n/2) + n$ —①

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

$T(n) = T(n/2^2) + n/2 + n$ —②

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

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

⋮

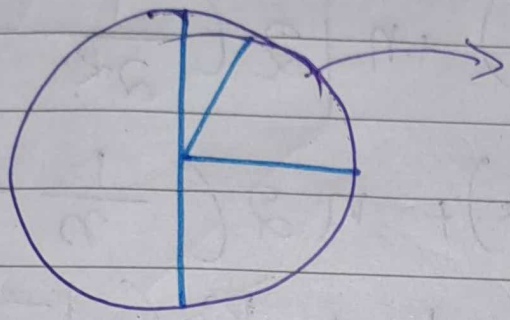
$T(n) = T(n/2^k) + n/2^{k-1} + \dots + n/2 + n$ → minimise.

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log n$

~~$T(n) = 1 + \frac{n}{2^{\log n - 1}} + \dots + n/2 + n$~~

$T(n) = T(n/2^k) + \frac{n}{2^{k-1}} + \dots + n/2 + n$

$= T(n/2^k) + n\left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \dots + \frac{1}{2} + 1\right]$



circle considered as one.

So assuming

$$\frac{1}{2^{k-1}} + \frac{1}{2^k} + \dots + \frac{1}{2^2} + \frac{1}{2^1} + \frac{1}{2^0}$$

is equal to 1.

Then, $T(n) = T\left(\frac{n}{2^k}\right) + n(1)$

$= 1 + n \quad 2\left(\frac{1}{2^k} - 1\right)$

$= O(n)$

GP would be more complicated...

$a = \frac{1}{\quad}$ $r = \frac{1}{2}$ $k = k$ $\frac{1\left(\frac{1}{2}^k - 1\right)}{\frac{1}{2}}$

$GP = \frac{1\left(\left(\frac{1}{2}\right)^k - 1\right)}{\frac{1}{2} - 1} = \frac{-2\left(\left(\frac{1}{2}\right)^k - 1\right)}{= \frac{1}{2^k}}$

$= T(n/2^k) + n\left(\frac{1}{2^k}\right)$ $\quad n/2^k = 1$

$= 1 + n$ $\quad 2^k = n$

$= 1 + n$

$= T(n/2^k) + n\left[2\left(\frac{1}{2^k} - 1\right)\right]$

$= T(n/2^k) + n\,2\left(\frac{1}{n} - 1\right)$

$= 1 + n\left(\frac{1-n}{n}\right)$

$= O(n)$

Q3.
```
void Test(int n)          — T(n)
{
if(n>1)                   — 1
{
for(int i=0; i<n; i++)    } n
{
//stmt;
}
Test(n/2);                — T(n/2)
Test(n/2);                — T(n/2)
}
```

Camlin

Q4: $T(n) = 1 + n + 2T(n/2)$

$$T(n) = \begin{cases} 1 & ; n = 1 \\ 2T(n/2) + n & ; n > 1 \end{cases}$$

$T(n) = 2T(n/2) + n$ —— (1)

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

$T(n) = 2\left[2T(n/2^2) + n/2\right] + n$

$T(n) = 2^2 T(n/2^2) + n + n$ —— (2)

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

$T(n) = 2^2\left[2T(n/2^3) + \frac{n}{2^2}\right] + 2n$

$T(n) = 2^3 T(n/2^3) + 3n$ —— (3)

$\vdots$

$T(n) = 2^k T(n/2^k) + 2^{k-1}n + \cdots + 2n + n$

$T(n) = 2^k T(n/2^k) + 2^{k-1}n + 2^{k-2}n + \cdots + 2n + 2^0 n$

$T(n) = 2^k T(n/2^k) + k \cdot n$

Camlin

$T(n) = 2\left(T\left(\frac{n}{2^k}\right)\right)$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$

↳ minimised.

$\frac{n}{2^k} = 1$

$n = 2^k \qquad k = \log n$

$T(n) = n \cdot 1 + n\log n$

$T(n) = n\log n$

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

Camlin

$T(n) = 2^k T(n/2^k) + n\left[2^{k-1} + 2^{k-2} + \dots + 2^1 + 2^0\right]$

$a = 1 \quad r = 2 \quad k = k$

$GP = \frac{a(r^k - 1)}{r - 1} = \frac{1(2^k - 1)}{2 - 1}$

$= 2^k - 1$

$T(n) = 2^k \left[T(n/2^k)\right] + n \cdot (2^k - 1)$

$T(n) = 2^k T(n/2^k) + kn$

↳ minimised.

$\frac{n}{2^k} = 1$

$n = 2^k \qquad k = \log n$

$T(n) = n \cdot 1 + n \log n$

$T(n) = n \log n$

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

Camlin

22/05/24/Wednesday.

Master's Theorem for Dividing Function:

$T(n) = aT(n/b) + f(n)$

where:

$a \geq 1$ a should be a constant

$b > 1$

$f(n) = O(n^k \log^p n)$

1. Case 1: if $\log_b a > k$; then $O(n^{\log_b a})$

2. Case 2: if $\log_b a = k$
   - i) if $p > -1$; $O(n^k \log^{p+1} n)$
   - ii) if $p = -1$; $O(n^k \log \log n)$
   - iii) if $p < -1$; $O(n^k)$

3. Case 3: if $\log_b a < k$
   - i) if $p \geq 0$; $O(n^k \log^p n)$
   - ii) if $p < 0$; $O(n^k)$

Q. $T(n) = 2T(n/2) + 1$

A: $a = 2 \quad b = 2 \quad f(n) = 1$] write in form of $n^k \log^p n$

$f(n) = n^0 \log^0 n$

↓

So $k = 0 \quad p = 0$

$\log_b a = \log_2 2 = 1 > k = 0$ So case 1

Camlin

So $O(n^{\log_b a})$

$= O(n^1) = \underline{O(n)}$

Q2. $T(n) = 4T(n/2) + n$

A: $a = 4 \quad b = 2 \quad f(n) = n$

$= n^1 \log^0 n$

$k = 1 \quad p = 0$ ←

$\log_b a = \log_2 4 = 2 > k = 1$

So case 1,

$O(n^{\log_b a}) = \underline{O(n^2)}$

Q3. $T(n) = 8T(n/2) + n$

A: $a = 8 \quad b = 2 \quad f(n) = n$

$k = 1 \quad p = 0$ ← $= n^1 \log^0 n$

$\log_b a = \log_2 8 = 3 > k = 1$

So case 1, ~~$O(\log_b a$~~ $O(n^{\log_b a})$

$= \underline{O(n^3)}$

Q4. $T(n) = 9T(n/3) + 1$

A: $a = 9 \quad b = 3 \quad f(n) = 1$

$k = 0 \quad p = 0 \quad = n^0 \log^0 n$

$\log_b a = \log_3 9 = 2 > k = 0$

$O(n^{\log_b a}) = \underline{O(n^2)}$

Q5. $T(n) = 9T(n/3) + n$

A: $a = 9 \quad b = 3 \quad f(n) = n$

$k = 1 \quad p = 0 \quad = n^1 \log^0 n$

$\log_b a = \log_3 9 = 2 > k = 1$

So case 1,

$\underline{O(n^2)}$

Q6. $T(n) = 9T(n/3) + n^2$

A: $a = 9 \quad b = 3 \quad f(n) = n^2$

$k = 2 \quad p = 0 \quad = n^2 \log^0 n$

$\log_b a = \log_3 9 = 2 = k = 2$

So Case 2 ∴ $p > -1$

$O(n^k \log^{p+1} n)$

$= \underline{O(n^2 \log n)}$

Q7. $T(n) = 4T(n/2) + n$

A: $a = 4 \quad b = 2 \quad f(n) = n$

$k = 1 \quad p = 0 \quad = n^1 \log^0 n$

$\log_b a = \log_2 4 = 2 > k = 1$

So case 1, $O(n^2)$

Q8. $T(n) = 8T(n/2) + n \log n$

A: $a = 8 \quad b = 2 \quad f(n) = n \log n$

$k = 1 \quad p = 1 \quad = n^1 \log^1 n$

$\log_b a = \log_2 8 = 3 > k = 1$

Case 1 so, $\Rightarrow O(n^3)$

Q9. $T(n) = 2T(n/2) + n$

$a = 2 \quad b = 2 \quad f(n) = n$

$k = 1 \quad p = 0 \quad = n^1 \log^0 n$

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

So case 2, $p = 0 > -1$ so $O(n^k \log^{p+1} n) = O(n \log n)$

Q10. $T(n) = 4T(n/2) + n^2$

A: $a = 4 \quad b = 2 \quad f(n) = n^2$

$k = 2 \quad p = 0 \quad = n^2 \log^0 n$

$\log_b a = \log_2 4 = 2 = k = 2$

So case 2, $p > -1$, $O(n^k \log^{p+1} n)$

$= O(n^2 \log n)$

Q11. $T(n) = 4T(n/2) + n^2 \log n$

A: $a = 4 \quad b = 2 \quad f(n) = n^2 \log^1 n$

$k = 2 \quad p = 1$

$\log_b a = \log_2 4 = 2 = k = 2$

So case 2, $p > -1$

$O(n^k \log^{p+1} n) = O(n^2 \log^2 n)$

Q12. $T(n) = 2T(n/2) + \frac{n}{\log n}$

A: $a = 2 \quad b = 2 \quad f(n) = n \log^{-1} n$

$k = 1 \quad p = -1$

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

So case 2, $p = -1$,

$O(n^k \log\log n)$

$= O(n \log\log n)$

Q13. $T(n) = 2T(n/2) + n/\log^2 n$

A: $a=2 \quad b=2 \quad f(n) = n \log^{-2} n$

$k=1 \quad p=-2$

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

So case 2, $p < -1$,

$O(n^k) = O(n)$

Q14. $T(n) = T(n/2) + n^2$

A: $a=1 \quad b=2 \quad f(n) = n^2 \log^0 n$

$k=2 \quad p=0$

$\log_b a = \log_2 1 = 0 < k=2$

So case 3, $p=0$

$O(n^k \log^p n) = O(n^2)$

Q15. $T(n) = 2T(n/2) + n^2 \log^2 n$

A: $a=2 \quad b=2 \quad f(n) = n^2 \log^2 n$

$k=2 \quad p=2$

$\log_2 2 = 1 < k=2$

So case 3, $p \geq 0$

$O(n^k \log^p n)$

$= O(n^2 \log^2 n)$

Q16. $T(n) = 4T(n/2) + n^3$

A: $a=4 \quad b=2 \quad f(n) = n^3 \log^0 n$

$k=3 \quad p=0$

$\log_2 4 = 2 < k=3$

So case 3, $p=0$

$O(n^k \log^p n) = O(n^3 \log^0 n)$

$= O(n^3)$

Q17. $T(n) = 4T(n/2) + n^3/\log n$

A: $a=4 \quad b=2 \quad f(n) = n^3 \log^{-1} n$

$k=3 \quad p=-1$

$\log_2 4 = 2 < k=3$

So case 3, $p < 0$ So $O(n^k)$

$= O(n^3)$

Page :
Date :

Q18. $T(n) = 4T(n/2) + (n \log n)^2$

A: $a = 4 \quad b = 2 \quad k = 2 \quad p = 2$

$\log_2 4 = 2 = k = 2$

So case 2, $p > -1$

So $O(n^k \log^{p+1} n)$

$= O(n^2 \log^3 n)$

Q19. $T(n) = 3T(n/2) + n^2$

A: $a = 3 \quad b = 2 \quad k = 2 \quad p = 0$

$\log_2 3 = 1.585 < 2 \; k = 2$

So case 3, $p = 0$ so $O(n^k \log^p n)$

$= O(n^2 \log^0 n)$

$= O(n^2)$

Q20. $T(n) = 3T(n/2) + n$

A: $a = 3 \quad b = 2 \quad k = 1 \quad p = 0$

$\log_2 3 = 1.585 > k = 1$

So case 1, $O(n^{\log_b a}) = O(n^{1.585})$

23/06/24 Thursday

21. $T(n) = 3^n T(n/2) + n^2$

A: Since $a = 3^n$ which makes it a non-constant, we can't apply master's theorem.
So we try using back substitution.
We don't need to do it...

22. $T(n) = 2T(n/4) + n^{0.51}$

A: $a = 2 \quad b = 4 \quad f(n) = n^{0.51}$
$k = 0.51 \quad p = 0$

$\log_b a = \log_4 2 = 0.5 < k = 0.51$

~~Case 2, $p \geq -1$~~
~~So case~~ ~~$O(n^k \log^{p+1} n)$~~

So, Case 3, $p = 0$

$O(n^k \log^p n)$

$O(n^{0.51})$

$O(n^{0.51})$

Q23. $T(n) = 64 T(n/8) - n^2 \log n$

A: Here $f(n)$ is negative. Not in form of $T(n) = aT(n/b) + f(n)$
So no master's theorem.

Q24. $T(n) = T(n/2) + n(2 - \cos n)$

A: Here also we cannot apply master's because $f(n)$ is not in the form of $n \log n$.

Recurrence Relations: Root Functions:

Q. ~~void (Test~~

```
void Test (int n) {          —— T(n)
{
  if (n > 2)                 —— 1
  {
    stmt;                    —— 1
    Test(√n);                —— T(√n)
  }
}
```

A: $T(n) = 1 + 1 + T(\sqrt{n})$

$= 2 + T(\sqrt{n})$

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

$$T(n) = \begin{cases} 1 & ; n = 2 \\ T(\sqrt{n}) + 1 & ; n > 2 \end{cases}$$

By back substitution,

$T(n) = T(\sqrt{n}) + 1$ —— ①

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

$T(n) = T((n^{1/2})^{1/2}) + 1 + 1$

$T(n) = T(n^{1/4}) + 2$ —— ②

$T(n^{1/4}) = T((n^{1/4})^{1/2}) + 1$

$T(n) = T(n^{1/8}) + 1 + 2$

$T(n) = T(n^{1/8}) + 3$ —— ③

$\vdots$

$T(n) = T(n^{1/2^k}) + k$

$$T(n) = \begin{cases} T(n^{1/2^k}) + k & ; n > 2 \\ 1 & ; n = 2 \end{cases}$$

smallest value of $n = 2$,

$T(2) + k$; ie, $n^{1/2^k} = 2$

$\frac{1}{2^k} \log n = \log 2$

$\frac{1}{2^k} \log n = 1$

$\log n = 2^k$

$\log \log n = k \log_2 2$

$k = \log \log n$

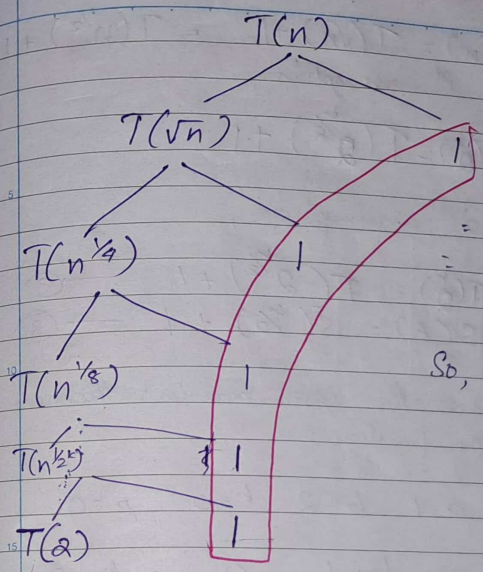
So, $T(2) + k$

$= 1 + k$

$= 1 + \log \log n$

$= O(\log \log n)$

By using Tree method :



$= 1 + 1 + \ldots + 1$ (k^{th})

$=$ upto $n^{1/2^k} = 2$

$k = \log \log n$

So, $O(\log \log n)$

This is not coming for exam

Master's Theorem for Root Function :

Assume 1. $n = 2^k$

2. the growth rate of $T(2^k) = k$.

so that of $T(2^{k/2}) = k/2$

$T(n) = T(\sqrt{n}) + 1 = T(n^{1/2}) + 1$

After 1st assumption,

$T(2^k) = T(2^{k/2}) + 1$

After 2nd assumption,

$T(2^k) = T(2^{k/2}) + 1$ — (2)

$S(k) = S(k/2) + 1$ — (3)

Now use the same master's theorem as Dividing function.

Same for this

→ Then $a=1$ $b=2$ $k=0$ $p=0$

$\log_b a = \log_2 1 = 0 = k = 0$

So case 2, $p = 0 > -1$

$O(n^k \log^{p+1} n) = O(k^0 \log^{0+1} n)$

$= O(\log k)$

$k = 2^k$

$k = \log n \rightarrow O(\log \log n)$

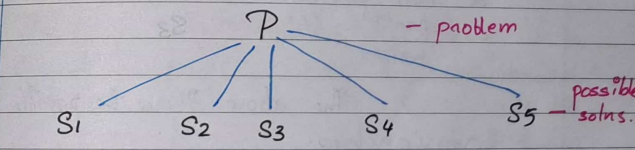
Camlin

Greedy Approach:

problem → size ~~of solns~~

```
Greedy (a, n)
for i=1 to n do
    x = select(a)
    if feasible(x), then
        solun = solun + x;
```

- This is another strategy like Divide & conquer.
- This is used for optimisation problems.



Now it checks among the possible solns, it checks which all are feasible. Then they choose the soln which is most optimal one amongst the feasible solns.

There is only 1 optimised soln.

Camlin

We can use branch and bound as well as dynamic programming methods to solve optimisation problems.

Fractional knap-sack problem:

Object	O_1	O_2	O_3	O_4	O_5	O_6	O_7
Profit	5	10	15	7	8	9	4
Weight	1	3	5	4	1	3	2.
P/W	5	3.3	3	1.75	8	3	2

Constraint: carry bag capacity limit = 15kg.
carry bag should have max. profit

here $n = 7$

Here you can select objects by:
1. Min weight — S_1
2. Max profit — S_2
3. $\frac{\text{profit}}{\text{weight}}$ ratio — S_3.

The above 3 are the possible soln approaches.

Now check each soln,

1# S_1: ↓ based on ↓ wt.

object	profit	weight (kg)	remaining wt. (kg)
O_1	5	1	15-1 = 14
O_5	8	1	14-1 = 13
O_7	4	2	13-2 = 11
O_2	10	3	11-3 = 8
O_6	9	3	8-3 = 5
O_4	7	4	5-4 = 1
O_3	$15 \times \frac{1}{5} = 3$	1	1-1 = 0

After O_4, we had only 1 kg left. So we took a fraction of O_3. That is,
wt of O_3 = 5kg ; 1 kg of O_3 = 3 profit.
↓
15 profit.

Total profit = 46

2# S_2 ↓ based on ↑ profit.

Object	profit	weight	remaining weight
O_3	15	5	15-5 = 10
O_2	10	3	10-3 = 7
O_6	9	3	7-3 = 4
O_5	8	1	4-1 = 3
O_4	$7 \times \frac{3}{4} = 5.25$	$4 \times \frac{3}{4} = 3$	3-3 = 0

Total profit = 47.25

3# S3:

Object	Profit	Weight	Remaining wt.
			15-1 = 14
O5	8	1	14-1 = 13
O1	5	1	13-3 = 10
O2	10	3	10-5 = 5
O3	15	5	5-3 = 2
O6	9	3	2-2 = 0
O7	4	2	

Total profit = 51

So,

S1 = 46
S2 = 47.25
S3 = 51 ✓

We choose S3 because it is most optimised.

In exam if this qn comes, directly do S3. No need to do S1 and S2 as those are there only for proof.

Do S1 if they ask specifically for it.

Camlin

Q1.

Objects	1	2	3	4
wt	3	4	5	6
profit	4	6	7	11

constraints = ~~wt~~ max wt = 9kg
max profit should be there.

A. Using P/W ratio.

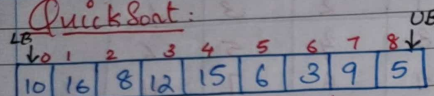
Objects	1	2	3	4
profit	4	6	7	11
wt	3	4	5	6
P/W	1.33	1.5	1.4	1.83

Object	profit	wt	rem. wt
4	11	6	9-6 = 3
2	$6 \times \frac{3}{4} = 4.5$	$4 \times \frac{3}{4} = 3$	3-3 = 0

Max profit = 11 + 4.5
= 15.5

Camlin

QuickSort:



We use divide and conquer strategy and fix the position of

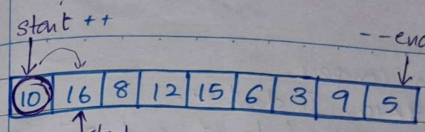
Step 1: Select a pivot element. It may be first, last, random, or median element.

Step 2: Partition step. In this elements less than pivot element should go to the left of pivot and those greater should go to its right.
Repeat this multiple times if you get the list sorted.

We need two pointers in this. First we fix A[0] as start pointer and A[8] as end pointer.
start pointer - either increments (+) or stops.
end pointer - either decrements (-) or stops.

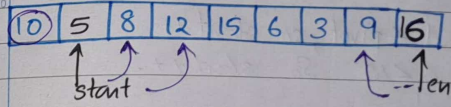
start - increment if element is lesser than pivot
end - decrement if element is greater than pivot.

- In the above example, say (10) is the pivot selected.

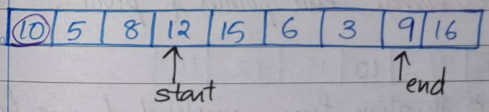


if pivot is the first element, then we increment the start pointer immediately.

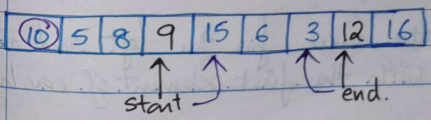
Now 16 > 10, so decrement end pointer. but at end pointer, 5 < 10. So now exchange start & end pointer values.



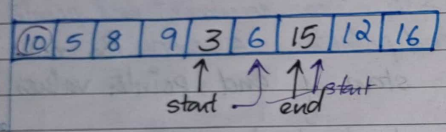
Now again check with start pointer. 5 < 10. So start ++.
Now A is 8 < 10, Again start ++.
Now it is 12 > 10. Go to end pointer.
16 > 10. So decrement end pointer.



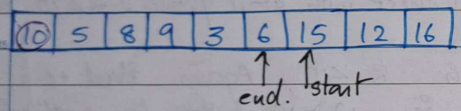
Now we exchange. Because end < pivot & start > pivot.



Now again compare.
9 < 10. So start ++.
Now 15 > 10. So look at end pointer.
12 > 10. So decrement end pointer. end--.
Now 3 < 10. That is end < pivot & start > pivot. So exchange.



Now start < pivot, so start ++
Now 6 < 10. So start ++.
Now end --.

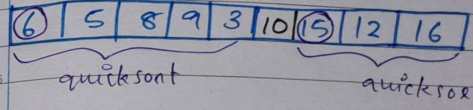


Since start pointer passed the end pointer,
Now exchange end pointer & pivot.

6	5	8	9	3	10	15	12	16

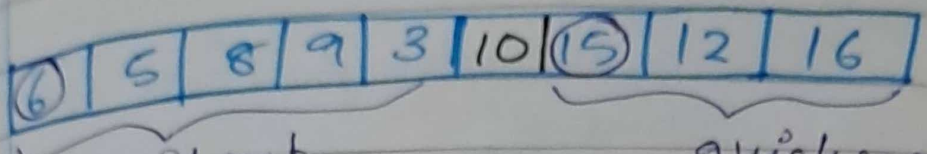
Now 1st iteration has completed & we have fixed position of pivot

Now do the same for the left and right subarrays with the first element of each subarray as the pivot (or as whatever you wish on pivot).



Repeat until you get sorted array.

subarray as the pivot (or as whatever you wish as pivot).

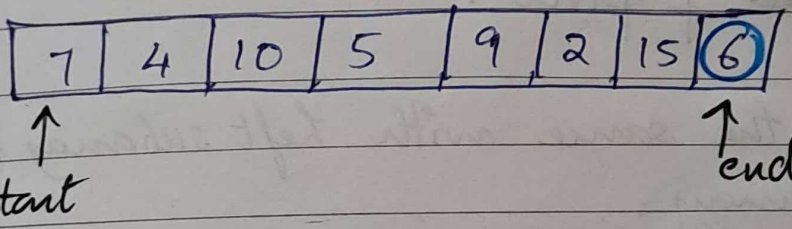


Repeat until you get sorted array.

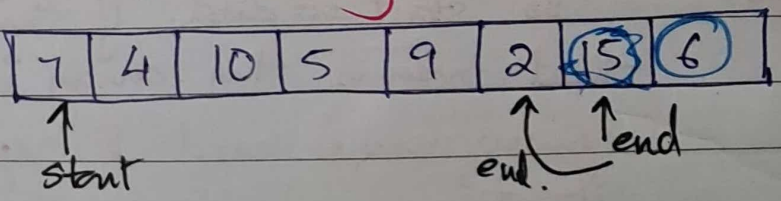
22/5/24/Wednesday.

Q. 7, 4, 10, 5, 9, 2, 15, 6. Take 6 as pivot & do quicksort?

A:

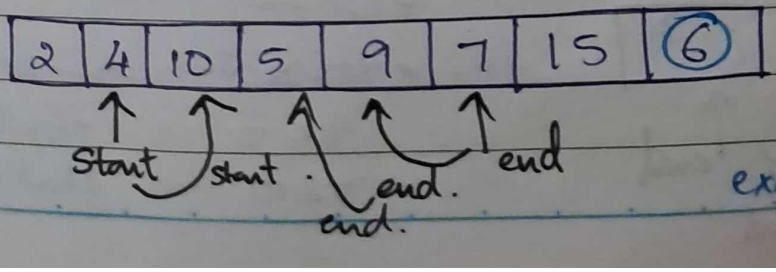


Here since pivot is at the end decrement end pointer first. Then do the same steps as done previously...

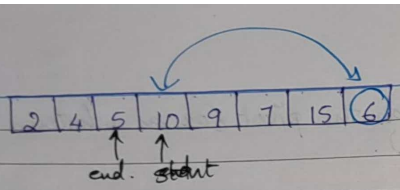


~~15~~ 15 > 7 so end--

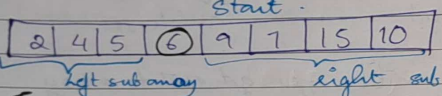
2 < 7 so exchange.



exchange 10 & 5.



So now exchange pivot with start.



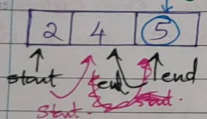
left sub array right sub array

First iteration is done.

Since pivot was the last element, exchange the start & pivot.

Now do the same with left subarray & right subarray.

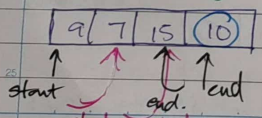
Left sub array:



last element is pivot.
end--
4 > 2 so start++

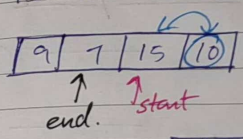
so start == end. If this happens, that means the array is sorted. So no more need to repeat the iterations.

Right sub array:

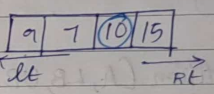


end
9 < 15 so
9 < 10 so start++
7 < 10 so start++

15 > 10 so end--

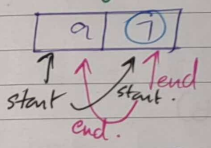


Now exchange start & pivot

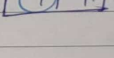


2nd iter done.

Lt sub array:

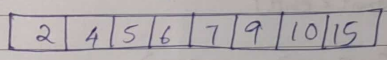


7 < 9 so end--
start++

3rd iter done 

Now exchange,

So final array.



* if the pivot was random in the first iter then it is random in remaining iters as well.

30/5/24 Thu.

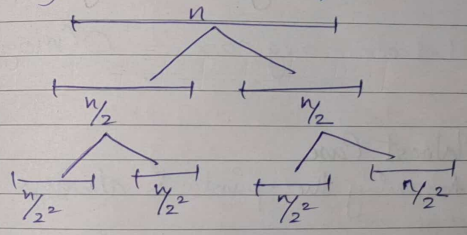
QuickSort Algorithm & Time Complexities

```
QuickSort(A, LB, UB)                          ——— T(n)
{
  if (LB < UB)                                ——— 1
  {
    loc = partition (A, LB, UB);              ——— partition time
    QuickSort (A, LB, loc-1);                 } depends on case.
    QuickSort (A, loc+1, UB);
  }
}

partition (A, LB, UB)
{
  pivot = A[LB];
  start = LB;
  end = UB;
  while(start < end)
  {
    while (A[start] <= pivot)
    {
      start ++;
    }
    while (A[end] > pivot)
    {
      end--;
    }
    if (start < end)
    {
      swap (A[start], A[end]);
    }
  }
  swap (A[LB], A[end]);
  return end;
}
```

Best Case & Average Case.

Assuming pivot position is always in the middle. That is in each iteration we are dividing into 2 equal parts.



So,

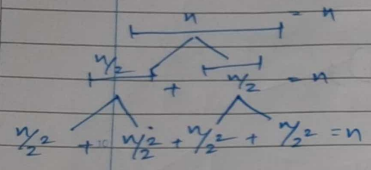
QuickSort (A, LB, loc-1); ——— $T(n/2)$
QuickSort (A, loc+1, UB); ——— $T(n/2)$

So,

$T(n) = 1 + \text{partitionTime} + 2T(n/2)$

↓

each iteration takes n time.



So, $T(n) = 1 + n + 2T(n/2)$

↓ asymptotically

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

time complexity (TC)

$a = 2 \quad b = 2 \quad \log_b a = 1 = k = 1 \quad p = 0 \Rightarrow p > -1$

$TC = n^k \log^{P+1} n = n \log n$

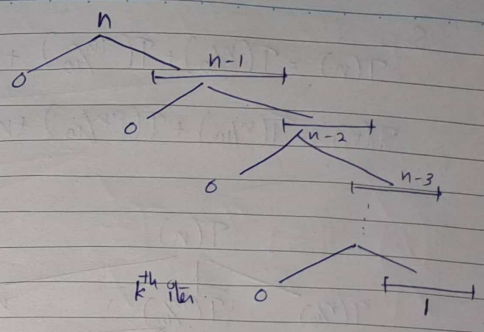
So bestcase = avg case = $O(n \log n)$

2# Worst Case

Assuming the pivot is always at the extreme end.

For calculating time complexity, lets consider it being at LB all the time...

(pivot)



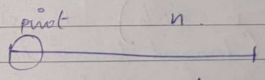
So time $= n + (n-1) + (n-2) + \cdots + 2 + 1$

$= \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$

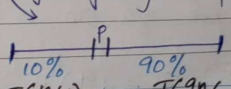
$= O(n^2)$

∴ Worst case time complexity = $O(n^2)$

3# Special Case - Unbalanced Tree.



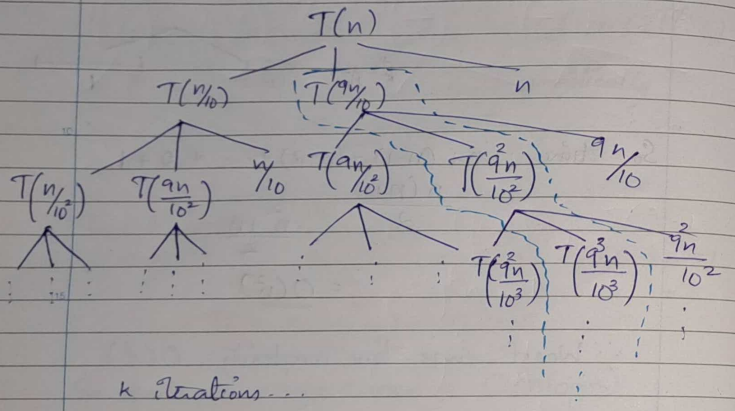
(after fitting the pivot's pos.



in every case you compare with pivot as n

So, $T(n) = T(n/10) + T(9n/10) + \text{partition time}$

$T(n) = T(n/10) + T(9n/10) + n$ → Recurrence Relation.



k iterations...

Since these are imbalanced trees (one part is 10% & other is 90%) so we take the longest branch (longest branch has the greatest value among all the branches).

So we go as ⇒ $\frac{9n}{10} \to \frac{9^2n}{10^2} \to \frac{9^3n}{10^3} \to \cdots$

So, K^{th} iter. $= \left(\frac{9}{10}\right)^k n = T\left[\left(\frac{9}{10}\right)^k n\right]$ //

Now equate to min: value min value = 1 @ $n/b = 1$ for $T(n/b)$.

ie; $\left(\frac{9}{10}\right)^k n = 1$

$\left(\frac{9}{10}\right)^k = \frac{1}{n}$

$n = \left(\frac{10}{9}\right)^k$

$k = \log_{10/9} n$

Time taken for : (add the non-TC branches)

1^{st} iter $= n$

2^{nd} iter $= \frac{n}{10} + \frac{9n}{10} = n$

3^{rd} iter $= \frac{n}{10^2} + \frac{9n}{10^2} + \frac{9n}{10^2} + \frac{9^2n}{10^2}$

$= n$

$\vdots$

k^{th} iter $= n$

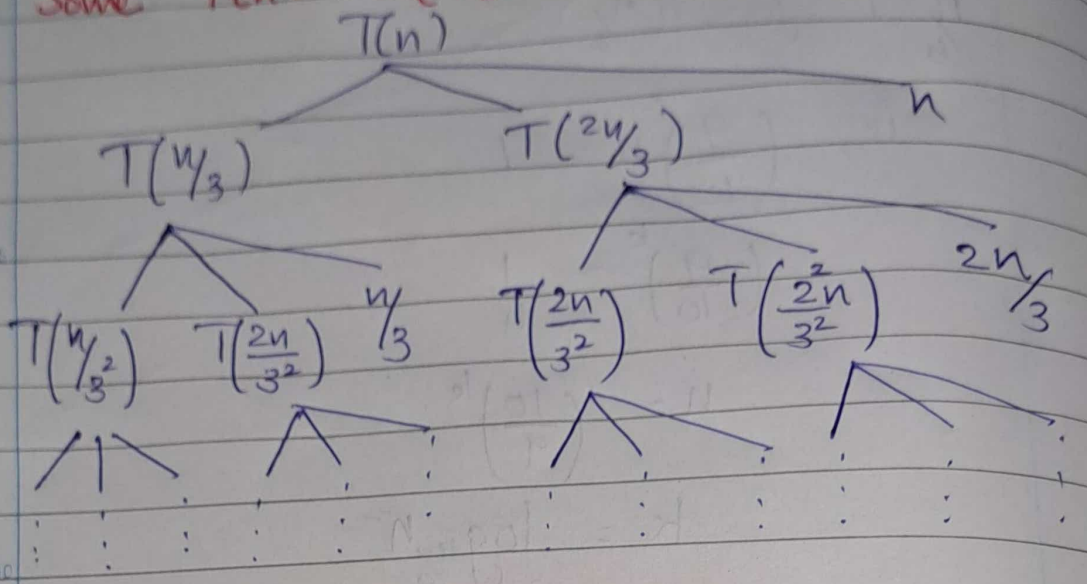
So each level time taken = n

There are k such levels, so,

Time complexity = $k \cdot n$

$TC = n \log_{10/9} n$

Q. Solve $T(n) = T(n/3) + T(2n/3) + n$



k levels ...

Taking longest branch, $T\left(\left(\frac{2}{3}\right)^k n\right) = T(1) = 1$

$$\left(\frac{2}{3}\right)^k n = 1$$

$$\left(\frac{2}{3}\right)^k = \frac{1}{n}$$

$$n = \left(\frac{3}{2}\right)^k$$

$$k = \log_{3/2} n$$

Similarly each iteration takes time $= n$

So time complexity $= n \cdot \log_{3/2} n$

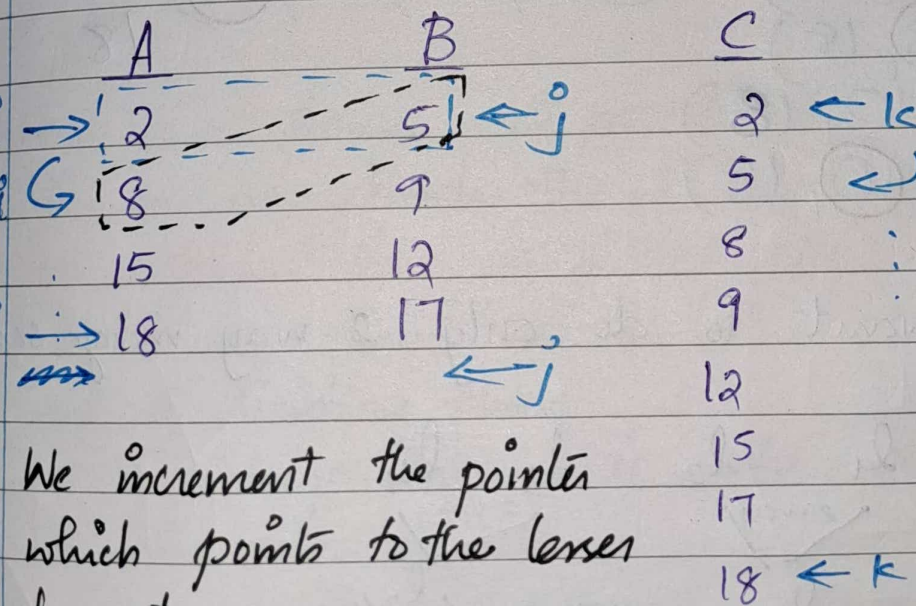
mid sem questions over...

(Unbalanced tree not present) ----

Merge Sort

- 2 Way Merge Sort → iterative
- Merge Sort → recursive.

- We use divide & conquer approach.
- We will have 2 sorted lists & we merge both & create another sorted list.



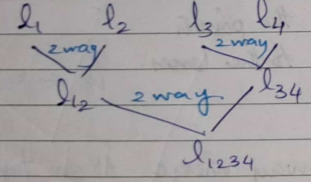
We increment the pointer which points to the lesser element.

This is 2 way merge. At a time we are comparing 2 elements so it is 2 way merge. If at a time 4 elements are compared then it is 4-way merge.

A	B	C	D	F
				2
4	3	8	2	3
6	5	10	4	4
12	9	16	18	4
				5
(4, 3, 8, ②)				6
(4, ③, 8, 4)				8
(④, 5, 8, 4)				9
(6, 5, 8, ④)	(⑫, —, 16, 18)			10
(6, ⑤, 8, 18)	(—, —, ⑯, 18)			12
(⑥, 9, 8, 18)	(—, —, —, ⑱)			16
(12, 9, ⑧, 18)				18
(12, ⑨, 10, 18)				
(12, —, ⑩, 18)				

- follow FIFO nxt

If you want to do only 2-way merge sort then,



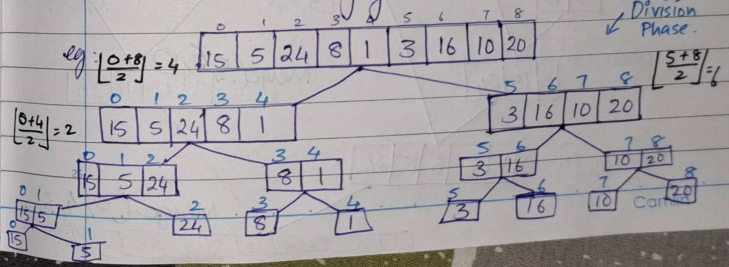
Merge Sort Algorithm:

```
MergeSort (A, lb, ub)
{
  if (lb < ub)
  {
    mid = ⌊(lb+ub)/2⌋      → take floor value.
    MergeSort (A, lb, mid);
    MergeSort (A, mid+1, ub);
    merge (A, lb, mid, ub);
  }
}
```

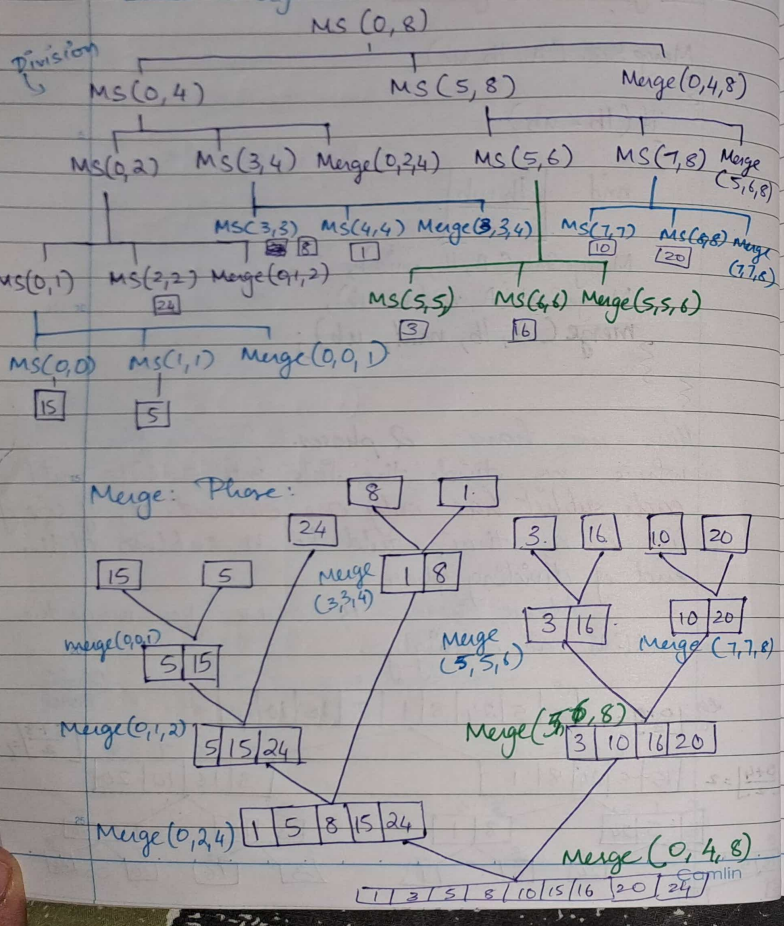
Here we have 2 phases. Dividing Phase where we divide the lists into sublists until each sublist has only one element. ie; if size of list is n, there would be n sublists at the end of dividing phase.

Merge Phase is where you merge the sublists accordingly.

eg:



Based on algorithm:



∴ Final array is: 1, 3, 5, 8, 10, 15, 16, 20, 24.

31/5/24 Friday.

Merge Algorithm

```
Merge(A, lb, mid, ub)
{
  i = lb;
  j = mid+1;
  k = lb;
  while(i<=mid && j<=ub)
  {
    if (a[i] <= a[j])
    {
      b[k] = a[i];
      i++;
      k++;
    }
    else
    {
      b[k] = a[j];
      j++;
      k++;
    }
```

```
if (i>mid)
{
  while (j<=ub)
  {
    b[k] = a[j];
    j++;
  }
  k++;
}
else
{
  while (i<=mid)
  {
    b[k] = a[i];
    i++;
    k++;
  }
}
for (k=lb; k<=ub; k++)
{
  a[k] = b[k];
}
```

```
MergeSort (A, lb, ub)  ——— T(n)
{
  if (lb < ub)  ——— 1
  {
    mid = ⌊(lb+ub)/2⌋  ——— 1
    MergeSort (A, lb, mid);  ——— T(n/2)
    MergeSort (A, mid+1, ub);  ——— T(n/2)
    Merge (A, lb, mid, ub);  ——— n
  }
}
```

For Merge algorithm, the time taken is always n because we are comparing each and every element and merging them one sublist at a time if initially all sublists have only one element. Like this n sublists are compared.

As for the MergeSort recursive calls, we always divide the list into half. So both the calls take $T(n/2)$ time.

Recurrence Relation ⇒

$$T(n) = 1 + 1 + T(n/2) + T(n/2) + n$$

$$\boxed{T(n) = 2T(n/2) + n}$$

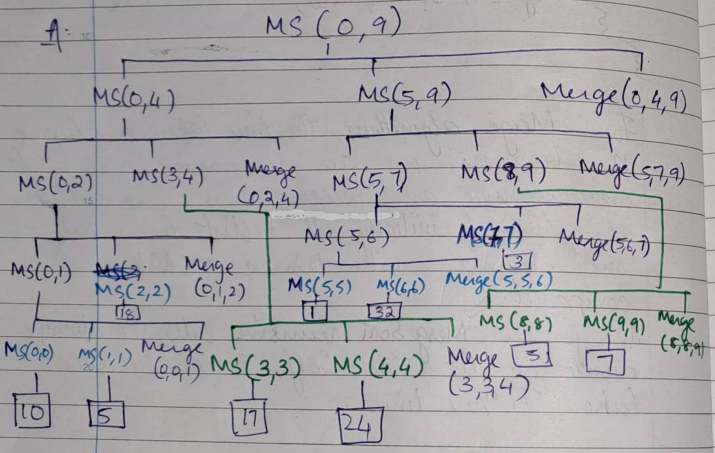
$$\boxed{TC = n \log n}$$

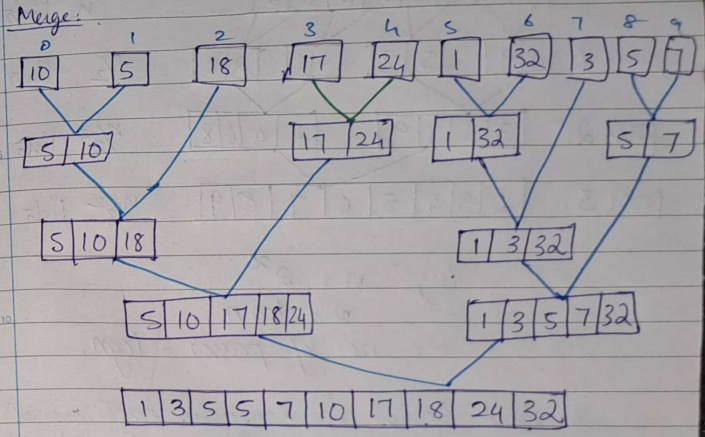
That is, MergeSort has same time complexity as best/avg case of quicksort.

This time complexity is same for best, avg and worst case of merge sort.

Q. Sort these elements using MergeSort?
{10, 5, 18, 17, 24, 1, 32, 3, 5, 7}

A:





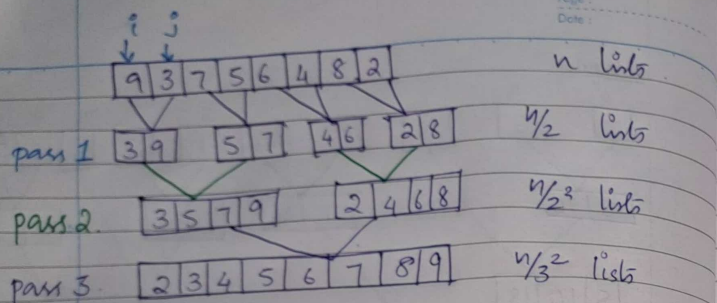
1/6/24 | Saturday

2 Way Merge Sort:

	0	1	2	3	4	5	6	7
A	9	3	7	5	6	4	8	2

Here there is no division phase. There is only merge phase present.

Here each element is considered as a list & the 2 way merge sort is done... Merge operation...



ie; $n = 2^x$

$x = \log n$

∴ no. of passes = $\log n$.

In each pass, we compare each and every element each time. So time in each pass = n.

∴ Time complexity = $n \cdot$ (no. of passes)

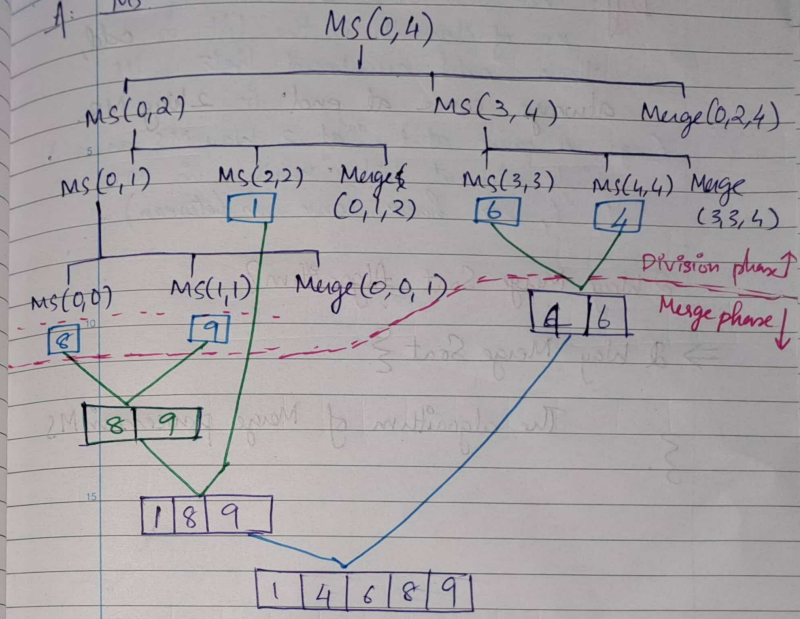
$= n \cdot \log n$

For best, avg, & worst cases.

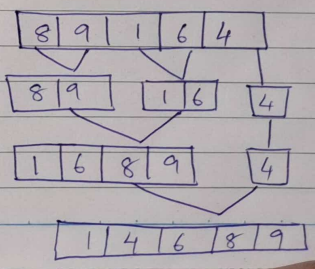
Q. Sort $\{8, 9, 1, 6, 4\}$ using 2 way Merge Sort & Merge Sort. Explain the diff: observed?

Camlin

A: MS



2 Way MS



Camlin

Page :
Date :

Diff: We can observe that if the no: of elements in the list is odd, then odd numbered lists will always come at end in 2 Way MS. (eg: 4 came at the end of 2 Way MS as a single element list while in MS, 1, 6, etc have come in between).

2 Way Merge Sort Algorithm?

⇒ 2 Way Merge Sort {

The algorithm of Merge process in MS

}.