Design & Analysis of Algorithm.

To complete revisions (4 Days)

S.R No.	Topic	Module	Difficulty level	No. of Revision
1	Time & Space Analysis	1	M+	
2	Sorting Tech.	2	M+	
3	Seaching Algo.			
4	Greedy Algo.	3	M+	
5	Dijkstra Algo.	4	M+	
6	Bellman-ford			
7	Shortest path in DAG's			
8	Dynamic Prog.	5	M+	
9	Matrix-chain Multiplication			
10	Longest common subsequence	6	M+	
11	Multi stage Graph		M+	
12	0/1 Knapsack		M+	
13	Subset Sum		M+	
14	Travelling Salesman Problem	7	M+	
15	Allpairs shortest path		M+	
16	Np- Completeness	8 Not in Gate Syllabus	Just for sake of Knowledge	
17	Problems on Np.			

Move to 2nd Notebook

Time and space Analysis (Module-1)

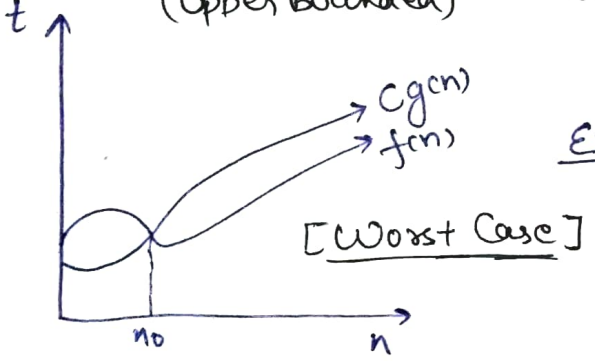
Introduction to Asymptotic Notations

Informal description of problem. (Algorithms)

Every Program can have many Algo. but we choose best in case of 'time and memory'.

i) Big (Oh) → O

(Upper Bounded)



Condition:- $f(n) \leq cg(n)$ for $n \geq n_0, c > 0, n_0 \geq 1$

$\boxed{f(n) = O(g(n))}$

Ex → Let us say $f(n) = 3n+2$ and $g(n) = n$

then for, $3n+2 \leq cn$

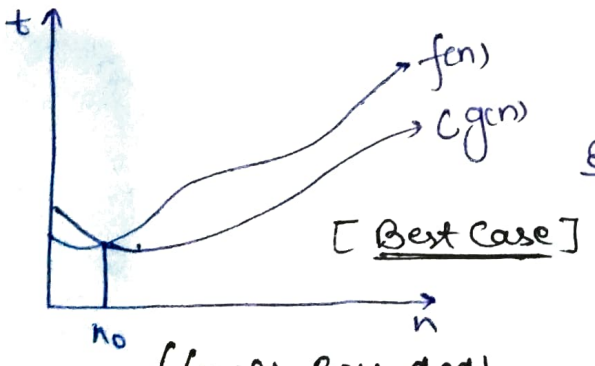
say $\boxed{c=4}$.

$\Rightarrow 3n+2 \leq 4n$

$\Rightarrow \boxed{n \geq 2}$.

So, $f(n) = O(g(n))$ and it will true for $n^2, n^3, n^n \ldots 2^n$ or anyone but we reach for highest bound.

ii) Big Omega (Ω)



(Lower Bounded)

Condition:-

$f(n) \geq cg(n)$, $n \geq n_0, c > 0, n_0 \geq 1$

$n_0 \rightarrow$ input so it has to be atleast 1.

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

If $f(n) = \Omega g(n)$ so,

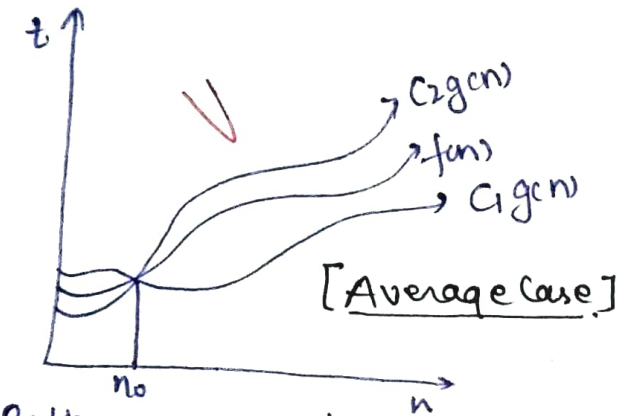
$3n+2 \geq nc$

True for $c = 1$ or $n_0 \geq 1$

So, $\boxed{3n+2 = \Omega(n)}$

If $f(n) = \Omega(n)$ so it may have $\log n$, $\log \log n$ and many more that is less than $f(n)$. But we go for the tightest bound.

Big theta (Θ)



[Average Case]

[Both upper and lower bounded.]

Condition:

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

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

$c_1, c_2 > 0$

$n \geq n_0, \; n_0 \geq 1$

$f(n) = 3n+2 \quad g(n) = n$

$f(n) \leq c g(n)$ for $c_2 = 4, \; n_0 \geq 1$

$f(n) \geq c g(n)$ for $c_1 = 1, \; n_0 \geq 1$

So, $\boxed{c_1 = 1, \; c_2 = 4}$

Also called as asymptotic equal as.

$3n^2 + n + 1 = \Theta(n^2)$

$2n^3 + n = \Theta(n^3)$

> We opt max. power term only.

We always interested in O because it is worst.

If in any case O and Ω are same then we go for it.

Ex

5	7	1	2	4	10	11

Let a element to be search in array using linear search

① Best Case → You are searching for 5 and found at 1st. $\Omega(1)$.

② Worst Case → You are searching for 11 and found at last. $O(n)$.

③ Avg. Case → You are searching for 2 and found in between $\Theta(n/2) = \Theta(n)$

Time Complexity analysis of Iterative programs

$f(n)$ → not a real time but approx. time

Algorithms are of two types -

i) Iterative Algo.

```
A(){
 for i = 1 to n
   max(a,b)
}
```

In iterative type of program we count no. of steps repeated.

② Recursive Algo

```
A(n){
  if()
  A(n/2) }
```

These Algo. are analysed using recursive equations.

Both the types are similar but the way of analysis is different.

If there is no any iterative or recursive call then program will take constant time of execution O(1).

Ex

```
A() {
  i=1; s=1;
while (s<=n)
{ i++;
  s=s+i;
  Pf("ravi");
}
```

S	1	3	6	10	15	21	...	n
i	1	2	3	4	5	6	...	k

So, S = sum of i^{th} natural number

Let us say i changes to value k so

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

In worst case

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

$$\Rightarrow k^2 > n$$

$$\Rightarrow k > \sqrt{n}$$

So,

$$\boxed{k = O(\sqrt{n})}$$

②

```
for (i=0; i*i<=n; i++) {
  Pf("");
}
```

$i <= \sqrt{n}$

$\boxed{O(\sqrt{n})}$

③

```
A() {
  int i, j, k, n;
for (i=1; i<=n; i++)
{
 for (j=1; j<=i; j++)
 {
 for (k=1; k<=100; k++)
  { Pf("ravi")
  }
 }
}
```

i = 1	i = 2	i = 3
j = 1 times	j = 2 time	j = 3 times
k = 100 times	k = 2×100	k = 3×100

... i = n

j = n times

k = n×100 times

Total no. of times loop 3 will run is -

$$n*100 + (n-1)100 + \dots 3*100 + 2*100 + 100$$

$$= 100(1+2+3+\dots+n)$$

$$= 50(n(n+1))$$

$$= O(n^2)$$

Not $(O(n^3))$

It is necessary to analyse the question before respond.

④ A() {
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++) {
Pf ("ravi")
}

i = 1	i = 2	i = 3
j = 1	j = 2 time	j = 9 time
k = n/2 * 1	k = n/2 * 4	k = n/2 * 9

i = n
j = n^2
k = n/2 * n^2

So, k will execute :

$\frac{n}{2}(1+4+9+\ldots+n^2)$

$= \frac{n}{2}\left(\frac{n(n+1)(2n+1)}{6}\right) = \boxed{O(n^4)}$

⑤ for (i=1 ; i<n ; i = i*2)
Pf (ravi);

~~if~~ i = i*3 → $\log_3 n$
i = i*6 → $\log_6 n$
i = i*k → $\log_k n$

i = 1, 2, 4, 8, 16 ... n
2^0 2^1 2^2 2^3 2^4 ... 2^k
After k iteration

$2^k = n$
$k = \log_2 n$
$O(\log(n))$.

⑥ A(){
int i, j, k;
for (i=n/2 ; i<n ; i++) { ⟶ n/2 times
for (j=1 ; j<=n/2 ; j++) { ⟶ n/2 times
for (k=1 ; k<=n ; k=k*2) { ⟶ $\log_2 n$ times
Pf ("ravi")
}}}

All the loops are independent from other loop.

total $\frac{n}{2} * \frac{n}{2} * \log_2 n$
$= O(n^2 \log_2 n)$

⑦ for (i= n/2 ; i<=n ; i++) → n/2
for (j= 1 ; j<n/2 ; j=2*j) ; → $\log_2(n/2)$
for (k= 1 ; k<n ; k=k*2) ; → $\log_2(n)$
Pf (ravi)
}}}

$n/2 * \log_2(n/2) * \log_2 n$
$= O(n \log n/2 \log n)$.

⑧ while (n>1)
{ n = n/2
}
Asume n>=2.

If n = 1 2 3 ... n
Loop execute 2^1 2^2 2^3 ... 2^k
$n = 2^k$
$\Rightarrow \boxed{k = O(\log_2 n)}$

But for $n = 20$

20 → 10 → 5 → 2 → 1 → 4 times

so $\lfloor \log_2 20 \rfloor$

So, complete ans $O(\lfloor \log_2 n \rfloor)$

(9)

```
for(i=1; i<=n; i++)
for(j=1; j<=n; j=j+1)
    Pf("ravi");
}
```

$i=1$	$i=2$	$i=3$	...	$i=k$	$i=n$
$j=1$ to n	$j=1$ to n	$j=1$ to n		$j=1$ to n	$j=1$ to n
n times	$n/2$ times	$n/3$ times		n/k time	$n/n=1$ time

$\Rightarrow$ total no. of times loop execute $= n + \frac{n}{2} + \cdots \frac{n}{k}$

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

$\Rightarrow n \log n = \boxed{O(n \log n)}$

(10)

```
Al(){
  int n = 2^k;
for(i=1; i<=n; i++) ⟶ n times
  {
   j=2;
   while(j<=n)
   { j = j^2;
     Pf("ravi");
   }
  }
}
```

for $k=1$	$k=2$	$k=3$
$n=4$	$n=16$	$n=256$
$j=2,4$	$j=2,4,16$	$j=2,4,16,256$
$n*2$ times	$n*3$ times	$n*4$ times

total $n*(k+1)$ after k iterations

say $n = 2^{2^k}$

$\Rightarrow k = \log\log n$

$\Rightarrow n(\log\log n + 1)$

$\Rightarrow \boxed{O(n \log\log n)}$

Time Complexity for recursive Program

Ex ① A(n){
if ( )
return (A(n/2) + A(n/2))
}

$T(n) = C + 2T(n/2)$ → Recursive Eqn
↳ for if ( )

② A(n){
if (n>1)
return (A(n-1))}

$T(n) = 1 + T(n-1)$ — ①

Using Back Substitution method (slow method)

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

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

$T(n) = 1 ; n = 1$
Base condn

So $n-k=1 \Rightarrow k = n+1$.

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

$\boxed{T(n) = O(n)}$

③ $T(n) = n + T(n-1) ; n>1$
$= 1 ; n = 1$

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

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

for $n-(k+1) = 1$
$\Rightarrow k = n-2$

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

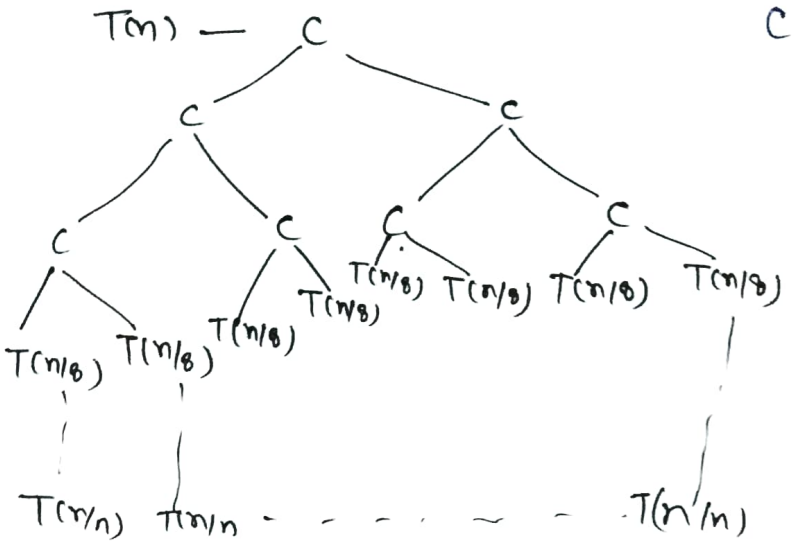
$\Rightarrow$ ~~$T(n) = n(n-1)+(n-2)+1$~~
$= (n^2-n+2n+3$
$= (n^2+2n+3$
$= O(n^2)$.

$= n+(n-1)+(n-2)+ \ldots (n-(n-2)) + T(n-(n-2+1))$
$= n(n-1)+(n-2)+ \ldots 2+1$
$= \frac{n(n+1)}{2} = \boxed{O(n^2)}$

Recursion Tree method

$T(n) = 2T(n/2) + c \; ; \; n>1$

$= c \; ; \; n=1$



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

Total time taken =

$c + 2c + 4c + \dots + nc$

$c(1+2+4+\dots n)$

Let us say $n = 2^k$

$c(1+2+4+\dots+2^k)$

$= \frac{c(1(2^{k+1}-1))}{2-1}$

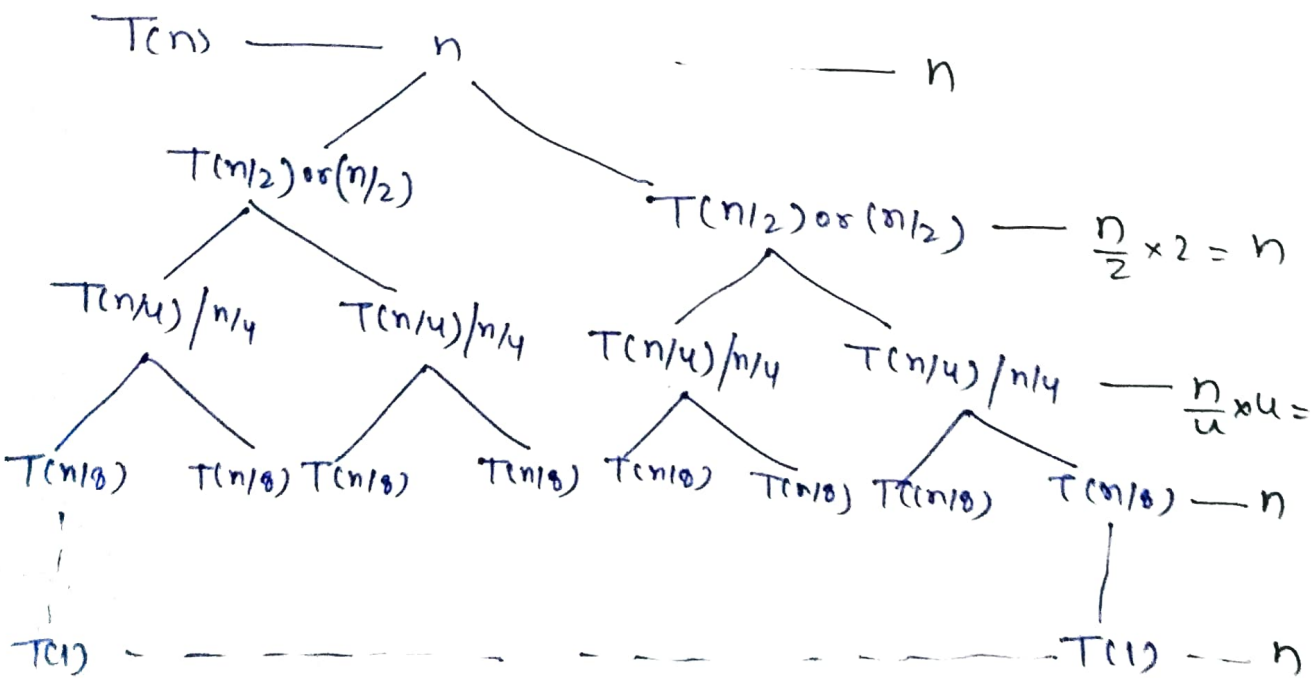
$= c(2^{k+1}-1)$

$= c(2n-1)$

$= \boxed{O(n)}$.

Ex $T(n) = 2T(n/2) + n \; ; \; n>1$

$= 1 \; ; \; n=1$



Total amount of work = n + total no. of levels.

Tree is shrinking in a way $= \frac{n}{2^0} + \frac{n}{2^1} + \dots + \left(\frac{n}{2^k}\right) \to 1$

$\Rightarrow n = 2^k$

$\Rightarrow \boxed{k = \log_2 n}$

Total no. of levels = $k+1 = \log_2 n + 1$

total time taken = $\boxed{n*(\log_2 n+1)}$

$= \boxed{O(n \log n)}$.

Comparing various functions to analyse time complexity:-

Master's Theorem

Compare two functions → Cancel the common part

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

Apply log on both sides

$n \log_2 2$ $2 \log n$

n $2 \log_2 n$

Put $n = 2^{100}$

2^{100} 200

So, $f(n) > g(n)$

② $f(n) = 3^n$ $g(n) = 2^n$

$\log 3$ $\log 2$

$\log 3 > \log 2$

So $f(n) > g(n)$

③ $f(n) = n^2$ $g(n) = n \log n$

n $\log n$

$\boxed{f(n) \geq g(n)}$

④ $f(n) = n$ $g(n) = (\log n)^{100}$

$\log n$ $100 \log(\log n)$

Put $n = 2^{128}$

$f(n) = 128$ $100 + 7 = 700$

Put $n = 2^{1024}$

$f(n) = 1024$ 100×10

1000

After some extent n is growing more. So,

$f(n) > g(n)$

⑤ $f(n) = n^{\log n}$

$g(n) = n \log n$

Apply log on both sides

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

$(\log n + \log \log n) = g(n)$

$n = 2^{1024}$

$f(n) = 1024 * 1024$

$g(n) = (1024 + 10)$

So, $f(n) > g(n)$

$O(n^{\log n})$

⑥ $f(n) = \sqrt{\log n}$ $g(n) = \log \log n$

$\frac{1}{2} \log \log n$ $\log \log \log n$

$n = 2^{2^{10}}$

$f(n) = \frac{10}{2} = 5$

$g(n) \approx 3.5$

So $f(n) > g(n)$

⑦ $f(n) = n^{\sqrt{n}}$ $g(n) = n^{\log n}$

$\sqrt{n}\log n$ $\log n \log n$

$\sqrt{n}$ $\log n$

$\frac{1}{2}\log n$ $\log\log n$

$\boxed{n^{\sqrt{n}} > n^{\log n}}$

put $n = 2^{128}$

$\frac{1}{2}(128) = 64 = f(n)$

$g(n) = 7$

Take more than one value of n and check

⑧ $f(n) = \begin{cases} n^3 & 0 < n < 10000 \\ n^2 & n \geq 10000 \end{cases}$ $g(n) = \begin{cases} n & 0 < n < 100 \\ n^3 & n > 100 \end{cases}$

	0-99	100-9999	10000,----
f(n)	n^3	n^3	n^2
g(n)	n	n^3	n^3

So for $n_0 \geq 10000$

$g(n) > f(n)$

$\boxed{f(n) = O(g(n))}$

⑨ $f_1 = 2^n$, $f_2 = n^{3/2}$, $f_3 = n\log n$, $f_4 = n^{\log n}$

Compare f_1 and f_2

$n\log 2$ $3/2 \log n$

$\downarrow$ $\downarrow$

n $\frac{3}{2}\log n$

put $n = 2^{128}$

$f_1 = 2^{128}$ $f_2 = \frac{3}{2} \times 128 = 64$

So, $\boxed{f_1 > f_2}$

Compare f_1 and f_3

$n\log_2 2$ $(\log n + \log\log n)$

$\downarrow$

n

$n = 2^{128}$

$f_1 = 2^{128}$ $f_3 = (128 + 7)$

So, $\boxed{f_1 > f_3}$

Compare f_1 and f_4

$n\log 2$ $\log n \log n$

$\downarrow$

n

put $n = 2^{128}$

$f_1 = 2^{128}$

$f_4 = 128 \times 128 = 2^{14}$

So, $\boxed{f_1 > f_4}$

So, $f_1 = 2^n$ is greater than all.

Compare f_2 and f_4 $\quad$ so, $f_4 > f_2$

$n^{\log n} \quad n^{3/2}$

Compare f_4 and f_3 $\quad$ put $n = 2^{128}$

$n^{\log n} \quad n \log n$ $\quad$ $f_4 = 128 \times 128$

$(\log n \log n) \quad (\log n + \log\log n)$ $\quad$ $f_3 = 128 + 3$

So, f_4 is second standing fn

Compare f_2 and f_3 $\quad$ put $n = 2^{128}$

$3/2 \log n \quad (\log n + \log\log n)$ $\quad$ $f_2 = \frac{3}{2} \times 128$

$f_3 = 128 + 7$

So, $f_2 > f_3$

So, $\boxed{f_1 > f_4 > f_2 > f_3}$

Note → (1) Always put large value of n.

(2) Compare with min. two values of n.

$(\log n)^2 \rightarrow \log n \log n$ } Both are same

$\log^2 n \rightarrow \log\log n$

Masters Theorem

$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$, $\forall\, a, b \in$ constant, $a \geq 1$.

1) If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$

2) If $a = b^k$,
   a) if $p > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$
   b) if $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log\log n)$
   c) if $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

3) If $a < b^k$
   a) if $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b) if $p < 0$, then $T(n) = \Theta(n^k)$

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

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

3.a) $a < b^k$

and $p=0$ so $\boxed{T(n) = \Theta(n^2)}$

② $T(n) = 4T(n/2) + n^2$

$a=4 \quad b=2 \quad k=2 \quad p=0$

$a = b^k$ and $p > -1$

So, 2.a)

$\boxed{T(n) = \Theta(n^2 \log n)}$

③ $T(n) = T(n/2) + n^2$

$a=1 \quad b=2 \quad k=2 \quad p=0$

$a < b^k \quad p=0$

$\boxed{T(n) = \Theta(n^2)}$

④ $T(n) = 2^n T(n/2) + n^n$

$a = 2^n$ which is not a constent So we can't apply masters Theorem here.

⑤ $T(n) = 16 T(n/4) + n$

$a=16, \; b=4, \; k=1, \; p=0$

$a > b^k$

$\boxed{T(n) = \Theta(n^2)}$

⑥ $T(n) = 2T(n/2) + n \log n$

$a=2 \quad b=2 \quad k=1 \quad p=1$

$a = b^k, \; p > -1$

$\boxed{T(n) = \Theta(n \log^2 n)}$

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

⑦ $T(n) = 2T(n/2) + n/\log n$

$a=2 \quad b=2 \quad k=1 \quad p=-1$

$a = b^k \quad p=-1$

$\boxed{T(n) = \Theta(n \log \log n)}$

⑧ $T(n) = 2T(n/4) + n^{0.51}$

$a=2 \quad b=4 \quad k=0.51 \quad p=0$

$a < b^k \quad p > 0$

$\boxed{T(n) = \Theta(n^{0.51})}$

⑨ $T(n) = 0.5 T(n/2) + n^{-1}$

$a=0.5 \quad b=2 \quad k=-1$

Here $a<1$ so not valid bcoz $a>1$.

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

$a=6 \quad b=3 \quad k=2 \quad p=1$

$a < b^k \quad p > 0$

$\boxed{T(n) = \Theta(n^2 \log n)}$

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

↳ So we can't apply master theorem.

(12) $T(n) = 7T(n/3) + n^2$

$a=7,\ b=3\ \ k=2\ \ p=0 \qquad a < b^k \quad p>0$

$\boxed{T(n) = \Theta(n^2)}$

(13) $T(n) = 4T(n/2) + \log n$

$a=4\ \ b=2,\ k=0\ \ p=1$

$a \geq b^k \quad p=1$

$\boxed{T(n) = \Theta(n^2)}$

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

$a=\sqrt{2}\ \ b=2\ \ k=0,\ p>1$

$a > b^k$

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

$= \boxed{\Theta(\sqrt{n})}$

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

$a=2\ \ b=2\ \ k=1/2$

$a \geq b^k \quad p=0$

$\boxed{T(n) = \Theta(n)}$

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

$a=3\ \ b=2\ \ k=1\ \ p=0$

$a > b^k \quad p \neq 0$

$\boxed{T(n) = \Theta(n^{\log_2 3})}$

(17) $T(n) = 3T(n/3) + \sqrt{n}$

$a=3\ \ b=3\ \ k=1/2\ \ p=0$

$a > b^k$

$\boxed{T(n) = \Theta(n)}$

(18) $T(n) = 4T(n/2) + cn$

$a=4\ \ b=2\ \ k=1$

$p=0$

$a > b^k$

$\boxed{T(n) = \Theta(n^2)}$

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

$a=3\ \ b=4\ \ k=1\ \ p>1$

$a < b^k \quad p \geq 0$

$\boxed{T(n) = \Theta(n \log n)}$

(19) $T(n) \leq T(\lfloor \sqrt{n} \rfloor) + 1$

$\lfloor\ \rfloor \rightarrow$ floor fn

put $\rightarrow n = 2^{\log n}$

Analyzing space complexity of Iterative and recursive Algorithms

Amount of m/m cells required in form of given input n to solve the algorithm.

Time-Space trade off :- Sometimes less time but more space or vice-versa.

i) **Iterative Algorithm** :-

```
Algo(A,1,n)
{
  int i;
for(i=1 to n)
  A[i]=0;
}
```

Total space require = n + 1

n → for Array of size n

1 → variable i

But n is given in input so we can cancelled it out bcoz we are calculating the extra space,

So, S(n) = ~~n~~ + 1

= O(1).

Note → If there exist two variable i and j then n+2 will occur.

But Ignore n and S(n) = O(1) bcoz either 1 or 2 it is always related to constant space.

ii)
```
Algo (A,1,n)
{
  int i; Create B[n];
for(i=1 to n)
  B[i] = A[i];
}
```

Total amount of extra space require = n + 1

n → for B[n]

1 → for i.

So, $\boxed{S(n) = O(n)}$

Here we not consider the space for A[n] bcoz it given as input and we are calculating extra space.

iii) Algo (A, 1, n) {

Create B[n,n] —— n^2

int i, j; ——→ 2

for (i = 1 to n)

for (j = 1 to n)

B[i,j] = A[i];

}

Extra space require $= n^2 + 2$

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

Recursive Algorithm :-

A(n) {

if (n ≥ 1) { → # Anchor / Handling Condition

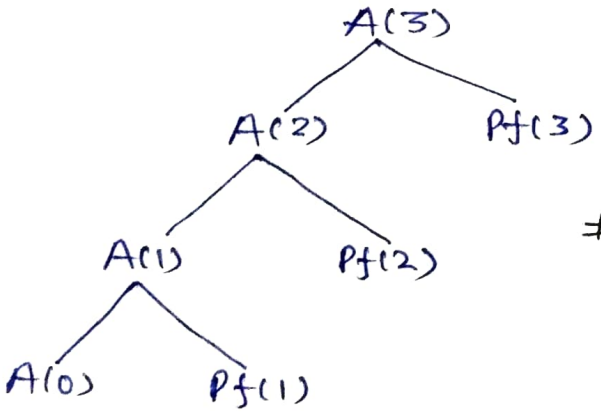
A(n-1); →# Head Recursion.

Pf(n);

} }

for small size → Use tree method

Soln Let n = 3

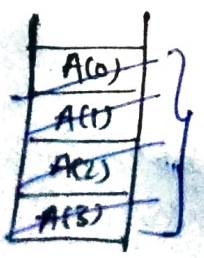


for (n=3) four recursive calls required.

So for n, n+1 recursive calls required.

On traversing from top to bottom and left to right if you got function call as node, insert it inside the stack and otherwise all the instruction are executed.

At the last visit of fn call pop it off from stack.



o/p - 1 2 3

for n → we need n+1 fn calls so (n+1) space in stack.

So total space $S(n) = O(n)$.

Time complexity

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

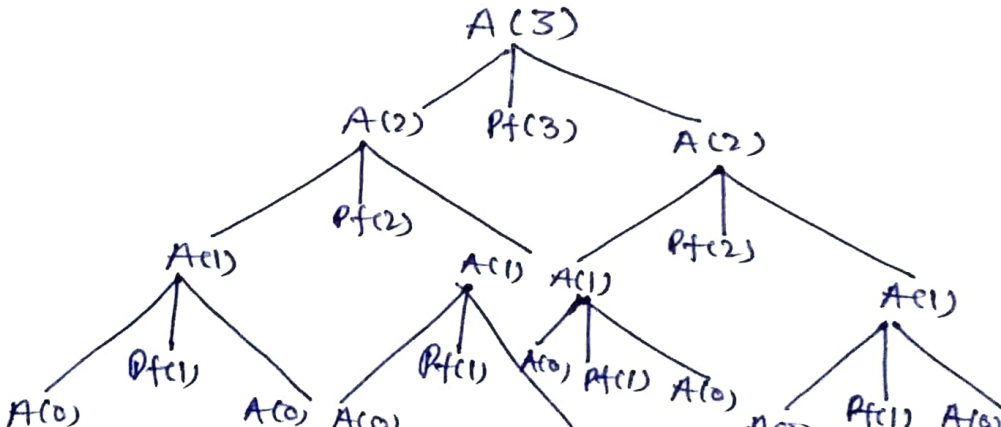
$$T(n) = T(n-3) + 3$$
$$\vdots$$
$$T(n) = T(n-k) + k$$

for $n-k \geq 0$, process stops $T(0) = 1$

$$T(n) = T(0) + n$$
$$T(n) = 1 + n = \boxed{O(n)}.$$

Q
```
A(n){
  if (n>=1){
    A(n-1);
     Pf(n);
    A(n-1);
  } }
```

for n = 3



Total no. of function calls. or recursive calls.

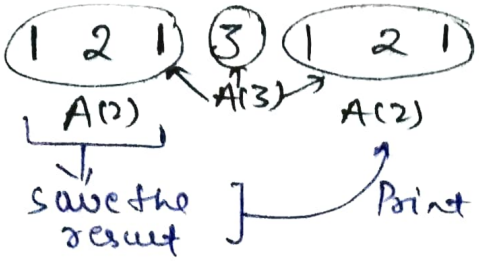
$A(3) \rightarrow 15 = 2^{3+1} - 1$
$A(2) \rightarrow 7 = 2^{2+1} - 1$
$A(1) \rightarrow 3 = 2^{1+1} - 1$
$A(n) \rightarrow \underline{2^{n+1} - 1}$.

O/P → 1213121

~~A(0)~~ ~~A(0)~~ ~~A(0)~~ ~~A(0)~~ ~~A(0)~~ ~~A(0)~~ ~~A(0)~~
~~A(1)~~ ~~A(1)~~ ~~A(1)~~
~~A(2)~~

max cells in a stack required = 4 (for n=3)
for A(n) times → n+1 cells required in stack.

So, space complexity $\boxed{S(n) = O(n)}$



we can do it by dynamic programming to reduce time complexity.

Time Complexity

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

$$T(n) = 1 \quad ; (n=0)$$

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

and

$$T(n) = 8T(n-3) + 2^2 + 2 + 1$$

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

$$\vdots$$

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

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

$$= \frac{1(2^{n+1} - 1)}{2-1} = 2^{n+1} - 1 = \boxed{O(2^n)}.$$

Amortized Analysis

In an amortized analysis, we average the time required to perform a sequence of data structure operation over all the operations perform.

→ Using an amortized analysis, we can show that the average cost of an operation is small, even though a single operation within the sequence might be expensive.

→ Used to analyze time complexities of hash tables, splay trees and disjoint sets.

→ Techniques used in amortized analysis:-

- Aggregate Method.
- Accounting Method
- Potential Method.

Aggregate Analysis :- $\frac{T_1 + T_2 + T_3 + \dots + T_n}{n}$

Hash Table Insertion

① Increase the size of table whenever it becomes full.

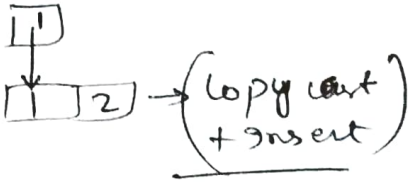
↳ Allocate m/m for a large table of size double the old table

→ Copy the contents of old table to new table

→ Free the old table

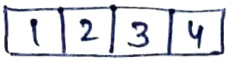
Example Initially table is empty and size is 0

Insert 1 (overflow) 



Insert 2 (overflow) 

Insert 3 (overflow) 

Insert 4 

Insert 5 (overflow) 

Item no :	1	2	3	4	5	6	7	8	9	10
Table size :	1	2	4	4	8	8	8	8	16	16
Cost :	1	2	3	1	5	1	1	1	9	1

(1 move 1 9nsert) (2 move 1 9nsert) (4 moves 1 9nsert) (8 moves 1 9nsert)

9nworst case for entering n element $T(n) = O(n)$

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

for n element performing n operations total cost

$= n * n = n^2$

But amortized cost $= \left(\frac{1+2+3+1+5+1+1+1+1+9\cdots}{n}\right)$

$= \frac{(\underbrace{1+1+1+\cdots+1}_{n \text{ times}}) + (1+2+4+8+\cdots)}{n}$ $\lfloor \log_2 n-1 \rfloor + 1$ time

$= \frac{n+2n}{n} = 3$

Amortized cost $= O(1)$

① Generating fn of fibonacci series $= \boxed{\frac{z}{1-z-z^2}}$

② If $T(n) = T(\sqrt{n}) + c$ → Substitute $n = 2^m$ and $T(2^m) = S(m)$
then, $S(m) = S(\frac{m}{2}) + c$.

③ $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$

④ $a^{\log_b n}$ → No of leaves in recursion Tree (Masters theorem)

⑤ $\log x = [1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}]$

→ $O(\log n!) = O(n \log n)$