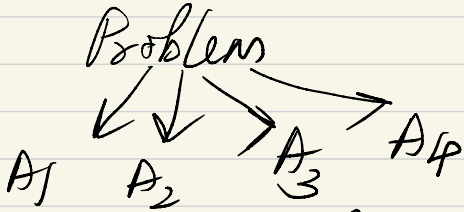
Design And Analysis of Algorithm

Algorithm

Algorithm ⟶ Program [Implementation of algo in some Programming language]

Problem ⟶ Algorithm ⟶ Program



Which one is better than other ?

In Analysis of algorithm we answer the following questions

1. When an algorithm is said to be better than other algorithm
   - better = will give me best
2. What is the criteria of Measurement

Complexity of code

Time → Running Time

Space ⟹ Not Bothered about very much

We will analyze algorithms to calculate the Running time of Entire Program

How to Measure time → Experimental Method

Drawback of Experimental Method

- Machine Dependent
- Language used Dependency
- We will get time only for a particular value of n
- We will have to implement the code

Hence we need a method which should have the following properties :-

- No machine Dependency
- No language Dependency
- No need of actual implementation
- For all inputs

We are interested only in the growth of the function

Asymptotic analysis will categorize function based on their growth

① Big O Notation

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

When growth of $g(n)$ is more or equal to $f(n)$

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

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

if c is > 0 & $n \geq n_0$

eg:

$f(n) = 3n+3$

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

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

We study algorithm for large value of n

	n	$f(n)$	$g(n)$
	1	6	3
	2	9	6
Critical Point	3	12	11
	4	15	18 → greater
	5	18	27
	6	21	38

(+3 applies to each $f(n)$ value)

Q. $f_1 = 2n$
$f_2 = 20n$ ⟹

n	f_1	f_2
1	2	20
2	4	40
4	8	80
8	16	160

(×2 down each column)

⟹ At Each step value is doubling The growth of f_1 & f_2 is same.

Q. $f_1 = 3n+3$
$f_2 = 2n^2+5n+4$

Method 2 [we this]
① Drop Lower Order Terms
② Drop constants

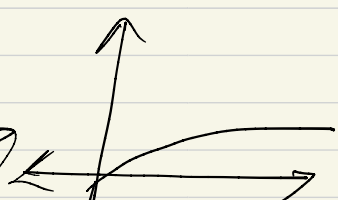
Method 1

n	f_1	f_2
1	6	11
2	9	22
3	12	37
4	15	56
5	18	79
6	21	106

(×3 down f_1)

⟹ $\boxed{f_1(n) = O(f_2(n))}$

Q $f_1 = \log_2(n)$
$f_2 = 2n$ ⟶
$f_1(n) = O[f_2(n)]$



By observing graph

Q $f_1 = 30n$
$f_2 = n^2 + 2n + 10$
$\Rightarrow f_1(n) = O[f_2(n)]$

Q. $f_1 = 2on$
$f_2 = 2n^2 + 3n + 5$
$f_3 = n^2$
$f_1(n) = O[f_2(n)]$
same growth functions

Q. $f_1 = 2n$ $f_3 = n$
$f_2 = 4n$
f_1, f_2 & f_3 have same growth rate

Q. $f_1 = n$ $f_3 = \log(n) \cdot 30$
$f_2 = 2n^2$
$\Rightarrow f_2 > f_1 > f_3$
$f_3(n) = O[f_1(n)]$
$f_1(n) = O[f_2(n)]$

Q $f_1 = 2^n$
$f_2 = 3^n$ $3^n > 2^n$

Q. $f_1 = \log(n)$, $f_2 = \log[\log(n)]$, $f_3 = 3n^2$, $f_4 = 100n + 20$, $f_5 = 100n \cdot \log_2(n)$

$f_2 < f_1 < f_4 < f_5 < f_3$

Q $f_1 = \sqrt{n}$
$f_2 = \log_2(n)$
$f_2(n) = O[f_1(n)]$

Q $f_1 = \log_2(n)$ $\quad f_3 = \sqrt{\log_2(n)}$

$f_2 = \lceil \log(\log_2(n)) \rceil$ $\quad f_4 = 3n+2$

$f_5 = 100n \log(n)$

$f_6 = n \log n / 100$

$f_7 = n^2$

$f_8 = 2^n$

Let $\log n = x$ $\quad \log n$ $\quad x$ $\quad \sqrt{x}$

$\sqrt{x} < x$

$\Rightarrow \log n > \sqrt{\log_2(n)}$

$\log x$ $\quad \log \log(n) < \log(n)$

$\log(x)$ $\quad \sqrt{x}$

$\sqrt{x} > \log(x)$

$\Rightarrow \sqrt{\log n} > \log \lceil \log n \rceil$

$\Rightarrow f_2 < f_3 < f_1$

ascending order

$\log \lceil \log_2(n) \rceil$, $\sqrt{\log_2(n)}$, $\log_2(n)$, $3n+2$, $100n \log_2(n) = n \log_2(n)/100$, n^2, 2^n

Q $f_1 = 2^n$ $f_2 = 4^n \Rightarrow [(2)^2]^n$

$\Rightarrow 2^n < e^7 < 4^n < 5^n < 6^n$

Q $f_1 = 2^n \Rightarrow 2 \times \cdots n$ times
$f_2 = n^n \Rightarrow n \times \cdots n$ times
$\Rightarrow n^n > 2^n$ or $2^n < n^n$

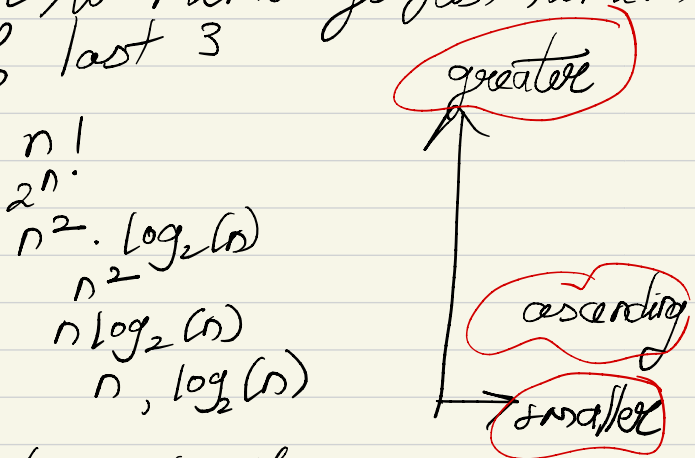
Q Arrange in increasing order of growth

smaller to greater

$\log_2 \lceil \log_2(n) \rceil$, $\log_2(n)$, $\lceil \log_2(n) \rceil$, $\lceil \log_2(n) \rceil^3$, $n^{1/10}$, n, n^2, n^3, 2^n, e^n, n^n

- But in Programming Questions input size limits will be given hence what to choose when

No need to memorize just Remember the flow of last 3



Now n here is the maximum size of the input

We need to choose such a function such that after replacing n the final value should be less than 10^8

eg $n = 10^3$

then we get $n^2 \Rightarrow (10^2)^3 = 10^6$ ↳ acceptable

$n = 10^5$

then we get $n^2 \Rightarrow (10^5)^2 = 10^{10}$ this not acceptable

hence go down

$\log_2[10] = 3.32$

$\log_2[100] \Rightarrow \log_2[10]^2 = 2 \times \log_2[10]$

$= 2 \times 3.32 = 6.6$

Q. $f_1 = n^{\log_2(n)}$ $f_2 = (\log_2 n)^n$

n	f1	f2
2	2	1
4	$16 = 2^4$	16 2^4
8	$512 = 2^9$	38 6400
16	16^4	4^{16}
32	$(32)^5$	$(5)^{32}$
64	$(64)^6$	$(6)^{64}$

$\Rightarrow f_1(n) = O[f_2(n)]$

Analyzing Algorithms

Types of algorithms

- Iterative loop
- Recursive Algorithm
- Constant Time algorithms $O(1)$

Loop analysis

We will be using for loop

for loop syntax

```
for (exp1 ; exp2 ; exp3)
{
    code
}
```

(1), (2), (4) → Optional

(3) code

Simple Maths

(1) Arithmetic Progression

$1+2+3\cdots n = \sum_{i=1}^{n} i = n(n+1)/2$

$1^2+2^2+\cdots n^2 = \sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$

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

2, 4, 6, 8, 10 ···· $d=2$ for all term ⇒ constant

⇒ $a,\ a+2d \cdots a+nd$

$d = T_2 - T_1$

$T_n = a+(n-1)d$ ⇒ n^{th} term a → first term

Sum of an AP

$S_n = \frac{n}{2}[\text{first term} + \text{last term}]$

$= n/2[2a+(n-1)\cdot d]$

$d = T_2 - T_1$

a → first term

(2) Geometric Progression

Ratio is same

n^{th} in GP ⇒ ar^{n-1}

$S_n = \frac{a(r^n-1)}{r-1}$ $r>1$ ⇒ $r = T_2/T_1$

or $S_n = a(1-r^n)/1-r$ $r<1$ $S_\infty = \frac{a}{1-r}$

a = first term

For loop analysis

for loop works for values of i

```
for(i=1; i<=n; i++)
{
    printf("Pankaj");
}
```

Will work n times

hence O(n)

1, 2, 3 ... n is n times

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

For each time outer loop work inner loop is working n time

$\Rightarrow O(3n)$

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

Be careful while analysing for nested loop

Nested loop $\longrightarrow O(n^2)$

Simple loop $\longrightarrow O(n)$

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

$i*3$

$\log_3(n)$

Value of i

$1, 2, 2^1, \ldots\ldots 2^k$ → GP

$1, 2, 4, 8 \ldots$

semi/avg

$2^k <= n$

$k <= \log_2(n)$

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

$n = 4$ → $8, 4, 2, 1$

last value when loop ends

$n, n/2, n/2^2, \ldots \frac{n}{2^k}$

$4, \frac{4}{2}=2, 1$

$4, 2, 1$

$2^2, 2^1, 2^0$

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

or $n >= 2^k$

$\boxed{\log_2(n) = k}$

Q.

```
for(i=1; i<=n; i++)
{
    for(j=1; j<=i; j++)
    {
        code
    }
}
```

when $i = n$

j will run from $j = 1$ to n

$\Rightarrow 1+2+\ldots$? times

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

```
for(i=1; i<=n; i=i*2)
{
  for(j=1; j<=i; j++)
  {
    for(k=1; k<=10; k++)
    {
      constant
      loop
    }
  }
}
```

$\log_2(n)$

loop unfolding $\Longrightarrow$ $n^2 \cdot \log_2(n) \cdot 10$

10 is constant

$n^2 \cdot \log_2(n)$

When

$i=1$	$i=2$	$i=n$
$j=1 \rightarrow$ 1 time	$j=1,2$	$1+2+3 \cdots +n$
	2 time	$n(n+1)/2 \Rightarrow n^2$

this is wrong because i is changing as 2K

$i = i*2$

$i=1 \rightarrow i=2 \rightarrow i=4$

so

$i=1$	$i=2$	$i=4$
$j=1$	$j=4$	
10	10 + 10	10

Q. for(i=1; i<=n; i++)
for(j=1; j=i*i; j++)
for(k=1; k<=n; k++)

i=1
j=1
k → n

i=2
j=1, 2, 3, 4
n n n n

i=3
j=1 to 3^2
for each n
9n

When $i = n$
$j = 1$ to n^2
for each n times
n^3

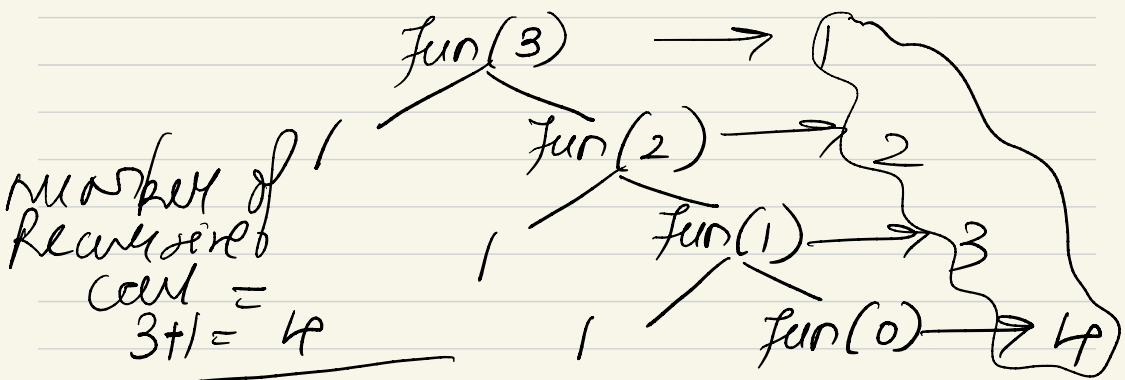
total
$n + 4n \ldots n^3$
$O(n^4)$

- Always analyze algorithms for worst case

Recursive Algorithm

eg
```
void fun(int n){
    if(n>0)
    {
        printf("pankaj");
        fun(n-1);
    }
}
```

Work done ← printf("pankaj")
Recursive call ← fun(n-1)



Q. What do we count in Recursion?

→ We count the function calls in Recursion

Asymptotic notation says Running Time of an algorithm with input T(n)

→ $T(n) = 1 + T(n-1)$

```
fun(int n)
{
  if(n>0)
  {
    printf("fun");
    fun(n/2);
  }
}
```

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

```
fun(int n)
{
  n>0
  for(i=1; i<n; i=i*2)
  {
    printf("T");
  }
  fun(n-1)
}
```

$T(n) = \log_2(n) + fun(n-1)$

```
Q  void fun(int n)
{
  int i;
  if(n>0)
  {
    for(i=1; i<=n; i++)
      printf("Pankaj");
    fun(n-1);
  }
}
```

n ← for(i=1; i<=n; i++)

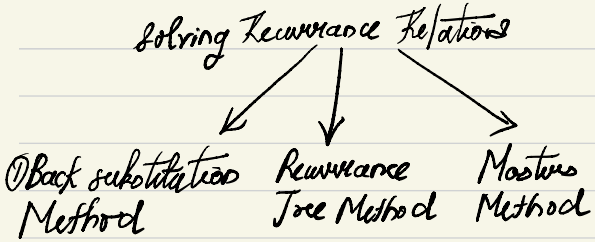
$n \cdot C$ ← printf("Pankaj")

$T(n-1)$ ← fun(n-1);

$\Rightarrow T(n) = n + f(n-1)$ when $n > 0$

$= 1$ when $n = 0$

Solving Recurrence Relations



① Back substitution Method

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

Put $n = n-1$

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

Put $n = 1$

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

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

$$\vdots$$

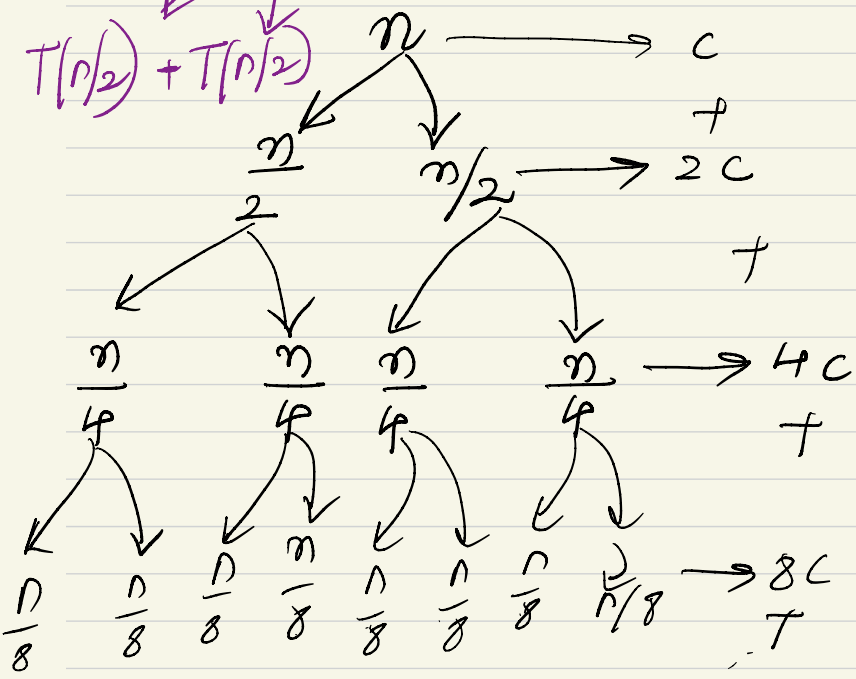
Put $n - k = 1$

$$\boxed{k = n-1}$$

Reaverance Tree Method

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

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



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

$$S = c + 2^2 c + 2c + \cdots 2^k c$$
$$= c\left[1 + 2^2 \cdots 2^k\right]$$
$$= c(2 \cdot 2^k - 1)$$
$$\Rightarrow O(n)$$

But $k = n$

2