DSA Course By GeeksForGeeks.

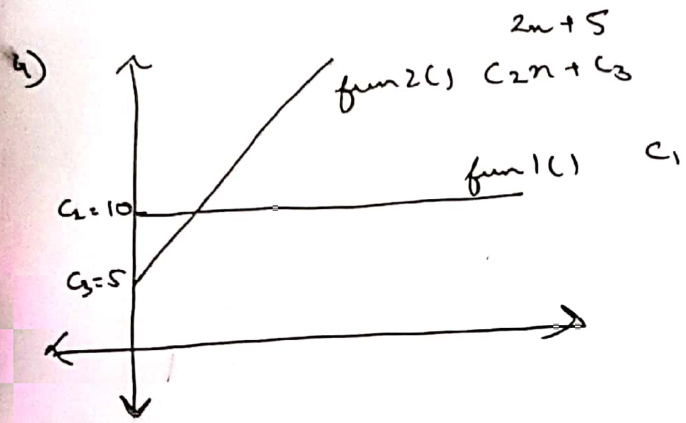
Lecture 1: Analysis of Algo.

1) def sum_num(num):
       return num*(num+1)/2

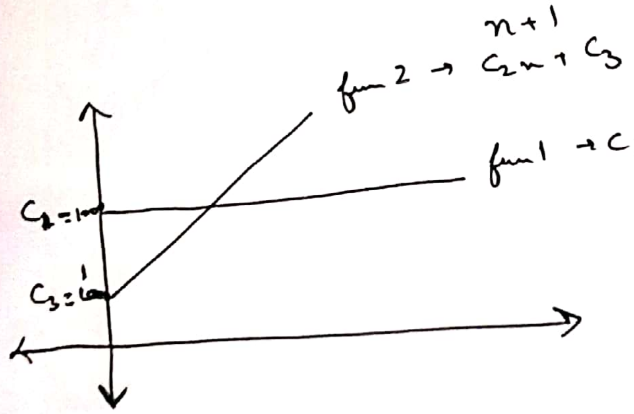
2) Asymptotic analysis → ~~Depe~~ Study of time complexity.

3) Constant : C_1
linear : $C_1 n + C_2$
quardratic : where ~~coff~~ co-eff ~~of~~ of $C_1 > 1$.

4)



$2n+5 \geq 10$
$n \geq 2.5$ $\boxed{n \geq 3}$.



$n+1 \geq 1000$
$\boxed{n \geq 999}$.

The linear graph will always have more value at one point when compared to const.

Lecture 2: Order of Growth.

1) $f(n)$ is growing faster if →

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

However when it will intersect depends on language and system.

Assumption $f(n), g(n), n \geq 0$.

OR

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

2) $f(n) = 2n^2 + n + 6$

$g(n) = 2n + 5$

$$\lim_{n\to\infty} \frac{2 + 1/n + 6/n^2}{2/n + 5/n^2} = \frac{2}{0} = \infty$$

or

$$\lim_{n\to\infty} \frac{(2n+5)/n^2}{(2n^2+n+6)/n^2} = \frac{2/n + 5/n^2}{2 + 1/n + 6/n^2} = \frac{0}{2} = 0.$$

3) $c < \log\log n < \log n < n^{1/3} < n^{1/2} < n < n^2 < n^3 < n^4 < 2^n < n^n$

which is better → i) $2n^2 + n + 6$ Order of growth = n^2

$100n + 3$ OG: n.

OG: $\log n$ → This is better.

ii) $c_1 \log n + c_2$

$c_3 n + c_4 \log\log n + c_5$ OG: n.

$\frac{n^2}{n \log n}$ → $\log n$ better

iii) $c_1 n^2 + c_2 n + c_4$ OG: n^2

$c_5 n \log n + c_6 n + c_4$ OG: $n \log n$ → better

Lecture 3: Asymptotic Analysis.

Best, Avg & worst case.

i)
```
int getsum(int arr[], int n)
{
    int sum = 0;

    if (n%2 != 0)
        return 0;
    for(int i = 0; i<n; i++)
        sum = sum + i;
    return sum;
}
```

Best → C_1

Average → $\frac{Cn+C}{2} = \frac{n}{2}$ = linear (If they are equally likely)

Worst : $Cn+C_2$

ii) Best Case → Bogus

Average + Worst case → Considered for product.

iii) Big O : Exact or upper bound.

Theta : Exact bound

Omega : Exact or lower bound.

→ Mathematical Tool to rep → order of growth

Lecture 4: Big O Notation.

1) $f(n) = 3n^2+2n+100 = O(n^2)$

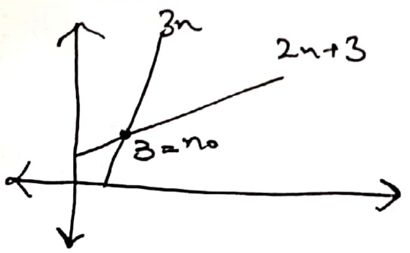
$f(n) = 4n + \log n + 30 = O(n)$.

$f(n) = O(g(n))$ iff exist constant c and n_0 such that $f(n) \le c\,g(n)$ for all $n \ge n_0$.

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

$f(n) \le c\,g(n) \rightarrow 2n+3 \le cn$

$\boxed{c=3}$ $(2+1)$ $\quad 2n+3 \le 3n \quad 3 \le n \rightarrow \boxed{n_0 \ge 3}$.



$\{n/4,\ 2n+3,\ n/100 + \log n,\ \underset{c_1}{100}\} \in O(n)$

$\{n^2+n,\ 2n^2,\ \log n,\ 2 \ldots\} \in O(n^2)$.

$\{1000,\ 2,\ 1,\ \log \ldots\} \in O(1)$.

) for (int i=0; i<n; i++) .
if arr[i] == x
return i ;
return -1;
}

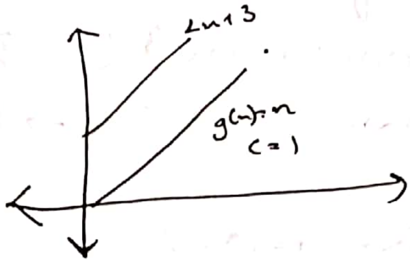
O(n). → Worst case.

Lecture 5: Omega notation : Best case.

(1) $f(n) = \Omega(g(n))$ iff there exist +ve const c and n_0 st
$0 \le cg(n) \le f(n)$ for all $n_0 \ge n_0$

Example.
$f(n) = 2n+3 = \Omega(n)$. | $f(n) = 2n^2+3n+6 = \Omega(n^2)$.

$c = 1$
$cg(n) = n$
$n \le 2n+3$
$\boxed{n_0 = 0}$.



$n^3, n^n \in \Omega(n)$.
2) $\{n/4, 2n+3, n^2,$

3) $\boxed{\text{If } f(n) = \Omega(g(n)) \text{ then } g(n) = O(f(n))}$

4) Used when there is no upper bound, which runs infinitely, like a game. So we use Ω here.

Lecture 6 Theta Notation : ~~Average~~ Case (Exact bound).

① $f(n) = \Theta(g(n))$ iff exist +ve const c_1, c_2, n_0 st
$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$
for all $n \ge n_0$.

Eg → $f(n) = 2n+3$ order of growth.
$= \Theta(n)$

$c_1 = 1$
$(2-1)$

$c_2 = 3$ $(2+1)$

$n \le 2n+3 \le 3n$
$n_0 \ge 0$ $n_0 \ge 3$

or $n_0 \ge 3$

② If $f(n) = \Theta(g(n))$

Then $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

and $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$

③ Application → When we need to traverse the whole array then we consider Θ notation.

→ Worst case of Quick Sort → $\Theta(n^2)$.

Best → $\Theta(n\log n)$.

Average → $\Theta(n \log n)$

④ $\{n^2/4, n^2/2, 2n^2 - - - 4n^2 + 2\log n + 6\} \in \Theta(n^2)$.

Lecture : Analysis of Common Loops.

floor $\lfloor n/c \rfloor$

1) for (int i = 0; i<n; i=i+c} { } → $\Theta(n/c) = \Theta(n)$

2) for (int i = 0; i<n; i=i-c) { } → $\Theta(n/c) = \Theta(n)$.

$\lceil n/c \rceil$ ← ceil.

3) for (int i = 1; i<n; i=i*c)

1, 2, 4,

$c^{k-1} < n$

$k-1 < \log_c n$

$k < \log_c n + 1$ or $\Theta(\log n)$.

→ Doesn't matter because we can divide by const and get value.

4) for (int i = 1; i<n; i= i/c)

[Same as 3]

5) for (int i = 2; i<n; i = pow(i,c))

$2^{c^{k-1}} < n$

⇒ $c^{k-1} < \log_2 n$

⇒ $k-1 < \log_c \log_2 n$

⇒ $k < \log_c \log_2 n + 1$.

⇒ $\boxed{k < \log \log n}$.

→ $\Theta(\log \log n)$.

⑥ ~~void~~ for (i=0; i<n; i++)] $\Theta(n)$

for (i=1; i<n; i*=2)] $\Theta(\log n)$.

for (i=1; i<100; i=i+1)] $\Theta(1)$

$\Theta(n) + \Theta(\log n) + \Theta(1) = \Theta(n)$.

Scanned with CamScanner

② for(int i=0; i<n; i++) $\Theta(n)$
for(int j=1; j<n; j*=2) $\Theta(\log n)$ $> \Theta(n\log n)$.

for(i=0; i<n; i++) $\Theta(n)$
for(j=0; j<n; j++) $\Theta(n)$ $> \Theta(n^2)$

$$\Theta(n^2) + \Theta(n\log n) = \Theta(n^2).$$

Lecture: Analysis of Recursion.

```
void fun(int n)
{  if (n <= 1) return
   for(i=0; i<n; i++) print("GFG");
   fun(n/2)
   fun(n/2)
}
```

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

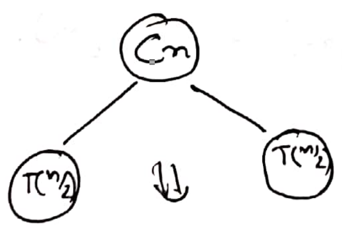
$$T(1) = C_1$$

Recursive ~~Methods~~. Tree Method

→ We ~~work~~ write non-recursive part as root and recursive as children.

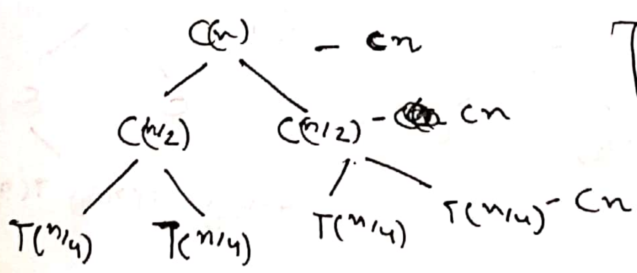
→ We keep expanding till we see a pattern.

①



$$\rightarrow \underbrace{Cn + Cn + Cn - - - Cn}_{\log_2 n}$$

②



height
→ $\log_2 n$

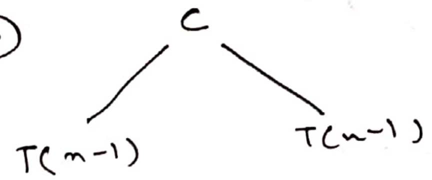
$Cn \times \log_2 n$

$O(n\log n)$
↓
Complexity.

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

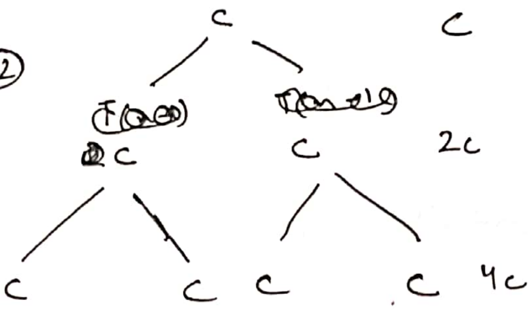
$T(1) = c.$

①



⇓

②



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

height = n

$c + 2c + 4c - - -$

$c(1+2+4 - - -)$

used for ending term.

$c(2^n - 1)$

$= \Theta(2^n)$

$\Theta(2^n)$

$$\underbrace{c + 2c + 4c - - - -}_{n}$$

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

$T(1) = c$

① c | $T(n/2)$ ⇒ c | $c - T(n/4)$

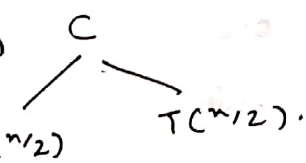
$$\underbrace{c + c + c + - - - -}_{\log_2 n}$$

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

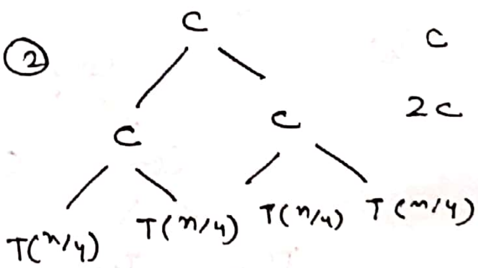
$T(n) = 2T(n/2) + c$

~~T(n) = 2~~ $T(1) = c$

①



②



$\cancel{c+2c+3c}$ $\underbrace{c + 2c + 4c - - - -}$

$c(2^{\log_2 n} - 1)$ $= 2^{\log_2 n}$ $\Theta(2^{\log_2 n}) = \Theta(n).$

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

$T(1) = Cn$

Cn

$T(n/2) \quad T(n/4)$

$\Rightarrow$

$Cn \qquad Cn$

$Cn/2 \qquad C(n/4) \qquad \frac{3Cn}{4}$

$T(n/4) \quad T(n/8) \quad T(n/8) \quad T(n/16)$

$Cn \qquad Cn$

$Cn/2 \qquad Cn/4 \qquad 3Cn/4$

$Cn/4 \quad Cn/8 \quad Cn/8 \quad Cn/16 \rightarrow 9Cn/16$.

$C\,n/2 + n/16$

$9Cn/16$

height $\rightarrow \log n$.

Here notation will be $\underline{\underline{O(n)}}$

$r = 3/4$ $\quad \frac{a}{1-r} = \frac{Cn}{(1-3/4)} = O(n)$.

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

$T(1) = C$.

C

$T(n-1) \qquad T(n-2)$.

$\Rightarrow$

$C \qquad C$

$C(n-1) \qquad C(n-2) \qquad 2C$

$T(n-2) \quad T(n-3) \quad T(n-3) \quad T(n-4)$

$\underbrace{C + 2C + 4C \; - - -}_{n}$

$\underline{C(1+2+4 - - -)} = \underline{C(2^n - 1)}$

$O(2^n)$

Lecture: Space Complexity.

```
int getsum(int n)
  return n*(n+1)/2
```

$\Theta(1)$ or $O(1)$

1 vars.

```
getsum2(int n)
  int sum = 0;
  for(i=0; i<=n; i++)
      sum = sum+i;
  return sum;
```

$O(1)$ or $\Theta(1)$.

3 vars

```
int arrsum(int arr[], n)
{ for(i=0; i<n; i++)
      sum += arr[i]
}
```

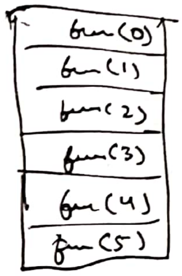
$\Theta(n)$

Auxillary Space : Order of growth of extra space or temp space in terms of i/p size.

Aux Space = $\Theta(1)$.
Space comp = $\Theta(n)$.

Aux Space → Came into existence cuz we needed to compare arr space wrt sort.

```
int fun(int n)
{  if (n<=0)
       return 0;
   return n + fun(n-1);
}
```

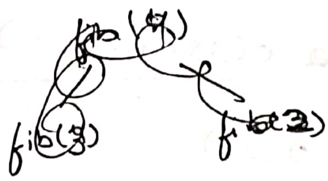


↳ Aux Space : n+1

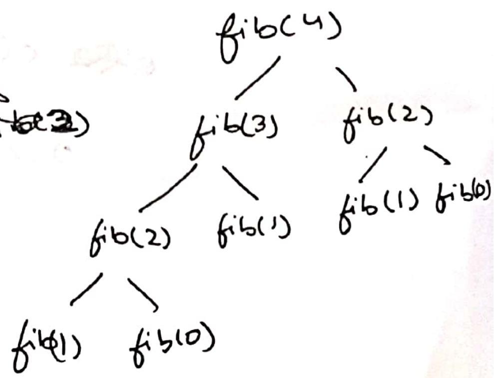
```
int fib(int n)
{  if(n==0 || n==1)
       return n
   return fib(n-1) + fib(n-2).
}
```

```
int fib(int n)
{ int f[n+1];
  f[0] = 0
  f[1] = 1
  for(i=2; i<=n; i++)
      f[i] = f[i-1] + f[i-2]
  return f[n];
}
```

Space & Aux → $O(n)$.



Aux → $\Theta(n)$



```
fib(int n)
{
  if(n==0 || n==1)
     return n;
  int a=0, b=1
  for(i=2; i<=n; i++)
   { c=a+b; a=b; b=c; }
  return c; }
```

Auxillary space → $\Theta(d)$
Space Comp → $O(1)$.

II Mathematics Module 2

Number of digits →

i) Iterative

```
int c(long n) {
   int count=0;
   while(n!=0) {
      n= n/10;
      count++;
   }
   return count;
}
```

ii) Recursive

```
int co(long n) {
   if(n==0)
      return 0;
   return 1+ countDigit(n/10);
}
```

iv)

```
int count(long n) {
   return floor(log10(n) + 1); }
```

Median →

6 Prime num → $6n \pm 1$ where n is natural num