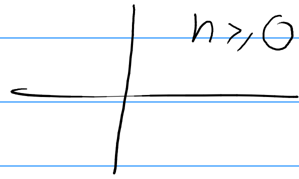
DSA

Order of growth

A function $f(n)$ is said to be growing faster than $g(n)$ if

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



OR

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

Directway:
1. Ignore lower order terms
2. Ignore leading constant

Remember:

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

Check for best, Average and worst case

eg:

```
int getSum(int arr[], int n)
{
    if (n%2 != 0) return 0;
    for(int i=0; i<n; i++)
        Sum = Sum + i;
    return sum;
}
```

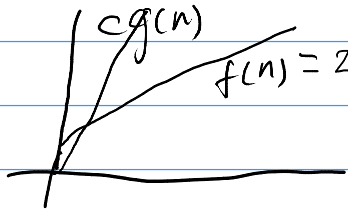
worst case: n
Best case: 1

Asymptotic Notations

Big O: Represents exact bound or upper Bound
Theta : Represent exact bound.
Omega : Represent exact or lower bound.

Big O Notation
(Upper bound or Order of growth)

We say $f(n) = O(g(n))$ if there exist constants c and n_0 such that $f(n) \le g(n)$ for all $n \ge n_0$



Example

$f(n) = 2n+3$

$2n+3 \le 3n$ for $n \ge 3$

$\{\frac{n}{4}, 2n+3, \frac{n}{100} + \log n, n+10000, n/10000, 100, \log n + 100\} \in O(n)$

$\{n^2+n, 2n^2+n^2+1004, n^2+2\log n\} \in O(n^2)$

$\{1000, 2, 3 \ldots\} \in O(1)$

Application

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

Time complexity : $O(n)$

Omega Notation

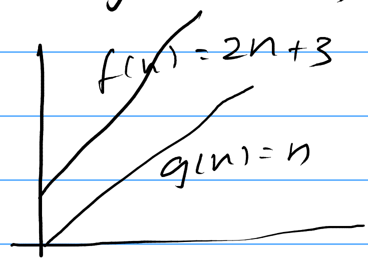
lower bound

$f(n) = \Omega(g(n))$ if there exist positive constant c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$

example:

$f(n) = 2n+3$

$= \Omega(n)$



① $\{n/4, n/2, 2n, 3n, 2n+3, n^2, 2n^2 \ldots n^n\} \in \Omega(n)$

② If $f(n) = \Omega(g(n))$
then $g(n) = O(f(n))$

③ Omega notation is useful when we have lower bound on time complexity

Theta Notation

$f(n) = \Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$

Example

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

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$

① 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 $g(n) = \Omega(f(n))$

② Theta is useful to represent time complexity when we know exact bound. For example time complexity to find sum, max and min in an array is $\Theta(n)$

③ $\{n^2, n^2/4, \ldots 2n^2 \ldots 4n^2 + 2n\log n \ldots\} \in \Theta(n^2)$

Analysis of common loops

① for(int i=0; i<n; i=i+c)

{ // Some Θ(1) work

}

$i=0$, $i=c$, $i=2c$

$\Rightarrow i_k = kc$ $\qquad kc<n \Rightarrow \boxed{k<\frac{n}{c}}$

No of iterations $= n/c$

Time complexity : Θ(n)

②

for(int i=n; i>0; i=i-c)

{

//some Θ(1) work)

}

Timecomplexity = Θ(n)

③

for (int i=1; i<n; i=i×c)

//some Θ(1) work;

$c^{k-1} < n$

$\ln_c n = k-1$

$k < \ln_c n + 1$

④ for (int i=n; i>0; i=i/c)

$n \quad \frac{n}{c} \quad \frac{n}{c^2} \ldots$

$\Rightarrow \ln_c n \geq k-1$ $\qquad \frac{n}{c^{k-1}} \geq 1$

$k \leq \log_c n + 1$ $\qquad n \geq c^{k-1}$

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

$2 \quad 2^c \quad (2^c)^c \quad ((2^c)^c)^c$

$2^{c^k} < n \qquad c=2 \quad (2^2)^2$

$2 \quad 2^2 \quad 2^{4^2} \quad 2^{8^3}$

$c^k \; k \log_2 n$

$k < \log(\log_2 n)$

(6) Void func(int n)
```
{ for(int i=0 ; i<n ; i++)
        // some Θ(1) work
```
$\Theta(n)$

```
  for(int i=1; i<n ; i=i*2)
        // some Θ(1) work
```
$\Theta(\log n)$

```
  for(int i=0 ; i<100; i++)
        // some Θ(1) work
}
```
$\Theta(1)$

(7) for(int i=0 ; i<n ; i++)
```
        for(int j=1; j<n; j=j*2)
            // same Θ(1) work
```

$\log n + \log n + \cdots$ n time

$\Theta(N \log N)$

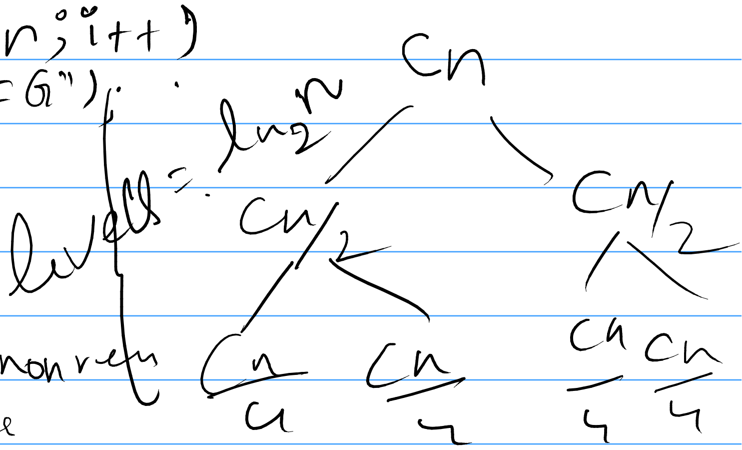
Analysis of Recursion

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

recursive non rec

$$T(n) = \underbrace{2T(n/2)}_{\text{recursive}} + \underbrace{cn}_{\text{non rec}}$$

$$T(1) = c$$



work at each level = CN

Time complexity $(N \log N)$

Recussion Tree method

→ we work non recursive part as root of tree and recursive part as children

→ we keep expanding childer until we see a pattern

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

$$T(1) = C$$

$$\Rightarrow C + 2C + 4C + \ldots$$

$$\frac{a(r^n - 1)}{r-1} \quad \frac{a\, 2^n - 1}{1}$$

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

→ Recursion tree method doesn't give exact bound always.

Space Complexity

Order of growth of memory (OR RAM) space in terms of input size

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

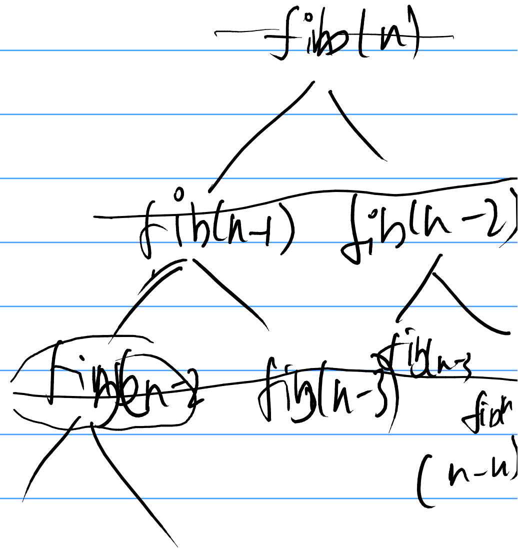
Auxilary space : order of growth of extra space or temporary space in terms of input size

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

Auxilary space: $\Theta(1)$

space complexity $\Theta(n)$

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



Auxilary space : height of tree

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

$\theta(n)$