10/1/23

1. Introduction

Performance is based on
1. Time complexity
2. Space complexity

Time complexity
↳ Empirical or Posterion
↳ Theoretical or Apriori

Theoretical or Apriori
- Frequency count

Program Statement	Frequency count
1.	
$x = x + 2$	1
Total frequency count	1

Time complexity is $O(1)$.

2.

for K = 1 to n do	(n+1)
$x = x + 2$;	n
end	n
Total frequency count	3n+1

Time complexity is $O(n)$.

3.

```
for j = 1 to n do                    (n+1)
    for k = 1 to n do                (n+1)n
        x = x+2;                     n^2
    end                              n^2
end                                  n
```

Total frequency Count $3n^2 + 3n + 1$

Time Complexity is $O(n^2)$.

* Time complexity is $O(\log n)$

Example:

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

2) for(int i = n; i > 0; i /= c)

3) Binary Search.

* Time complexity is $O(\log \log n)$

Example:

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

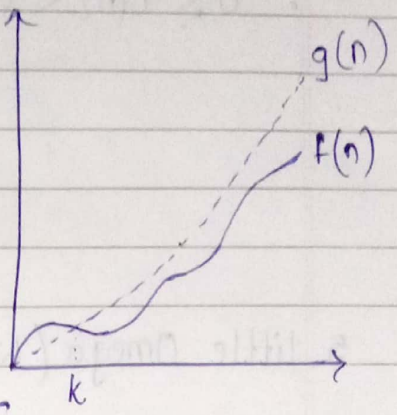
2) for(int i = n; i > 0; i = fun(i))

Asymptotic Notation

Asymptotic notations are to calculate running time complexity of an algorithm.

1. Big O Notation (least upper bound)
   - worst case scenario
   - largest value of time

Big Oh (O): $f(n) = O(g(n))$, if there exists a positive integer n_0 and a positive number c such that $|f(n)| \leq c * |g(n)|$



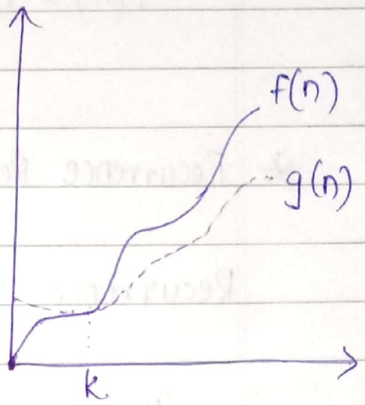
$f(n) \leq c * g(n)$

2. Big Omega (Ω) Notation (greatest lower bound)
   - best case scenario
   - lower bound of algorithm's running time.

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

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

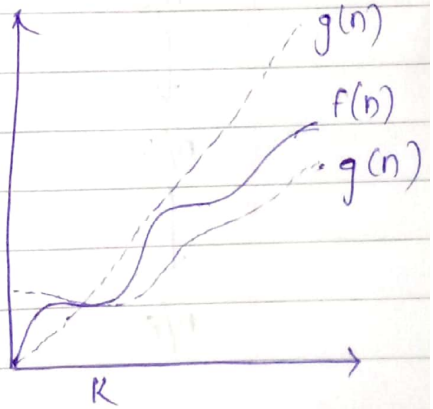
g(n) is the lower bound.



$16n^3 + 12n \rightarrow \Omega(n^3)$

3. Theta (Θ) Notation
   - average case
   - upper bound & lower bound

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

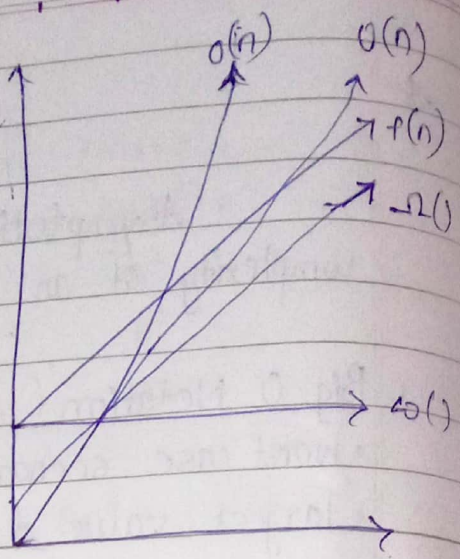


$2n^2 \leq 2n^2+2 \leq 3n^2$

Page No. Masters
Date

4. Little o (o) Loose Upper bound

- Big-O is used as a tight upper-bound.
- rough estimation of max. time
- $0 \leq f(n) < c * g(n)$



5. Little Omega (ω) loose lower bound

- rough estimation of min. time
- $f(n) > c * g(n)$

* Recurrence Relation

Recurrence relation of binary search

n
↓
n/2
↓
n/4
↓
n/8

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

where c is constant time taken while calculating(mid == x) or for comparisons.

- Substitution Method
- Recursive Method
- Master's method

1. Substitution Method

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

$T(n) = T(n/2) + c$ — (1)

$T(n/2) = T(n/4) + c$ — (2)

$T(n/4) = T(n/8) + c$ — (3)

Substituting (2) in (1)

$T(n) = T(n/4) + c + c$

$T(n) = T(n/4) + 2c$ — (4)

Substituting (3) in (4)

$= T(n/8) + c + 2c$

$= T(n/8) + 3c$

$= T\left(n/2^3\right) + 3c$

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

Assume $T\left(\frac{n}{2^k}\right) = 1$ & $n = 2^k$

$T\left(\frac{n}{n}\right) \neq k.c$

$T(1) + k.c$

$1 + k.c$

$n = 2^k$

$\log n = k \log_2 2$

$k = \log n$

$1 + \log n \cdot c$

$O(\log n)$

2. $T(n) = \begin{cases} 1 & n = 1 \\ n * T(n-1) & n > 1 \end{cases}$

$T(n) = n * T(n-1)$ — (1)

$T(n-1) = (n-1) * T(n-2)$ — (2)

$T(n-2) = (n-2) * T(n-3)$ — (3)

Substitute (2) in (1)

$T(n) = n * (n-1) * T(n-2)$ — (4)

Substitute (3) in (4)

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

$\vdots$

$T(n) = n * (n-1) * (n-2) \ldots T(n-(n-1))$

$T(n) = n * (n-1) * (n-2) \ldots T(1)$

$T(n) = n * (n-1) * (n-2) \ldots * 3 * 2 * 1$

Taking n out from each term.

$T(n) = n * n\left(1 - \frac{1}{n}\right) * n\left(1 - \frac{2}{n}\right) \ldots n\left(\frac{3}{n}\right) * n\left(\frac{2}{n}\right) *$

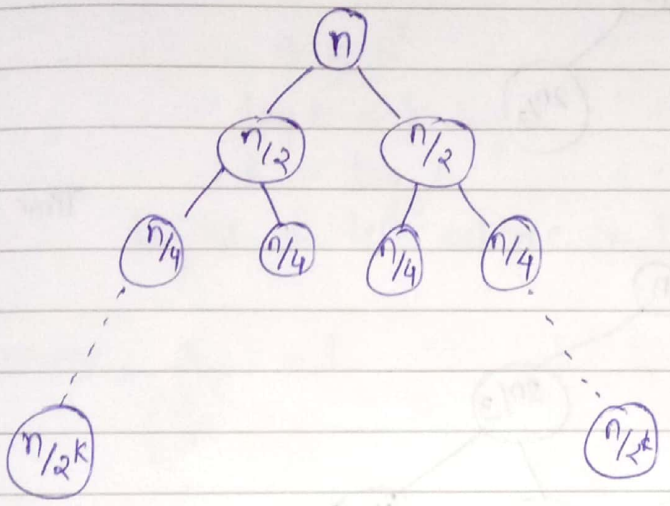
$$\Downarrow$$

$$T(n) = n^n$$

Time Complexity $= O(n^n)$

* Recursive Tree Method

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



Time req.

n

n

n

n.

Total no. of steps $= k$

Time req. to complete execution for each step $= n$

Time req. to solve given eqn. $= n * k$

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

$$n = 2^k$$

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

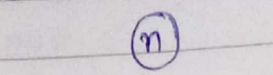
$$k = \log n$$

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

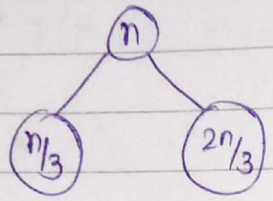
Time complexity is represented by n.

2. $T(n) = T(n/3) + T(2n/3) + n$ → no. of input.

Step 1 : Other than recursive terms we have term n
So n is the root of the tree

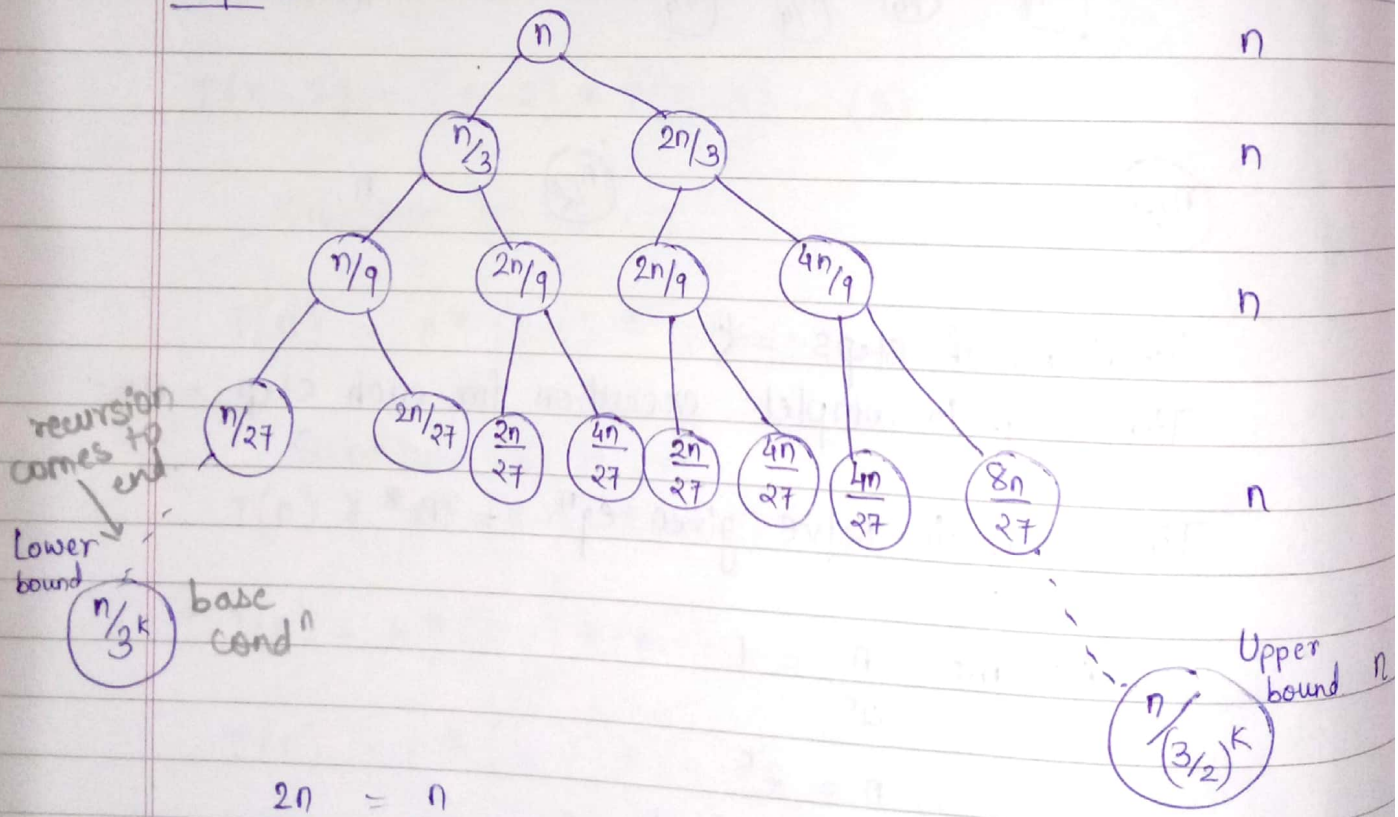


Step 2 : Two recursive calls $n/3$ & $2n/3$



Step 3 :

Time req. to comp



$$\frac{2n}{3} = \frac{n}{(3/2)}$$

$$\frac{n}{(3/2)^0} = 0$$

$$(3/2)^0 = 1$$

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

$$\frac{n}{(3/2)^2} = \frac{n}{(9/4)} = \frac{4n}{9} \Longrightarrow \boxed{\frac{n}{(3/2)^k}}$$ upper bound

Let's find height of tree i.e. K.

Assumption, $\frac{n}{3^k} = 1$

$$n = 3^k$$

$$\log n = k \log 3$$

$$K = \log_3 n$$

height of left subtree $= \log_3 n$

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

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

$$\log n = K \log\left(\frac{3}{2}\right)$$

$$K = \log_{\frac{3}{2}} n$$

height of right subtree $= \log_{\frac{3}{2}} n$

To solve the given recurrence relation time req. is $n * k$.

For worst case scenario time req. is

$$T(n) = O\left(n * \log_{\frac{3}{2}} n\right)$$

For best case time req. is

$$T(n) = \Omega(n * \log_3 n)$$

* Master's Method.

Equation has to be in

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$

$b > 1$

$f(n) \rightarrow$ (+ve) function

e.g. 1. $T(n) = T(n-1) + 1$

$a = 1$, $b = 1$, $f(n) = 1$

✓ ×

2. $T(n) = 8\,T(n/2) + n^2$

$a = 8$, $b = 2$, $f(n) = n^2$

✓ ✓ ✓

Can be solved using master's method.

* There are 3 cases for master's method.

Case 1:

There are 3 cases for master's method

Case 1 : If $T(n) = O(n^{\log_b a - \epsilon})$

for constant $\epsilon > 0$ then
$T(n) = \theta(n^{\log_b a})$

For Big O notation the fun^n we have is
$f(n) \leq c.g(n)$
we need to show $T(n) = \theta(n^{\log_b a})$
$n^1 \leq c.n^2$
$n^1 \not\geq n^{2-1}$

Case 2 : If $T(n) = \Omega(n^{\log_b a + \epsilon})$

for constant $\epsilon > 0$ then
$T(n) = \theta(f(n))$

for Ω notation (lower bound / best case)
The function is $f(n) \geq c.g(n)$
but we need to show
$T(n) = \theta(f(n))$
$f(n) \geq c.g(n)$
$n^3 \geq c.n$
$n^3 = n^{1+2}$

Thus the given example can be solved using case III

case III : If $T(n) = \theta(n^{\log_b a})$

then $T(n) = \theta(n^{\log_b a} * \log n)$

* Problems :

* Case 1.

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

$a = 8$, $b = 2$, $f(n) = n^2$

Find $g(n)$

$n^{\log_b a} = n^{\log_2 8} = n^3$

$f(n) = n^2$ & $g(n) = n^3$

As per case 1 $f(n) \leq c.g(n)$

$n^2 \leq c.n^3 \Longrightarrow T(n) = \Theta(n^{\log_b a})$

$T(n) = \Theta(n^3)$

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

$a = 9$, $b = 3$, $f(n) = n$

Find $g(n)$

$n^{\log_b a} = n^{\log_3 9} = n^2$

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

As per case 1 $f(n) \leq c.g(n)$

$n \leq c.n^2 \Longrightarrow T(n) = \Theta(n^{\log_b a})$

$= \Theta(n^2)$

Case 2

3. $T(n) = 64\,T\left(\frac{n}{2}\right) + n^7$

$T(n) = a.\,T\left(\frac{n}{b}\right) + f(n)$

$\therefore a = 64 \qquad a \geq 0$

$b = 2 \qquad b > 0$

$f(n) = n^7 \qquad f(n)$ is +ve function

can be solved using master's method

$g(n) = n^{\log_b a} = n^{\log_2 64}$

$g(n) = n^{\log_2 2^6} = n^6$

$f(n) = n^7$

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

$n^7 \geq c.g(n)$

$n^7 \geq c.n^6$

$n^7 = n^{6+1}$

If $T(n) = \Omega\left(n^{\log_b a + \epsilon}\right) \qquad \epsilon > 0$

then $T(n) = \Theta(f(n))$

$T(n) = \Theta(n^7)$

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

To get the given recurrence relation in the form req. for master's method we make two assumptions.

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

$n = 2^k$

$\log n = k \log_2 2$

$k = \log n$

The given relation is

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

Replace n with 2^k

$T(2^k) = 2T(\sqrt{2^k}) + c$

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

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

Still the eqn is not in the required form for solving

∴ Assume $T(2^k) = S(k)$

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

$S(k) = 2.S(\frac{k}{2}) + c$

$a = 2$

$b = 2$

$f(n) = c$

$g(n) = n^{\log_b a}$

$= n^{\log_2 2}$

$= n$

$\therefore g(k) = k^{\log_b a}$

$= k$

case I is used to solve the given recurrence

$S(k) = k$

$T(2^k) = k$

$T(n) = \log n$

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

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

$T(n) = a.T(\frac{n}{b}) + f(n)$

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

$k = \log n$

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

n with 2^k

$T(2^k) = 2T(2^k)^{\frac{1}{2}} + \log 2^k$

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

$T(2^k) = S(k)$

$S(k) = 2 \cdot S\left(\frac{k}{2}\right) + \log_2 2^k$

$= 2 \cdot S\left(\frac{k}{2}\right) + k$

$2^k = S(k)$

$2^k = k$

$2^{\frac{k}{2}} = \frac{k}{2}$.

$a = 2, \; b = 2, \; f(n) = k$

$g(n) = n^{\log_b a}$

$g(k) = k^{\log_2 2}$

$= k$

as $f(k) = g(k)$

$f(n) = g(n)$

As both terms are equal

case III

↳ $T(n) = \Theta(n^{\log_b a} * \log n)$

$S(k) = \Theta(k \cdot \log k)$

$T(2^k) = \Theta(\log n \cdot \log \log n)$

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

6. Solve the following using substitution method.

$$T(n) = \begin{cases} 1 & \text{, if } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{, if } n > 1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \text{—(1)}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{cn}{2} \quad \text{—(2)}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{cn}{4} \quad \text{—(3)}$$

Sub (2) in (1)

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right] + cn$$

$$= 4T\left(\frac{n}{4}\right) + cn + cn$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2cn \quad \text{—(4)}$$

Sub (3) in (4)

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right] + 2cn$$

$$= 8T\left(\frac{n}{8}\right) + cn + 2cn$$

$$= 8T\left(\frac{n}{8}\right) + 3cn$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3cn \qquad \frac{n}{2^k}$$

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

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

$= 2^k T(1) + kCn$ $\quad n = 2^k$

$= 2^k \times 1 + kCn$

$= 2^k + kCn$

$= n + \log n \cdot n$

$= n(1 + \log n)$

Time complexity is

7. Give asymptotic upper bound for $T(n)$ for following recurrences using master's method.

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

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

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

$K = \log n$

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

Replace n with 2^k

$T(2^k) = T(2^k - 1) + 2^k$

$T(2^k) = S(k)$

$S(k) = T(k-1) + K$