What is an Algorithm?

An algorithm is a step by step instructions that accomplishes a particular task, specifically related to solving a computational problem

An Algorithm has following properties :

1. **Input** :- Zero or more entities supplied as input.
2. **Output** :- Atleast one entity as output.
3. **Definiteness** :- Each instruction is clear and unambiguous.
4. **Finiteness** :- algorithm terminates after a finite number of steps.
5. **Effectiveness** :-

Note:

- All the steps of algorithm should be written in a human understandable language which does not depend on any programming language.
- Any programming language can be used to implement algorithm.

Why Analysis of Algorithms ?

In computer science, multiple algorithms are available for solving the same problem (for example, a sorting problem has many algorithms, like insertion sort, selection sort, quick sort and many more).

Algorithms analysis helps us to determine which algorithm is most efficient in terms of time and space consumed.

Goal of the Analysis of Algorithm is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.)

What is Running Time Analysis?

- It is the process of determining how processing time increases as the size of the problem (input size) increases.
- Input size is the number of elements in the input, and depending on the problem type, the input may be of different types.

12 Jan 2021

How to Analyse Algorithms?

1. Execution Time
2. Number of statements Executed
3. Ideal solution? (Running Time Analysis)

- Express running time of a given algorithm as a function of Input Size i.e. $f(n)$.

Performance of an Algorithm can be analysed in two ways:

1. Time complexity
2. Space complexity

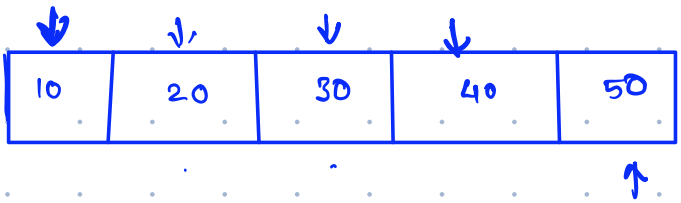
There are mainly 3 types of Time complexities :-

1.) Best case

2.) Worst case

3.) Average case

Example :- Liner / Sequential Search



Comparison operation

case 1 :- key Element = 40

Avg. case ✓

case 2 :- key Element = 10 ✓

Best case → 1 unit of time

case 3 :- Key Element = 50 ✓

Worst case → 5 units of time

Approaches to calculate Time / space complexity:

1. Frequency Count or Step Count
2. Asymptotic Notation

1. Frequency / step count

Rules :-

① For comments & declaration, count is 0.

② For return and assignment, count is 1.

③ Ignore lower order Exponents, when higher order Exponents are present.

④ Ignore constant multipliers

Example 1 :- Finding Sum of n elements of an Array.

```
Algorithm sum (int a[n], int n)
int s = 0                          ---> 1 unit
for ( i=0; i<n; i++ )              ---> n+1
  {
    s = s + a[i];                  ---> n
  }
return s;                          ---> 1
```

n = 3

i = 0, 1, 2, 3

$f(n) = 2n + 3$

Time complexity = $O(n)$

Space Complexity :- $2n + 1 + 2 + 2$

$= 2n + 5$

$O(n)$ $O(n)$

Example 2 :- (Addition of 2 matrices)

```
void main add (int a[][], int b[][])      // n x n , n x n
{
    int c[][];                            // n x n
    for (i=0; i<n; i++)                   ——— n+1
    {
        for (j=0; j<n; j++)               → n*(n+1)
        {
            c[i][j] = a[i][j] + b[i][j]   n*n
        }
    }
}
```

$n = 3$

$i=0$
$i=1$
$i=2$
$i=3$ ✗

$f(n) = n+1 + n^2+n + n^2$
$= 2n^2+2n+1$

TC $= O(n^2)$

Sp – $n^2+n^2+n^2+3 = 3n^2+3$
$\rightarrow O(n^2)$

Example 3 :- Multiplication of 2 matrices

```
void main  matmul (int a[][], int b[][])
{
  for (i=0; i<n; i++)          — n+1
  {
    for (j=0; j<n; j++)        — n*(n+1)
    {
      c[i][j] = 0;             → n x n
      for (k=0; k<n; k++)      — n x n x (n+1)
      {
        c[i][j] = a[i][k] * b[k][j]
                                 — n*n*n
      }
    }
  }
}
```

$n^3 + n^3 + n^2 + n^2 + n^2 + n + n + 1$

$= 2n^3 + 3n^2 + 2n + 1$

$= O(n^3)$

$$f(n) = 6n^3 + 10n^2 + 15n + 6$$

$$\Downarrow$$

$$6n^3$$

$$\Downarrow$$

Time complexity $\rightarrow O(n^3)$

order of n^3

13 Jan 2020

Examples:

```
for( i=0 ; i<n ; i=i+2)   — n
  {
    ———
  }
```

n/2

$f(n) = \frac{n+\frac{n}{2}}{2} = \frac{2n+n}{2} = \frac{3n}{2}$

$O(n)$

```
for( i=n ; i>0 ; i--)   n+1
  {
    —
  }
```

$O(n)$

```
for( i=0; i<n; i++)   — n
  {
    for (j=0; j<n; j++)  — n x n
      {
        ———   —— n x n
      }
```

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

$O(n^2)$ ✓

```
for ( i=0 ; j<n ; i++)        n-1
  {
  - for ( j=0 ; j<i ; j++)
    {
        ---------- statement
    }
```

i	j	s
0	x	0
1	0	1
	1x	
2	0	2
	1	
	2x	
3	0-	3
	1-	
	2-	
	3x	
⋮	⋮	
n		n

$1+2+3+\cdots+n$

$f(n) = 1+2+3+4-\cdots\cdots n$

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

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

$\downarrow$

$O(n^2)$

```
P=0
for(i=1 ; p<=n ; i++)
  {
    P=p+i
  }
```

i	p
1	0+1
2	1+2
3	1+2+3
4	1+2+3+4
⋮	
k	1+2+3+4⋯+k

$p > n$

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

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

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

$\frac{k^2+k}{2} = n$

$k^2 = n$

$k = \sqrt{n}$

$f(n) = \sqrt{n}$

$O(\sqrt{n})$

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

i	
1	
1×2	5
1×2^2	5^2
1×2^3	5^3
1×2^4	5^4
⋮	
2^k	5^k

$i >= n$

$2^k >= n$

$2^k = n$

$k = \log_2 n = f(n)$

$O(\log_2 n)$ ✓

$\log_5 n$ ✓

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

$i < 1$

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

$k = \log_2 n$

$O(\log_2 n)$

i
n
$\frac{n}{2}$
$\frac{n}{2^2}$
$\frac{n}{2^3}$
⋮
$\frac{n}{2^k}$

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

$i^2 >= n$

$k^2 >= n$

$k^2 = n$

$k = \sqrt{n}$

$O(\sqrt{n})$

i
0
1
.
.
.
k

```
p = 0
for (i=1 ; i<n ; i=i*2)
  {
    p++ ;
  }
for (j=1 ; j<p ; j=j*2)
  {
    Statement;
  }
```

$\log_2 p$

$O(\log_2 \log_2 n)$

$n \log n$

$n^2 + n + p$

Types of Time Function:-

$O(1)$ → constant

$O(\log n)$ → Logarithmic — $\log(\log n)$

$O(n)$ → Linear

$O(n^2)$ → Quadratic

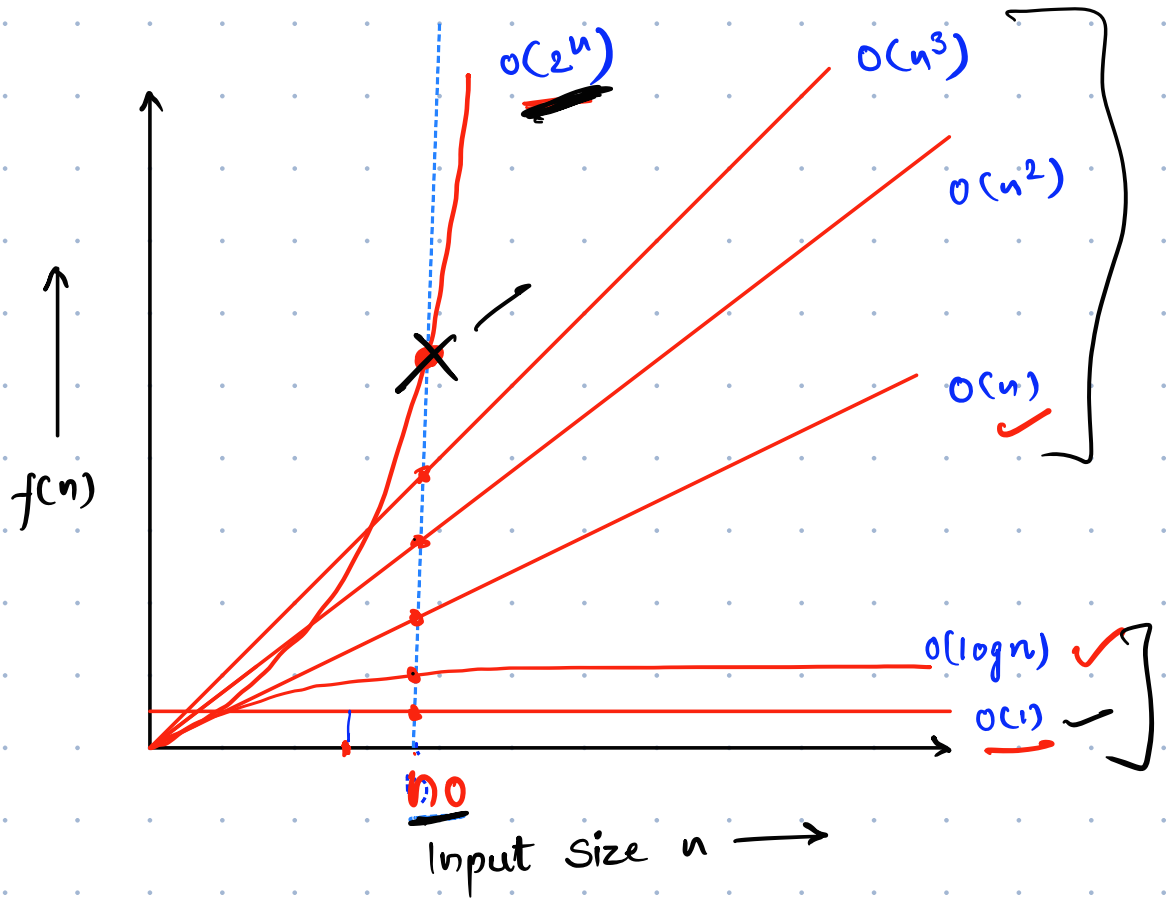
$O(n^3)$ → cubic.

$O(2^n)$, $O(3^n)$, $O(n^n)$ → Exponential

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 \ldots$

$< 2^n < 3^n \ldots < n^n$

→ worst case


O(2^n)
O(n^3)
O(n^2)
O(n)
O(log n)
O(1)
f(n)
no
Input Size n

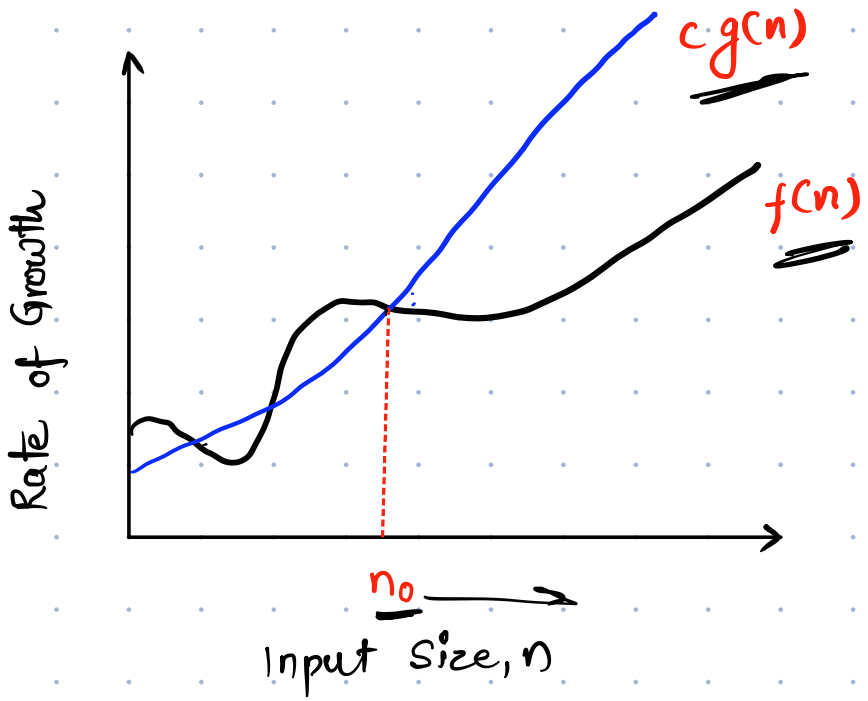
Asymptotic Notation:

Big-O Notation :

- Gives tight upper bound for all $n >= n_0$ of given function.
- Represented as, $f(n) = O(g(n))$.

Big O notation defined as $O(g(n)) = \{f(n):$ there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c\,g(n)$$



3n+8

$f(n) = 3n + 8$

let $g(n) = n$

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

c & n_0

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

$3n + 8 \leq 3n + 8n$

$3n + 8 \leq 11n$

$c = 11$

$n_0 = 1$

$11 \leq 11$

$n \geq 8$

$c = 4$

$3(8) + 8 \leq 4(8)$

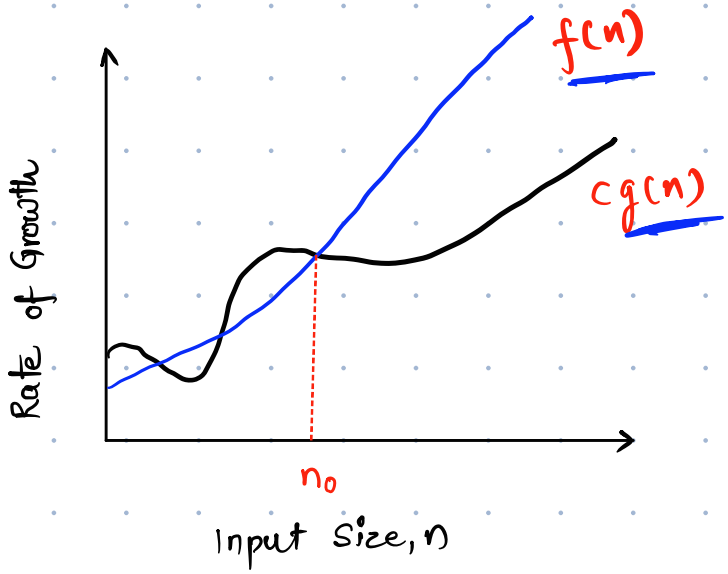
$32 \leq 32$

Big Omega (Ω) Notation:

- Similar to Big O discussion
- Gives lower bound of given function.
- Represented as, $f(n) = \Omega(g(n))$

i.e. at larger values of n, the tighter lower bound of $f(n)$ is $g(n)$.

The Big Omega notation can be defined as $\Omega(g(n)) = \{f(n)$: there exists positive constants c and n_0, such that $0 \leq c\,g(n) \leq f(n)$ for all $n > n_0\}$



$f(n) = 5n^2$

$c\, n^2 \leq 5n^2$

$g(n) = n^2$

$c = 5 \quad n = 1$

$n = 2$

$64 \leq 20$

$c = 5 \quad n = 2$

Big Theta (θ) Notation:

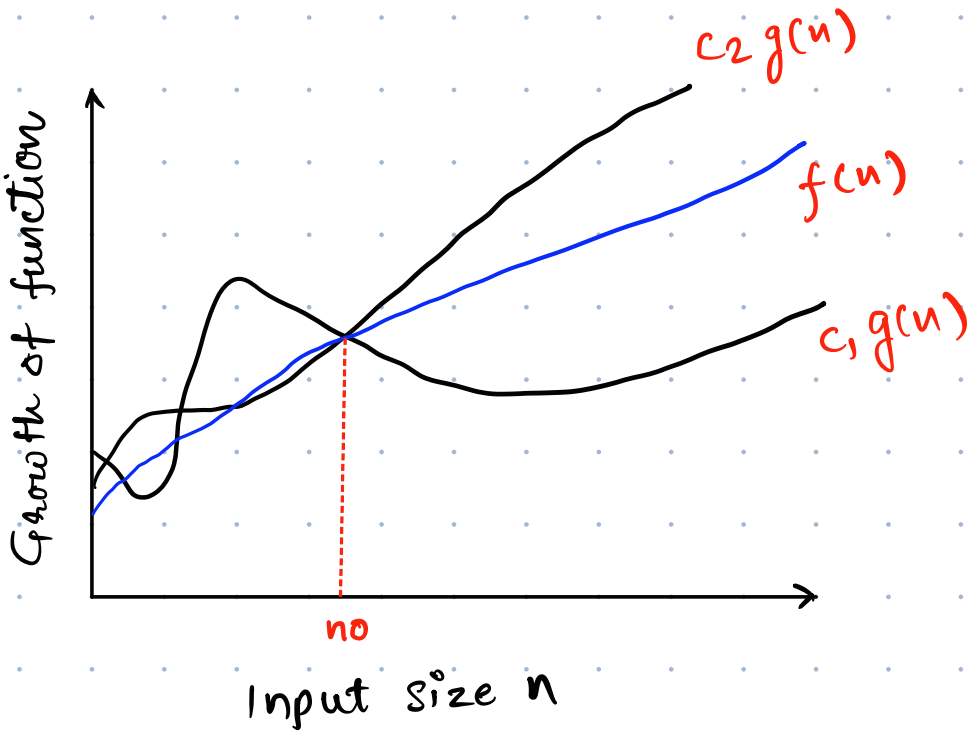
- Average running time of an algorithm is always between lower bound and upper bound.

Theta notation can be defined as,

$\theta(g(n)) = \{ f(n)$: there exists positive constants c1, c2 and no such that ,

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

for all $n >= no$.



2.9 PROPERTIES OF ASYMPTOTIC NOTATIONS

1. **Transitivity:**
   If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$.
   If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.
   If $f(n) = W(g(n))$ and $g(n) = W(h(n))$ then $f(n) = W(h(n))$.
   If $f(n) = o(g(n))$ and $g(n) = o(h(n))$ then $f(n) = o(h(n))$.
   If $f(n) = w(g(n))$ and $g(n) = w(h(n))$ then $f(n) = w(h(n))$.
2. **Reflexivity:**
   $f(n) = \Theta(f(n))$.
   $f(n) = O(f(n))$.
   $f(n) = W(f(n))$.
   "o" is not reflexive
   "ω" is not reflexive
3. **Symmetry and Transpose symmetry:**
   - **Symmetry:**
     $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.
   - **Transpose symmetry:**
     $f(n) = O(g(n))$ if and only if $g(n) = W(f(n))$.
     $f(n) = o(g(n))$ if and only if $g(n) = w(f(n))$.
4. **Analogy between asymptotic comparison of functions and comparison of real numbers.**
   $f(n) = O(g(n))$ » $a <= b$
   $f(n) = W(g(n))$ » $a >= b$
   $f(n) = \Theta(g(n))$ » $a = b$
   $f(n) = o(g(n))$ » $a < b$
   $f(n) = w(g(n))$ » $a > b$

   f(n) is asymptotically smaller than g(n) if $f(n) = o(g(n))$
   f(n) is asymptotically larger than g(n) if $f(n) = w(g(n))$
5. **Exponentials:** For all real $a > 0$, m, and n, we have the following identities:
   $a_0 = 1$,
   $a^1 = a$,
   $a^{-1} = 1/a$
   $(am)^n = am^n$
   $(am)^n = (an)^m$
   $a^m a^n = a^{m+n}$

Properties of Asymptotic Notations:

1. Transitivity:
2. Reflexivity
3. Symmetry
4. Transpose Symmetry

(1.) $f(n) = \underline{\Theta}(g(n))$ $\quad g(n) = \Theta(h(n))$

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

O, Ω

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

$h(n) = n^3$

$n = \underline{o}(n^2)$ $\quad n^2 = O(\underline{n^3})$

$\longrightarrow n = O(n^3)$ $\quad f(n) \leq O(g(n))$

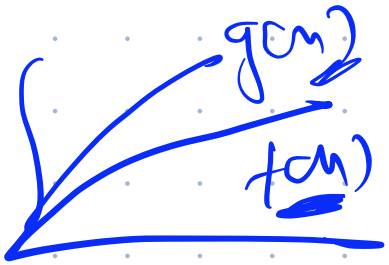
(2.) $f(n) = \Theta(f(n))$

$f(n) = n^3$

$f = \Theta(n^3)$

$= O(n^3)$

Symmetry



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

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

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

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

Transpose Symmetry :-

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

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

$f(n) = n$

$g(n) = n^2$

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

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

①

$f(n) = O(kg(n))$

$k > 0$

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

② $f_1(n) = O(g_1(n)) = n$

$f_2(n) = O(g_2(n)) = n^2$

$n^2 \neq n$

$(f_1 + f_2)(n)$

$= O(\max(g_1(n), g_2(n)))$

$= O(n^2)$

$f_1(n) = O(g_1(n)) = n$

$f_2(n) = O(g_2(n)) = n^2$

$f_1(n) \cdot f_2(n) = n^3$

$= O(g_1(n) \cdot g_2(n))$

$n^2 + n)$

$= O(n^3)$

Recursion:

- Recursion is an ability of an algorithm to repeatedly call itself until a certain condition is met. Such condition is called the base condition.
- The algorithm which calls itself is called a recursive algorithm.

The recursive algorithms must satisfy the following two conditions:

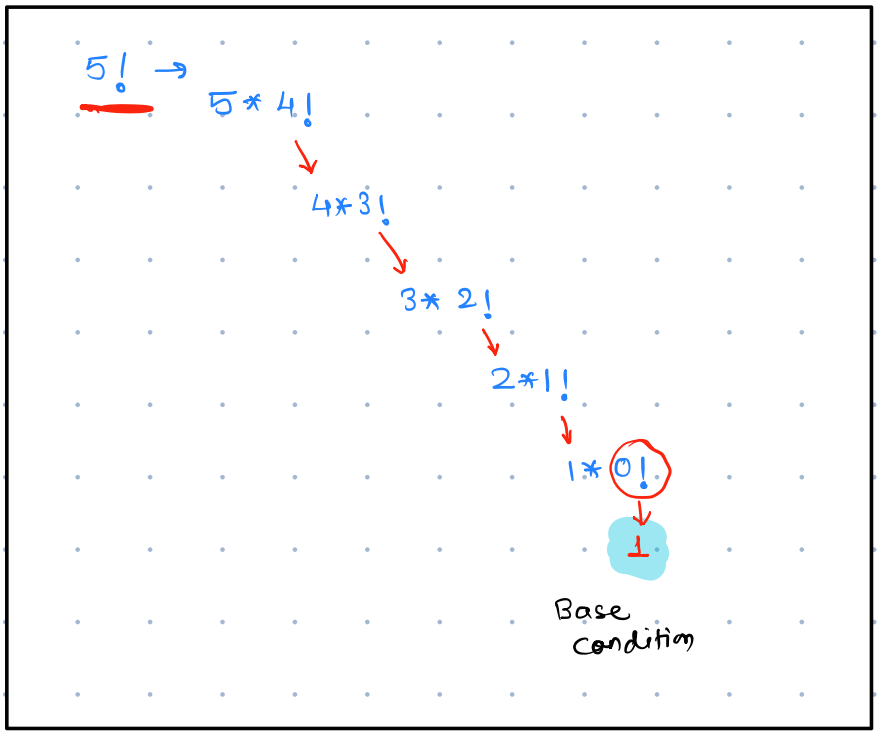
1. It must have the base case: The value of which algorithm does not call itself and can be evaluated without recursion.
2. Each recursive call must be to a case that eventually leads toward a base case.

Recurrence Relation:

- An algorithm is said to be recursive if it can be defined in terms of itself.
- The running time of recursive algorithm is expressed by means of recurrence relations.
- A recurrence relation is an equation of inequality that describes a function in terms of its value on smaller inputs.

- It is generally denoted by T(n) where n is the size of the input data of the problem.
- The recurrence relation satisfies both the conditions of recursion, that is, it has both the base case as well as the recursive case.
  - The portion of the recurrence relation that does not contain T is called the base case of the recurrence relation and
  - The portion of the recurrence relation that contains T is called the recursive case of the recurrence relation.

$$T(n) = \begin{cases} d & n = 1 \\ T(n-2) + c & n > 1 \end{cases}$$



$n!$
$= n * (n-1) * (n-2) * (n-3) \cdots \cdots 3 * 2 * 1$

$5! = 5 * 4 * 3 * 2 * 1$

$5! = 5 * 4!$

$4! = 4 * 3!$

Example: Factorial

There are various methods to solve recurrence:

1. Iterative Method / Substitution Method
3. Recurrence Tree
4. Master Method

$$T(n) = \begin{cases} d & ; n = 1 \\ T(n-1) + c & ; n > 1 \end{cases}$$

Factorial Algorithm:

```
Algo fact(N)
If (n<0) then return ("error");
  else
if (n<2) then return (1);
  else
Return (n* fact (n-1));
End if
End if
End fact
```

Recursive Method

$$T(n) = \begin{cases} d & ; n=1 \\ T(n-1)+c & n>1 \end{cases}$$

```
Algo fact(N)
If (n<0) then return ("error")
  else
if (n<2) then return (1);
  else
prod=1;
End if
End if

For (i=n down to 1)
  do prod= prod*i
End for
Return (prod)
End fact
```

Iterative Method

1 unit

1 unit + 1 unit = 2 unit

prod=1; → 1 unit

For (i=n down to 1) → n+1

do prod= prod*i → n

$O(n)$

Example 1: Factorial Example

$n! = n \times (n-1) \times (n-2) \times (n-3) \cdots \times 3 \times 2 \times 1$

$n! = n \times (n-1)!$

$(n-1) = (n-1) \times (n-2)!$

$(n-2) = (n-2)(n-3)!$

$\vdots$

$1! = 1 * 0!$

$0! = 1$ —— Base case

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ T(n-1) + c & \text{if } n > 1 \end{cases}$$

$T(1)$

$T(1) = d$

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

$= c + d$

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

$= c + c + d = 2c + d$

$T(4) = 3c + d$

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

$= c + T(n-3)$

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

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

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

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

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

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

$$\vdots$$

$$T(n) = kc + T(n-k)$$

$$n - k = 1$$

$$k = n - 1$$

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

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

$$= cn - c + d$$

$$\Rightarrow O(n)$$

$$T(n) = \begin{cases} d & n = 1 \\ T(n-1) + c & n > 1 \end{cases}$$

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

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

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

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

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

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

$$\vdots$$

$$T(n) = T(n-k) + kc \rightarrow$$

$$n - k = 1$$

$$k = n - 1$$

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

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

$O(n)$

Example 2:

Solve the recurrence $T(n) = T(n+1) + n$ using iterative method.

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

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

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

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

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

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

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

$$\vdots$$

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

$$+ (n-(k-2)) \cdots + n$$

$$n - k = 1$$

$$T(n) = T(1) + 2 + 3 + 4 \cdots + n$$

$$= 1 + 2 + 3 + 4 \cdots + n$$

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

Example 3:

Solve the recurrence using iterative method.

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

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

$$T(n) = \begin{cases} 1 \text{ or } d & \text{if } n = 1 \\ n + 2T(n/2) & n > 1 \end{cases}$$

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

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

$$T(n) = n + 2\left(\frac{n}{2} + 2T(n/4)\right)$$

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

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

$$T(n/4) = \frac{n}{4} + 2T\left(\frac{n/4}{2}\right)$$

$$= \frac{n}{4} + 2T(n/8)$$

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

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

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

$$\vdots$$

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

$$T(1) = 1$$

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

$$n = 2^k$$

$$\log n = \log 2^k$$

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

$$\log n = k \, (1)$$

$$T(n) = n \log n + n T(1)$$

$$= n \log n + n \cdot 1$$

$$O(n \log n)$$

Remember :- [Decreasing Function]

$$T(n) = T(n-1) + 1 \quad \rightarrow \quad O(n)$$

$$T(n) = T(n-1) + n \quad \rightarrow \quad O(n^2)$$

$$T(n) = T(n-1) + \log n \quad \rightarrow \quad O(n \log n)$$

$$T(n) = 2T(n-1) + 1 \quad \rightarrow \quad O(2^n)$$

$$T(n) = 3T(n-1) + 1 \quad \rightarrow \quad O(3^n)$$

$$T(n) = 2T(n-1) + n \quad \rightarrow \quad O(n2^n)$$

Logrithm :-

1.) $a^{\log_b x} = x^{\log_b a}$

2.) $\log_b a = \frac{\log_x a}{\log_x b}$

Master Theorem:

- For Divide and Conquer recurrences
- For subtracting/Decreasing recurrences

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad n > 1$$

$$= O(1) \qquad O(1) \qquad n = 1$$

$$\boxed{f(n) = O\left(n^k \log^p n\right)}$$

$a \geq 1$; $b > 1$; $k \geq 0$

p is a real number

case 1 :-

$$a > b^k \Rightarrow \boxed{\log_b a > k}$$

$$\log a > \log b^k \Rightarrow \log a > k \log b$$

$$\frac{\log_2 a}{\log_2 b} > k$$

$$\log_b a > k$$

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

case 2 if $a = b^k$ or $\log_b a = k$

then,

① If $P > -1$, then

$$T(n) = \Theta(n^{\log_b a} \log^{P+1} n)$$

$$\Theta(n^k \log^{P+1} n)$$

② If $P = -1$, then

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

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

(g) If $p < -1$, then

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^k)$

case 3 :- If $a < b^k$ or

$\log_b a < k$

(a) If $p \geq 0$, then $f(n)$

$T(n) = \Theta(n^k \log^p n)$

(b) If $p < 0$, then

$T(n) = O(n^k)$

① $T(n) = 2T(n/2) + 1$

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

$a = 2$; $b = 2$

$f(n) = 1$

$n^k \log^p n = 1$

$k = 0$ $\quad p = 0$

$\log_b a = \log_2 2 = 1$

$\log_b a > k$

1 0

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

$$= \Theta(n)$$

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

$a = 4$; $b = 2$

$f(n) = n$

$\log_b a = \log_2 4 = 2$

$(n^k \log^p n) = n$

$k = 1$ $\quad p = 0$

$\therefore \log_b a > k$

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta(n^2)$$

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

$$\log_b a = \log_2 2 = 1$$

$$f(n) = n$$

$$\left(n^k \log^p n\right) = n$$

$k = 1$ $\quad p = 0$ $\quad p > -1$

$$\log_b a = k$$

$$T(n) = \Theta\left(n^k \log^{p+1} n\right)$$

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

(4.) $T(n) = 2T(n/2) + n^3$

$a = 2 \quad b = 2 \quad \log_b a = 1$

$f(n) = n^3$

$k = 3 \quad p = 0$

$\log_b a < k$ ✓

$p = 0$ ie $p \geq 0$

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^3)$

5. $T(n) = 2T(n/2) + n^3 \log n$

6. $T(n) = 2T(n/2) + \frac{n^3}{\log^2 n}$

7. Solve following recurrence relation:

$$T(n) = 2T(\sqrt{n}) + \log n \quad \text{---(1)}$$

$T(n) = aT(n/b) + f(n)$ ---(1)

let $n = 2^m$

$m = \log_2 n$

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

$= 2T(2^{m/2}) + m \log 2$

$= 2T(2^{m/2}) + m$

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

$= 2S(m/2) + m$

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

$a = 2$; $b = 2$ $\quad f(n) = n^1$

$$f(n) = (n^k \log^p n)$$

$k = 1$

$$\log_b a = \log_2 2 = 1$$
$$= k$$

$p = 0$

case 2 :- $\quad p > -1$

$$S(n) = \Theta(n^k \log^{p+1} n)$$
$$= \Theta(n \log n)$$

$$T(n) = T(2^n)$$

$$2^m = n$$

$$m = \log_2 n$$

$$T(n) = \theta(\log n * \log(\log n))$$

$$= \theta(\log n \log\log n)$$

Master Method (for Decreasing Functions/ Algorithms)

$$T(n) = \begin{cases} c & n \le 1 \\ aT(n-b) + f(n) & n > 1 \end{cases}$$

$a > 0 \quad b > 0 \quad k \ge 0$

$$f(n) = O(n^k)$$

$T(n) = O(n^k) - [f(n)]$ $\quad a < 1$

$T(n) = O(n^{k+1})$ $\quad a = 1$

$= O(n^k \cdot n)$

$= O(n \cdot f(n))$

$\geq O(n)$

$= O(n^k a^{n/b})$ $\quad a > 1$

$= O(a^{n/b} \cdot f(n))$

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

$a = 1 \quad : b = 1 \quad f(n) = 1$

$O(n^k) = 1$

$k = 0$

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

$= O(n^1) = O(n)$

(2.)

$T(n) = T(n-1) + n$ $\quad O(n^2)$

$a = 1$ $\quad b = 1$ $\quad f(n) = n = O(n^k)$

$k = 1$

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

$= O(n^2)$

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

$O(n^3)$

$T(n) = T(n-1) + \log n$ → recurrence

a^k

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

$a = 2 \quad b = 1 \quad f(n) = O(n^k)$

$k = 2$

$a > 1$

$$T(n) = O(n^k a^{n/b})$$

$$= O(n^2 2^{n/1})$$

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

$$= O(2^n)$$

$$T(n) = \frac{T(n-5)}{3} + n$$

$a = 1/3 \quad b = 5 \quad f(n) = O(n^k)$

$k = 1$

O(n)

Recursion Tree

$$T(n) = aT(n/b) + f(n)$$

$f(n)$ → root node → cost

a — Number of subproblems

b — size of subproblem

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

2 → a

2 → b

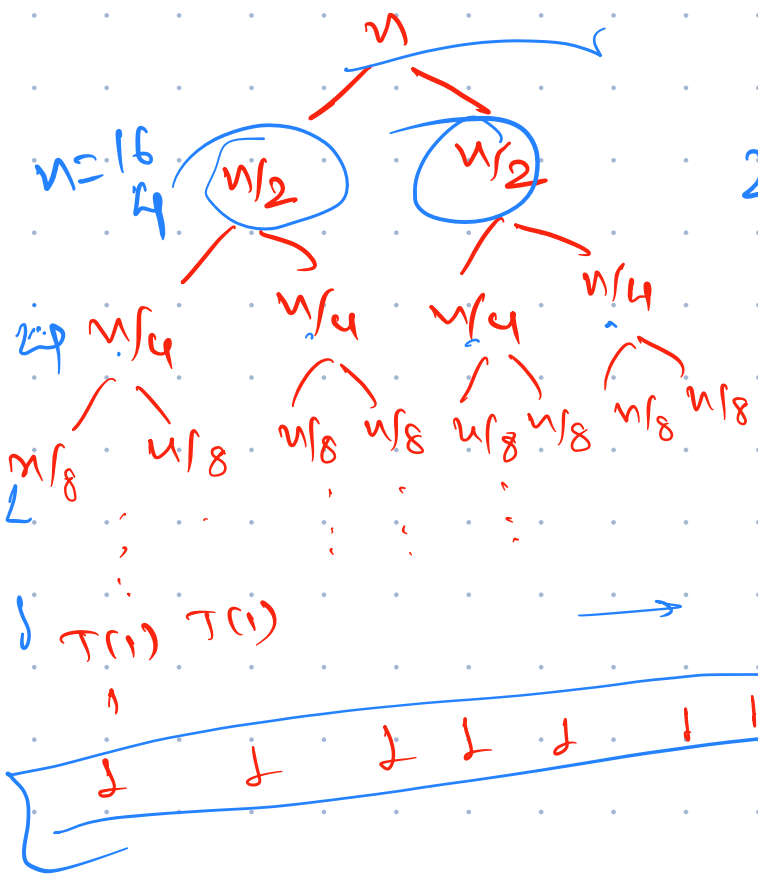
n → $f(n)$

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

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

$T(n/4) = 2T(n/8) + n/4$

$T(1) = 1$



Level	No of nodes	cost
0	1	n
1	2	n
2	4	n
3	8	n
⋮	⋮	
k	2^k	n
		kn

Total cost = kn

Base condition = $T(1) = 1$

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

$n = 2^k$

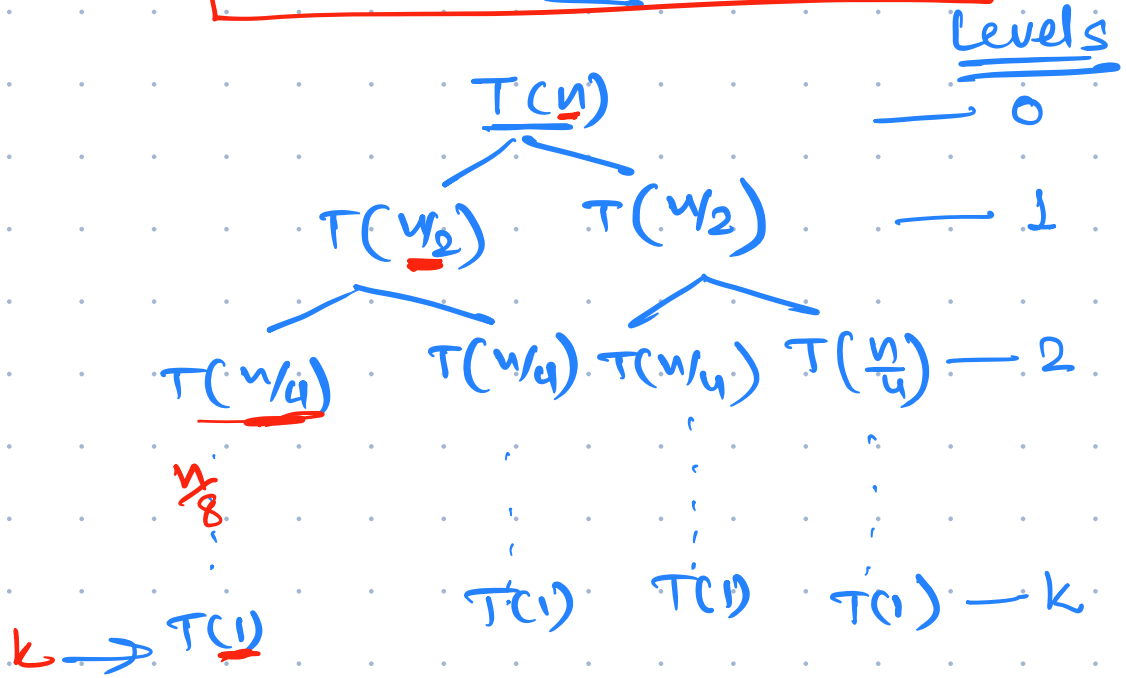
$k = \log_2 n$

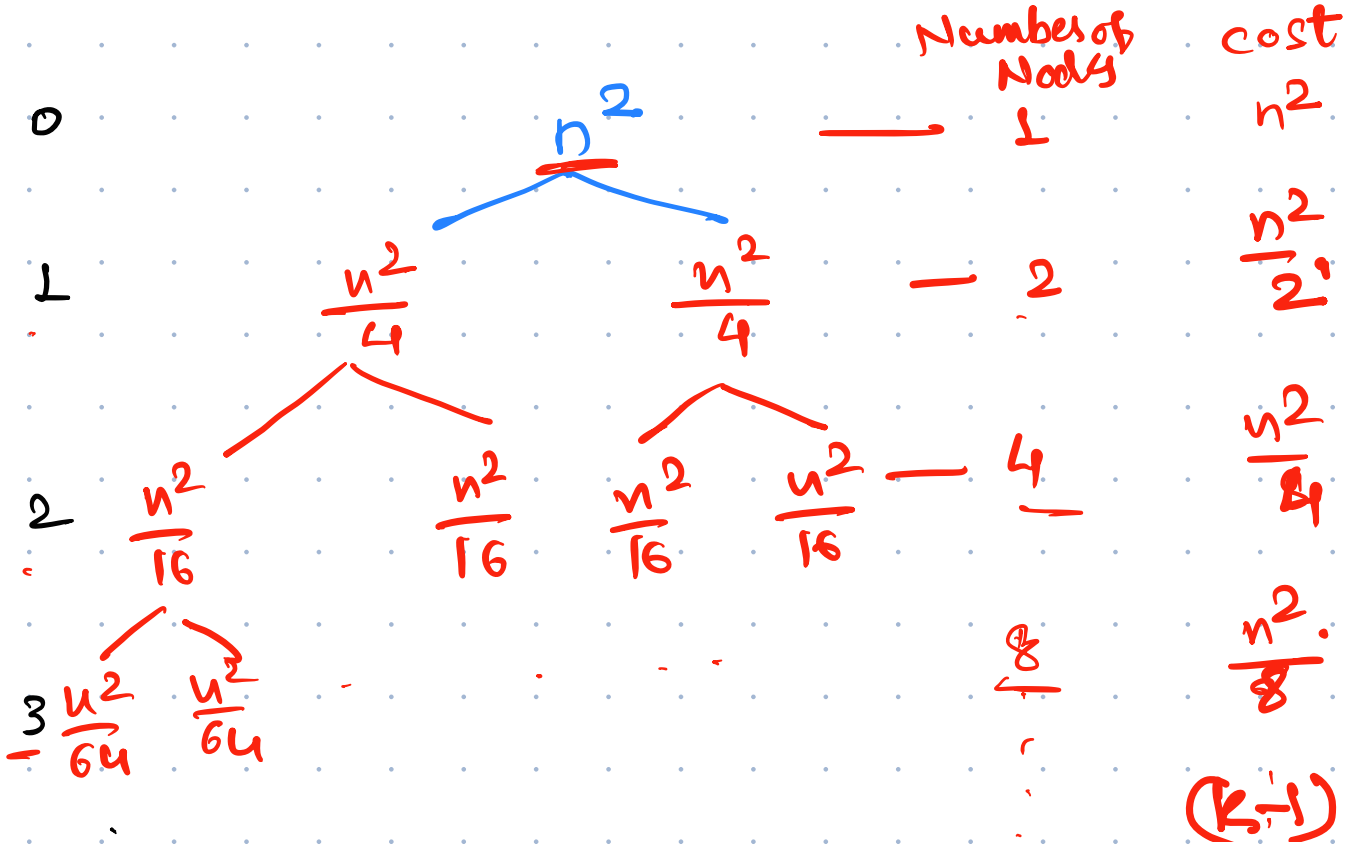
Total cost $= n \log n$

$T(n) = O(n \log n)$

Example 2 :-

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





k | T(1) T(1) - - - - - - T(1) | 2^k | $\frac{n^2}{2^k}$

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

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

$$n = 2^k$$

$k = \log_2 n$ → Level

Total cost → cost of non-leaf nodes (N_c)
+
cost of leaf nodes (L_c)

$= N_c + L_c$

$$N_c = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \cdots \frac{n^2}{2^{k-1}}$$

$$= n^2 \left(1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} \cdots + \frac{1}{2^{k-1}}\right)$$

Geometric Series

$x = \frac{1}{2} < 1$

$n \to \infty \rightarrow$

$$N_c = n^2 \left(\frac{1}{1 - \frac{1}{2}}\right)$$

$$\doteq 2n^2$$

$$L_c = 2^k \rightarrow 2^{\log_2 n} = n^{\log_2 2} = n$$

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

$$= \frac{n^2}{2^k} = \frac{n^2}{n} = \underline{n}$$

Total cost $= 2n^2 + n$

$= O(n^2)$

Example 3

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{4}\right) + n^2 & n > 1 \end{cases}$$

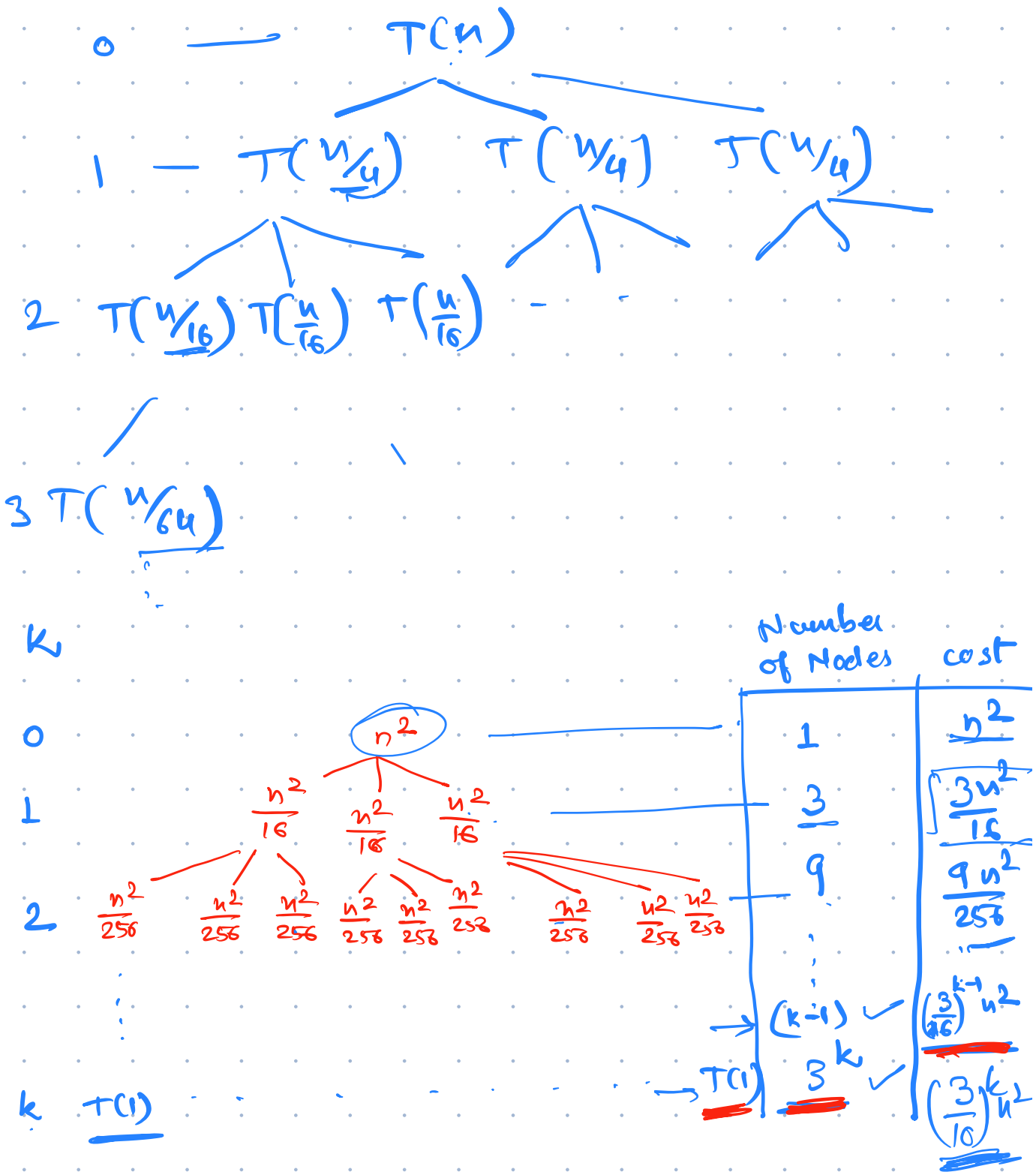
0 $T(n) = 3T\left(\frac{n}{4}\right) + n^2$

1 $T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{16}\right) + \frac{n^2}{16}$

2 $T\left(\frac{n}{16}\right) = 3T\left(\frac{n}{64}\right) + \frac{n^2}{256}$

3 $T\left(\frac{n}{64}\right) = 3T\left(\frac{n}{256}\right) + \frac{n^2}{64^2}$

$T(1)$ $\frac{n}{4^k}$


0
T(n)
1
T(n/4)
T(n/4)
T(n/4)
2
T(n/16)
T(n/16)
T(n/16)
3
T(n/64)
k
T(1)
k
Number of Nodes
cost
0
1
n²
1
3
3n²/16
2
9
9n²/256
(k-1)
(3/16)^(k-1) n²
3^k
T(1)
(3/16)^k n²

$$\frac{n}{4^k} = 1$$

$$n = 4^k$$

$$\boxed{k = \log_4 n}$$

Total cost $=$ $N_c + L_c$

$$N_c = n^2 + \frac{3}{16}n^2 + \frac{9}{256}n^2 + \cdots + \left(\frac{3}{16}\right)^{k-1} n^2$$

$$= n^2 \left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 \cdots \left(\frac{3}{16}\right)^{k-1} \right]$$

$$= n^2 \left(\frac{1}{1-x}\right)$$

$$x = \frac{3}{16}$$

$$= n^2 \left(\frac{1}{1 - \frac{3}{16}} \right)$$

$$N_c = n^2 \left(\frac{16}{13} \right)$$

$$L_c = 3^k = 3^{\log_4 n} = n^{\log_4 3}$$

$$\text{Total cost} = \frac{16}{13} n^2 + n^{\log_4 3}$$

$$\log_4 3 < 1$$

$$\log_4 4$$

$$= O(n^2)$$

Example 4

$$T(n) = \begin{cases} 1 & n=1 \\ T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n & n>1 \end{cases}$$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$$

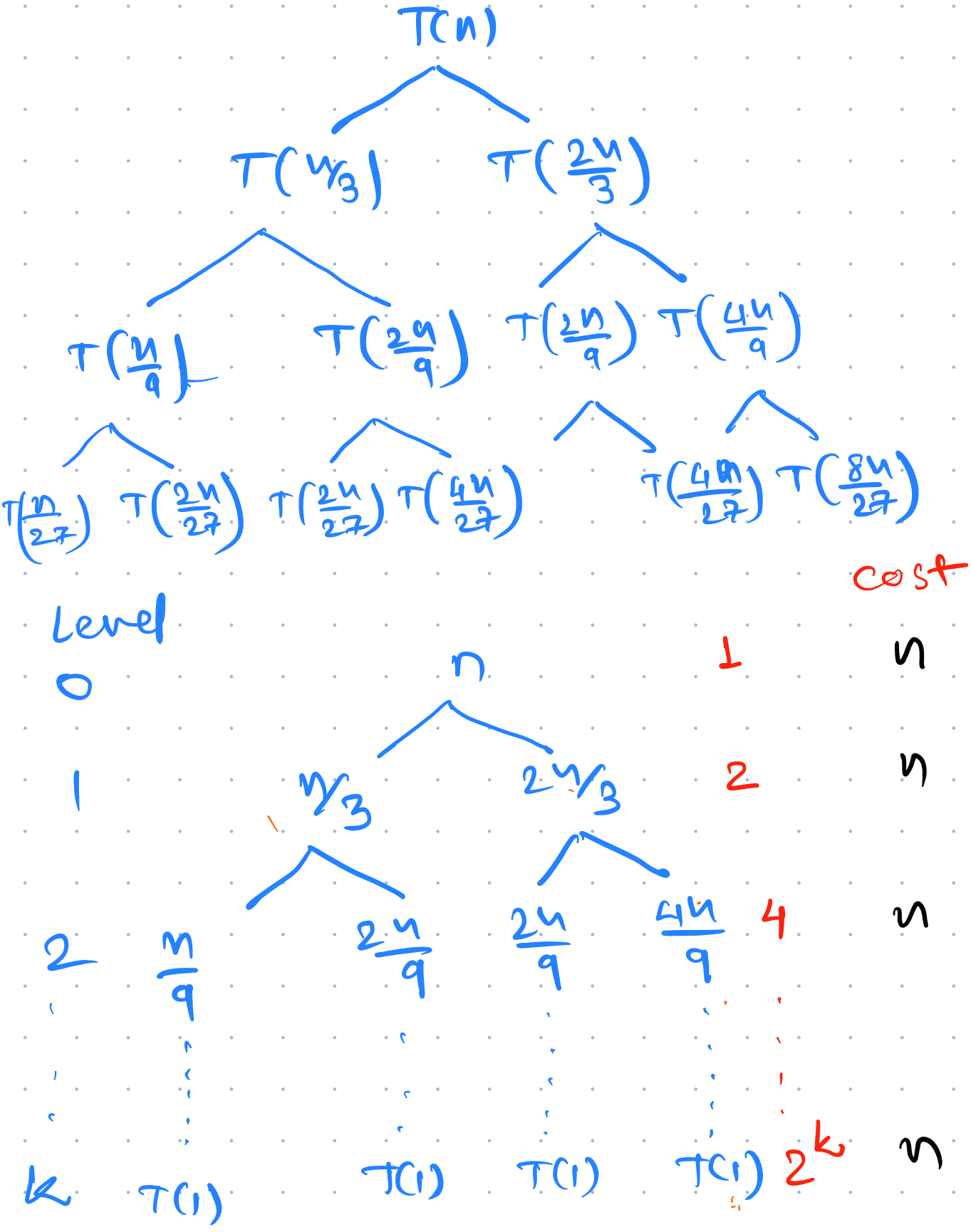
$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{9}\right) + T\left(\frac{2n}{9}\right) + \frac{n}{3}$$

$$T\left(\frac{2n}{3}\right) = T\left(\frac{2n}{9}\right) + T\left(\frac{4n}{9}\right) + \frac{2n}{3}$$

$$T\left(\frac{n}{9}\right) = T\left(\frac{n}{27}\right) + T\left(\frac{2n}{27}\right) + \frac{n}{9}$$

$$T\left(\frac{2n}{9}\right) = T\left(\frac{2n}{27}\right) + T\left(\frac{4n}{27}\right) + \frac{2n}{9}$$

$$T\left(\frac{4n}{9}\right) = T\left(\frac{4n}{27}\right) + T\left(\frac{8n}{27}\right) + \frac{4n}{9}$$


T(n)
T(n/3)
T(2n/3)
T(n/9)
T(2n/9)
T(2n/9)
T(4n/9)
T(n/27)
T(2n/27)
T(2n/27)
T(4n/27)
T(4n/27)
T(8n/27)
Level
cost
0
n
1
n
1
n/3
2n/3
2
n
2
n/9
2n/9
2n/9
4n/9
4
n
k
T(1)
T(1)
T(1)
T(1)
2^k
n

$\frac{2n}{3}$ $\frac{4n}{9}$ =

Level k = $\frac{2^k}{3^k} n = 1$

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

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

$k = \log_{3/2} n$

$Nc = n + n + n \cdots n$

$$= kw$$

$$= w \log_{3/2} n$$

$$LC = 2^k \tau(1)$$

$$= 2^{\log_{3/2} w}$$

$$= n^{\log_{3/2} 2}$$

Total cost $= w \log_{3/2} n + n^{\log_{3/2} 2}$