19/8/20

Algorithm :-

Def:- Algorithm is a step by step procedure to solve the computering program.

Algorithm	Program
1. Design	1. Implementation
2. Domain Knowledge	2. Programming
3. Any language	3. Any programming language.
4. Independent on hard ware / Operating system.	4. Depends on hard ware / Operating system
5. Analysis → Time Complexity, Space complexity	5. Testing → Out put → Size of the Software in bytes.

Time Complexity → function of time

$f(n) \rightarrow O(n), O(n^2)$ → Order of time

Priori Analysis	Posteriori Testing
1. Algorithm	1. Program
2. Independent of language	2. language dependent
3. Independent of hd ware	3. Depends on hardware.
4. Time / Space	4. Watch time / bytes.

Scanned with CamScanner

Characteristics of Algorithm :-

1. Input → 0 (or) more inputs
2. Output → at least 1 Output
3. Definiteness → Should use only known step.
4. finiteness → Should terminate in finite loop
5. Effectiness → Correctness.

How to write an Algorithm :-

for Swapping

```
Algo Swap (a, b)
{
    temp = a;
    a = b;
    b = temp;
}
```

How to Analysis on Algorithm ?

1. Time Complexity
2. Space Complexity
3. Network, Band width
4. power Consumptions.
5. Cpu registers
6. Security
7. Other parameters

used for hand held devices.

Scanned with CamScanner

Analysis of Swapping

Algo Swap (a, b)	Time	Space
{		
temp ← a	1	1 UM
a ← b	1	1 UM
b ← temp	1	1 UM

$f(n) = 3$ times units

↳ $O(1)$ → Order of function

$T(n) = 3$

$S(n) = 3UM$

↳ $O(1)$

20/8/20

* for Complex statements like $x = 5*a + 6*b$, still it takes constant amount of time and the Order of the function is also one.

frequency Count method:-

① for (i=0; i<n; i++) → break at (n+1)-times

{

Statement → n times

}

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

↓

Dominant term

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

② for (i=n; i>0; i--)

{

Statement → n times

}

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

Scanned with CamScanner

③ for (i=1; i<n; i+2) → {1, 3, 5, ---- n}

{ statement → $n/2$ times

}

$T(n) = n/2$ ⇒ as the dominant term is still n.

The Order of function is again $n \rightarrow O(n)$

④ for (i=0; i<n; i++) → (n+1) times

nesting loops

{ for (j=0; j<n; j++) → (n+1)×n times

n times ← { statement → n×n times

}

}

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

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

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

↳ Dominant term is n^2

$O(n^2) \rightarrow$ Order of n^2

⑤ for (i=0; i<n; i++)

{ for (j=0; j<i; j++)

{ statement

}

i	j	no: of time
0	* 0	0
1	0	1
	* 1	0
2	0	1
	1	1
	2*	0

for n^{th} term it gives n time (runs)

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

$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$ → Dominant term is n^2

Order of Time Complexity $= O(n^2)$

Scanned with CamScanner

(ii) p = 0

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

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

When the Value of p is greater than n then the loop breaks

$$P > n$$

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

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

$$k^2 > n \Rightarrow k = \sqrt{n}$$

$\therefore$ Order of time Complexity $= O(\sqrt{n})$

21/8/20

* Algorithm Add(A, B, n)

```
{
  for (i=0; i<n; i++)        → Outer loop
  {
    for (j=0; j<n; j++)      → inner loop
    {
      C[i,j] = A[i,j] + B[i,j]
    }
  }
}
```

n times ← (outer loop)

↓ n×n times

Time Complexity for matrix addition $= O(n^2)$

Space Complexity :-

$A \rightarrow n \times n = n^2$

$B \rightarrow n \times n = n^2$

$C \rightarrow n \times n = n^2$

$i \rightarrow 1,\ j \rightarrow 1,\ n \rightarrow 1$

$S(n) = 3n^2 + 3$

So, the space Complexity $= O(n^2)$

Scanned with CamScanner

* Algorithm multiply (A, B, n)

```
{
  for (i=0; i<n; i++)
  {
    for (j=0; j<n; j++)
    {
      C[i,j] = 0
      for (k=0; k<n; k++)
      {
        C[i,j] = C[i,j] + A[i,k] * B[k,j];
      }
    }
  }
}
```

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

$f(n) = O(n^3)$

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

↓

Space Complexity

* for (i=1; i<n; i=i*2)

```
{
  Statement
}
```

$1 \times 2 = 2$

$2 \times 2 = 2^2$

$2^2 \times 2 = 2^3$

⋮

2^k

Assume:-

$i > n$

$2^k \geq n$

$2^k = n$

$k = \log_2 n$

Time Complexity = $O(\log_2 n)$ → Celling / floor

↑ float / decimal

md:- ②

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

$i = 1+1+1+ \cdots n \Rightarrow O(n)$

Similarly for the above program

$i = 1 \times 2 \times 2 \times 2 \times \cdots k = 2^k$

$2^k = n \Rightarrow k = \log_2 n$

Scanned with CamScanner

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

$i = i/2$

$= n/2, n/2^2, n/2^3, \ldots n/2^k$

$i = n/2^k$ $\quad (i<1) \Rightarrow$ loop breaks

$n/2^k < 1 \Rightarrow n/2^k = 1 \Rightarrow k = \log_2 n$

$\therefore$ Time Complexity is of Order $\Rightarrow O(\log_2 n)$

* p=0
for (i=1; i<n; i=i*2)
{
p++ $\rightarrow \log n = p$
}
for (j=1; j<p; j=j*2)
{
statement $\rightarrow \log p$
} $\rightarrow \log(\log n)$

$\therefore$ Time Complexity $= O(\log(\log n))$.

* for (i=0; i<n; i++)
{
for (j=1; j<n; j=j*2) $\rightarrow n$
{
statement $\rightarrow n * \log n$
}
}

$\therefore$ Time Complexity of order $= O(n \log n)$ //

Scanned with CamScanner

22/8/20

Analysis of if and While

* While (condition)
  {
  statement
  }

  ⇓

  first statement then it check for condition

do
{
statement
}
While (condition)

⇓

Whether the condition is true/false the statem excutes 1 time.

* $i = 0 \rightarrow 1$

  While $(i<n) \rightarrow n+1$
  {
  statement $\rightarrow n$
  $i++ \rightarrow n$
  }

  $\Rightarrow T(n) = 3n+2$

  $O(n) \rightarrow$ Time Complexity

  ⇓

  linear time Complexity

  ⇓

  for $(i=0; i<n; i++) \rightarrow n+1$
  {
  statement $\rightarrow n$
  }

  $\Rightarrow T(n) = 2n+1$

  $O(n) \rightarrow$ Time Complexity

⇒ for while loop and for loop we will get same time Complexity but it has little change in the time function.

* a = 1;
  While (a<b);
  {
  statement;
  a = a*2
  }

a
1
$1 \times 2 = 2$
$2 \times 2 = 2^2$
$2^3 \times 2 = 2^4$
⋮
2^k

at $a \geq b$ the loop breaks

$a \geq b$

$2^k \geq b$

$2^k = b$

$k = \log_2 b$

∴ Time Complexity of Order $= O(\log_2 b)$

Scanned with CamScanner

* $i = n$

```
While (i>1)
{
  Statement
  i=i/2
}
```

$i = 1/2, 1/2^2, 1/2^3, \ldots 1/2^k$

$\Rightarrow n/2^k \le 1 \Rightarrow n = 2^k \Rightarrow k = \log(n)$

$\Rightarrow$ Time Complexity of Order $= O(\log n)$

at $n/2^k \le 1$ the loop breaks.

* $i = 1;$

$K = 1;$

```
While (K<n)
{
  Statement
    K=K+i
    i++
}
```

i	K
1	1
2	1+1 = 2
3	2+2 = 4
4	2+2+3 = 7
⋮	
m	2+2+3+4+ ... m

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

$$= 1 + \frac{m^2+m}{2} = n$$

As the dominant is $m^2 \Rightarrow m^2 = n \Rightarrow m = \sqrt{n}$.

$\therefore$ Time Complexity of Order $= O(\sqrt{n})$

* While (m!=n)

```
{
  if (m>n)
  m=m-n;
  else
    n=n-m;
}
```

m	m	m	n
6	3	16	2
3	3	14	2
	↓ O(1)	12	2
		10	2
		8	2
	O(m/2) ←	6	2
		4	2
		2	2

$\Rightarrow$ These type of programs has two time Complexity one is best case time Complexity and the Other is worst Case time Complexity.

Scanned with CamScanner

In Worst case, $T(n) = n/5$

$= O(n)$

Best case, $O(1)$

* Test (n)

```
{
  if (n < 5)
  {
    Printf ("%d", n)
  }
  else
  {
    for (i=0; i<n; i++)
    {
      Printf ("%d", i)
    }
  }
}
```

$n > 6$

$\Downarrow$

$O(n)$

(Order of Time Complex

$n = 4$

$\Downarrow$

$O(1)$

(Order of TC)

Types of Time functions :-

$O(1)$ = constant

$O(\log n)$ = logarithmic

$O(n) \rightarrow$ linear

$O(n^2) \rightarrow$ Quadratic

$O(n^3) \rightarrow$ Cubic

$O(2^n) \rightarrow$ Exponential

If

$f(n) = 2$

$f(n) = 5$

$f(n) = 5000$

$\Downarrow$

Time Complexity of Order $= O(1)$

If $f(n) = 3n+7$

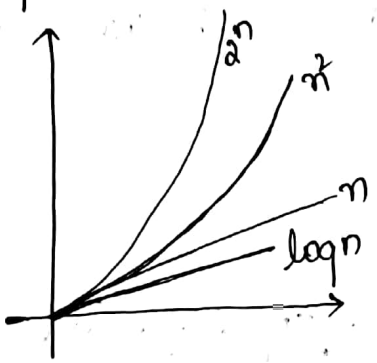
$f(n) = n/3000 + 6$

, still the time Complexity is of Order $n \rightarrow O(n)$

Scanned with CamScanner

* Comparison of function

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



⇒ Nature of the functions.

26/8/20

$O \rightarrow$ Big-O $\rightarrow$ Upper bound

$\Omega \rightarrow$ Big-Omega $\rightarrow$ lower bound

$\theta \rightarrow$ Theta $\rightarrow$ Average bound

Definition:-

1. $O \rightarrow$ Big-O :-

The function $f(n) = O(g(n))$ if and only if $\exists$ +ve constants c & n_0, such that $f(n) \leq c * g(n)$, for all $n \geq n_0$.

Eg:-

$f(n) = 2n+3$ (replace constant Values with linear Values)

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

$f(n) \leq \underset{c}{5}\,\underset{g(n)}{n}$

$f(n) = O(n)$

Similarly, we can write for Quadratic, Cubic, Logrithmic, Exponential.

* Should take most recommended Closest bound Values to write.

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

$f(n) = O(n^3)$.

Scanned with CamScanner

2. $\Omega \rightarrow$ Omega :-

The function $f(n) = \Omega(g(n))$ if and only if $\exists$ +ve c, and n_0 such that $f(n) \geq c * g(n)$, for all value $n \geq n_0$

Ex:-

$f(n) = 2n+3$

$$\underset{f(n)}{2n+3} \geq \underset{c}{1} \times \underset{g(n)}{n}, \text{ for } \forall\, n \geq 1$$

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

$$\underset{f(n)}{2n+3} \geq \underset{c}{1} \times \underset{g(n)}{\log n}, \text{ for } \forall\, n \geq 1$$

$\rightarrow f(n) = \Omega(\log n)$

* If we take $2n+3 \geq n^2$, the statement is not true for $n \geq 4$, so, it is not allowed to use Omega notation for the function n^2.

3. $\theta \rightarrow$ theta :- (Tide function)

The function $f(n) = \theta(g(n))$ if and only if $\exists$ +ve c_1, c_2 and n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

Eg:- $f(n) = 2n+3$

$$\underset{c_1}{1} * \underset{g(n)}{n} \leq \underset{f(n)}{2n+3} \leq \underset{c_2}{5} * \underset{g(n)}{n}$$

$f(n) = \theta(n)$

Average Bound.

Scanned with CamScanner

Ex:-

1. $f(n) = 2n^2+3n+4$

$$2n^2+3n+4 \le 2n^2+3n^2+4n^2$$

$$f(n) \le \underset{c}{\underset{\downarrow}{9n^2}} \mapsto g(n)$$

$$f(n) = O(n^2) \rightarrow \text{Upper bound}$$

$$1 \times n^2 \le 2n^2+3n+4$$

$$1 * n^2 \le f(n)$$

$$f(n) = \Omega(n^2) \rightarrow \text{Lower bound}$$

$$1 \times n^2 \le 2n^2+3n+4 \le 9 \times n^2$$

$$1 * n^2 \le f(n) \le 9 * n^2$$

$$f(n) = \Theta(n^2) \rightarrow \text{Average bound.}$$

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

$$\underset{\downarrow}{1 * n^2 \log n} \le \underset{\downarrow}{n^2 \log n + n} \le \underset{\downarrow}{10 \times n^2 \log n} \quad \forall\, n \ge 1$$

$$O(n^2 \log n) \qquad f(n) \qquad \Omega(n^2 \log n)$$

$$\Rightarrow O(n^2 \log n),\ \Omega(n^2 \log n),\ \Theta(n^2 \log n)$$

3. $f(n) = n!$

$$1 \times 1 \times 1 \times 1 \times \ldots \le 1 \times 2 \times 3 \times \ldots n \le n \times n \times n \times \ldots n$$

$$1 \le n! \le n^n$$

$$\Rightarrow \Omega(1),\ O(n^n) \rightarrow O(n^n)$$

$\downarrow$ lower bound $\qquad \downarrow$ Upper bound

* As the two function of upper bound and lower bound are not equal we cannot write the Θ notation of Time Complexity.

Scanned with CamScanner

4. $f(n) = \log(n!)$

$$\log(1) \le \log(n!) \le \log(n^n)$$

$$0 \le \log(n!) \le n \log n.$$

As we got the time complexity of the lower bound as 0, but the time complexity of the lower bound should not be less than one. So, we take $\Omega(1)$ even if we get 0.

$\therefore$ Time Complexity $= \Omega(1), O(n \log n)$ //.

28/8/20

Properties of Asymptotic Notation :-

General Properties:

If $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$

Eg:- $f(n) = 2n^2 + 5$ is $O(n^2)$

then $7 f(n) = 7(2n^2 + 5)$

$= 14n^2 + 35 = O(n^2)$

This property is true for Ω, θ also.

Reflexive Property:

If $f(n)$ is given, then $f(n)$ is $O(g(n))$

Eg:- $f(n) = n^2 \rightarrow O(n^2)$

This property hold true for $\Omega(f(n))$

Transitive Property:

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$

then $f(n) = O(h(n))$

Scanned with CamScanner

Eg:- $f(n) = n$, $g(n) = n^2$, $h(n) = n^3$

n is $O(n^2)$ and n is $O(n^3)$

then n is $O(n^3)$.

Symmetric :-

If $f(n)$ is $\theta(g(n))$ then $g(n)$ is $\theta(f(n))$

Eg:- $f(n) = n^2$, $g(n) = n^2$

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

$g(n) = \theta(n^2)$

Transpose Symmetric :-

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

Eg:- $f(n) = n$, $g(n) = n^2$

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

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

Exp:-

1. If $f(n) = O(g(n))$, $d(n) = O(e(n))$ then $f(n) + d(n)$

$f(n) = n$, $d(n) = n^2$

$n + n^2 \rightarrow O(n^2)$

$= O(\max(g(n), e(n)))$

2. If $f(n) = O(g(n))$, $d(n) = O(e(n))$ then $f(n) * d(n)$

$f(n) = n$, $d(n) = n^2$

$f(n) * d(n) = n * n^2$

$= n^3 \rightarrow O(n^3)$

$\therefore$ The time Complexity is Order of (n^3)

$T(n) = O(n^3)$ //.

Scanned with CamScanner

Best, Worst and Average Case Analysis

Linear Search

A

8	5	13	6	10	7	2	13	12	11

Best case → Searching a key element Present at first index

Best case Time — 1 $O(1)$

$B(n) = 1 = O(1)$

Worst case → Searching a key element Present at last index

worst case Time = n

$w(n) = n = O(n)$

$$\text{Average case} = \frac{\text{all possible case time}}{\text{no. of cases}}$$

$$\text{Avg. time} = \frac{1+2+3+\cdots n}{n}$$

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

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

Asymtotic Representation

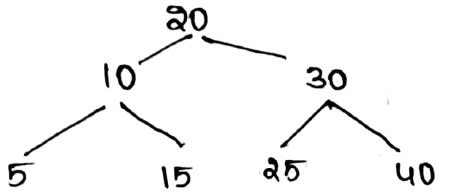
$B(n) = O(1)$
$= \Omega(1)$
$= \Theta(1)$

$w(n) = n$
$= O(n)$
$= \Omega(n)$
$= \Theta(n)$

$A(n) = n$
$= O(n)$
$= \Omega(n)$
$= \Theta(n)$

Scanned with CamScanner

Binary Search Tree :- (Time complexity = log(n))



Best case – searching the root element

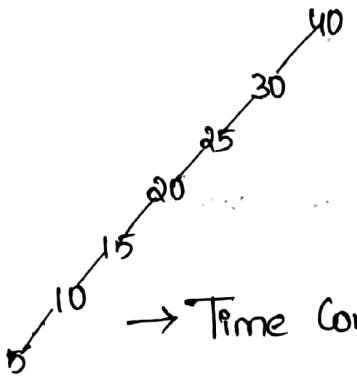
Best case time → B(n) = 1

worst case time → searching the leaf element

worst case time = w(n)
= height of Tree

min w(n) = log n

max w(n) = n



→ Time Complexity = n

B(n) = O(1), A(n) = log n, w(n) = O(n)

* Algorithm are of two types

1. Iterative
2. Recursive

Iterative

```
Algo ( )
{
  for (i=0; i<n; i++)
    {
      statement
    }
}
```

⇓

looping Programs

Recursive.

```
Algo ( )
{
  if ( )
    A= n/2
}
```

⇓

Recursive Program

Scanned with CamScanner

```
* A( )
  {
    i=1, s=1
    While (s<=n)
      {
        i++
        s=s+i
        statement
      }
  }
```

i	s	no. of times
1	1	1
2	1+1	
3	1+1+2	
4	1+1+2+3	
⋮		
K	1+1+2+3+---K	

$$s \geq n$$

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

$$K^2 = n$$

$$K = \sqrt{n}$$

$\therefore$ Time Complexity is of Order $= O(\sqrt{n})$

Scanned with CamScanner

27/8/20

Recursive Analysis

1. Tree method
2. Back substitution method.

Ex:-

```
* Test (n)          → T(n)
  {
    if (n>0)        → 1
    {                   } → Constants
      statement     → 1
      test (n-1)    → T(n-1)
    }
  }
```

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

$= 1 + T(n-1) \quad n>0$

$T(n) = 1 \quad n=0$

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

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

$T(n-2) = T(n-3) + 1$ and so, on

Substitute the value of $T(n-1)$ in $T(n)$

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

$T(n) = [T(n-3)+1] + 1 + 1$ and so, on

at k times assume the loop breaks

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

(Assumed) $n-k = 0 \Rightarrow n = k$

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

$T(n) = 1 + n \Rightarrow T(n) = O(n)$ //.

Scanned with CamScanner

```
* Test (n)          ---> T(n)
  {
    if (n>0)        --> 1
    {
      for (i=0; i<n; i++)  --> n+1
      {
        statement  --> n
      }
      Test (n-1)  --> T(n-1)
    }
  }
```

Equation $= T(n) = T(n-1) + (2n+2)$ (because time complexity is $O(n)$)
↳ replace it with n

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

$$\therefore T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+n & n>0 \end{cases}$$

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

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

$$T(n-2) = T(n-3) + n - 2$$ and so continues.

Substitute the value of $T(n-1)$ in $T(n)$

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

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

$\vdots$

for k^{th} term,

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

assume, $n-k=0$

$n=k$

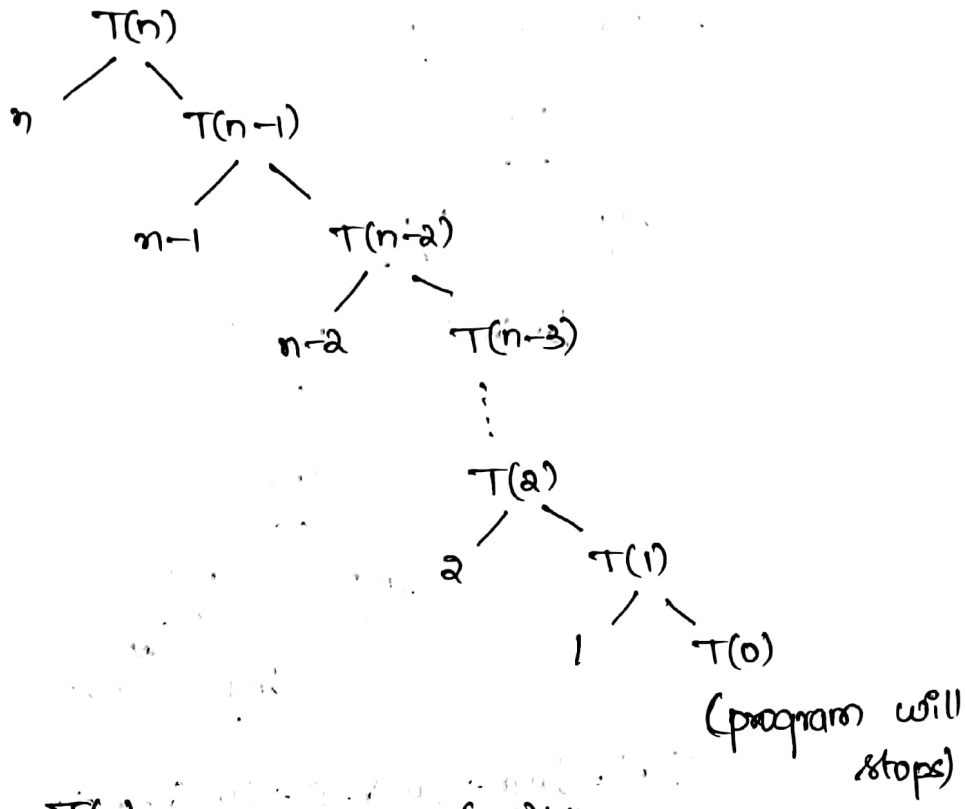
$$T(n) = T(0) + 1 + 2 + 3 + \cdots (n-1) + n$$

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

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

Scanned with CamScanner

*



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

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

$\Downarrow$

Tree method.

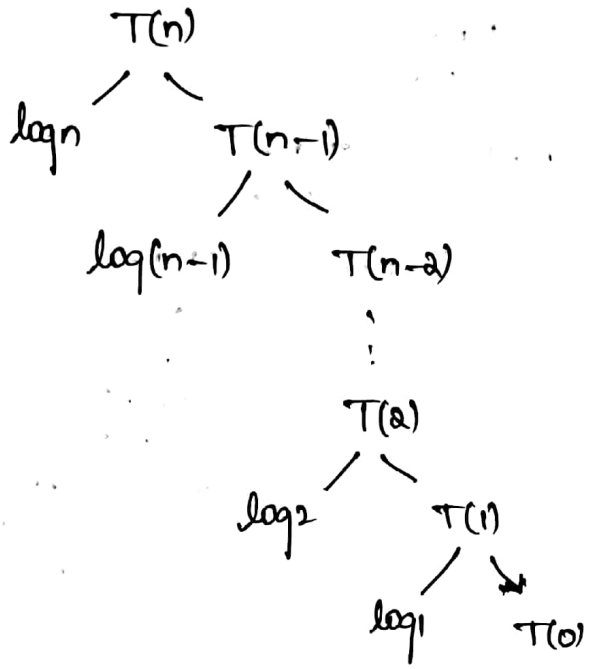
```
* Test (n)          → T(n)
  {
    if (n>0)        → 1
    {
      for (i=1; i<n; i=i*2)
      {
        statement
      }                      → logn
      Test (n-1)    → T(n-1)
    }
  }
```

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

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + \log n \end{cases}$$

Scanned with CamScanner

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



$T(n) = \log n + \log(n-1) + \log(n-2) + \cdots \log 2 + \log 1$

$T(n) = \log(n * (n-1) * (n-2) * \cdots 1)$

$= \log(n!) = O(\log n^n)$

$= O(n \log n)$ $\Rightarrow$ Back substitution method.

Summary:-

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

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

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

Similarly,

$T(n) = T(n-1) + n^2 \longrightarrow O(n^3)$

If $T(n) = T(n-2) + 1 - n/2 \rightarrow O(n)$

$T(n) = T(n-100) + n \longrightarrow O(n^2)$

It may be any value still the Order of time Complexity $= O(n^2)$

Scanned with CamScanner

4/9/20

Master's theorem :-

The Time Complexity $T(n) = aT(n-b) + f(n)$

where $a>0$, $b>0$ and $f(n) = O(n^k)$

where $k \geq 0$.

Case :-

1. if $a=1 \rightarrow O(n \times f(n))$
2. if $a>1 \rightarrow O(a^{n/b} \times f(n))$
3. if $a<1 \rightarrow O(f(n))$

Recurrence Relation for Dividing function :-

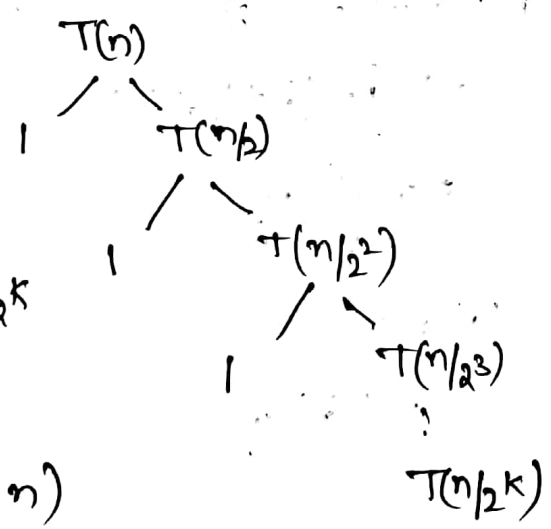
```
Algo (n)
{
  if (n>1)
  {
    statement
    Algo (n/2)
  }
}
```

Recurrence relation,

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

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

Tree method:



$n-k=0$

$T(0)=1$

assume,

$n/2^k = 1$, $n=2^k$

$k = \log n$

$= O(\log n)$

Scanned with CamScanner

By substitution,

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

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

$$T(n) = T(n/2^3) + 1 + 1 + 1$$

$$\vdots$$

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

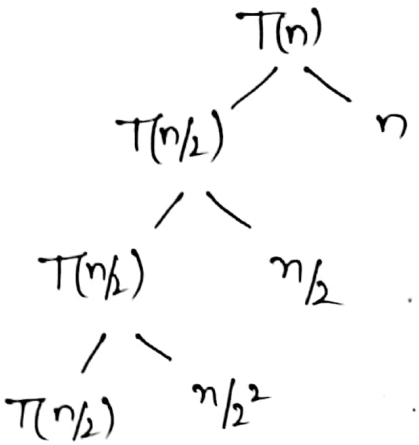
assume, $n/2^k = 1$

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

$$= O(\log n)$$

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

Tree method:-



$$\vdots$$

$$T(n) = n + n/2 + n/2^2 + \cdots n/2^k$$

$$= n\left[1 + 1/2 + 1/2^2 + \cdots 1/2^k\right]$$

$$= n \sum_{i=0}^{k} 1/2^i$$

$$T(n) = n$$

$$= O(n)$$

Scanned with CamScanner

```
* Algo (n)
  {
    if (n>1)
    {
      for (i=0; i<n; i++)
      {
        statement
      }
      Algo (n/2)
      algo (n/2)
    }
  }
```

5/9/20

Master theorem for Dividing functions

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

$a \geq 1$
$b > 1$ $\quad f(n) = \theta(n^k \log^p n)$

Case 1:- if $\log_b^a > k$ then $\theta(n^{\log_b^a})$

Case 2:-

If $\log_b^a = k$

if $p > -1$, then $\theta(n^k \log^{p+1} n)$

if $p = -1$, then $\theta(n^k \log \log n)$

if $p < -1$, then $\theta(n^k)$

Case 3:-

If $\log_b^a < k$

if $p \geq 0 \Rightarrow \theta(n^k \log^p n)$

if $p < 0 \Rightarrow \theta(n^k)$ //.

Scanned with CamScanner

* $T(n) = 2T(n/2)+1$

By Comparing with $T(n) = a(T(n/b)) + f(n)$

$a = 2,\ b = 2$

$f(n) = O(1)$

$= O(n^0 \log^0 n)$

$K=0,\ p=0$

$\log_b^a \rightarrow 1 \Rightarrow K \rightarrow 0$

$\therefore$ Time Complexity $= \Theta(n \log_b^a)$

$= \Theta(n)$ //

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

By Comparing with $T(n) = a(T(n/b)) + f(n)$

$a=4, b=2,\ f(n) = n$

$= O(n^1 \log^0 n)$

$K=1,\ p=0$

$\log_b^a \rightarrow \log_2^4 \rightarrow 2 > K$

Time Complexity $= \Theta(n \log_b^a)$

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

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

By comparing with $T(n) = a(T(n/b)) + f(n)$

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

$= O(n^1 \log^0 n)$

$\log_b^a = 1,\ K=1,\ p=0$ (which is greater than -1)

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

Time Complexity $= \Theta[(n^K) \times \log^p(1/n)]$

$= \Theta[(n) \times \log^0 n]$

$= \Theta[n \log^0 n] = \Theta(n)$

Scanned with CamScanner

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

By comparing with $T(n) = a[T(n/b)] + f(n)$

$a = 2,\ b = 2,\ n = 1,\ p = -1$

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

$k = 1,\ p = -1$

$\log_b^a = 1 = k$

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

* $T(n) = 2T(n/2) + n/\log^2 n$

By comparing with $T(n) = a[T(n/b)] + f(n)$

$a = 1,\ b = 1,\ k = 1,\ p = -2 < -1$

$\log_b^a = 1 = k$ $\quad f(n) = n/\log^2 n$

Time Complexity $= \Theta(n^k)$

$= \Theta(n)$

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

By comparing with $T(n) = a[T(n/b)] + f(n)$

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

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

$= O(n^2 \log^0 n)$

$k = 2,\ p = 1$

$\log_b^a = 1$ (which is less than k)

and p is greater than 0

$\therefore$ Time Complexity $= \Theta(n^k \log^p n)$

$= O(n^2 \log n)$ //.

Scanned with CamScanner

Summary:-

Case 1:-

$T(n) = 2T(n/2) + 1 \rightarrow \theta(n)$

$T(n) = 4T(n/2) + 1 \rightarrow \theta(n^2)$

$\theta(n^{\log_b^a}) = \log_b^a = 2 \Rightarrow \theta(n^2)$

$T(n) = 4T(n/2) + n \rightarrow \theta(n^2)$

$T(n) = 8T(n/2) + n^2 \rightarrow \theta(n^3)$

Case 2:-

$T(n) = T(n/2) + 1 \rightarrow \theta(\log n)$

$\boxed{\log_b^a = K}$

$T(n) = 2T(n/2) + n \rightarrow \theta(n \log n)$

$= 2T(n/2) + n \log n \rightarrow \theta(n \log^2 n)$

19/20

Scanned with CamScanner

9/9/20

Root function:-

(Analysis of root function)

Ex:-

```
* Algo(n)        → T(n)
  {
    if (n>2)
    {
      statement        → 1
      Algo(√n)         → T(√n)
    }
  }
```

$$T(n) = T(\sqrt{n}) + 1$$

$n=0 \rightarrow$ decrementing function

$n=1 \rightarrow$ dividing functions

$n=2 \rightarrow$ Root function

$T(0)=1, \quad T(1)=1, \quad T(2)=1 \rightarrow$ root function.

$$T(n) = \begin{cases} 1 & n=2 \\ T(\sqrt{n})+1 & n \geq 2 \end{cases}$$

$$T(n) = T(n^{1/2}) + 1 \rightarrow (1)$$

$$T(n^{1/2}) = T(n^{1/4}) + 1 \rightarrow (2)$$

$$T(n^{1/4}) = T(n^{1/8}) + 1 \rightarrow (3)$$

$$\vdots$$

$$T(n^{1/2^k}) = T(n^{1/2^{k+1}}) + 1 \rightarrow (4)$$

By substituting (2), (3) ... (4) inequation (1)

$$T(n) = T(n^{1/2^2}) + 1 + 1$$

$$T(n) = T(n^{1/2^3}) + 1 + 1 + 1$$

$$\vdots$$

$$T(n) = T(n^{1/2^k}) + k \rightarrow (5)$$

Scanned with CamScanner

Assume, $n = 2^m$

Now, put this n Value in eq. 5

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

Assume $m/2^k = 1 \Rightarrow m = 2^k$

$$k = \log_2 m$$

$$n = 2^m \Rightarrow m = \log_2 n$$

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

$\therefore$ The complexity $= O(\log(\log n))$ //.

Binary Search :-

(Iterative method)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	3	6	8	12	14	17	25	29	31	36	42	47	53	55	62

↓ mid position.

↑ l ↑ h

Key = 42

l	h	mid
1	15	$\frac{l+h}{2} = 8$

* If key > mid then increment the lower point to mid + 1 $\Rightarrow l = mid + 1$

Key > mid Value
$l = mid + 1$

l	h	mid
1	15	$\frac{l+h}{2} = 8$
9	15	$\frac{9+15}{2} = 12$

* If key value < mid value then the higher Point to mid − 1 $\Rightarrow h = mid - 1$

l	h	mid
9	15	12
9	11	$9+11/2 = 10$

Scanned with CamScanner

Now,

$l = mid + 1 = 11$, $h = 11$, $mid = \frac{11+11}{2} = 11$

$\therefore$ the 11th position is Our required KeyValue.

(Key = 12)

l	h	M
1	15	8

As key < mid $\Rightarrow$ $h = mid - 1 = 7$

$1 \quad 7 \quad \frac{1+7}{2} = 4$

$\therefore$ The 4th position is the key = 12.

(Key = 30) (Where this key is not present in array)

l	h	M
1	15	8
9	15	12
9	11	10
9	9	9
9	8	

(l > h)

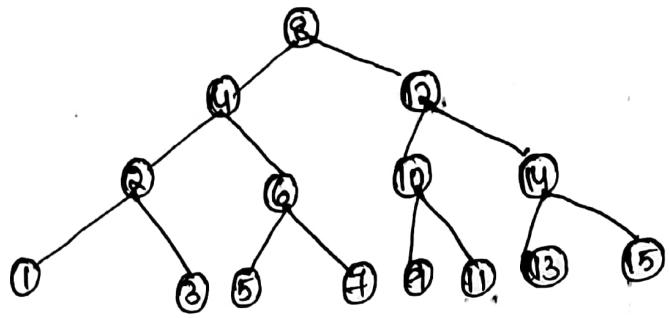
(Element not found)

```
* int (BinSearch (A, n, Key))
  {
    l = 1 ; h = n
        mid = (l+h)/2
        if (Key = A[mid])
          return mid.
        if (Key < A[mid])
          h = mid - 1
        else
          l = mid + 1
    }
    return 0;
  }
```

Scanned with CamScanner

10/9/20

Binary Search tree for mid Values of the above



The height of the tree = $\log_2^n$

No: of Comparisons = $\log_2^n$

Best case Complexity

Key = MID → O(1)

Root of tree

Worst case Complexity.

Exactly equal to height of tree → $O(\log_2^n)$

* If the tree is skewed tree then the time Complexity is Order of $n \rightarrow O(n)$

Recursive Method:- (Binary Search)

```
RBS (l, h, Key)
  {
    if (l==h)
      { if (A[l] == Key)
          return l;
        else
          return 0;
      }
    else
      mid = (l+h)/2
      { if (Key == A[mid])
          return mid
```

Scanned with CamScanner

```
if (key < A [mid])
    return (RBS (l, mid-1, key))
else
    return (RBS (mid+1, h, key))
}
}
```

Complexity :-

16/9/20

Two way merge sort :-

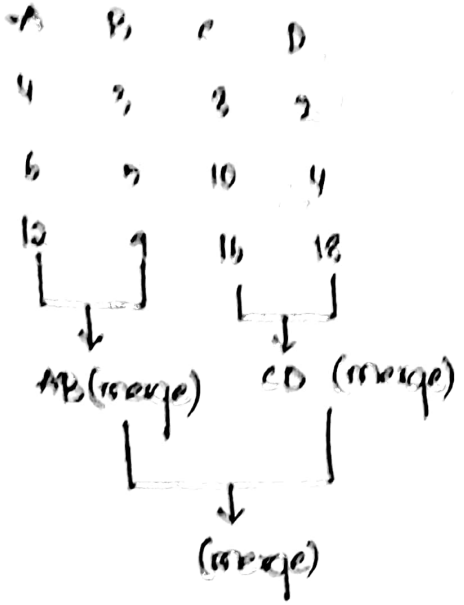
A	B	C
2	5	2
8	9	5
15	12	8
18	17	9
(m)	(n)	12
		15
		17
		18

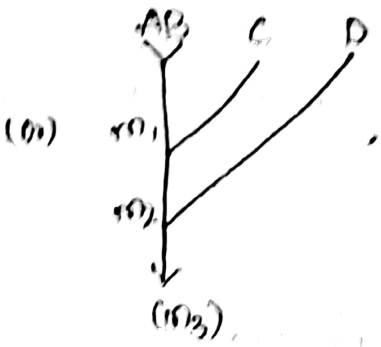
$i \rightarrow$ 8, 15, 18 ; $j \rightarrow$ 9, 12, 17

$\therefore$ Time Complexity $= O(m+n)$

```
Merge (A,B,m,n)
{
  i=1, j=1, k=1;
  if (A[i] < B[j])
    C[k++] = A(i++);
    else
    C(k++) = B(j++);
  for (i=0; i<=m; i++)
    C(k++) = A[i]
  for (j=0; j<=n; j++)
    C(k++) = B[j]
}
```

Scanned with CamScanner

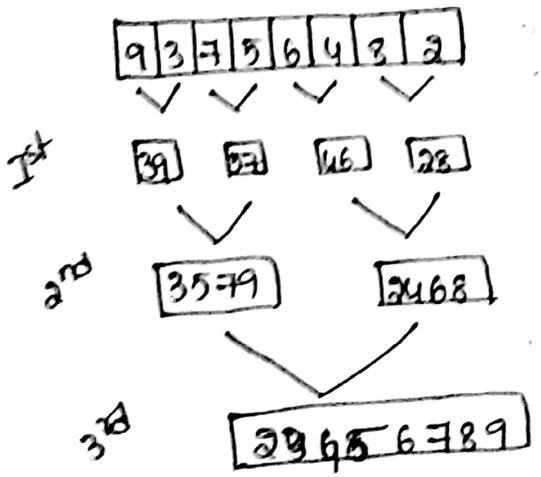




* We can merge more than 2 arrays in n different ways.

Ex:- A [9|3|7|5|6|4|8|2]

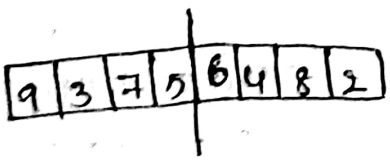
Assume each element as an array of single element.



Analysis:-

We have array of 8 elements

Now, divide



$8/2 \rightarrow 4, 4$

$4/2 \rightarrow 2, 2$

$2/2 \rightarrow 1, 1$

$8 = 2^3$

$3 \rightarrow$ no: of passes

$T(n) = \log_2 n \times n$

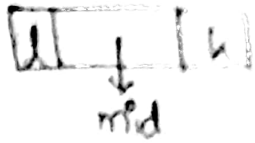
$= n \log_2 n$

Scanned with CamScanner

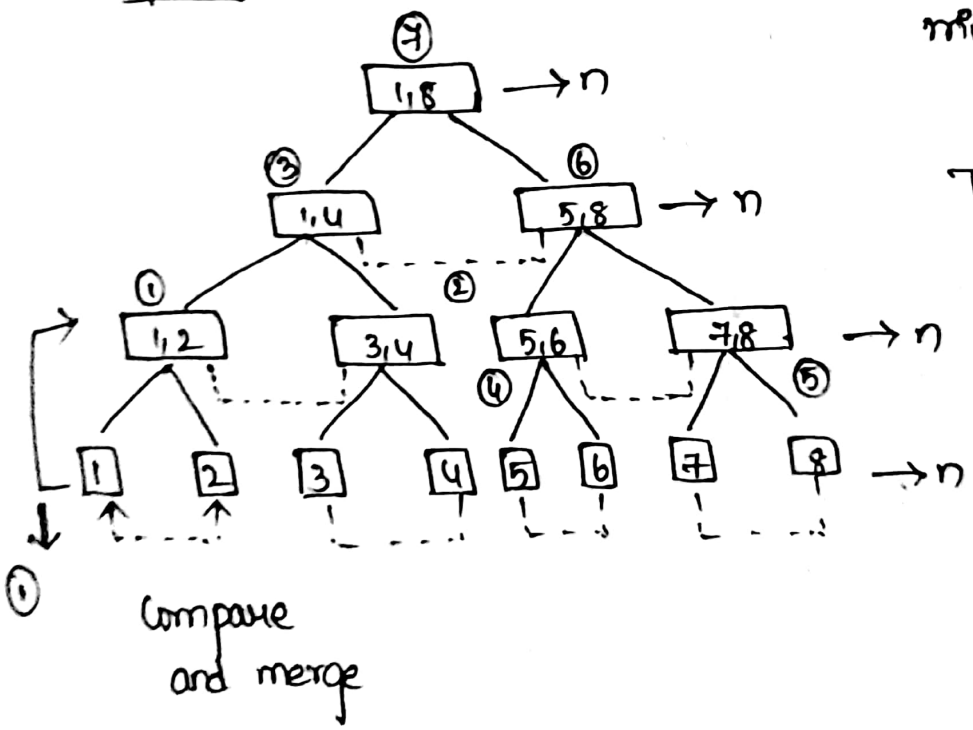
Algorithm :-

```
Merge Sort (a, l, h)
{ if (l < h)
  {
    mid = (l+h)/2

    merge sort (l, mid)  → T(n/2)
    merge sort (mid+1, h) → T(n/2)
    merge (l, mid, h) → n
  }
}
```



Explanation:-



$mid = \frac{8+1}{2} = 4.5$

These are the index positions of the array

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

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

//.

Scanned with CamScanner

18/9/20

* For extracting the elements from large data set, it will take huge amount of time. So sorting is required.

For unsorted data it will take 2^{60} sec

For sorted data it will take $\log_2 2^{60} = 60$ sec.

Prons and Cons of Merge sort :-

1. Prons :-

a) Insertion, Selection and bubble sorting.

these have time complexity $O(n^2)$

Merge sort has $n \log n$:

time complexity $O(n \log n)$ has more efficient algorithm than $O(n^2)$.

b) External Sorting.

c) Stable Sorting

8, 6, 4, 3, 6, 5, 9

↓

8, 4, 3, 5, 6, 6, 9

2. Cons :-

a) It takes extra space, not inplace sorting

b) No small problem defined.

c) Merge sort is slow algorithm (Recursive method)

Space = Complexity = $n + \log n$

$= O(n)$ //.

Scanned with CamScanner

Quick sort:-

6, 3, 5, 4, 2, 1, 9, 10, 12

↓

Pivot element

All the elements less than or equal to pivot elements will fall left of pivot element and greater elements will fall right.

3, 5, 4, 2, 1, (6), 9, 10, 12

19/9/20

↑ lower pointer ↑ higher pointer

→ 10, 16, 8, 12, 15, 6, 3, 9, ∞

i j

$\boxed{\text{Pivot}=10}$ End of list

* i → find the element greater than pivot element
* j → find the element lesser than pivot element
* Interchange the values present at ith location and jth location (swap).
* Repeat the procedure untile i<j
* The last location j is interchange with the pivot element.

Ex:- 10, 16, 8, 12, 15, 6, 3, 9, 16, ∞

i j

Pivot = 10

16>5

Interchange 10, 5, 8, 12, 15, 6, 3, 9, 16, ∞

i i j

Interchange 10, 5, 8, 9, 15, 6, 3, 12, 16, ∞

i j

Scanned with CamScanner

Interchange 10, 5, 8, 9, 31 6, 15, 12, 16, ∞

i > j

i > j and loop breakes.

last location interchanges with pivot element.

5, 8, 9, 3, 10, 15, 12, 16, ∞

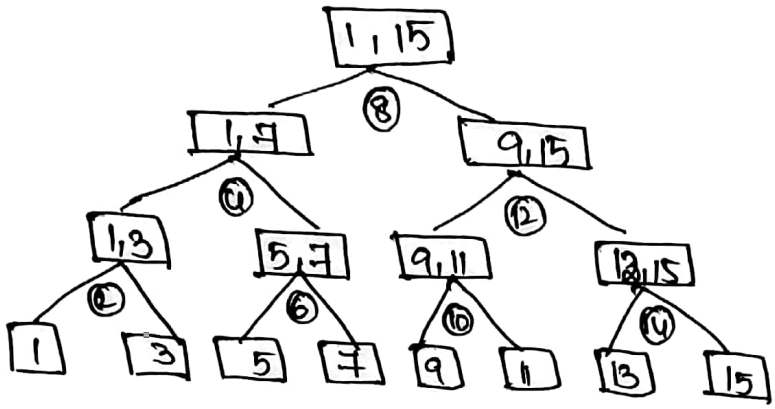
Partition Algorithm :-

```
Partition (l, h)
  {
    Pivot = A[i]
    i = l, j = h
      do
        { i++;
        } while (A[i] ≤ pivot)

      do { j--;
         } while (A[j] > pivot)
    if (i<j)
      {
        Swap (A[i], A[j])
      }
      Swap (A[i], A[j]);
      return j;
    }
  Quick sort (l, h)
    if (l<h)
    { j = partition (l, h);
      Quick sort (l, j);
      Quick sort (j+1, h);
    }
```

Scanned with CamScanner

Analysis of Quick Sort:-



8, 4, 12, 2, 6, 10, 14 are Pivot elements.

$\log n \rightarrow n + \log n$

* Best case (or) Average Case analysis of quick Sort.

Scanned with CamScanner