ALGORITHMS

1. INTRODUCTION TO ALGORITHMS

① Characteristics of Algorithm:-

Priori Analysis	Posteriori Testing
1. Algorithm	1. Program.
2. Independent of Language.	2. Language dependent
3. Hardware Independent.	3. Hardware dependent
4. Time & Space Function.	4. Watch time & bytes.

② How to write an algorithms:-

```
algorithm swap(a, b) {
    temp = a
    a = b
    b = temp.
}
```

③ How to analyze an algorithm:-

1. Time (as a function)
2. Space (as a function)
3. N/w.
4. power.
5. CPU Registers.

```
algorithm swap(a, b) {          Space
    temp = a ——— 1              a — 1
    a = b ——— 1                 b — 1
    b = temp ——— 1              temp — 1
}
f(n) = 3.                       S(n) = 3
O(1)                            O(1).
```

④ Frequency count method :-

① Sum in array

```
Algorithm sum (A, n) {                                  Space
    S=0 ——————————— 1                                   A — n
    for(i=0; i<n; i++) { ——— n+1 (precisely 2n+2)       n — 1
        S = S+A[i]; ——————— n                           S — 1
    }                                                   i — 1
    return S; ——————————— 1                             ———
}                                                       n+3
f(n) = 2n+3 ⇒ O(n)                                      S(n) = n+3 ⇒ O(n)
```

Eg ② Sum of two matrices.

```
Algorithm Add (A, B, n) {
    for (i=0; i<n; i++) {                    ——— n+1
        for (j=0; j<n; j++) {                ——— (n+1)*n
            c[i,j] = A[i,j] + B[i,j];        ——— n*n
        }
    }
}
```

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

$O(n^2)$

Space

A — n^2
B — n^2
C — n^2
n — 1
i — 1
j — 1

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

$O(n^2)$

Eg ③ Multiplication of matrices

```
Algorithm Multiply (A, B, n) {
n+1 ——— for (i=0; i<n; i++) {
(n+1)n ——— for (j=0; j<n; j++) {
n×n ——— c[i,j] = 0;
(n+1)×n×n ——— for (k=0; k<n; k++) {
n×n×n ——— c[i,j] = c[i,j] + A[i,k] * B[k,j];
    }
}
}
}
```

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

$O(n) = n^3$

Space

A — n^2
B — n^2
C — n^2
i — 1
j — 1
n — 1
k — 1

$S(n) = 3n^2 + 4$

$O(n^2)$.

⑤ Time Complexity #1

① `for (i=1; i<n; i++) {` `i+=2` `i+=20`

`stmt;` ——— n ↓ $\frac{n}{2}$ ↓ $\frac{n}{20}$

`}`

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

②
```
for(int i=0; i<n; i++) {          ——— n+1
    for(int j=0; j<n; j++) {      ——— n(n+1)
        stmt;                     ——— nn
    }
}                                 ———
                                  O(n²).
```

③
```
for(int i=0; i<n; i++) {
    for(int j=0; j<i; j++) {
        stmt;
    }
}
```

i	j	no. of times
0	0 ✗	0
1	0 ✓	1
	1 ✗	
2	0 ✓	2
	1 ✓	
	2 ✗	
3	0	3
	1	
	2	
	3	
⋮		⋮
n		n

$1+2+3+\ldots\ldots+n = \frac{n(n+1)}{2}$

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

$O(n^2)$.

④
```
p=0;
for(i=1; p<=n; i++) {
    p = p+i;
}
```

Assume $p > n$ (to the loop stops).

i	p
1	0+1 = 1
2	1+2 = 3
3	1+2+3 = 6
4	1+2+3+4
⋮	⋮
k	1+2+3+4+ ....+k

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

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

$k^2 > n$

$k > \sqrt{n}$.

$O(\sqrt{n})$

⑥ Time Complexity #2:-

① `for (i=1; i<n; i = i+2) {`
`stmt;`
`}`

Assume $i >= n$

$\therefore i = 2^k$

$2^k >= n$.

$k = \log_2 n$

$O(\log_2 n)$

i
1
2
$2^2 = 4$
$2^3 = 8$
⋮
2^k

(where the loop repeated k times).

Another way:-

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

[$1 \times 2 \times 2 \times 2 \times \ldots = n$

$2^k = n$

$k = \log_2 n$]

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

[$i = 1+1+1+\ldots+1$ (k times) $= n$.

$\therefore k = n$.]

When using log, obtain ceil value:

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

n=8

i	
1	3
2	
4	
8 X	

$\log_2 n = \log_2 8$

$= 3$

n=10

i	
1	4
2	
4	
8	
16 X	

$\log_2 n = \log_2 10$

$= 3.2$

≈ 4

ceil value.

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

i
n
n/2
n/2^2
n/2^3
⋮
n/2^k

Assume i<1

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

$n < 2^k$

$k = \log_2 n$

$O(\log_2 n)$.

③ for(i=0; i*i<n; i++) {
    stmt;
}

i*i < n.

$i^2 < n$

$i = \sqrt{n}$

$O(\sqrt{n})$

④ for (i=0; i<n; i++) {
    stmt;
} — n

for (j=0; j<n; j++) {
    stmt;
} — n

2n

O(n)

(10) (5)

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

p++; ——— log n.

```
for (j = 1; j < p; j = j * 2) {
    stmt;
}
```

stmt; ——— log p.

log (log n).

O(log (log n)).

(11) (6)

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

for (i = 0; i < n; i++) { ——— n

for (j = 1; j < n; j = j * 2) { —— n × log n

stmt; ——— n × log n

2n log n + n.

O(n log n)

* for (i = 0; i < n; i++) —— O(n)
* for (i = 0; i < n; i = i + 2) —— O(n)
* for (i = n; i > 1; i--) —— O(n)
* for (i = 1; i < n; i = i * 2) —— $O(\log_2 n)$
* for (i = 1; i < n; i = i * 3) —— $O(\log_3 n)$
* for (i = n; i > 1; i = i/2) —— $O(\log_2 n)$.

(7) Time Complexity of While and if #3.

(1)

```
a = 1;
while (a<b) {
    stmt;
    a = a * 2;
}
```

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

Terminate

$a \geq b$.

$a = 2^k$.

$2^k \geq b$.

$k = \log_2 b$

$O(\log n)$.

```
for (a=1; a<b; a=a*2)
{
    stmt;
}
```

(2)

```
i = n
while (i > 1) {
    stmt;
    i = i/2;
}
```

$O(\log n)$

```
for (i=n; i>1; i--) {
    stmt;
}
```

(3)

```
i = 1;
k = 1;
while (k<n) {
    stmt;
    k = k+i;
    i++;
}
```

i	k
1	1
2	$1+1=2$
3	$2+2=4$
4	$4+3=7$
5	$7+5=12$
⋮	⋮
l	$2+2+3+4+\cdots+l$

$k \geq n$.

$\frac{l(l+1)}{2} \geq n$

$l^2 \geq n$.

$l = \sqrt{n}$.

$2+2+3+4+\cdots+l$

$\frac{l(l+1)}{2}$

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

④ GCD code.

```
for( ; m!=n; ) {
[while(m!=n) {]
    if (m>n) {
        m=m-n;
    } else {
        n=n-m;
    }
}
```

m	n
6	3
3	3

m	n
5	5

m	n	
16	2	
14	2	
12	2	
10	2	
8	2	
6	2	$\left(\frac{16}{2}\right)$
4	2	
2	2	$\frac{n}{2}$

$O(n)$

min = $O(1)$.

⑤ Analysis of if & while:-

```
Algorithm Test(n) {
    if (n<5) {
        printf("%d", n);          ——— 1      O(1) best case
    } else {
        for(i=0; i<n; i++) {
            printf("%d", i);      ——— n      O(n) worst case.
        }
    }
}
```

(8) Classes of functions.

$O(1)$ —— constant

$O(\log n)$ —— logarithmic

$O(n)$ —— linear.

$O(n^2)$ —— quadratic.

$O(n^3)$ —— cubic.

$O(2^n)$ —— exponential.

$O(3^n)$

$O(n^n)$

$f(n) = 2 \rightarrow O(1)$

$f(n) = 5 \rightarrow O(1)$

$f(n) = 5000 \rightarrow O(1)$

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

$f(n) = 500n + 700 \rightarrow O(n)$

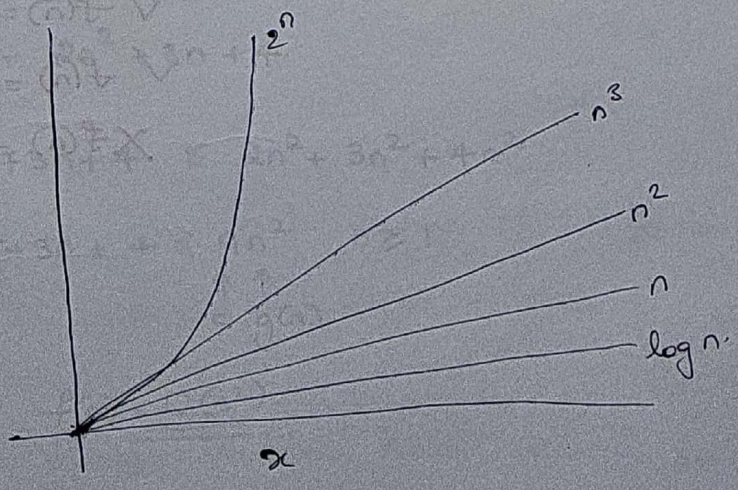
$f(n) = \frac{n}{5000} + 6 \rightarrow O(n)$

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

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4
3	8	64	256
2	4	16	16

slower ← faster.

$$\log n < n < n^2 < 2^n.$$



⑨ Asymptotic Notations Big Oh – Omega – Theta #1.

O big-oh upper bound.

Ω big-omega lower bound.

Θ theta average bound.

Big Oh

The function $f(n) = O(g(n))$ iff $\exists$ +ve constants c and n_0

$\ni$ $f(n) \leq c * g(n)$ $\forall$ $n \geq n_0$.

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

$2n+3 \leq \underbrace{10n}_{c\ g(n)}$ $n \geq 1$ (n_0).

$\therefore f(n) = O(n)$.

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

$2n+3 \leq 5(n)$. $n \geq 1$.
$g(n)$.

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

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

lower bound. avg bound. upper bound

✓ $f(n) = O(n)$ preferable.

✓ $f(n) = O(n^2)$

✓ $f(n) = O(2^n)$

✗ $f(n) = O(\log n)$

Omega

The function $f(n) = \Omega(g(n))$ iff $\exists$ +ve constants c and n_0

$\ni$ $f(n) \geq c * g(n)$ $\forall$ $n \geq n_0$.

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

$2n+3 \geq 1 * n$ $\forall$ $n \geq 1$

($2n+3$: $f(n)$, 1 : c, n : $g(n)$)

preferrable because closest.

$\therefore f(n) = \Omega(n)$.

✓ $f(n) = \Omega(\log n)$.

✓ $f(n) = \Omega(\sqrt{n})$

✗ $f(n) = \Omega(n^2)$.

Theta Notation

The function $f(n) = \Theta(g(n))$ iff $\exists$ +ve constants c_1, c_2 and n_0

$\ni$ $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$.

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

$1 * n \leq 2n+3 \leq 5 * n$.

(c_1, $g(n)$, c_2, $g(n)$)

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

✗ $f(n) = \Theta(n^2)$

Examples :-

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

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

$2n^2+3n+4 \leq 9n^2$ $\quad n \geq 1$

(c, $g(n)$)

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

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

$\Omega(n^2)$.

$1 \times n^2 \leq 2n^2+3n+4 \leq 9n^2$.

$\Theta(n^2)$

② $f(n) = n^2 \log n + n$.

$1 \times n^2 \log n \leq n^2 \log n + n \leq 10 n^2 \log n$.

$O(n^2 \log n)$

$\Omega(n^2 \log n)$

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

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

③ $f(n) = n!$

$f(n) = n! = n * (n-1) * (n-2) * (n-3) \ldots * 3 * 2 * 1$.

$\leq 1.2.3.4 \ldots n. \leq n.n.n.n \ldots n.$

$1.1.1.1.1 \ldots 1$

$1 \leq n! < n^n$.

$\Omega(1)$. $O(n^n)$

For smaller values of n the value of n! will be at 1 and for larger values, it may go upto n^n. So there is no place for it in the order.

If we cannot mention average bound, we will specify upper and lower bounds.

④ $f(n) = \log n!$

$\log(1\times1\times1\times1\ldots\times1) \leq \log(1\times2\times3\times\ldots\times n) \leq \log(n\times n\times n\times\ldots\times n)$

$1 \leq \log n! \leq \log n^n.$

$1 \leq \log n! \leq n\log n.$

$O(1)$ $O(n\log n)$

There is no tight bound.

Upper bound: $O(n\log n)$

Lower bound: $O(1)$.

(10) Properties of Asymptotic Notations:-

General Properties:-

① 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 $7f(n) = 7(2n^2+5)$

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

Same properties apply to other notations as well.

Reflexive:-

If $f(n)$ is given then $f(n)$ is $O(f(n))$.

Eg: $f(n) = n^2$ $O(n^2)$.

Transitive:-

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n) = O(h(n))$.

Eg: $f(n) = n$ $g(n) = n^2$ $h(n) = n^3$ n is $O(n^2)$ & n^2 is $O(n^3)$ then n is $O(n^3)$.

Symmetric :-

If $f(n)$ is $\Theta(g(n))$ then $g(n) = \Theta(f(n))$.

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

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

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

This is true only in case of Θ.

Transpose Symmetric :-

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

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

then n is $O(n^2)$.

n^2 is $\Omega(n)$.

* If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

$g(n) \leq f(n) \leq g(n)$.

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

* If $f(n) = O(g(n))$ and $d(n) = O(e(n))$,
then $f(n) + d(n) = O(\max(g(n), e(n)))$. — [from ① & ②].

Eg:- $f(n) = n = O(n)$.

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

$f(n) + d(n) = n + n^2 = O(n^2)$. — ①

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

$d(n) = n = O(n)$

$f(n) + d(n) = n^2 + n = O(n^2)$ — ②.

* If $f(n) = O(g(n))$ and $d(n) = O(e(n))$,

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

(11) Comparison of Functions # 1.

① $n < n^2 < n^3$

2	$2^2 = 4$	$2^3 = 8$
3	$3^2 = 9$	$3^3 = 27$
4	$4^2 = 16$	$4^3 = 64$.

② $n^2 \quad n^3$

Apply log on both sides

$\log n^2 \quad \log n^3$

$2 \log n < 3 \log n$.

$\log ab = \log a + \log b$.

$\log \frac{a}{b} = \log a - \log b$.

$\log a^b = b \log a$.

$a^{\log_c b} = b^{\log_c a}$

$a^b = n$ then $b = \log_a n$

Eg:- ① $f(n) = n^2 \log n \quad g(n) = n (\log n)^{10}$.

Apply log

$\log [n^2 \log n] \quad \log [n (\log n)^{10}]$

$\log n^2 + \log \log n \quad \log n + \log (\log n)^{10}$

$2 \log n + \log \log n > \log n + 10 \log \log n$.

② $f(n) = 3n^{\sqrt{n}}$ $\quad g(n) = 2^{\sqrt{n}\log n}$.

$3n^{\sqrt{n}}$

$2^{\log_2 n^{\sqrt{n}}}$

$n^{\log_2 2^{\sqrt{n}}} = n^{\sqrt{n}\log_2 2}$

$(n^{\sqrt{n}})^{\log_2 2}$

$3n^{\sqrt{n}} > n^{\sqrt{n}}$. $\Rightarrow$ values wise $3n^{\sqrt{n}} > n^{\sqrt{n}}$.

$3n^{\sqrt{n}} = n^{\sqrt{n}}$ $\Rightarrow$ asymptotically equals

③ $f(n) = n^{\log n}$ $\quad g(n) = 2^{\sqrt{n}}$.

apply log:

$\log n^{\log n}$ $\quad \log_2 2^{\sqrt{n}}$.

$\log n \cdot \log n$ $\quad \sqrt{n}\log_2 2$

$\log^2 n$ $\quad \sqrt{n} = n^{1/2}$

apply log:

$2\log\log n < \frac{1}{2}\log n$.

④ $f(n) = 2^{\log n}$ $\quad g(n) = n^{\sqrt{n}}$.

$\log n \log_2 2$ $\quad \sqrt{n}\log n$

$\log n < \sqrt{n}\log n$

⑤ $f(n) = 2n$ $\quad g(n) = 3n$

asymptotically equal.

(6) $f(n) = 2^n$ $g(n) = 2^{2n}$.

$n \log_2 2$ $2n \log_2 2$.

n $2n$.

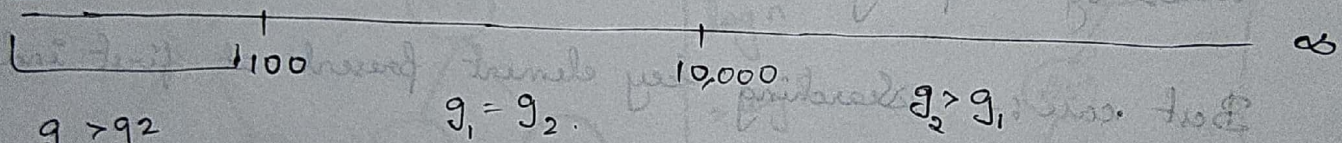
We cannot say they are equal. Because after applying log we arrived at n, $2n$. So $f(n) < g(n)$.

(7)

$$g_1(n) = \begin{cases} n^3 & n < 100 \\ n^2 & n \geq 100 \end{cases}$$

$$g_2(n) = \begin{cases} n^2 & n < 10000 \\ n^3 & n \geq 10000 \end{cases}$$

Ans:-



$g_1 > g_2$ $g_1 = g_2$. $g_2 > g_1$

$g_2 > g_1$ [we will check for all values of n till ∞].

* True or False

(1) $(n+k)^m = \Theta(n^m)$. ✓ $(n+3)^2 = \Theta(n^2)$.

(2) $2^{n+1} = O(2^n)$ ✓ $2 \cdot 2^n = O(2^n)$.

(3) $2^{2n} = O(2^n)$ ✗ $(2^2)^n = 4^n$ $4^n > 2^n$.

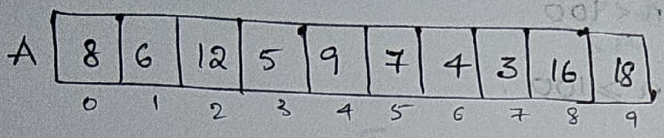
(4) $\sqrt{\log n} = O(\log \log n)$ ✗ Apply log.

(5) $n^{\log n} = O(2^n)$ ✓ $\log n \log n < n$

(12) Best, Worst and Average Case Analysis :-

1. Linear Search.
2. Binary Search Tree.

Linear Search :-



key = 7 → found at 5.

key = 20 → not found

Works by comparing each element with the key.

Best case :- Searching key element present at first index.

Best case Time — $O(1)$

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

Worst case :- Searching key at last index

Worst case Time — $O(n)$.

$$W(n) = O(n).$$

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

Average case time :- $\frac{1+2+3+\cdots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$

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

$B(n) = 1$ $\quad W(n) = n.$

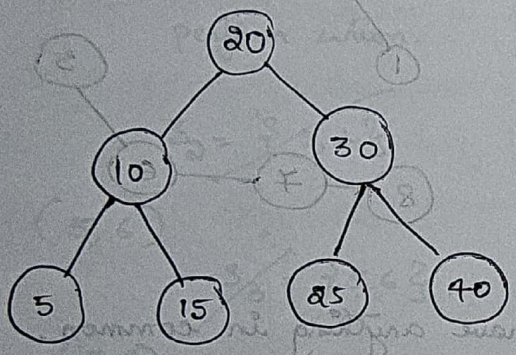
$B(n) = O(1)$ $\quad W(n) = O(n)$

$B(n) = \Omega(1)$ $\quad W(n) = \Omega(n).$

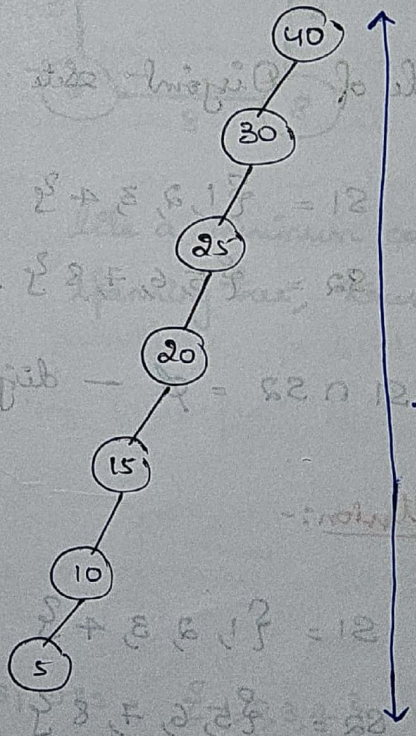
$B(n) = \Theta(1)$ $\quad W(n) = \Theta(n).$

Best case and worst case can be applied on any notation.

* Binary Search Tree :-



$\log n$
$= \log 8$
$= 3.$



Best Case : Search root element.

Best Case Time : $B(n) = 1.$

Worst Case : Search leaf element.

Worst Case Time : $W(n) = \log n.$

$\min W(n) = \log n.$

$\max W(n) = n.$