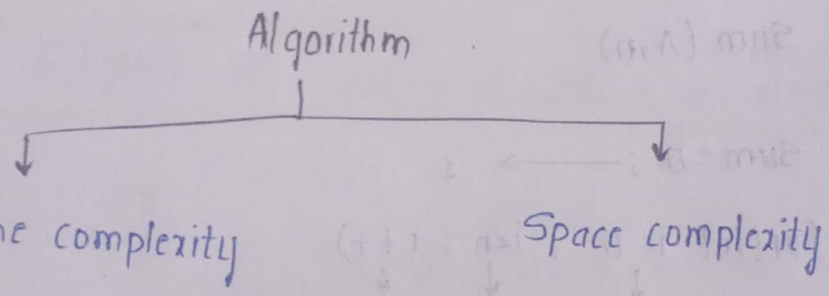
* Algorithm :- Step by step procedure to solve a problem.

* Properties of Algorithm :-

1) Input specified
2) Output specified
3) Definiteness
4) Effectiveness
5) Finiteness



* Space Complexity 3 ~~times~~ types. They are :

1) Instruction Space
2) Data Space
3) Environment Space

* Time Complexity (T(n)) :-

> It should be machine independent.
> We don't count exact time. Just we go with approximation.

Time complexity is further divided into two types :

1) Constant time Complexity

Ex:- Sum of two numbers

```
int sum(a,b)
{
    int sum = a+b;
    return sum;
}
```

→ This whole term is considered as one.

$T(n) = 1$

2) Linear time Complexity

Ex:- Sum of array Elements

```
int Sum (A,n)
{
    Sum = 0 ;  ⟶ 1
    for(int i= 0 ; i<n ; i++)
          ↓      ↓     ↓
          1     n+1   n times
    Sum = Sum + A[i] ; ⟶ n
    return sum ; ⟶ 1
}
```

cost	Repetition
1	1
1	1
1	n+1
1	n
1	n
1	1
	T(n) = O(n) = 3n+4

T(n) = Step count function.

⇒ Time complexity depends on input size.

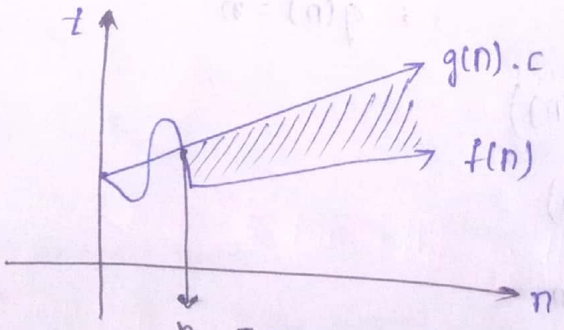
* Asymptotic notations :- 5 types

1) Bigoh [O] { lesser than }
2) Omega [Ω] { Greater than }
3) Theta [Θ] { Equal }
4) Small oh [o] { strictly less than }
5) Small Omega [ω] { strictly greater than }

1]. Bigoh [O] :-

f(n) = Our function ; g(n) = A simple function.

$f(n) = O(g(n))$ iff $f(n) \leq c.g(n)$, $c>0$, $n \geq n_0 \geq 1$



n_0 [Because here it is not satisfying inequality]

Ex:- $f(n) = T(n) = 3n+4$; $g(n) = n$

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

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

$3n+4 \leq c.n$

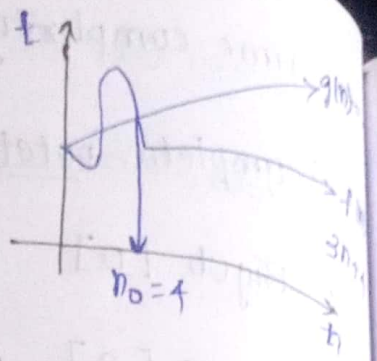
Assume 'c' value that inequality becomes true.

So, $c = 4$

$3n + 4 \leq 4n$

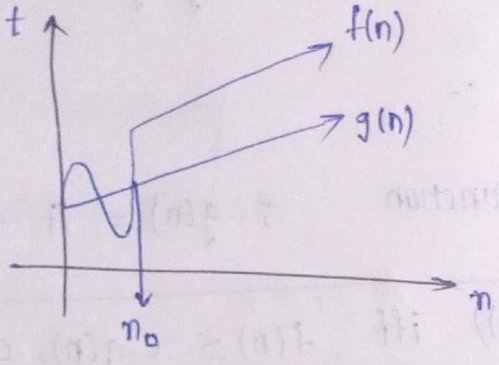
$4 \leq n$

$\boxed{n_0 = 4}$



2) Omega (Ω) :-

$\boxed{f(n) = \Omega(g(n)) \text{ iff } f(n) \geq c.g(n),\ c > 0,\ n \geq n_0 \geq 1}$



Ex:- $f(n) = 3n+4$; $g(n) = n$

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

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

$3n+4 \geq c.n$

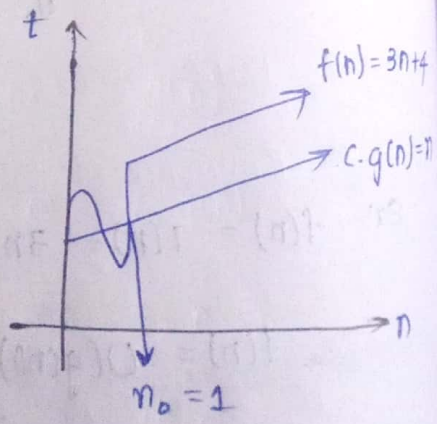
for $c = 1$ $\quad 3n+4 \geq n$

$2n+4 \geq 0$

$n+2 \geq 0$

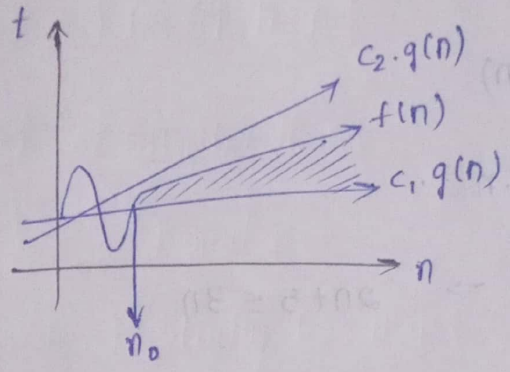
$n \geq -2 \geq 1$

$\boxed{n_0 = 1}$



3) Theta (Θ) :-

$f(n) = \Theta(g(n))$ iff $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$, $c_1, c_2 > 0$, $n \ge n_0 \ge 1$



Ex:- $f(n) = T(n) = 3n+4$; $g(n) = n$

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

$c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$

$c_1 \cdot g(n) \le f(n)$

$c_1 \cdot n \le 3n+4$

$n \le 3n+4$

for $c_1 = 1$, $n_0 = 1$

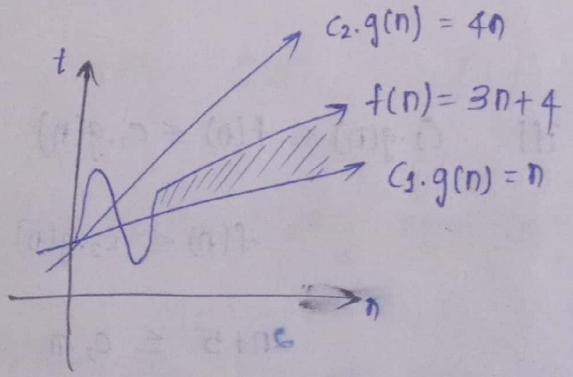
$f(n) \le c_2 \cdot g(n)$

$3n+4 \le c_2 \cdot n$

$3n+4 \le 4n$

for $c_2 = 4$, $n_0 = 4$

$T(n) = 3n+4 = \Theta(n)$



Note:- $3n+4$ is example for $O(n)$, $\Omega(n)$, $\Theta(n)$.

H/w :- $T(n) = 2n+5)$; $g(n) = n$

Then find $O(n)$, $\Omega(n)$, $\Theta(n)$.

Sol:- 1) Big Oh (O) :-

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

$2n+5 \leq c.n$

For $\boxed{c=3}$ $\Rightarrow$ $2n+5 \leq 3n$

$5 \leq n$

So, $\boxed{n_0 = 5}$

2) Omega(Ω) :-

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

$2n+5 \geq c.n$

for $\boxed{c=1}$, $2n+5 \geq n$

$n \geq -5 \geq 1$

$\boxed{n_0 = 1}$

3) Theta (Θ) :-

$f(n) = \Theta(g(n))$ iff $c_1.g(n) \leq f(n) \leq c_2.g(n)$

$c_1.g(n) \leq f(n)$

$c_1.n \leq 2n+5$

$\boxed{c_1 = 1}$. Then

$n \leq 2n+5$

$\boxed{n_0 = 1}$

$f(n) \leq c_2.g(n)$

$2n+5 \leq c_2.n$

$\boxed{c_2 = 3}$ Then

$2n+5 \leq 3n$

$5 \leq n$

$\boxed{n_0 = 5}$

Dt:- 29/03/2023

* Sum of array Elements using recursions:-

```
1) Rsum (A,n) {
2) if(n==0)  return 0;
3) return Rsum (A, n-1) + A[n]
4) }
```

Time complexity:

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

$$T(n) = \begin{cases} T(n-1)+2 & n>0 \\ 2 & n=0 \end{cases}$$

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

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

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

⋮

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

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

* 1) Fibonacci series using recurrences:-

```
Fib(n) {
    fib1 = 0;
    fib2 = 1;
    for(i=2 to n) do {
```

```
        fib3 = fib1 + fib2;
        fib1 = fib2;
        fib2 = fib3;
    }
}
```

Time complexity :

cost	Repetition
1	1
1	1
1	1
1	n+1
1	n-2
1	n-2
1	n-2
1	n-2
	5n-5 = O(n)

$\therefore$ $\boxed{T(n) = O(n)}$

2) Addition of two matrices

```
Add ( A, B, C, m, n) {
    for (i = 0 to m) {
        for ( j = 0 to n) {
            C[i,j] = A(i,j) + B(i,j);
        }
    }
}
```

cost	Repetition
i=0	1
i<m	m+1
i++	m
j=0	m
j<n	m(n+1)
j++	mn
	mn
	3mn+4m+2

$T(n) = 3mn + 4m + 2 = O(mn)$

$\boxed{T(n) = O(mn)}$

3) Linear Search

```
LinearSearch (A, n, key) {
    for (i = 0 to n) do {
        if (key == A[i])
            return i;
    }
    return -1;
}
```

Time complexity : → Best case : $T(n) = O(1)$

→ Average case : $T(n) = \frac{n}{2} = O(n)$

→ Worst case : $T(n) = O(n)$

Last

Not found

4) Binary search :- [Recursive Algorithm]

```
BinarySearch (A, start, end, key) {
    if (start > end) return -1;
    mid = (start + end)/2;
    if (key == A[mid]) return mid;
    else if (key < A[mid]) {
        return BinarySearch (A, start, mid-1, key);
    }
```

return BinarySearch (A, mid+1, end, key);

}

Time complexity :

$$T(n) = \begin{cases} 2 & n=0 \\ 4 & n=1 \\ 5 + T(n/2) & n>0 \end{cases}$$

$T(n) = 5 + T(n/2)$ (1)

$= T(n/4) + 5 + 5$ (2)

$= T(n/8) + 5 + 5 + 5$ (3)

⋮

$= T\left(\frac{n}{2^k}\right) + k(5)$ [k-times]

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

$k = \log n$

After $\boxed{\log n}$ times $T(n) = T(1) + \log n(5)$

$= 5 \cdot \log n + 4$

$\boxed{T(n) = O(\log n)}$

Q) Arrange the below Time complexities in Ascending order

$\log n, n, n^2, 1, 2^n, n^n, n^3$

Ans:-

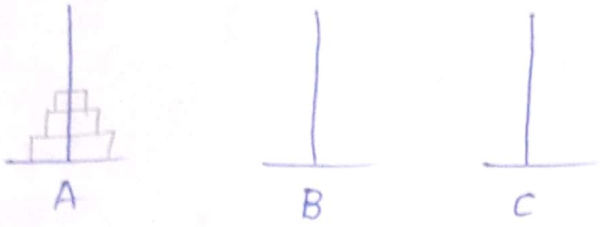
$1 < \log n < n < n^2 < n^3 < 2^n < n^n$

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

① → Also can be written as $O(n)$ & $O(n^2)$.

Dt:- 03/04/2023

* Towers of Hanoi:-



for 'n' disc's we need $2^n - 1$ steps to move from A to C.

Algorithm :

```
TOH (n, A, C, B) {
    if (n == 1)  move (A, C);
    if (n > 1) {
        TOH(n-1, A, B, C);
        TOH (1, A, C, B);
        TOH (n-1, B, C, A);
    }
}
```

Ex:- For n = 3

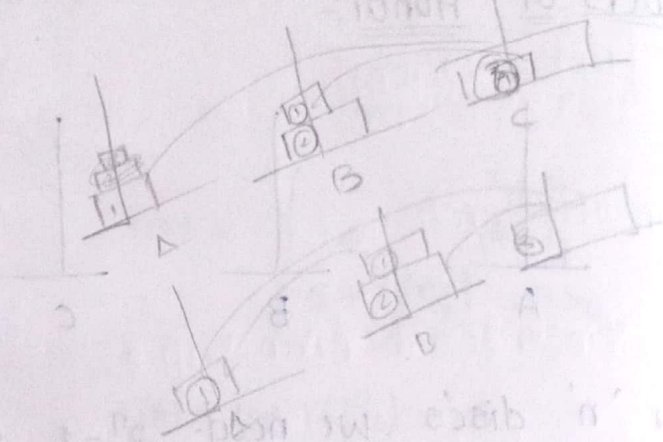
```
TOH (3, A, C, B) {
    TOH (2, A, B, C) {  // As n > 3
    TOH (1, A, C, B)
    TOH (1, A, B, C)
    TOH (1, C, B, A)
}
TOH (1, A, C, B)
```

TOH (2, B, C, A)

T(1, B, A, C)

T(1, B, C, A)

T(1, A, C, B)



Steps : 1) m (A, C)

2) A → B

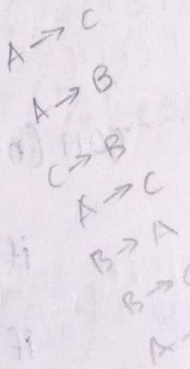
3) C → B

4) A → C

5) B → A

6) B → C

7) A → C



Time complexity :-

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

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

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

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

$= 2^2[2 \cdot T(n-3) + 1] + 2 + 1$

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

for k steps, $T(n) = 2^k T(n-k) + 2^k - 1$

$n - k = 1 \quad k = n - 1$

After n-1 steps, $T(n) = 2^{n-1} T(1) + 2^{n-1} - 1$

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

$T(n) = 2 \cdot 2^{n-1} - 1 = 2^{n-1} - 1$

$\boxed{T(n) = O(2^n)}$

* Masters theorem :-

(i) If the accurssive function is → dividing accurrence relation

$$\boxed{T(n) = a \cdot T(n/b) + \theta(n^k \log^p n) \quad a \geq b, b > 0, k \geq 0}$$

The timecomplexity is based on the following conditions:

i) $a > b^k$: $\boxed{T(n) = \theta(n^{\log_b a})}$

ii) $a = b^k$: a) $p > -1$: $\boxed{T(n) = \theta(n^{\log_b a} \cdot \log^{p+1} n)}$

b) $p = -1$: $\boxed{T(n) = \theta(n^{\log_b a} \cdot \log \cdot \log n)}$

c) $p < -1$: $\boxed{T(n) = \theta(n^{\log_b a})}$

iii) $a < b^k$: a) $p >= 0$: $\boxed{T(n) = \theta(n^k \log^p n)}$

b) $p < 0$: $\boxed{T(n) = \theta(n^k)}$

Ex:- $T(n) = a \cdot T(n/2) + 2$

Here $a = 1, b = 2, k = 0, p = 0$

$1 = 2^0 = 1 \quad a = b^k$

$T(n) = n^{\log_2 2^0} \cdot \log^1 n$

$= n^0 \cdot \log n$

$= \theta(\log n)$

$= O(\log n)$

Hlw: ① $T(n) = 2 \cdot T(n/2) + n$ ② $T(n) = 2 \cdot T(n/2) + n \log n$

① Sol:- $T(n) = 2 \cdot T(n/2) + n$

$a = 2, \; b = 2, \; k = 1, \; p = 0$

Here, $p > -1$. So,

$T(n) = \theta(n^{\log_2 2} \cdot \log^{0+1} n)$

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

② Sol:- $T(n) = 2 \cdot T(n/2) + n \log n$

$a = 2 \; ; \; b = 2 \; ; \; k = 1 \; ; \; p = 1$

$a = b^k \Rightarrow 2 = 2^1$

$p = 1$ & $p > -1$.

So, $T(n) = \theta(n^{\log_2 2} \cdot \log^{1+1} n)$

$T(n) = \theta(n \cdot \log^2 n)$

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

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

(ii) For decreasing recurrence relation :-

If the recursive relation is

$$T(n) = a.T(n-b) + f(n)$$

$$f(n) = O(n^k) \quad , a>0, b>0, k \geq 0$$

(1) $a<b1$: $\boxed{T(n) = f(n)}$

(2) $a=1$: $\boxed{T(n) = O(n.f(n))}$

(3) $a>1$: $\boxed{T(n) = O(a^{n/b}.f(n))}$

Ex:- ① $T(n) = 2T(n-1) + 1$

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

$a>1 \Rightarrow O\left(2^{n/1}.(1)\right) = \boxed{O(2^n)}$

② $T(n) = T(n-1) + n$

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

$a=1 \Rightarrow O(n.n) = \boxed{O(n^2)}$

Dt:- 10/04/2023

* Divide and Conquer :-

→ Dividing problem into two halves and combined to get overall solution.

Steps :

1) Dividing the problem into subparts.
2) Finding the solution to subparts.
3) Combining these solutions to get final solution.

Ex:- 1) Merge sort → $T(n) = 2T(n/2) + n$, $n > 1$ [recurrence relation]
2) Quick sort
3) Binary search
4) Tree traversals
5) Strassen's matrix multiplication.

2) Quick sort :- Pivot = A[start]
Pivot = A[end]
Pivot = A[mid]

Recurrence relation :- $T(n) = T(n-1) + n$ → worst case (pivot is max or min)

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

$T(n) = 2T(n/2) + n$ → Best case (Pivot is mid)

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

* Randomization :- choosing pivot element randomly uniform distribution to optimize time complexity.

* Differences b/w Merge and Quick sort :

Merge	Quick
1) $T(n) = \Theta(n \log n)$	1) $T(n) = O(n^2)$ [worst case]
2) Extra space	2) No extra space
	3) Spread sheets, Programming languages.

5). Strassen's Matrix Multiplication :-

→ Multiplication is the costiest operator comparing to Addition.

Ex:- $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$$A \times B = \begin{bmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{bmatrix}$$

For this we need, 8 *, 4 +

Size of Matrix	No. of *
[2x2]	8
[4x4]	64
[8x8]	512

} NMM

For 2x2, $\begin{cases} T(n) = 8T(n/2) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$ $\boxed{T(n) = O(n^3)}$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

For this method, we need 7 matrices:

$P = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$

$Q = (A_{21} + A_{22}) \cdot B_{11}$

$R = A_{11} \cdot (B_{12} - B_{22})$ $A_{11} \cdot (B_{12} - B_{22})$

$S = A_{22} \cdot (B_{21} - B_{11})$ $A_{22} (B_{21} - B_{11})$

$T = (A_{11} + A_{12}) \cdot B_{22}$

$U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$

$V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

To remember 7 matrices

B_{11}	A_{11}	A_{22}	B_{22}

1) Add [rows]

2) Sub [columns]

Then,

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

For this, the recurrence relation is:

$$T(n) = \begin{cases} 7T(n/2) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

$$T(n) = \theta\left(n^{\log_2 7}\right) = O(n^{2.81})$$

Size of matrix	No. of * operations
2 x 2	7
4x4	49
8x8	343

} Strassen's Multiplication

Date :- 11/04/2023

* Pseudocode for strassen's Matrix Multiplication :-

We apply this method ~~for~~ only when $n \times n = 2^k$ [2x2, 4x4, 8x8]

⇒ Strassens (A, B, n) {

```
    if(n == 1){
        C ← new matrix nxn
        C[0][0] = A[0][0] x B[0][0] ;
    }
    C ← new matrix nxn
    A11 , A12 , A21 , A22 ← A.split_matrix ();
    B11 , B12 , B21 , B22 ← B.split_matrix ();
    P = strassens (A11 + A22 , B11 + B22 , n/2);
    Q = strassens (A21 + A22 , B11 , n/2);
    R = strassens (A11 , B12 - B22 , n/2) ;
    S = strassens (A22 , B21 - B11 , n/2) ;
    T = strassens (A11 + A12 , B22 , n/2);
    U = strassens (A21 - A11 , B11 + B12 , n/2) ;
    V = strassens (A12 - A22 , B21 + B22 , n/2) ;
```

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

C = combine_quadrants (C11, C12, C21, C22);

}

return C;

}

Ex:- Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2\times 2}$ $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2\times 2}$

Sol:-

$P = (1+4).(1+4) = 25$

$Q = (3+4).(1) = 7$

$R = (1)(2-4) = -2$

$S = (4).(3-1) = 8$

$T = (1+2).(4) = 12$

$U = (3-1)(1+2) = 6$

$V = (2-4)(3+4) = -14$

$C_{11} = 25 + 8 - 12 - 14 = 7$

$C_{12} = -2 + 12 = 10$

$C_{21} = 7 + 8 = 15$

$C_{22} = 25 - 2 - 7 + 6 = 22$

$$A \times B = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$T(n) = O(n^{2.81}) = O(n^{\log 7})$

Advantages:-> Time complexity is better

> No. of multiplication operations are less.

Disadvantages :-

- 7*, 18+ [Matrix addition]
- It is only applicable for $2^k \times 2^k$.
- Requirement of extra space.
- Precission with float type of matrix.
- Difficult to implement.