Chapter No 1: Algorithms Analysis

Algorithms

Design
First we have to design a solution for a problem.

Person with the domain knowledge.

Any Language for humans (Preferebly Mathematical Notation)

H/W & OS Independent

Analyze Algorithm

Programs:

Implementation

Programmer will write the program, can be a designer of Algorithms also

Using Programming Language.

H/W & OS Dependent

Run/Testing

Priori Analysis	Posteriori Testing
Algorithm based	Program based
Independent of Language	Language Independent
H/W independent	H/W dependent
Time complexity & space Function	Watch Time (Total seconds by program) bytes of space occupied.

Characteristics of Algorithm:

1- Input => Can Take 0, or more inputs

2- Outputs => Must have at least 1 output

3- Definiteness => Every single line have a single & exact meaning => Cannot use ($\sqrt{-1}$) =>

4- Finiteness => Limited set of steps.

5- Effectiveness => It should serve some purpose.

How to Analyze an Algorithm (Factors)

1- Time
2- Space
3- Network ⇒ Data Transfer
4- Power consumption
5- CPU Registers

Example:

```
Algorithm swap(a, b)
{
    temp = a ——— Takes 1 unit of time
    a = b    ———  "  "  "  "  "
    b = temp ———  "  "  "  "  "
}
```

1- Time function: $f(n) = 3$, $f(n) = 10000$

Constant → $O(1)$

2- Space → temp ⇒ Local variable ⇒ 3

a 1, b 2

$S(n) = 3$ — $O(1)$

$S(n) = 400$ — $S(n) = 10000$ 2 3, 4

:Frequency Count Method:

Example:

n=8

A | 1 | 3 | 9 | 27 | 81 | 243 | 729 | 2187 |

```
Algorithm  Sum(A,n)
{
    S=0;          ——  1 (unit of time)
    for(i=0; i<n; i++)  ——>  n+1
    {
        S=S+A[i];  ——>  n times
    }
    return S;  ——  1
}
```

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

(Degree of Polynomial: n)

Time Complexity: order of $= O(n)$

Space Complexity:

Variables: A = n, S = 1, n = 1, i = 1

$S(n) = n+3 \Rightarrow$ order of: $O(n)$

Algorithm add (A, B, n)

n x n matrices

{

for (i=0; i<n; i++) → n+1

{

for (j=0; j<n; j++) → n x (n+1)

{

c[i] = A[i] + B[i]; → n x n

}

}

Time Complexity

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

↓ Highest degree

↓

$O(n^2)$

Space Complexity

A ⇒ n^2, B ⇒ n^2, C ⇒ n^2

n → 1, i → 1, j → 1

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

↓ Highest Degree

$O(n^2)$

①

```
for (i=n; i>=0; i--), (i=n; i>=0; i+=2);
{                                          n+1
    stmt;                                  n
}
```

O(n)

②

```
for (i=0; i<n; i++)                        n+1
{
    for(j=0; j<n; j++)                     n → n*(n+1) ⇒ n²+n
    {
        stmt;                              n*n ⇒ n²
    }
}
```

Thus O(n²)

③

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

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

n → Degree = n²

0+1+2+3+...+n = $\frac{n(n+1)}{2}$ ⇒ $\frac{n^2+n}{n}$

Time Complexity = O(n²)

(4) P=0

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

i	P	no of Times
1	0+1=1	
2	3	
3	6	
4		
⋮		
K	1+2+3+...+K	

P > n

And P = 0+1+2+3 ... +k

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

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

$\Rightarrow k^2 > n$

$\Rightarrow k > \sqrt{n}$

$O(\sqrt{n})$

Let n = 9

⇒ First time

i = 1, P = 0 < n ⇒ T

⇓

P = 0+1

⇓

No 2: i = 2; P = 1 < 9 ⇒ T

P = 1+2 = 3

⇓

No 3: P = 3 < 9 ⇒ T, i = 3

⇓

P = 3+3 = 6

⇓

No 4: P = 6 < 9 ⇒ T, i = 4

⇓

P = 6+4 = 10

⇓

P = 10 < 9 ⇒ False

Thus $O(\sqrt{n})$ ⇒ n = 9 ⇒ $\sqrt{9} = 3$

⑤

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

i
1
1×2 = 2
2×2 = 2^2
2^2×2 = 2^3
2^3×2 = 2^4
!
2^k

Assume $i >= n$

Since $i = 2^k$

$\therefore 2^k >= n$

$2^k = n$

$k = \log_2 n$

$\Rightarrow O(\log_2 n)$

⑥

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

$\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}$
!
$\frac{n}{2^k}$

Assume $i < 1$

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

$n = 2^k$

$k = \log_2 n \Rightarrow O(\log_2 n)$

⑦

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

$i*i <= n \Rightarrow i^2 >= n \Rightarrow i^2 = n$

$i = \sqrt{n}$

$O(\sqrt{n})$

(8)
```
for (i=0; i<n; i++)
{
;
;
for(j=0; j<n; j++)
{
}
}
```

— n

— n

2n ⇒ O(n

(9)
```
P=0
for (i=1; i<n; i= i*2)
{ P++ i;}
for (j=1; j<P; j= j*2)
{ stmt; }
```

— log n = P

— log P

⇒ O(log P) ⇒ O(log log n)

(10)
```
for (i=0; i<n; i++)
{
  for(j=1; j<n; j= j*2)
  {
    stm
  }
}
```

— n

— n × log n

— n × log n

⇒ f(n) = 2n log n + n

O(n log n)

Classes of Time Function

$O(1) \Rightarrow$ Constant
$O(\log n) \Rightarrow$ Logarithmic

$O(n) \Rightarrow$ Linear
$O(n^2) \Rightarrow$ Quadratic
$O(n^3) \Rightarrow$ Cubic
$O(2^n) \Rightarrow$ Exponential

Comparison

Based on Weightage

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

Asymptotic Notations:

Big Oh $[O(n)]$ ⇒ Worst Case (Upper Bound)

Big Omega $[\Omega(n)]$ ⇒ Best Case (Lower Bound)

Theta $[\Theta(n)]$ ⇒ Average Case (Average Bound)

Big Oh - When to use

$f(n) = O(g(n))$ if and only if there exists postive constant c and n_0

such that

$f(n) \leq c * g(n)$ for all n's (numbers or cases) greater than n_0

Example:

$f(n) = 2n+3$

Form:

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

$2n+3 \leq 7n$

یہاں آپ کچھ بھی لکھ سکتے ہیں بشرطیکہ وہ f(n)

سے greater ہو۔

n_0 is starting value from where

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

To get this,

$2n+3 \le 2n+3*(n)$

$\Rightarrow 2n+3 \le 2n+3n \Rightarrow 2n+3 \le 5n$

Agar ham n ki jagah value put karen

$\Rightarrow$ for all values ≥ 1

Yahan $\because f(n) = O(n)$ because polynomial degree

Now, if Big-Oh is $f(n) = O(n)$

Time function سے جو greater ہے اس کا مطلب ہے
Big-Oh میں آ سکتا ہے۔

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

Lower bound: $\Omega(\sqrt{n})$, $\Omega(\log n)$ all true

Upper bound: $O(n^2)$, $O(n\log n)$ all are true
→ لیکن تمام میں سے close والا ہے۔

Average Bound: $\Theta(n)$ → We cannot use any other

Not Related to Best, Worst or Average Case.

Example Asymptotic Notation

① $f(n) = 2n^2 + 3n + 4$

For Big-Oh - O

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

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

$\downarrow$

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

For Big-Omega - Ω

$2n^2 + 3n + 4 \geq 1 \times n^2 \Rightarrow f(n) = \Omega(n^2)$

For Theta - θ

$1(n^2) \leq 2n^2 + 3n + 4 \leq 9(n^2)$

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

② $f(n) = n^2 \log n + n$

$1(n^2 \log n) \leq n^2 \log n + n \leq 10 [n^2 \log n]$

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

③ $f(n) = n!$

$c_1 * [g(n)] \leq f(n) \leq c_2 * g(n)$

$1 \times 1 \times 1 \times 1 \times 1 \times \ldots 1 \leq n! \leq n \times n \times n \times n \times n \times n$

$1 \leq n! \leq n^n$

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

جب کوئی common نہ آئے تو θ کی جگہ
Big-Oh اور omega استعمال کرتے ہیں

④ $f(n) = \log n!$

$\log 1 \times \log 1 \times \log 1 \times \ldots \log 1 \leq \log n! \leq \log n \times \log n \times \log n \times \ldots \log n$

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

$\Rightarrow 1 \leq \log n! \leq n \log n$

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

Properties of Asymptotic Notations

General:

- If $f(n)$ is $O(g(n))$ (Ω, Θ) than $a*(f(n))$ is $O(g(n))$

  e.g if $f(n) = 2n^2+5 \Rightarrow O(n^2)$

  then $7 \cdot f(n) = 7(2n^2+5) = 14n^2+35$

  $\Rightarrow O(n^2)$

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

  e.g $f(n) = n^2 \Rightarrow O(n^2)$

Transitive:

(True for all 3 notations)

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

e.g.

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

$\Rightarrow$ n is $O(n^2)$ and n^2 is $O(n^3)$ then n is $O(n^3)$

Symmetric:

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

e.g

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

$f(n) \Rightarrow \Theta(n^2)$, $g(n) \Rightarrow \Theta(n^2)$

Only True for Θ

Transpose Symmetric: (True for O and Ω)

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

e.g

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

upper Bound $O(n^2)$

lower Bound $\Omega(n)$

Another:

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

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

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

- **Addition:**

If $f(n) = O(g(n))$
$d(n) = O(h(n))$

$$f(n) + d(n) = O(\max(g(n), h(n)))$$

ع lot expression لو ادخل اكبر ع example ادي

e.g

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

$$f(n) + d(n) = n + n^2 \Rightarrow O(n^2)$$

Higher degree polynomial

ف Big-O notation مش تفصيل ع

Multiplication:

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

$$f(n) * d(n) = O(g(n) * h(n))$$

e.g

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

$$\Rightarrow f(n) * d(n) = n * n^2 = n^3 \Rightarrow O(n^3)$$

Comparisions of Functions:

We compare function to see which is greater (upper bound) and which is smaller (lower bound).

Function 1	Function 2
n	n^2

directly کہیں گے simple

Function 1 is smaller and
Function 2 is greater.

i.e

$$n < n^2$$
$$O(n) \leq O(n^2)$$

Prove کرنے کے لیے Mathematically

طریقے ہیں۔

Way 1:

Put sample values

Sample values	Function 1	Function 2
n	n^2	n^3
2	$2^2 = 4$	$2^3 = 8$
3	$3^2 = 9$	$3^3 = 27$
4	$4^2 = 16$	$4^3 = 64$

Thus,

$n^2 < n^3$ (As per value)

Second Way: (Applying log):

n^2	n^3
$\log n^2$	$\log n^3$
$2 \log n$	$3 \log n$
∴ Rule of log	
$2 \log n$	$< 3 \log n$

Some log Rules:

1: $\log ab = \log a + \log b$

2: $\log a/b = \log a - \log b$

3: $\log a^b = b \log a$

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

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

Examples:

$f(n) = n^2 \log n$ $\quad$ $g(n) = n(\log n)^{10}$

Applying log on both sides

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

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

$2\log n + \log\log n$ $\quad$ $\log n + 10 \log\log n$

Bigger Term

$2\log n \geq \log n$

Thus

$f(n) > g(n)$

Example

$f(n) = 3n^{\sqrt{n}}$ $\quad$ $g(n) = 2^{\sqrt{n}\log_2 n}$

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

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

$(n^{\sqrt{n}})^{\log_2 2 = 1}$

$= n^{\sqrt{n}}$ using 4th formula

$\Rightarrow (3)\ n^{\sqrt{n}}$ $\quad$ $n^{\sqrt{n}}$

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

But asymptotically they have

$O(n^{\sqrt{n}}) = O(n^{\sqrt{n}})$

Example:

$n^{\log n}$ $\qquad$ $2^{\sqrt{n}}$

Applying log

$\log(n^{\log n})$ $\qquad$ $\log 2^{\sqrt{n}}$

$\log n \times \log n$ $\qquad$ $\sqrt{n} \log 2$

$\log^2 n$ $\qquad$ $\sqrt{n}$

ابھی بھی کچھ واضح نہیں ہے تو ہم ایک بار اور log

Apply کرتے ہیں

$\log(\log^2 n)$ $\qquad$ $\log n^{1/2}$

$\Rightarrow 2 \log \log n$ $\qquad$ $\frac{1}{2} \log n$

smaller $\qquad$ Greater

Examples

$f(n) = 2^{\log n}$, $g(n) = n^{\sqrt{n}}$

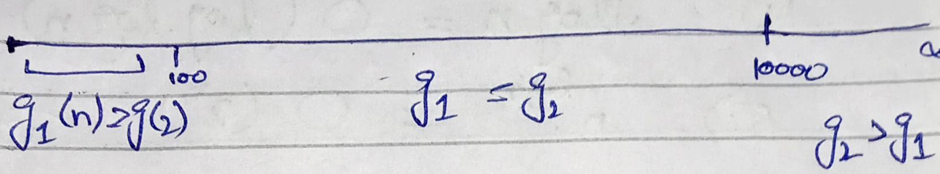
$f(n) = 2n$, $g(n) = 3n$

$f(n) = 2^n$, $g(n) = 2^{2n}$

Last Type

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

$$g_2(n) = \begin{cases} n^2 & < 10{,}000 \\ n^3 & \geq 10{,}000 \end{cases}$$



جب infinity کو approach کرنے والا

function میں جو greater ہو تو مکمل طور

ہو وہ function؛

is greater

$\Rightarrow \quad g_1(n) < g_2(n)$

True or False

1) $(n+k)^m = O(n^m)$ True

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

3) $2^{2n} = O(2^n)$ False

4) $\sqrt{\log n} = O(\log \log n)$ False

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

Best, Worst, Average Case:

For this we will take example of

A: Linear Search
B: Binary Search Tree

A: Linear Search

a

8	6	12	5	9	8	7	10	15	19	20
0	1	2	3	4	5	6	7	8	9	10

Case 1: Key = 7, Found in 7 iteration on Index 6

Case 2: Key = 21, Not Found

Best Case:

جب ہمیں پہلا Element ملے

Index "0" پر

Best Case Time — 1 ⇒ O(1)

B(n) = O(1)

Worst Case:

Last Index Element i.e

Index 10 = 20

Worst Case Time — n ⇒ O(n)

W(n) = O(n)

Average Case:

All Possible Cases time
No. of Cases

This is very difficult to determine.

$$\text{Avg. time} = \frac{1+2+3+\cdots+n}{n} \Rightarrow \frac{n(n+1)}{2} \Big/ n$$

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

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

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

Using Asymptotic Notation:

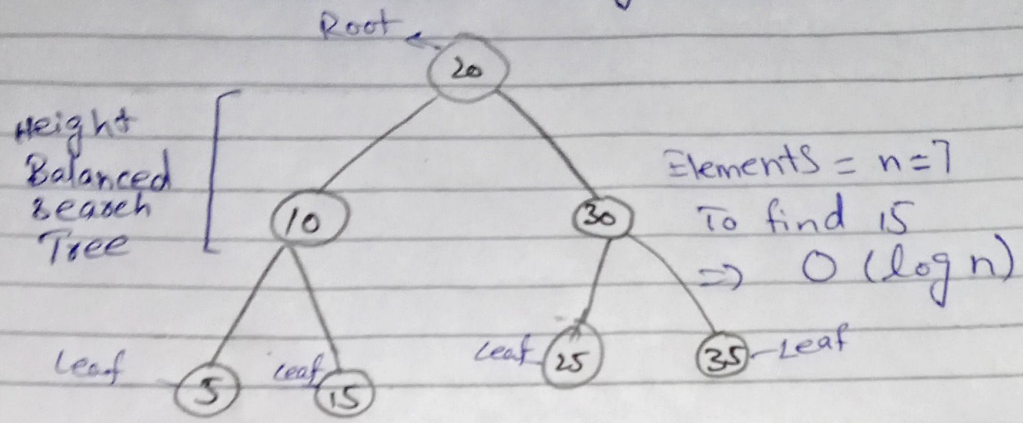
$B(n) = 1$, $B(n) \leq O(1) \Rightarrow B(n) = \Omega(1) \Rightarrow \Theta(1)$

$W(n) = n$, $W(n) = O(n) \Rightarrow W(n) = \Omega(n) \Rightarrow \Theta(n)$

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

3- Binary Search

Elements are already sorted.



Best Case – Searching for Root Ele

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

Worst Case – Searching for Leaf Ele

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

⇒ min $W(n) = \log n$

max $W(n) = n$