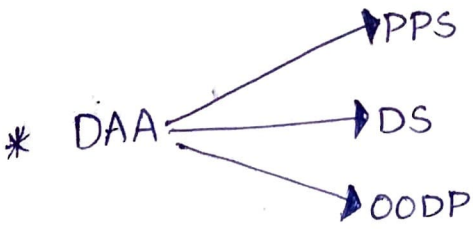


Algorithm

* Design
* Domain Knowledge.
* Any Language.
* Independent of OS and Hardware

Program

* Implementation
* Program knowledge
* Program Language
* dependent of OS and Hardware.

* Design Algorithm
  ↓
  Analysis of Algorithm ⟹ Which Algorithm is best using
  ↓ Time Complexity
  Create Program Space Complexity

* Analysis of Algorithm.
  → Time. (How fast the program runs)
  → Space. (How much memory program occupy)
  → other factors → power consumption
  CPU Register.
  Network.

Eg:-

* A → 10sec → 2GB
  B → 5sec → 1.5GB Best one.

(1) Algo Swap(a,b)

Time | Space

0

Begin

temp ← a; 1 | a 1

a ← b; 1 | b 1

b ← temp; 1 | temp 1

End.

f(n) = 3 ↓ Time function O(1) constant Time

S(n) = 3

S(n) = 3

O(1)

* if a = (a*b) + (b+5)

4th 3rd 1st 2nd

4 unit of time O(1)

* Space Complexity ⇒ int a,b ⇒ int – 4bty

a,b 8 bty for this line.

Variable need memory in RAM

(2) Algo sum (A,n)

Time → 0

{

s=0; → 1

for (i=0; i<n; i++) → n+1

{

s = s+A[i] → n

}

return s; → 1

}

1+n+1+n+1

f(n) = 2n+3

O(n)

if n=5

i=0 0<5 (1)

1<5 (2)

2<5 (3)

3<5 (4)

4<5 (5)

5<5 X

(Include it)

5+1 times

Space

A → n

n → 1

s → 1

i → 1

S(n) = n+3

* Declaration, {, } 1 unit time

* Simple Initialization 1 unit time

Algorithm Add (A, B, n)

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

$f(n) = 2n^2 + 2n + 1 \Rightarrow O(n^2)$

Space Complexity

A	n×n
B	n×n
n	1
i	1
j	1
C	n×n

$S(n) = 3n^2 + 3 \quad O(n^2)$

Time Comp

Algorithm Multiply (A, B, n)

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

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

$O(n^3)$

Higher order in $f(n)$ is taken.

Space Complexity

A	n*n
B	n*n
C	n*n
n	1
i	1
j	1
k	1

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

$O(n^2)$

Time Complexity

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

n+1
n
O(n)

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

n
O(n)

(3)
```
for (i=1; i<n; i+2)
{
    stmt
}
```

n/20
O(n)

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

n*n
f(n) = n²
O(n²)

(5)
```
for (i=0; i<n; i++)
{
    for (j=0; j<1; j++)
    {
        stmt;
    }
}
```

n = anyvalue.

⑥
```
p=0
for (i=1; p<=n; i++)
{
    p=p+i;
}
```

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

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

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

$k^2 = n$

$k = \sqrt{n}$

$O(\sqrt{n})$

Linear Search

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

one by one

```
Linear Search (A, n, k)
{
    flaag=0;
    for(i=0, i<n; i++)
    {
        if (a[i]==k)
        {
            printf("Element found");
            flag=1;
            break;
        }
    }
}
```

Binary Search

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
5	6	10	12	91	81	95	97

l. ——— h.

$\frac{0+7}{2} = 3.5$

$= 3$

a[mid] == key

12 == 6

key < a(mid)

l = 0

n = mid

```
Binary Search (A, n, k)
{
    flag = 0;
    l = 0;
    h = n-1;
    mid = l+h/2;
    while (l <= h)
    {
        if (a[mid] == k)
        {
            printf("Element found")
            flag = 1;
            break;
        }
        elseif (K < a[mid])
        {
            h = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }
```

```
if (flag == 0)
{
print ("Not Found")
}
}
```

Time Complexity Cases

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

$i > n$

$2^k = n$

$k = \log_2 n$

$f(x) = \log_2 n$

$O(\log_2 n)$

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

---

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

Assume

$i <= 1$

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

$n = 2^k$

$k = \log_2 n$

i/n
$n/2$
$n/2^2$
$n/2^3$

---

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

Assume

$i^2 > n$

$i^2 = n$

$i = \sqrt{n}$

$O(\sqrt{n})$

④ for(i=0; i<n; i++) ⟶ n+1

{ stmt; ⟶ n.

}

⑤ for (j=0; j<n ; j++) ⟶ n+1

{ stmt; ⟶ n.

}

⑥ p=0;

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

{ p++; ⟶ $p = \log_2 n$

stmt

}

⑦ for(j=1; j<p; j=j*2)

{ stmt;

}

logP

LogP

$O(\log \log_2 n)$

⑧ for(i=0; i<n; i++) ⟶ n+1

{ for(j=1; j<n; j*2) ⟶ n logn.

{ stmt; ⟶ nlogn

}

}

2nlogn+n+1

O(nlogn)

⑨ 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) → $O(\log_2 n)$

Analysis of while and if

①

```
i=0                    → 1
while(i<n)             → n+1
{
    stmt               → n
    i++                → n
}
```

$3n+2$

$O(n)$

---

②

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

a
1
1*2 = 2
2*2 = 4
4*2 = 8
.
.
.
1
2^k

$a > b$

$2^k \geq b$

$2^k = b$

$k = \log_2 n$

$O(\log n)$

③
```
i=n;
while(i>1)
{
  stmt;
  i = i/2;
}
```

$O(\log n)$

④
```
i=1
k=1
while(k<n)
{
  stmt;
  k=k+i;
  i++;
}
```

Analysis with while and if

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

m	n	m	n	m	2
6	3	5	5	16	2

If condition

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

Types of time functions (or) Classes of function

$O(1)$ → constant

$O(\log n)$ → logarithmic

$O(n)$ → Lineal

$O(n^2)$ → Quadratic

$O(n^3)$ → Cubic

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

Eg:- $f(n)=2$, $f(n)=5$, $f(n)=5000$ } $O(1)$

Eg:- $f(n)=2n+3$, $f(n)=500n+700$, $f(n)=\frac{n}{5000}+6$ } $O(n)$

Eg:- Matrix addition

Eg:- Matrix Multiplication.

Classes of function in increasing order of derivates

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

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256
3.16	9	81	512

Asymptotic Notations

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

Notation

- O Big-oh = upper bound of a function
- Ω Big-omega = lower bound of a function
- Θ Theta = Average band of a function

Big-oh O

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

The function $f(n) = O(g(n))$ if $\exists$ +ve constants c and n_0 such that $f(n) \leq c * g(n)$ $\forall n \geq n_0$

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

$2n+3 \leq 10n \quad \forall n \geq 1$

$n=1 \quad 2(1)+3 \leq 10(1)$

$5 \leq 10$

$f(n) = O(n)$

$f(n) = 2n+3$

$2n+3 \leq n(2n+3)$

$2n+3 \leq 2n^2+3n \quad \forall n \geq 1$

Eg:- $2(1)+3 \leq 2(1)+3(1)$

$5 \leq 5$

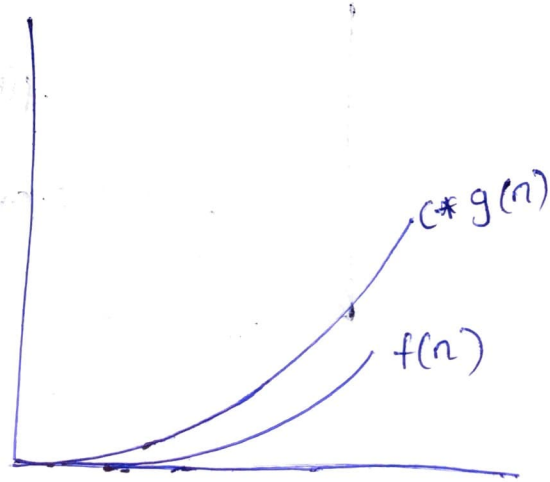
$f(n) = O(n^2)$

$f(n) = 2n+3$

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

$f(n) \quad c \quad g(n)$

$7 \leq 5n^2$



Omeaga

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

The function $f(n) = \Omega(g(n))$ if $\exists$ +ve constants C and C_0 such that $f(n) \geq C * g(n) \; \forall \; n \geq n_0$

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

$2n+3 \geq 1 * n$

$f(n) = C * g(n)$

$n=1 \quad 2(1)+3 <= 1 \times 1$

$5 <= 1$

$f(n) = 2n+3$

$f(n) \leftarrow 2n+3 \geq \underset{\downarrow C}{1} * \log n \longrightarrow g(n)$

$n=1 \quad 5 \geq 1 * \log_2 1$

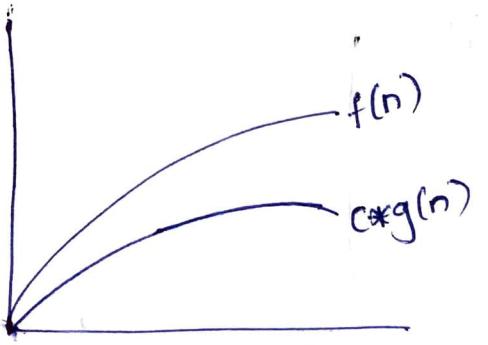
$5 \geq 0$

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

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

$f(n) = \Omega(1)$

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



Theta

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

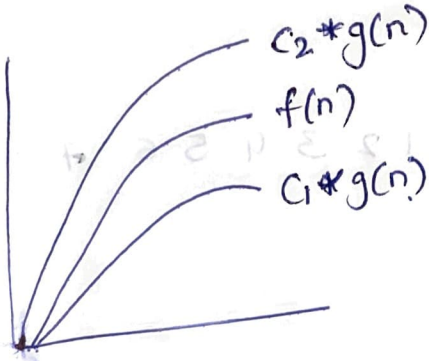
Eg :- $f(n) = 2n+3$

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

$c*g(n) \quad f(n) \quad c_2*g(n)$

$n=1 \quad 1 \leq 5 \leq 5$

$n=2. \quad 2 \leq 7 \leq 10$



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

$f(n) = n!$

$= n*(n-1)*(n-2) - - - 3 \times 2 \times 1$

$1*1*1*1 \leq 1*2*3 - - - *n \leq n*n*n*n$

Best, worst / Average case always

Linear Search

```
Linear Search (A, n, K)
{
    if (key == A[i])
    {
        printf ("Elemet key found")
        flag = 1;
        break;
    }
```

```
if (flag==0)
{
  printf("key Not found")
}
```

Best Case :-

$O(1)$

$\Omega(1)$

$\Theta(1)$

Worst Case :- $O(n)$

$\Omega(n)$

$\Theta(n)$

Average Case :- $\frac{7}{8}$

1 2 3 4 5 6 7 8 9

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

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

$= O(n)$

$\Omega(n)$

$\Theta(n)$