Frequency Count Method

(1)

```
Algorith Sum (A,n)
{
  s=0;                          ——— 1 time
  for (i=0; i<n; i++)           ——— n+1 time
  {   1     n+1   n
    s = s + A[i];               ——— n times
  }
  return s;                     ——— 1 time
}
```

time function $f(n) = 2n+3 = O(n)$

Space Complexity:

A — n
n — 1
s — 1
i — 1

$S(n) = n+3 = O(n)$

(2)

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

Space

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

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

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

③ 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];
                }}}
```

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

Space:

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

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

(2)

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

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

$O(n)$

(5)

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

(5.)

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

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

we Assome that $p > n$

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

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

$k^2 > n \qquad k = \sqrt{n} = O(\sqrt{n})$

⑥ for (i = n ; i >= 1 ; i = i/2)

{ Stmt; }

$\frac{i}{n}$
$n/2$
$n/2^2$
$n/2^3$
⋮
$n/2^k$

Assume $i < 1$

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

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

$n = 2^k$

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

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

$\log_2 n$

$\log_2 p$

$O(\log \log_2 n)$

classes of functions: (types of time functions)

$O(1)$ — Constant

$O(\log n)$ — logrithmic

$O(n)$ — Linear

$O(n^2)$ — Quadratic

$O(n^3)$ — Cubic

$O(k^n)$ OR $O(2^n)$ — Exponential
$k>1$

$k \geq 2$ $O(n^k)$ — polynomial complexity

$f(n) = 2$ $O(1)$
$f(n) = 5$
$f(n) = 500$

$f(n) = 2n+3 = O(n)$
$f(n) = 500n+700 = O(n)$

Rate of growth: the growth rate for an algo is the rate at which the cost of the algo grows as the size of its input grows

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

this the class of function in increasing order of their weightage.

Observe that the functions are listed in the order of their rates of growth. the logarithmic function $\log_2 n$ grows most slowly, the exponential function 2^n grows most rapidly.

n / g(n)	log n	n	n log n	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}

```
A()
{ for (i=1; i<=n; i++)
  {
    for (j=1; j<=n; j=j+i)
      { printf("Hello");
      }
  }
```

i	1	2	3	4	. . . .	n
j	1 to n	1 to n	1 to n	1 to n		1 to n
printf	n time	$n/2$ time	$n/3$ time	$n/4$ time		n/n time

$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots \frac{1}{n}\right) = n(\log n)$

$= O(n \log n)$.