Frequency count method for complexity

① ~~Sum of Array elements~~

Framwork for analysis of Non-recursive functions.

+ Finding complexity of non-recursive algorithm is simpler than that of recursive algorithm.

Step ① Determine size of problem / input
Step ② Find primitive / elementary operations.
Step ③ Find count of primitive operations for best, worst and average case.
Step ④ simplify the summation by dropping multiplicative and divisive constants of highest degree polynomial term in sum.

① Sum of array elements

```
Algorithm sumarray (A, n)
{
    s ← 0;                          → 1
    for (i=0; i<n; i++)             → 2n+2
        (1)  (n+1)  (n)
    {
        s = s + A[i];               → n
    }
    return s;                       → 1
}                                   ⟹ 3n+4
```

for statement $i<n$

if $n=5$

$i=0$
$i=1$
$i=2$
$i=3$
$i=4$ } comparison is true.

$i=5$ false

if $n=5 \Rightarrow i<n$ is executed $n+1=6$ times i.e. 5 times it is true and 6th time is false.

$f(n) = 3n+4$ ← Highest degree $\Rightarrow n$

Time complexity : $T(n) = O(n)$

Space complexity → A, n, s, i variables used

Size of A = n
n = 1
s = 1
i = 1

$S(n) = n+3$

i.e. $S(n) = O(n)$

② Sum of two matrices

```
Algorithm (A, B, n)
{
    for (i=0 ; i<n ; i++)          ⇒ 2n+2 But we consider n+1
    {                                  1   n+1   n
        for (j=0 ; j<n ; j++)      → n(n+1) times
        {                                  inner
            C[i][j] = A[i][j] + B[i][j] ;   → n times → n×n
        }
    }
}
```

$f(n) = (n+1) + n(n+1) + n \times n$

$= n + 1 + n^2 + n + n^2$

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

Time complexity — Highest Degree $\Rightarrow \boxed{T(n) = O(n^2)}$

Space complexity

Variables $\Rightarrow$

A	$\Rightarrow n^2$	$n \times n$ matrix
B	$\Rightarrow n^2$	
C	$\Rightarrow n^2$	
n	$\Rightarrow 1$	
i	$\Rightarrow 1$	
j	$\Rightarrow 1$	

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

Space

③ **Matrix multiplication**

Algorithm multiply (A, B, n)

```
{
① n+1      for (i=0; i<n; i++)
           {
② n*(n+1)      for (j=0; j<n; j++)
               {
③ n*n              C[i][j] = 0;
④ n*(n+1)          for (k=0; k<n; k++)
                   {
⑤ n*n*n                C[i][j] = C[i][j] + A[i][k] * B[k][j];
                   }
               }
           }
}
```

①	$n+1$	=	$n+1$
②	$n*(n+1)$	=	n^2+n
③	$n*n$	=	n^2
④	$n*n*(n+1)$	=	n^3+n^2
⑤	$n*n*n$	=	n^3

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

$T(n) = O(n^3)$

Space ⇒ A = n^2, B ⇒ n^2, C = n^2 } matrix

n = 1, i = 1, j = 1, k = 1 } $3n^2+4 \Rightarrow S(n) = O(n^2)$

1.5.1 Time complexity #1

① `for(i=0 ; i<n ; i++)` → n+1

```
{
    statement;   → n
}
```

$f(n) = 2n + 1$

$T(n) = O(n)$

② `for(i=n ; i>0 ; i--)`

```
{
    statement;   → n times
}
```

$O(n)$

③ `for(i=0 ; i<n ; i=i+20)`

```
{
    statement;   → n/20 times
}
```

$f(n) = \frac{n}{2} = O(n)$ Same

④ for (i=0; i<n; i++) ⟶ n+1
{
for (j=0; j<n; j++) ⟶ n × (n+1)
{
statement; ⟶ n × n
}
}

$O(n^2)$

⑤ for (i=0; i<n; i++)
{
for (j=0; j<i; i++)
Here is difference
{
statement;
}
}

Lets trace values

i	j	No of times statement executed
0	0 × condition false	0 no execution
1	0 ✓ 1 ×	1
2	0 ✓ 1 ✓ 2 ×	2 sum of this
!		
n	0 ✓ 1 ✓ 2 ✓ ...	n

1.5.1 Time complexity

① `for (i=0 ; i<n ; i++)` → n+1

```
{
    statement;        → n
}
```

$f(n) = 2n + 1$

$T(n) = O(n)$

② `for (i=n ; i>0 ; i--)`

```
{
    statement;        → n times
}
```

$O(n)$

③ `for (i=0 ; i<n ; i=i+20)`

```
{
    statement;        → n/20 times
}
```

$f(n) = \frac{n}{2} = O(n)$

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

$O(n^2)$

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

Here is difference

Lets trace values

i	j	No of times statement executed
0	0 × (condition false)	0 (no execution)
1	0 ✓, 1 ×	1
2	0 ✓, 1 ✓, 2 ×	2
⋮	⋮	sum of this
n	0 ✓, 1 ✓, 2 ✓, ..., n ×	n

$$T(n) = 0+1+2+3+\cdots+n$$

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

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

⑥

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

this is not i<n, so not n times

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

How many times it will execute? Say k times

not 'n' times but for some time say k time

Assume $P > n$ then execution stops.

The moment comes that $P = 1+2+\cdots+k$

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

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

$$\frac{K^2+K}{2} > n$$

$$\geq K^2 > n$$

$$\therefore K > \sqrt{n}$$

$$\therefore \boxed{O(\sqrt{n})}$$

---

(7)

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

This is not executing for n times
Then suppose it executes for some (K) time

i
1
$1*2 = 2$
$2*2 = 2^2$
$2^2*2 = 2^3$
$2^3*2 = 2^4$
:
$= 2^k$

Assume when $i >= n$

execution stops.

$\therefore 2^k >= n$

$\therefore 2^k = n$

$\boxed{k = \log_2 n}$

$O(\log_2 n)$

// value reached to 2^k

⑧
```
for(i=0 ; i<n ; i++)          ——— n+1
{
    for(j=1 ; j<n ; j=j*2)    → n x log n
    {
        statement;            → n x log n
    }
}
```

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

$O(n \log n)$

---

⑨ **Bubble sort.**

```
Algorithm Bubblesort(A, n)
{
    for(i=n-1 ; i>=1 ; i--)          n-1
    {
        for(j=0 ; j<i ; j++)         ⇒ n(n-1)/2
        {
            if(A[j] > A[j+1])
            {
                temp = A[j]
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
}
```

Bubble sort Algorithm

Algorithm Bubblesort(A, n)

```
for i ← 0 to n-1 do                    → (n-1)
    for j ← 0 to n-i-1 do
        if A[j] > A[j+1] then
            Swap(A[i], A[j])
        end
    end
end
```

Time

n=5

i	j	(n-1-i)
0	0,1,2,3,4	(n-1)
1	0,1,2,3	(n-2)
2	0,1,2	(n-3)
3	0,1	1
4	0	0

$$0+1+2+3+4+\cdots+(n-1)$$

$$\therefore \frac{(n-1)(n-1+1)}{2}$$

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

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

Pass 1

1	2	3	4	5	6
5	1	4	2	8	9

$n = 6$

$n-1 = 5$

$i = 0$ | $j = 0, 1, 2, 3, 4, 5$

1	5	4	2	8	9

1	4	5	2	8	9

1	4	2	5	8	9

Skip

1	4	2	5	8	9

⇒

1	2	4	5	8	9

⇒