Contents

- Analysis of Algorithms
- Order of Growth
- Best Case, Average Case, Worst Case
- Calculating The Running Time of a program
- Analysing the Time Efficiency of Non-recursive Algorithms
- Analysing the Time Efficiency of Recursive Algorithms
- Empirical analysis of time efficiency

Characteristics of the Algorithms

- Input
- Output
- Definiteness
- Finiteness
- Effectiveness

Analysis of algorithms

- **Dimensions**:
  - Simplicity
  - Time efficiency
  - Space efficiency
- The term "analysis of algorithms" is usually used in a narrower, technical sense to mean an investigation of an algorithm's efficiency with respect to two resources: running time and memory space
- Approaches:
  - Theoretical analysis
  - Empirical analysis

Analysis of algorithms

- **Time efficiency**, also called **time complexity**, indicates how fast an algorithm runs.
- **Space efficiency**, also called **space complexity**, refers to the amount of memory units required by the algorithm in addition to the space needed for its input and output.

Comparing Algorithms

Q. Given two algorithms **A** and **B**, how do we know which is faster?

A. Implement and run both and compare the time each takes!

To compare two algorithms, we can implement them, run them and compare their running times.

Challenges!

- The running time of a program is hardware and software dependent.

We need to run both algorithms on the same machine (or on machines with the same specs), using the same programming language, the same compiler, etc.

- The running time of a program depends on the input size and on the input type.

We need to run the programs as many times as needed to cover all possible input sizes and types that might affect the behavior of the programs.

- Running the programs might take a long time!

Takes as long as the fastest of the two programs requires.

Which program runs faster?

Program A:

```
x = 1;
y = 2;
sum = x + y;
```

4 operations

Program B:

```
x = 1;
y = 2;
z = 3;
k = 4;
m = 5;
n = 6;
x = x + y;
x = x + z;
x = x + k;
x = x + m;
X = x + n;
```

16 operations

Theoretical analysis

```
Algorithm Swap(a,b)
{       temp=a;  ------- 1
        a=b;        -------- 1
        b=temp;   -------- 1
}    ---------------------------------
       f(n) = 3
```

```
space
   a
   b
   temp
---------------
s(n)= 3
```

Frequency count method

```
Algorithm  sum(A,n)
{      s=0;
       for(i=0;i<n;i++)
        {
           s=s+A[i];
        }
Return s;

}
```

Frequency count method

```
Algorithm  sum(A,n)
{     s=0;   -------------------------   1
       for(i=0;i<n;i++)  ------------------n+1
        {
           s=s+A[i];      -------------------- n
        }
Return s;        ----------------------  1

}                                          -----------
                                            2n+3
```

Frequency count method

```
Algorithm  sum(A,n)                    space
{     s=0;                              A=n
      for(i=0;i<n;i++)                  n=1
      {                                 i=1
         s=s+A[i];                      s=1
      }                                 ____________
Return s;                               s(n)=  n+3

}
```

Frequency count method

```
Algorithm  sum(A,B,n)
{            for(i=0;i<n;i++)  ----------------n+1
       {
          for(j=0;j<n;j++)    ----------- n(n+1)
         {
          C[i,j]=A[i,j]+B[i,j];   ---------- n * n
           }
        }                                    -----------
 }                                          2 n^2 + 2n + 1
                                            0( n^2)
```

Frequency count method

```
Algorithm  sum(A,B,n)
{           for(i=0;i<n;i++)  ----------------n+1
            {
          for(j=0;j<n;j++)    ---------- n(n+1)
{
          C[i,j]=A[i,j]+B[i,j];   ---------- n * n
      } }
                                              -----------
                                          2 n^2 + 2n + 1
                                           0 n^2
```

space:

A	n^2
B	n^2
c	n^2
n	1
I	1
j	1
$3\ n^2+3$	$0\ n^2$

Frequency count method

```
Algorithm  sum(A,B,n)
{          for(i=0;i<n;i++) ----------------   n+1
           {
          for(j=0;j<n;j++)   ----------        n(n+1)
            { c[i,j]=0;  --------- -------      n * n
              for(j=0;j<n;j++) {  ------        n * n* (n+1)
          C[i,j]=A[i,j]+B[i,j];  ----------     n * n  *n
                 }                          f(n)= 2 n^3+  3 n^2 + 2n + 1
             }                                    0 n^3
           }
}                          -
```

space:

A	n^2
B	n^2
c	n^2
n	1
j	1
I	1
k	1
	$3\,n^2+4$
	$0\,n^2$

Runtime Analysis Procedure

requires tracing skills

code → Trace → summation → Answer

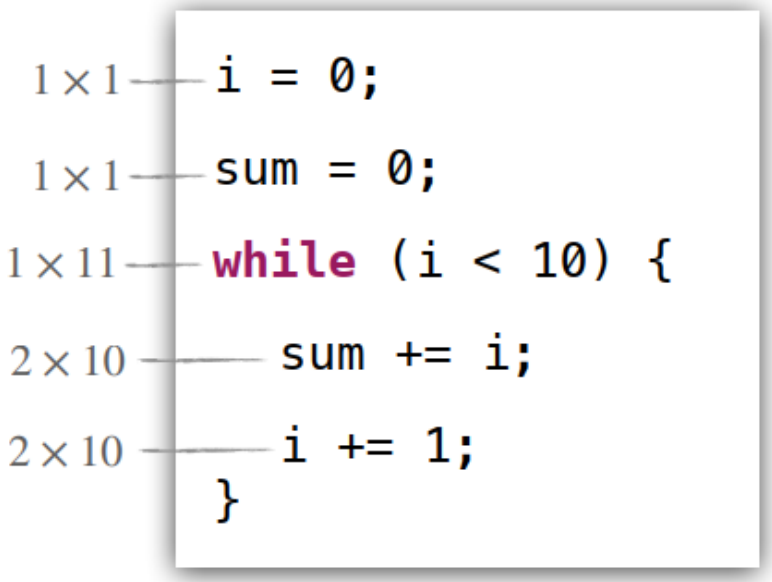
requires math skills

How Many Operations?

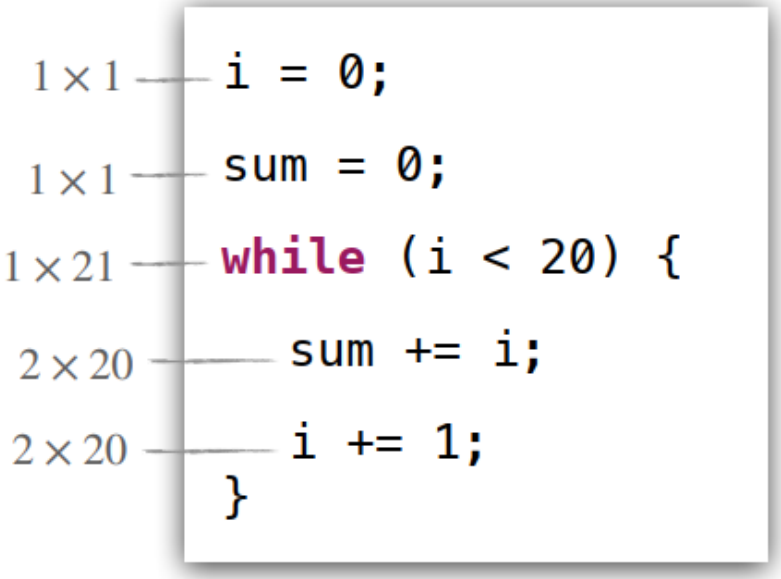
```
i = 0;
sum = 0;
while (i < 10) {
   sum += i;
   i += 1;
}
```

```
i = 0;
sum = 0;
while (i < 20) {
   sum += i;
   i += 1;
}
```

How many operation?



$2 + (1 \times 11) + (4 \times 10) =$
53 operations



$2 + (1 \times 21) + (4 \times 20) =$
103 operations

For simplicity, we will say:

- the left code performed the sum += i operation 10 times.
- the right code performed the sum += i operation 20 times.

We will always pick a certain operation to be the basis for our cost model.

How Many Operations?

How many times does sum += i get executed?

```
i = 0;
sum = 0;
while (i<5) {
   sum += i;
   i += 1;
}
```

5 times

```
i = 10;
sum = 0;
while (i>0) {
   sum += i;
   i -= 1;
}
```

10 times

```
i = 0;
sum = 0;
while (i<n) {
   sum += i;
   i += 1;
}
```

n times

Note: In all of the examples, n is assumed to be positive

How Many Operations

How many times does op() get called?

```
i = 100;
while (i<n) {
   op();
   i += 1;
}
```

$n - 100$ times

```
i = 0;
while (i<n) {
   op();
   i += 5;
}
```

$\lceil n / 5 \rceil$ times

```
i = 100;
while (i<n) {
   op();
   i += 5;
}
```

$\lceil (n - 100) / 5 \rceil$ times

How Many Operations

How many times does op() get executed?

```
for (int i=0; i<n; i++)
   op();
```

n

```
for (int i=0; i<n; i+=5)
   op();
```

$\lceil n/5 \rceil$

How Many Operations?

How many times does op() get called?

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

$2n$

```
for (int i=0; i<n; i+=3) {
   op();
   op();
   op();
}
```

n

assuming n is a multiple of 3. If not, then the answer is: $[n/3] \times 3$

How Many Operations?

How many times does op() get called?

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

n^2

```
for (int i=0; i<n; i++)
      op();
for (int j=0; j<n; j++)
      op();
```

$2n$

How Many Operations?

How many times does op() get called? (assuming n is a multiple of 2)

```
for (int i = 10; i < n; i++) {
   for (int j = 5; j < n; j += 2)
      op();
}
```

$$(n - 10) \times \frac{1}{2}(n - 5)$$

for all $n > 10$, 0 otherwise

How Many Operations?

How many times does op() get called?

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j += 2)
      op();
   for (int j = 0; j < n; j += 2)
      op();
}
```

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

How Many Operations?

How many times does op() get called? (assuming n is a multiple of 2)

```
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j += 2)
      for (int k = 10; k < n; k++)
         op();
```

$$n \times \frac{1}{2}n \times (n - 10) = \frac{1}{2}n^3 - 5n^2$$

for all $n > 10$, 0 otherwise

How Many Operations?

How many times does op() get called? (assuming n is a multiple of 2)

```
for (int i = 0; i < n*n; i++)
     op();

for (int i = 0; i < n; i += 2)
   for (int j = 0; j < n; j += 2)
     op();
```

$$n^2 + (\frac{1}{2}n \times \frac{1}{2}n)$$

$$= n^2 + \frac{1}{4}n^2$$

$$= \frac{5}{4}n^2$$

How Many Operations?

How many times does op() get called?

```
for (int i = 0; i < n; i++)
  for (int j = i; j < i + 7; j++)
    op();
```

$7n$ (the inner loop always repeats 7 times, regardless of what the value of i is)

```
for (int i = 0; i*i < n; i++)
    op();
```

$\sqrt{n}$ (the loop stops when *i*2 = *n* i.e. when *i* = *n*)

i* i<n

i* i >= n

$i^2 = n$

i= $\sqrt{n}$

Frequency count method

```
Algorithm  sum(A,n)
{     s=0;
      for(i=0;s<=n;i++)
       {
          s=s+ i;
       }

}
```

Assume s>n

S=k(k+1)/2

$k^2 > n$

$k > \sqrt{n}$

i	s
0	0+0=0
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 Operations?

How many times does op() get called? (assuming n is a power of 2)

```
for (int i = 1; i <= n; i *= 2)
    op();
```

$i = 1, \quad 2, \quad 4, \quad 8, \quad \ldots, \quad \frac{1}{2}n, \quad n$

$= 2^0, \quad 2^1, \quad 2^2, \quad 2^3, \quad \ldots, \quad 2^{k-1}, \quad 2^k$

These are $k+1$ steps, where $2^k = n$ i.e. $k = \log_2(n)$

Total number of times `op()` is called = $\log_2(n) + 1$

```
for (int i = n; i >= 1; i /= 2)
    op();
```

$i = n, \quad \frac{1}{2}n, \quad \frac{1}{4}n, \quad \ldots, 8, \quad 4, \quad 2, \quad 1$

$= 2^k, \quad 2^{k-1}, 2^{k-2}, \quad \ldots, 2^3, 2^2, 2^1, 2^0$

These are $k+1$ steps, where $2^k = n$ i.e. $k = \log_2(n)$

Total number of times `op()` is called = $\log_2(n) + 1$

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

$n/2^k$

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i *= 3)
    op();
```

$i = 1, \quad 3, \quad 9, \quad 27, \quad \ldots, \quad n$

$= 3^0, \; 3^1, \; 3^2, \; 3^3, \quad \ldots, \quad 3^k$

These are $k + 1$ steps, where $3^k = n$ i.e. $k = \log_3(n)$

Total number of times `op()` is called = $\log_3(n) + 1$

! In general:

```
for (i = 1; i <= whatever; i *= b)
    op();
```

$\lfloor \log_b(\text{whatever}) \rfloor + 1$

Frequency count method

```
s=0;
for(i=1; i< n;i=i*2)
 {
    s++;                  --------   s=log n
 }
for(j=1 ; j< s; j=j*2)
 {
    statement;  ---------- log s
 }
```

log log n

Frequency count method

```
for(i=0; i< n; i++)   -------- n
 {
   for(j=1;j< n; j=j*2)   ------n logn
    {
     statement;  ---------- n log n
 }                          ----------------
}                           2 n logn+ n
                            0 nlogn
```

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j *= 2)
    op();
```

i	*number of* op() *calls*
1	$\log_2(1) + 1$
2	$\log_2(2) + 1$
3	$\log_2(3) + 1$
...	...
n	$\log_2(n) + 1$

$$\text{Total} = \log_2(1) + \log_2(2) + \log_2(3) + \ \ldots \ + \log_2(n) + (n \times 1)$$

$$= \log_2(1 \times 2 \times 3 \times \ldots \times n) + (n \times 1) = \log_2(n!) + n$$

$$\sim n \log_2(n)$$ ← Stirling's Approximation (see the math cheatsheet)

Frequency count method

for(i=0; i< n; i++) ---------0(n)

for(j=0;j< n; j=j+2) ------ n/2 ----- 0(n)

for(i=n; I >1; i--) ------- 0(n)

for(j=1;j< n; j=j*2) ------ 0 $\log_2(n)$

for(j=1;j< n; j=j*3) ------ 0 $\log_3(n)$

for(j=n ; j> 1; j=j/2) ------ 0 $\log_2(n)$

How Many Operations?

How many times does op() get called? (assuming n is a multiple of 2)

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= i; j++)
      op();
```

If the nested loops are *dependent*, we can't analyze each loop separately and then multiply them!

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    op();
```

If the nested loops are *dependent*, we can't analyze each loop separately and then multiply them!

1 Trace

i	j	*number of* op() *calls*
1		
2		
3		
...		
n		

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    op();
```

1 Trace

i	j	*number of* op() *calls*
1	[1]	1
2	[1, 2]	2
3	[1, 2, 3]	3
...	...	...
n	[1, 2, 3, ..., n]	n

X If the nested loops are *dependent*, we can't analyze each loop separately and then multiply them!

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    op();
```

X If the nested loops are *dependent*, we can't analyze each loop separately and then multiply them!

1 Trace

i	j	*number of* op() *calls*
1	[1]	1
2	[1, 2]	2
3	[1, 2, 3]	3
...	...	...
n	[1, 2, 3, ..., n]	n

Formulate a sum 2 Total $= 1 + 2 + 3 + \ldots + n$

$$= \sum_{i=0}^{n} i$$

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    op();
```

X If the nested loops are *dependent*, we can't analyze each loop separately and then multiply them!

1 Trace

i	j	*number of* op() *calls*
1	[1]	1
2	[1, 2]	2
3	[1, 2, 3]	3
...	...	...
n	[1, 2, 3, ..., n]	n

Formulate a sum 2 $\text{Total} = 1 + 2 + 3 + \ldots + n$

$$= \sum_{i=0}^{n} i$$

$$= \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

3 Solve the sum

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n*n; i++)
  for (int j = 1; j <= i; j++)
    op();
```

i	j	*number of* op() *calls*
1	[1]	1
2	[1, 2]	2
3	[1, 2, 3]	3
...	...	...
n*n	[1, 2, 3, ..., n*n]	n*n

$$\text{Total} = 1 + 2 + 3 + \ldots + n^2$$

$$= \sum_{i=0}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

!

A very frequently encountered sum:

$$\sum_{i=0}^{\star} i = \frac{\star\,(\star+1)}{2}$$

How Many Operations?

How many times does op() get called?

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    for (int k = 1; k <= i; k++)
      op();
```

i	*number of* op() *calls*
1	1 x 1
2	2 x 2
3	3 x 3
...	...
n	n x n

Total = $1^2 + 2^2 + 3^2 + \ldots + n^2$

$$= \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

← see the math cheatsheet