Calculating Algorithm Efficiency

Nested Loops

Quadratic Loop

→ j*=2

j = j*2''

```
for(i=0;i<10;i++)
    for(j=1; j<10;j++)
        statement block;
```

$O(n \log n)$

f(n)= O(n²)

i	j	# iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3
⋮		
n	0, …, n-1	n

Dependent Quadratic Loop

= n

```
for(i=0;i<10;i++)
    for(j=0; j<=i;j++)
        statement block;
```

O(1)

$f(n)= 1 + 2 + \ldots + n = O\,(n\,(n + 1)/2)$

$= O(n^2)$

General Rules

- Simple statement: constant

  i++; i<n; etc.   O(1) i = i*2. i = i/2

- Simple loops: # of iterations times the cost of the loop body = O(1)

  i = 0; While (i < n) { … i++; …}   for

- Nested loops: (the product of # of iterations of outer and inner loops) times (the cost of the inner loop body)

```
for (i=0; i < n; i++)
   for(j=0; j < m; j++)
      k++;
```

O(n) · O(m) · O(1).

① when n = m => O(n²)
② m = f(n)
③ O(n) · O(log n) · O(1).

- Consecutive statements: count the more expensive one

```
i = 0;
while (i < n) { … i++; …}
for (i=0; i < n; i++)
   for(j=0; j < n; j++)
      k++;
```

O(1)
O(n)
O(n)·O(1)
O(n·m)
n·log n

independent combination

O(1) + O(n) + O(n·m) = O(n·m)
O(n·log n)
O(n·log n)

- If/else statement: count the more expensive branch (for worst-case analysis)

switch

if else
O(n) O(n²)

Supplementary

Some Useful Mathematical Equalities

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n*(n+1)}{2} \approx \frac{n^2}{2}$$

$\approx O\left(\frac{n^2}{2}\right) = O(n^2)$

$a_n - a_{n-1} = d$

$a_1 + a_2 + \cdots + a_n$

$= \boxed{\frac{n(a_1 + a_n)}{2}} \rightarrow$ average.

$$\sum_{i=1}^{n} i^2 = 1 + 4 + \ldots + n^2 = \frac{n*(n+1)*(2n+1)}{6} \approx \frac{n^3}{3}$$

$$\sum_{i=0}^{n-1} 2^i = 0 + 1 + 2 + \ldots + 2^{n-1} = 2^n - 1$$

Some Useful Mathematical Equalities

$$1 + x + x^2 + \dots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \dots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

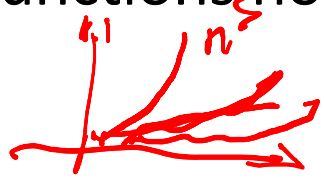
Omega Notation → best case

- Omega notation provides a tight lower bound for f(n). This means that the function can never do better than the specified value but it may do worse.
- Ω notation is simply written as, $f(n) = \Omega(g(n))$, where n is the problem size and $\Omega(g(n)) = \{h(n): \exists$ positive constants $c > 0$, n_0 such that $0 \leq cg(n) \leq h(n), \forall\, n \geq n_0\}$.
- Examples of functions in $\Omega(n^2)$ include: n^2, $n^{2.9}$, $n^3 + n$, $540n^2 + 10$

≤ g(n) ≤ f(n)

c · g(n) ≤ n^3 n^2

- Examples of functions not in $\Omega(n^3)$ include: n, $n^{2.9}$, n^2



O(n^3) Yes.

f(n) $n^a \leq n^b$ if $a \leq b$ & $n \geq 1$

Omega Ω

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

$\exists\ c > 0,\ n_0 > 0$

$0 \le c\, g(n) \le f(n)\ \forall n \ge n_0$

lower

best case.

Big O O

if $f(n) = O(g(n))$

$\exists\ c > 0,\ n_0 > 0$

$f(n) \le c \cdot g(n)\ \forall n \ge n_0$

upper

worst case