Algorithm Analysis

Big O $T(N) = O(f(n))$

If there are positive constants c & n_0 s.t.

$T(N) \leq c f(N)$ where $N \geq n_0$

Big O provides an upperbound on a function

But you have to get as close as possible

Big Ω $T(N) = \Omega(g(n))$

If there exist positive constants c and n_0 s.t.

$T(N) \geq c g(N)$ where $N \geq n_0$

Big Ω provides a lowerbound on a function

Proving a lower bound is very hard!

General-purpose sorting algorithms are lower-bounded by $N(\log N)$ or $\Omega(N(\log N))$

Big Θ $T(N) = \Theta(h(n))$

If and only if $T(N) = O(h(n))$ & $T(N) = \Omega(h(n))$

This is a "tight bound" on a function on a function

Why Big O? It's easier to show an upperbound rather than a tight bound?

Rules of Big O

Rule 1: $T_1(N) = O(f(N))$

$T_2(N) = O(g(N))$

a. $T_1(N) + T_2(N) = O(f(N) + g(N)) \approx O(\max(f(N), g(N)))$

↳ ∴ we're allowed to drop the lower terms

b. $T_1(N) * T_2(N) = O(f(N) * g(N))$

↳ outer loop ↳ inner loop ⇒ this rule corresponds to nested loops!

Rule 2: If $T(N)$ is a polynomial of degree k then

$T(N) = \Theta(N^k)$

Rule 3: $\log^k N = O(N)$ → doesn't matter how big k is!

↳ log will not grow faster than linear!

log is very slow!

It's upperbounded by $O(N)$

Heuristic Techniques

```
for (int i=0; i<N; i++) {
      a++;
}
```

constant cost * N — $O(N)$

```
for (int i=0; j<N*N*N; j++) {
      a++
}
```

$O(N^3)$

The cost of the whole thing would be: $O(N^3)$

```
for (int i=0; i<N; i++) {
      for (int j=0; j<N*N; j++) {
            a++;
      }
}
```

$O(N^3)$

```
for (int i=0; i<N; i++) {
      for (j=i; j<N; j++) {
            a++;
      }
}
```

$N + (N-1) + (N-2) + (N-3) + \dots + 1 = \frac{N(N+1)}{2} \Rightarrow O(N^2)$

```
if (condition) {
      S1
} else {
      S2
}
```

↳ Analyze S1 and S2 independently and choose whichever one is worse

↳ Sometimes the code gives us a clue which one occurs more frequently

```
for (i=0; i<N; i++) {
      if (i==3) {
            for (j=0; j<N*N*N; j++) {
                  a=a+1;
            }
      } else {
            for (j=0; j<N; j++) {
                  a=a+1;
            }
      }
}
```

this is the worst one

this happens once and then you do the rest

⋮

at the end it dominates the rest ($O(n^2)$)

This condition is worse but it happens only once!

↳ you can factor it out from the rest

↳ So it's not actually $O(n^4)$

$\Rightarrow O(N^3)$

$O(N^2)$ because $N*N$