BIRZEIT UNIVERSITY

Algorithm Analysis

Dr. Abdallah Karakra

Computer Science Department

COMP242

Tuesday, March 5, 2024

Mathematical Background

Logs and exponents

We will be dealing mostly with binary numbers (base 2)

Definition: $\log_X B = A$ means $X^A = B$

SELF STUDY

Mathematical Background

Properties of logs

We will assume logs to base 2 unless specified otherwise

$\log AB = \log A + \log B$ (note: $\log AB \neq \log A \bullet \log B$)

$\log A/B = \log A - \log B$ (note: $\log A/B \neq \log A / \log B$)

$\log A^B = B \log A$ (note: $\log A^B \neq (\log A)^B = \log^B A$)

SELF STUDY

Mathematical Background: Sums

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

$$f(n) = 1 + 3 + 5 + \ldots + (2n-1) = \sum_{i=1}^{n} (2i-1) = n^2$$

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

$$f(n) = 1^4 + 5^4 + 9^4 + \ldots + (4n-3)^4 = \sum_{i=1}^{n} (4i-3)^4$$

Sum of squares: $\displaystyle\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$ for large N

Mathematical Background: Sums

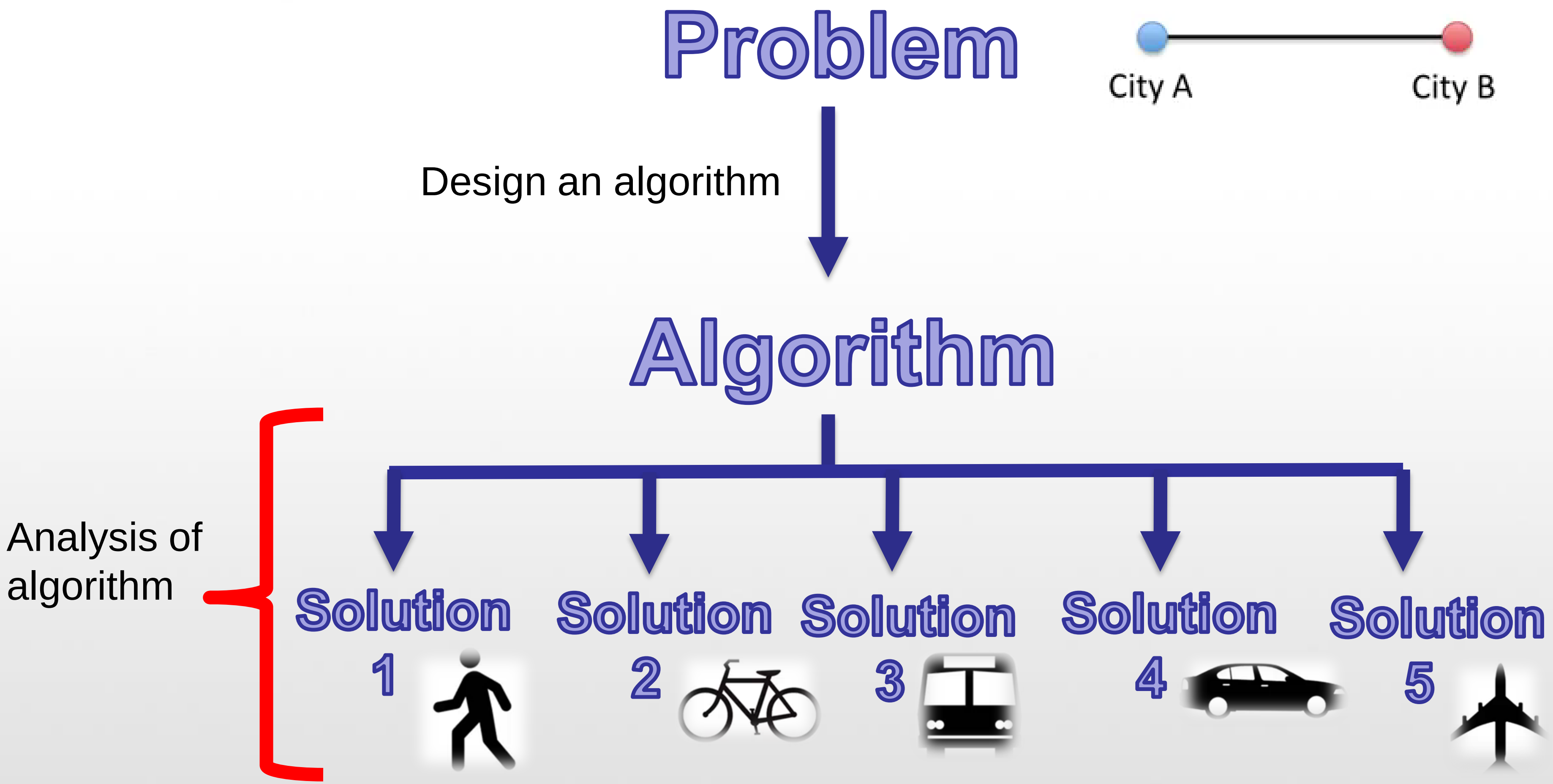
$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$

$$\sum_{k=0}^{n-1} x^k = \frac{x^n - 1}{x - 1} (x \neq 1)$$

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \quad \text{if } x < 1$$

Algorithm Analysis: Motivation



Algorithm Analysis: Motivation

To select the best algorithm from many you have to do Analyzing

❖ **Time:** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm.

> The better algorithm is the algorithm that run in less time

❖ **Space:** The number of memory cells which an algorithm needs.

> A good algorithm keeps this number as small as possible, too (consume less memory)

Algorithm Analysis: Motivation

In this course we will focus on time complexity instead of space.

How fast will your program run?

The running time of your program depends on:

1. Algorithm design
2. Input Size
3. Programming Language
4. The compiler you use
5. The OS on your Computer
6. Your computer Hardware (CPU, RAM, Busses)

We Need Fair Comparison

How fast will your program run?

Time complexity: computational complexity that measures or estimates the time taken for running an algorithm.

Complexity can be viewed as the maximum number of primitive operations that a program may execute.

Common time complexities

$O(1)$	Constant time
$O(\log n)$	Logarithmic time
$O(\log^2 n)$	Log-squared time
$O(n)$	Linear time
$O(n^2)$	Quadratic time
$O(n^3)$	Cubic time
$O(n^i)$ for some i	Polynomial time
$O(2^n)$	Exponential time

Higher order functions of n are normally considered less efficient.

How fast will your program run?

Approximate time to run a program with n inputs on 1GHz machine:

Function	n = 10	n = 50	n = 100	n = 1000	n = 10^6
O(n log n)	35 ns	200 ns	700 ns	10000 ns	20 ms
$O(n^2)$	100 ns	2500 ns	10000 ns	1 ms	17 min
$O(n^5)$	0.1 ms	0.3 s	10.8 s	11.6 days	3×10^{13} years
$O(2^n)$	1000 ns	13 days	4×10^{14} years	*Too long!*	*Too long!*
O(n!)	4 ms	*Too long!*	*Too long!*	*Too long!*	*Too long!*

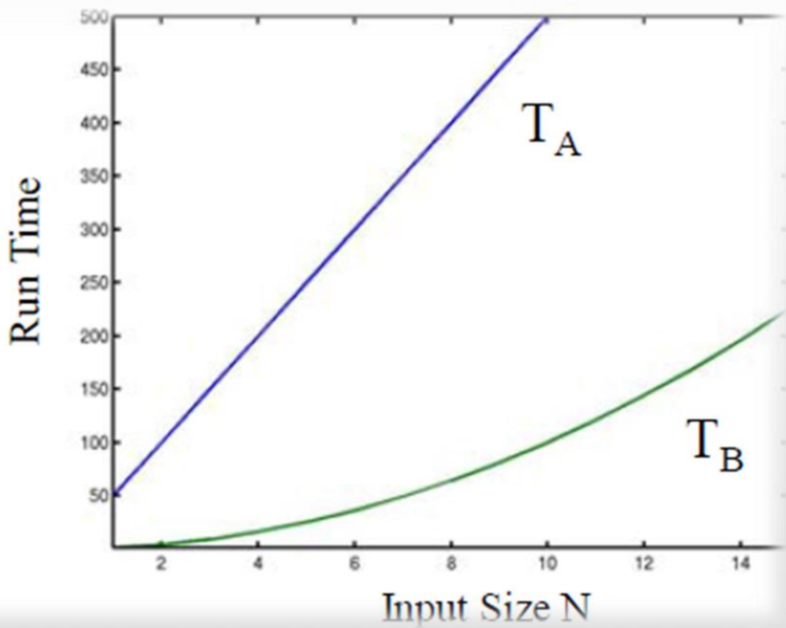
n: input size(data)

Algorithm Analysis: Motivation

Suppose you are given two algorithms A and B for solving a problem

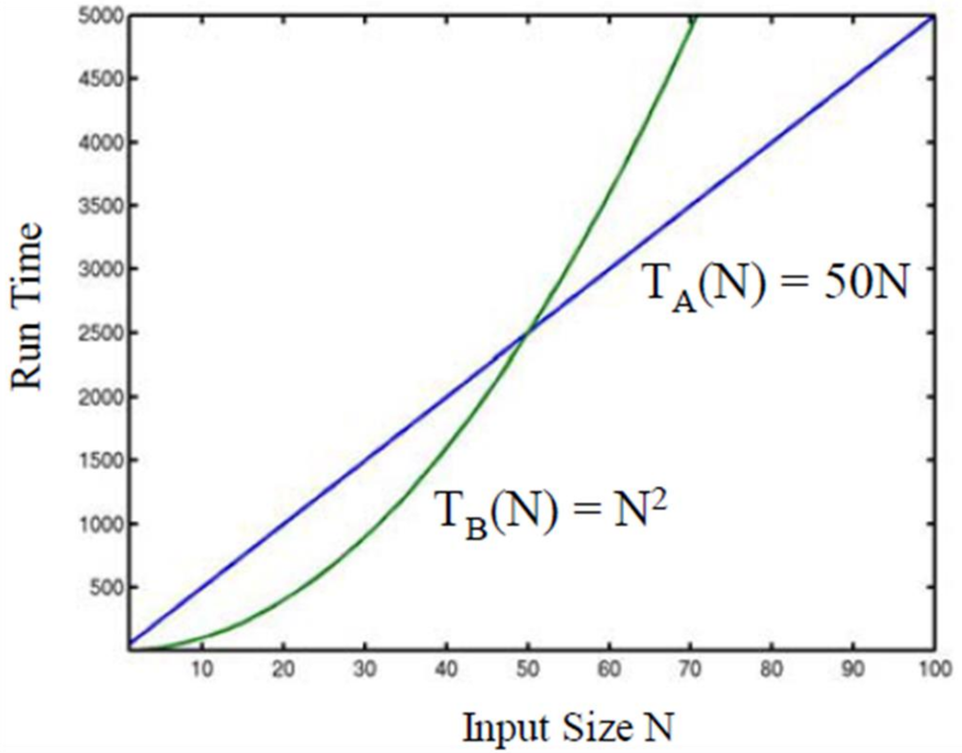
Here is the *running time* $T_A(N)$ and $T_B(N)$ of A and B as a function of input size N:

Which algorithm would you choose?



Algorithm Analysis: Motivation

For large N, the running time of A and B is:



Now which algorithm would you choose?

Algorithm Analysis: Running time Analysis

Suppose that T(n) is a function of time at a given input size n (size of data)

```
Algorithm Add (a,b){
  return a+b;          → 1 unit
}
```

In this case : T(n)=1

Constant

Running time is independent of the number of data items

```
Algorithm Add (a,n){
   s=0;              → 1
    for i=1 to n do {  → n +1
      s=s + A[i];    → n
   }
   return s;  → 1
}
```

In this case : T(n)=2n+3

Polynomial function with degree 1
Linear function

Algorithm Analysis: Running time Analysis

Suppose that T(n) is a function of time at a given input size n (size of data)

```
Algorithm MatAdd (a,b,c){
    for i=1 to n do {   → n+1
       for j=1to n do { → n(n+1)
           c[i][j]=a[i][j]+B[i][j]; → n^2
        }
    }

}
```

In this case : $T(n)=2n^2+2n+1$

Polynomial function with degree 2

Highest degree of this polynomial is n^2 , so we called quadratic

Algorithm Analysis: Running time Analysis

Suppose that T(n) is a function of time at a given input size n (size of data)

one machine an algorithm may take

$T(n) = 15n^3 + n^2 + 4$

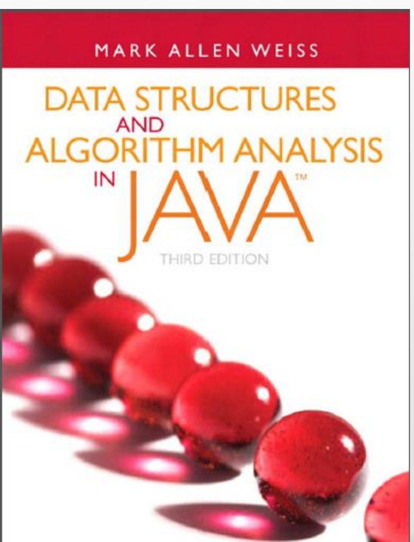
On another, $T(n) = 5n^3 + 4n + 5$

Both will belong to the same class of functions. Namely, "cubic functions of n".

Algorithm Analysis: 2.1 Mathematical Background

On board

p. 30,31,32 textbook



Asymptotic Notation

T(n)

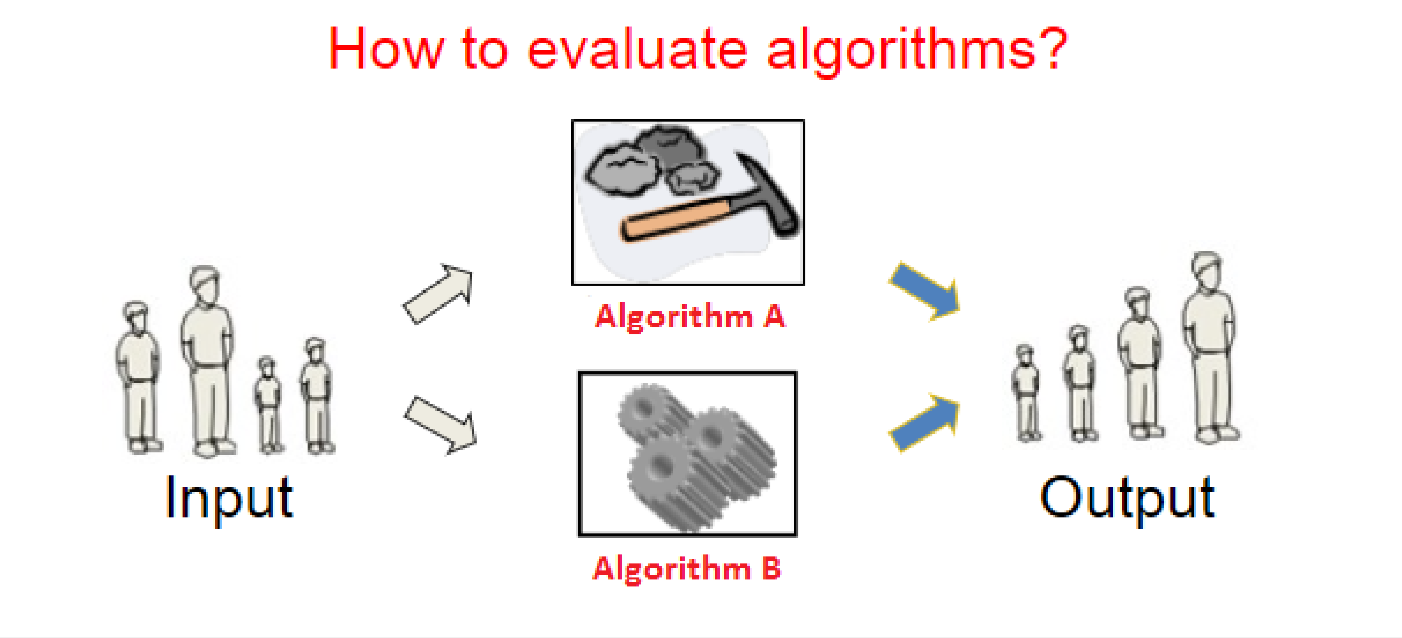
Asymptotic Notation is a formal notation for discussing and analyzing “classes of functions”.

- Its use in analyzing runtimes

- " Big-O" notation : $O(n)$
- " Big-Omega of n " : $\Omega(n)$
- " Theta of n" : $\Theta(n)$

Asymptotic Notation

- Its use in analyzing runtimes.



Asymptotic Notation

We will refer to the running time as *T(N)*

Definition 2.1.
$T(N) = O(f(N))$ if there are positive *constants* c and n_0 such that $T(N) \le cf(N)$ when $N \ge n_0$.

Definition 2.2.
$T(N) = \Omega(g(N))$ if there are positive *constants* c and n_0 such that $T(N) \ge cg(N)$ when $N \ge n_0$.

Definition 2.3.
$T(N) = \Theta(h(N))$ if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

Definition of Order Notation

- **Upper bound**: $T(n) = O(f(n))$ Big-O

  Exist constants c and n_0 such that

  $T(n) \le c\, f(n)$ for all $n \ge n_0$

- **Lower bound**: $T(n) = \Omega(g(n))$ Omega

  Exist constants c and n_0 such that

  $T(n) \ge c\, g(n)$ for all $n \ge n_0$

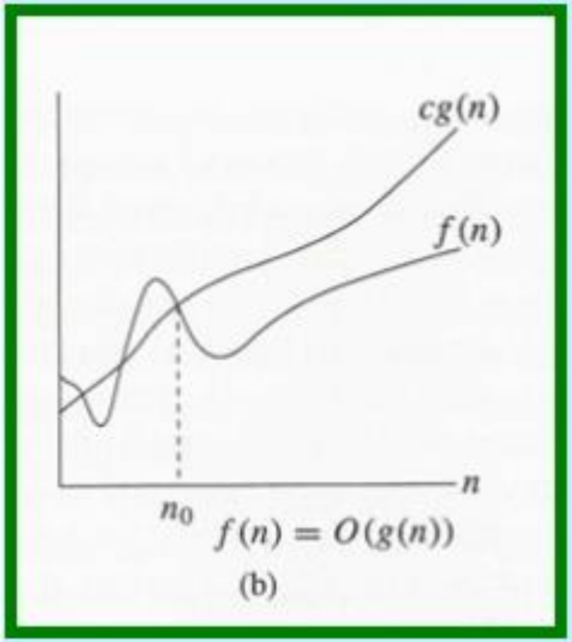
- **Tight bound**: $T(n) = \Theta(f(n))$ Theta

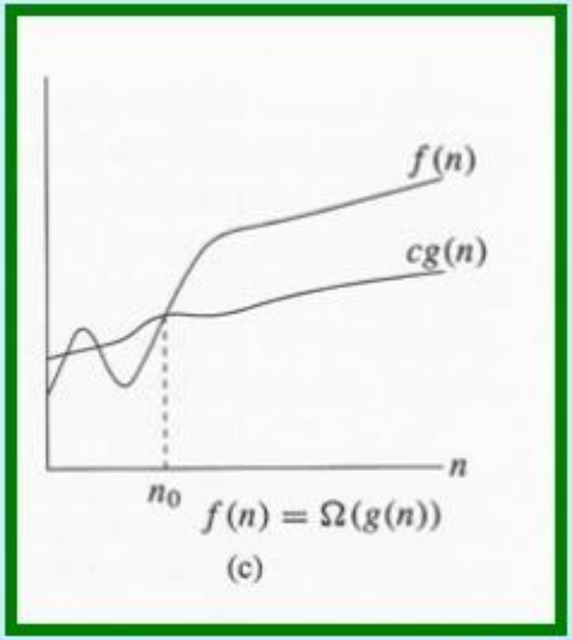
  When both hold:

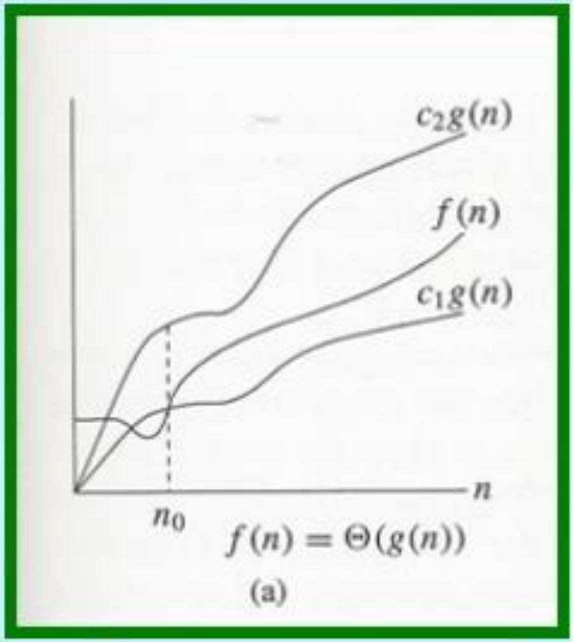
  $T(n) = O(f(n))$

  $T(n) = \Omega(f(n))$

Asymptotic Notation







Asymptotic Upper Bound

Big-O Notation

Asymptotic Lower Bound

Big-Ω Notation

Tight Asymptotic Bound

Big-Θ Notation

Example: Upper Bound

Claim: $n^2 + 100n = O(n^2)$

Proof: Must find c, n_0 such that for all $n > n_0$,

$$n^2 + 100n \le cn^2$$

Let's try setting $c = 2$. Then

$$n^2 + 100n \le 2n^2$$

$$100n \le n^2$$

$$100 \le n$$

So we can set $n_0 = 100$ and reverse the steps above.

Example: Lower Bound

Claim: $n^2 + 100n = \Omega(n^2)$

Proof: Must find c, n_0 such that for all $n > n_0$,

$$n^2 + 100n \geq cn^2$$

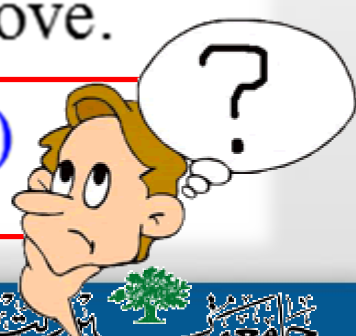
Let's try setting $c = 1$. Then

$$n^2 + 100n \geq n^2$$

$$n \geq 0$$

So we can set $n_0 = 0$ and reverse the steps above.

Thus we can also conclude $n^2 + 100n = \theta(n^2)$



Common time complexities

BETTER

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

WORSE

Better

Comparing the asymptotic running time

Ex: O(log n) is better than O(n)

$$\log n \ll n \ll n^2 \ll n^3 \ll 2^n$$

Rules

1. Eliminate low order terms

   - $4n + 5 \Rightarrow 4n$
   - $0.5\ n \log n - 2n + 7 \Rightarrow 0.5\ n \log n$
   - $2^n + n^3 + 3n \Rightarrow 2^n$

2. Eliminate constant coefficients

   - $4n \Rightarrow n$
   - $0.5\ n \log n \Rightarrow n \log n$
   - $n \log (n^2) = 2\ n \log n \Rightarrow n \log n$

Rules

If $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$, then

1. $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$

   $O(n^3) + O(n^4) = \max(O(n^3), O(n^4)) = O(n^4)$

2. $T_1(n) * T_2(n) = O(f(n) * g(n))$

   $O(n^3) * O(n^4) = O(n^3 * n^4) = O(n^7)$

Analyzing Code

I want to do some code examples, but first, how will we examine code.

- primitive operations
- consecutive statements
- function calls
- conditionals
- loops
- recursive functions