Analysis and Design of Algorithms

Session 1:

Luis Fernando Gutiérrez Preciado.

Analysis and Design of Algorithms

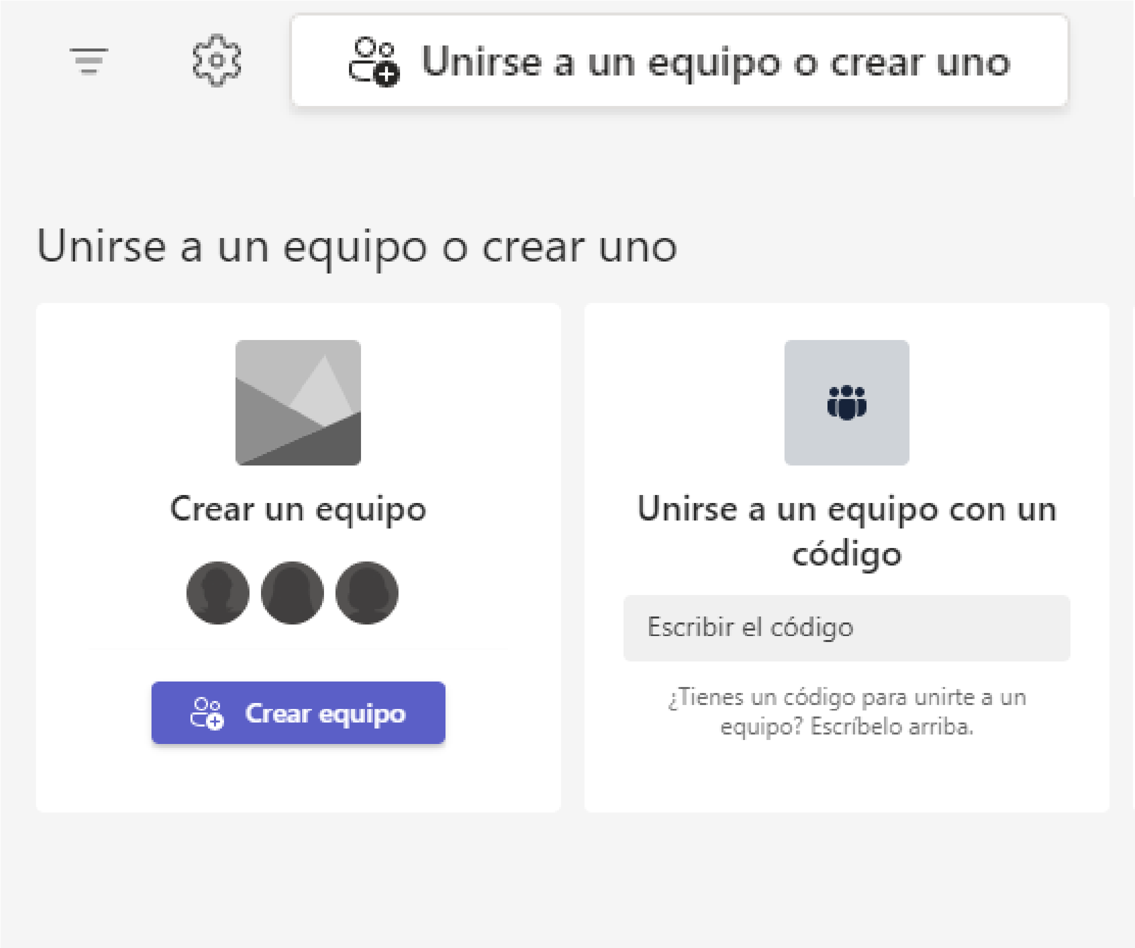
- Professor's Introduction
- Subject Presentation:
  - Learning Guide
- Introduction
- Motivation
- Algorithm Analysis
  - Classification
  - Types of Analysis
  - Asymptotic Notation

Why the class in English?

- To practice English
- As preparation for technical interviews
- I speak more slowly
- I don't feel bad if no one laughs at my bad jokes

MsTeams Group

- **5di591m**



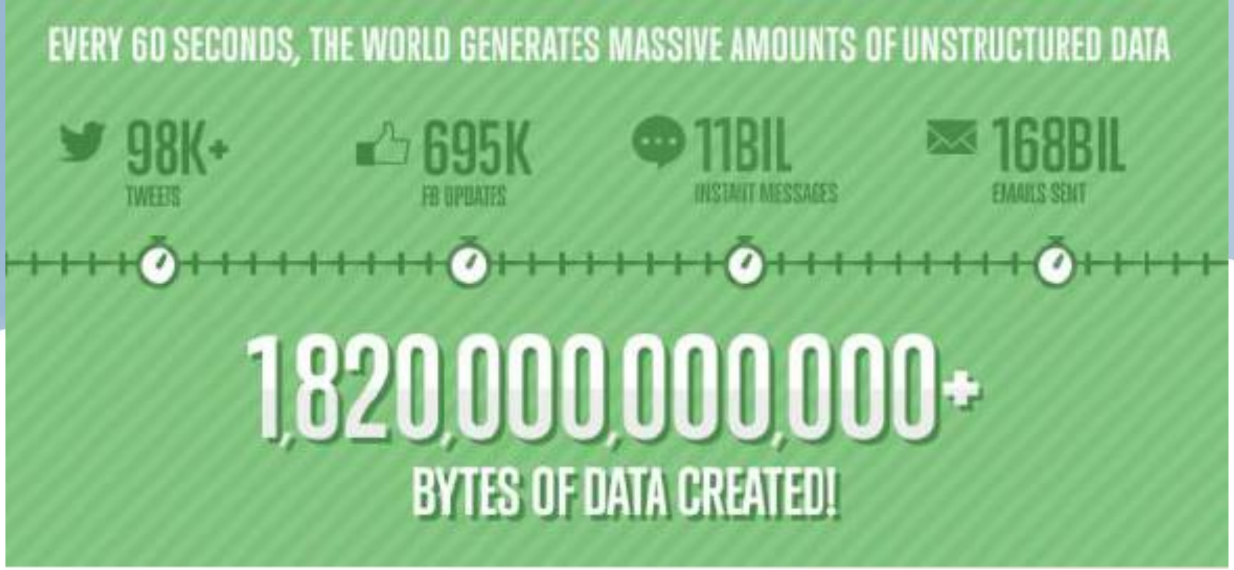
Objectives

- Importance of studying this subject
- To understand and apply asymptotic notation to describe the efficiency of algorithms.

Introduction

- Does analyzing algorithms still make sense given the technological advances?
  - New, faster processors are designed every year.
  - Programs can be parallelized using graphics processors (CUDA, OpenCL, Direct Compute).
  - Clusters of hundreds of interconnected machines can be utilized.
  - The software we use daily is often efficient enough for its intended purposes.
  - It seems like there are already efficient algorithms for everything.
  - AI already assists in coding.
- So, what now?

Motivation



- Big Data
- Unstructured Information
- Real-time Applications
- Leverage the technology you have now
- Cloud Computing - pay-as-you-go, the less resources you use, the lower the cost.

Motivation

- Retrieve information from the web
  - Google: search engines
- Cryptography
- Politics: where to invest more to gain more voters
- Bank: identify customers most likely to acquire a loan or credit card
- Security cameras:
  - detect intruders by analyzing their face
  - Perform a search in security videos
- Social networks:
  - identify the most influential users
  - What do they think about a certain topic?
- Photo-realism in computerized images
- More realistic video games
- Recommendation systems

Algorithm analysis

- One problem -> many solutions
  - If all are equally **effective** or **accurate**, which is the best?
  - The best is the most **efficient** one.
- A Mathematical Notation is proposed (**a priori** analysis):
  - Which algorithm is more likely to take less time under similar circumstances of HW, SW, and amount of information?
  - According to the resources used: the number of executed instructions, occupied memory, duration, and what is the maximum problem size that the solution allows.

Algorithm analysis

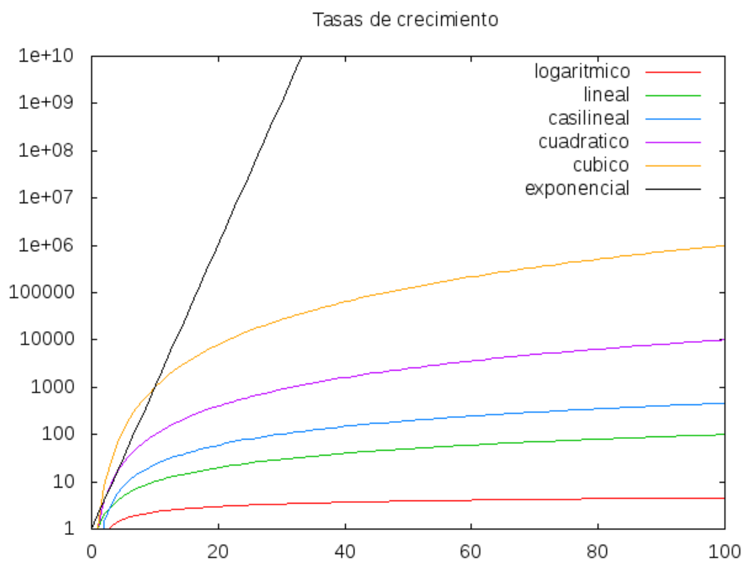
- For the analysis, we will consider two types of complexity:
  - **Temporal**: number of executed instructions
  - **Spatial**: occupied memory
- In many problems, we will be interested in analyzing the complexity of the solution in the **best**, **worst**, and **average** cases.
- For many algorithms, their *complexity* in these three cases has already been identified.
  - There are others for which the analysis is very complicated and there is no consensus.

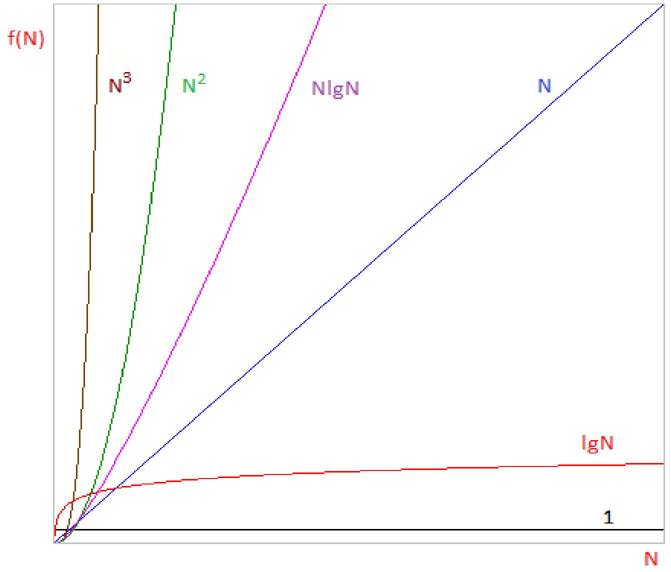
Classification of Algorithms

- The algorithms of interest have a parameter N
  - It represents the size of the problem
  - Affects the execution time
- N can represent
  - The number of rows of a square matrix
  - The size of a file to be sorted
  - The number of nodes in a graph
  - The degree of a polynomial ...

Classification of Algorithms

- Well-known execution times :
  - Constant (K)
  - Logarithmic ($\log_b N$)
  - Linear (N)
  - Quasilinear ($N \log_b N$)
  - Quadratic (N^2)
  - Cubic (N^3)
  - Exponential (b^N)





Classification of Algorithms

- Constant (K)
  - The algorithm's instructions are executed a fixed number of times, regardless of the value of N.
- Logarithmic ($\log_b N$)
  - As N increases, the execution time grows at a progressively slower rate.
  - Algorithms that solve large problems by breaking them down into smaller problems that are solved in constant time.
  - If b = 2, we write it as **lg N**. In this case, if N is doubled, how much does the execution time increase?

Classification of Algorithms

- Linear (N)
  - Each input data is processed a fixed number of times.
  - If N is doubled, the execution time doubles as well.
- Quasilinear ($N \log_b N$)
  - Algorithms that solve large problems by breaking them down into smaller problems that are solved in linear time.
  - If b = 2, N = 1024, what is the execution time? If N is doubled? What happens to the time?
    - The resulting time is greater than double but less than triple for all N > b: $2T(N) < T(2N) < 3T(N)$

Clasificación de algoritmos

- Quadratic (N^2)
  - Algorithms that involve two nested loops.
  - All or nearly all possible pairs of input data are processed in constant time.
  - If N is doubled, what happens to the time?
  - The resulting time becomes four times greater: $T(2N) = (2N)^2 = 4N^2$.
- Cubic (N^3)
  - Algorithms that involve three nested loops.
  - Tuples of data, or pairs of data, are processed in linear time.
  - If N is doubled, what happens to the time?
  - The resulting time becomes eight times greater: if $T(N) = N^3$, then $T(2N) = (2N)^3 = 8N^3$.

Clasificación de algoritmos

- Exponential (b^N)
  - Algorithms that use brute force to find one or more objects of size N that meet a certain requirement. Each element of the object can take b possible values.
    - Optimizing an N-dimensional function in a search space of size b for each dimension.
    - N-Queens Problem
    - Finding a way to exit a maze.
  - If N is doubled, what happens to the time?
    - The resulting time increases squared: if $T(N) = b^N$, then $T(2N) = b^{2N}$.

Time Comparison

$T(n)$	$n = 100$	$n = 200$
$k_1 \log n$	1 h	
$k_2\, n$	1 h	
$k_3\, n \log n$	1 h	
$k_4\, n^2$	1 h	
$k_5\, n^3$	1 h	
$K_6\, 2^n$	1 h	

$t = 1$ h	$t = 2$ h
$n = 100$	
$n = 100$	
$n = 100$	
$n = 100$	
$n = 100$	
$n = 100$	

Time Comparison

$T(n)$	$n = 100$	$n = 200$	$t = 1\ h$	$t = 2\ h$
$k_1 \log n$	$1\ h$	1	$n = 100$	All the teams
$k_2\ n$	$1\ h$		$n = 100$	
$k_3\ n \log n$	$1\ h$	2	$n = 100$	
$k_4\ n^2$	$1\ h$		$n = 100$	Optionals
$k_5\ n^3$	$1\ h$	3	$n = 100$	
$K_6\ 2^n$	$1\ h$		$n = 100$	

Time Comparison

	$T(n)$	$n = 100$	$n = 200$	$t = 1\ h$	$t = 2\ h$
Logarítmico	$k_1 \log n$	$1\ h$	$1{,}15\ h$	$n = 100$	$n = 10000$
Lineal	$k_2\ n$	$1\ h$	$2\ h$	$n = 100$	$n = 200$
Quasi-lineal	$k_3\ n \log n$	$1\ h$	$2{,}30\ h$	$n = 100$	$n = 178$
Cuadrático	$k_4\ n^2$	$1\ h$	$4\ h$	$n = 100$	$n = 141$
Cúbico	$k_5\ n^3$	$1\ h$	$8\ h$	$n = 100$	$n = 126$
Exponencial	$K_6\ 2^n$	$1\ h$	$1{,}27 x 10^{30}\ h$	$n = 100$	$n = 101$

Types of Analysis

- To measure the efficiency of an algorithm, we can conduct two types of analysis:

1. A priori
   - Applied during the algorithm design stage.
   - Obtains a mathematical expression that bounds the calculation time, using **asymptotic notation**.
2. A posteriori
   - Multiple runs of the already implemented algorithm are performed, using various values of N.
   - Statistics for time and space consumed and the number of (relevant) operations carried out in each case are reported.

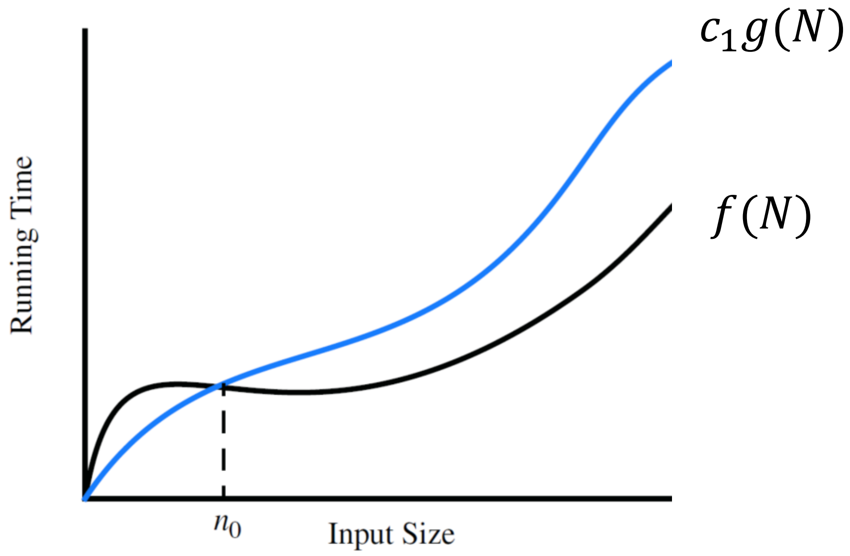
Asymptotic notation

- Purpose: To identify and specify the **temporal** and **spatial complexity** order to which an algorithm belongs.
- The execution time of an algorithm won't be exactly equal to any of those seen, but it will be proportional.
  - An algorithm that executes $2N^2$ instructions belongs to the quadratic complexity order.
  - It differs by a constant factor (2) from the reference time (N^2), and it's written like this: $2N^2 \in O(N^2)$.

Asymptotic notation

- A function f(N) is in the order of g(N), f(N) ∈ O(g(N)), if there exist positive constants c1 and N0 such that:

$$f(N) \le c_1\, g(N) \text{ for all } N > N_0.$$



- Prove: $2N^2 \in O(N^2)$

  $f(N) = ?$ $g(N) = ?$ $c_1 = ?$ $N_0 = ?$

Asymptotic notation

- Prove: $2N^2 \in O(N^2)$

  $f(N) = 2N^2$, $g(N) = N^2$

  $c_1 = 3$, $N_0 = 1$.

  $2N^2 < 3N^2$ holds for all $N > 1$

  [N = 2] 8 < 12, [N = 3] 18 < 27, …

  $\therefore$ **$2N^2 \in O(N^2)$**

P vs NP

- If the complexity order of an algorithm is **Exponential**, we say that the problem is solved in **Non-Polynomial (NP)** time.
- For the other six classes, the time is **Polynomial (P).**
- A time is Polynomial if it can be expressed through an equation where N is involved only in: additions, subtractions, multiplications, divisions, and/or logarithms.
  – For example: $3N^2 - 2 \log N + 5N$.

Exercises

- Prove that: $f(N) = 4N \in O(N^2)$.

- Prove that: $f(N) = 2N^4 \notin O(N^3)$.
  - By solving for c0, it is dependent on N, thus not constant.

- Prove that: $f(N) = N\log_2 N + 5N \in O(N^2)$

Analyzing an algorithm

- An algorithm is composed of one or more instances of the following
  1. Variable declarations, structures, functions, ...
  2. Assignment, arithmetic, relational, logical, and bitwise operations: =, +, -, *, /, %, +=, ++, >, ==, !, &&, ||, >>, &, |, ...
  3. Input/output operations and array access: printf, cin, cout, a[i] ...
  4. Conditional structures: if, switch, else if, case, ...
  5. Iterative structures: for, while, do/while, repeat, ...
  6. Function calls

Analyzing an algorithm

- For the analysis of time complexity, we will consider that the duration of any of the first three components (declarations and operations) will be bounded by a constant K
  - Its duration will not depend on the problem size (N)
  - To simplify the analysis, we can assign a value of **1** to the duration, which will not affect the result: the **order of time complexity** to which the solution belongs.
  - The purpose is not to measure how long an algorithm takes to perform an arithmetic operation (dependent on HW), but *how many arithmetic operations* it had to *perform* to arrive at the solution.

Analyzing an algorithm

- The duration of **conditional structures** will depend on the chosen path.
  - Hence, we consider the best, worst, and average cases.
  - If we analyze the worst case, the duration will be equal to a constant (logical expression defining the path) plus the duration of the longest possible path.
- The duration of **function calls** will be equal to a constant plus the duration of the function itself.

Analyzing an algorithm

- The duration of iterative structures will depend on the problem size (N):
  - `for(i = 0; i < N; i ++) cout << i;`
    - One assignment operation: i = 0
    - N + 1 relational operations: i < N
    - N arithmetic operations: i ++
    - As there are N iterations, there are N input/output operations: cout << i
    - Duration: 1 + (N + 1) + N + N = **3N + 2**

Analyzing an algorithm

- What is the duration of an algorithm that calculates the standard deviation of an array of N-given numbers?

$$\sigma^2 = {}^{1}/_{N} \sum_{i=1}^{N} (x_i - \hat{x})^2$$

```
double prom = 0, desv = 0;
for(int i = 0; i < N; i ++)
    prom += arr[i];
prom /= N;
for(int i = 0; i < N; i ++)
    desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

Analyzing an algorithm

- What is the duration of an algorithm that calculates the standard deviation of an array of N-given numbers?

$$\sigma^2 = {}^{1}\!/_{N} \sum_{i=1}^{N} (x_i - \hat{x})^2$$

`double prom = 0, desv = 0;`	2
`for(int i = 0; i < N; i ++)`	1+(N+1)+N
`   prom += arr[i];`	N (2)
`prom /= N;`	1
`for(int i = 0; i < N; i ++)`	1+(N+1)+N
`   desv += (arr[i] – prom) * (arr[i] - prom);`	N(6)
`desv = sqrt(desv / N);`	3

Analyzing an algorithm

- In conclusion, it requires **12N + 10** steps to calculate the standard deviation of an array of N elements.
- Prove that the time complexity order of the solution is **linear**, i.e., **12N + 10** $\in$ **O(N)**.

Analyzing an algorithm

- Now, how much memory does the algorithm that calculates the standard deviation of a given array of N numbers occupy?

```
double prom = 0, desv = 0;
for(int i = 0; i < N; i ++)
    prom += arr[i];
prom /= N;
for(int i = 0; i < N; i ++)
    desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

Analyzing an algorithm

- Now, how much memory does the same algorithm occupy?
  - Here, we will consider the maximum space occupied *at once*: hence, we count the variable **i** only once.

```
double prom = 0, desv = 0;                2: prom, desv
for(int i = 0; i < N; i ++)               1: i
   prom += arr[i];                        N: arr
prom /= N;
for(int i = 0; i < N; i ++)
   desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

Analyzing an algorithm

- In conclusion, it requires **N + 3** memory slots to calculate the standard deviation of an array of N elements.
  - N to store the list,
  - 3 for storing the mean, standard deviation, and the counter *i*
- Prove that the spatial complexity order of the solution is **linear**, i.e., $\mathbf{N + 3} \in \mathbf{O(N)}$.

Exercises

- What is the time and space complexity order of known algorithms that solve the following problems?
  1. Subtraction of N x N matrices.
  2. Summation of the first N natural numbers.
  3. Determining if the element located in the middle of an array of N numbers is divisible by 3.
  4. Calculation of the dot product of two vectors of size N.
  5. Storing all possible combinations <Shirt, Pants, Shoes> with N1 shirts, N2 pants, N3 shoes (N1 ≈ N2 ≈ N3).
  6. Given a drawer with N pieces of jewelry, in how many different ways can I fill a case that only fits 2 pieces?
  7. Storing each of the different ways referred to in problem 6.

Exercices

- What is the time and space complexity order of known algorithms that solve the following problems?
  1. **Subtraction of N x N matrices.**
     - a. Time. QUADRATIC: N^2 subtractions are performed through two nested loops from 1 to N, one iterating over rows and the other over columns.
     - b. Space. QUADRATIC: $3N^2$ to store 3 matrices.
  2. **Summation of the first N natural numbers.**
     - Time. CONSTANT: If the summation formula is known: N(N + 1) / 2. If not known, the time complexity is LINEAR (N additions are performed).
     - b. Space. CONSTANT: The value of N and the summation are stored.
  3. **Determining if the element located in the middle of an array of N numbers is divisible by 3.**
     - a. Time. CONSTANT: The operation is performed: array[N / 2] % 3.
     - b. Space. LINEAR: to store the array of N numbers.

Exercices

- What is the time and space complexity order of known algorithms that solve the following problems?

4. **Calculating the dot product of two vectors of size N**
   - a. Time. LINEAR: 2N operations are performed to multiply pairs of elements from both vectors in the same position and accumulate the result.
   - b. Space. LINEAR: to store the two vectors of size N.
5. **Storing all possible combinations <Shirt, Pants, Shoes> with N1 shirts, N2 pants, N3 shoes (N1 ≈ N2 ≈ N3)**
   - a. Time. CUBIC: N1xN2xN3 (3 cycles).
   - b. Space. CUBIC to store all possible triplets: (N1xN2xN3).

Excercices

- What is the time and space complexity order of known algorithms that solve the following problems?

6. **Given a drawer with N pieces of jewelry, in how many different ways can I fill a case that only fits 2 pieces?**
   - a. Time. CONSTANT: As only the number of different ways is requested, it can be calculated as the summation from 1 to N – 1: (N) (N – 1) / 2. For 4 pieces of jewelry, there are 3 + 2 + 1 different pairs. (Jewel1, Jewel2), (Jewel1, Jewel3), (Jewel1, Jewel4), (Jewel2, Jewel3), (Jewel2, Jewel4), (Jewel3, Jewel4).
   - b. Space. CONSTANT: Assuming the algorithm only receives the value of N.
7. **Storing each of the different ways referred to in problem 6.**
   - a. Time. QUADRATIC: consists of two nested loops, the first iterates from the first jewel to the penultimate, the second iterates from the next jewel to the last; the number of iterations of the second loop is: N – 1 + N – 2… + 2 + 1 = (N) (N – 1) / 2.
   - b. Space. QUADRATIC: to store 2((N) (N – 1) / 2) = 2N(N – 1) different pairs.

More about asymptotic notation

- Recalling the resulting equations from the spatial and temporal complexity analyses of the standard deviation algorithm.
- Do they hold true: **$12N + 9 \in O(N^2)$, $N + 3 \in O(N \log N)$**?
- Demonstrating the first (the second is left for the student):

  $f(N) = 12N + 9$, $g(N) = N^2$

  $c_1 = 12$, $N_0 = 1$.

  $12N + 9 < 12N^2$ holds true for all $N > 1$

  [N = 2] 33 < 48, [N = 3] 45 < 108, [N = 4] 57 < 192, …

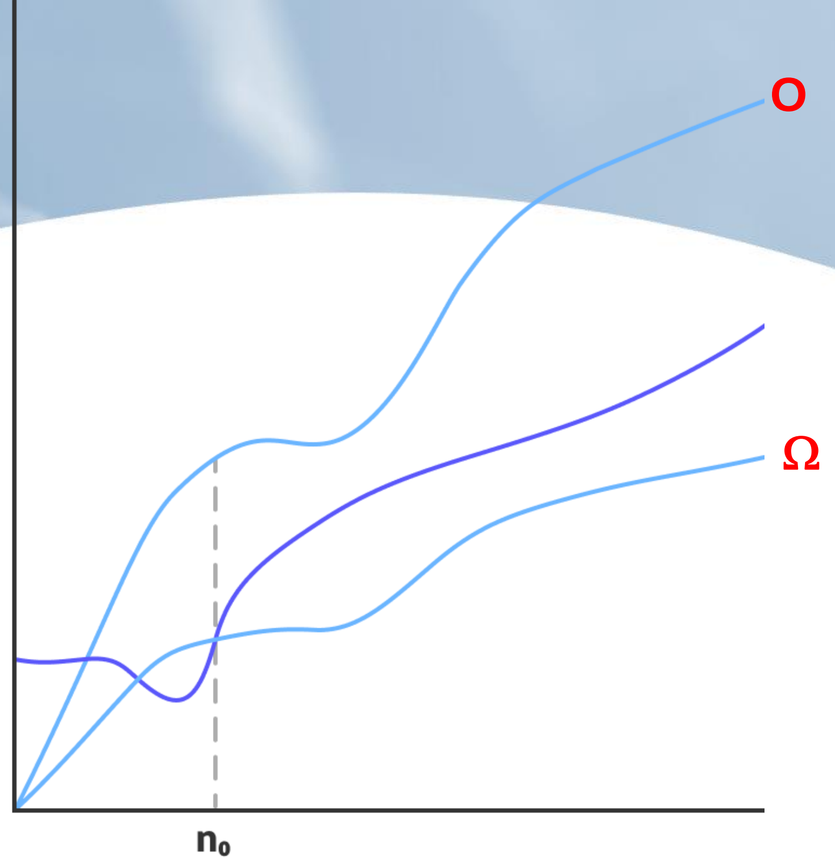
  $\therefore 12N + 9 \in O(N^2)$ …So, is 12N + 9 quadratic?

More about asymptotic notation

- What the notation O states is that the complexity of an algorithm will never exceed a given reference function.
  - If $f(N) \in O(g(N)) \rightarrow f(N)$ will never be greater than $g(N)$.
- For certain algorithms, the number of steps required to solve a problem varies depending on the input data.
  - This variation can cause the complexity to differ. (It is not the same in all cases)
- The most well-known algorithms susceptible to this variation are **sorting** and **searching algorithms**.

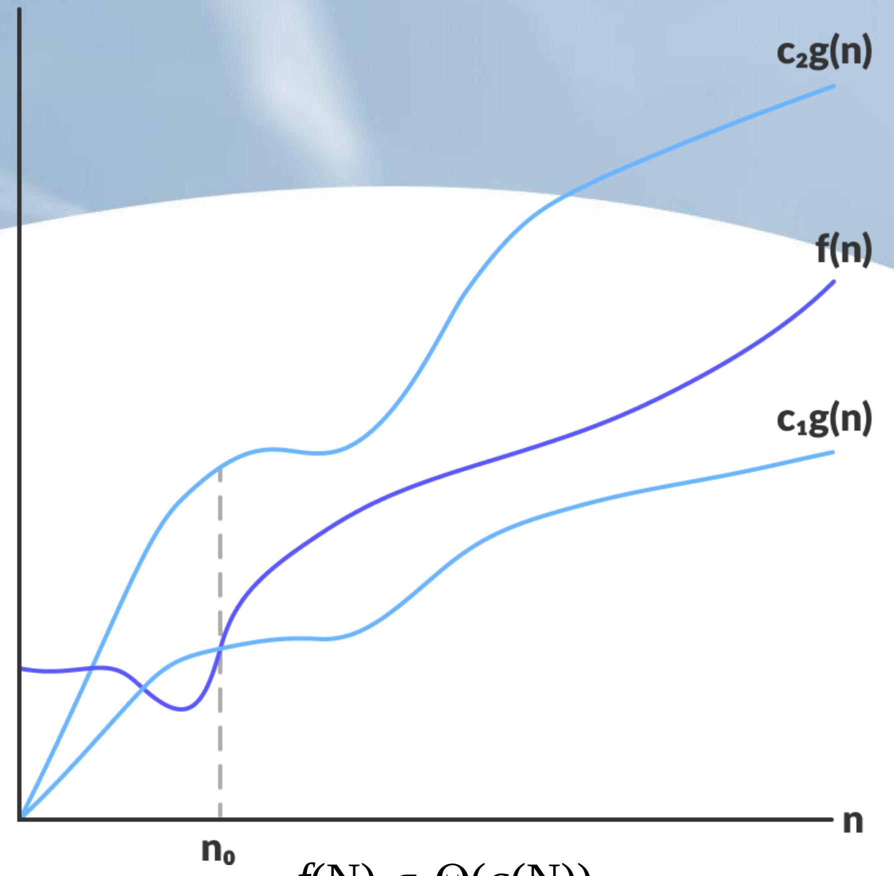
More about asymptotic notation

- There are complementary notations to address variation in algorithm complexity
  - The notation Ω defines a lower bound: **the complexity will not be less than** (or better than)
  - The notation Θ defines both *lower* and *upper bounds*: **the complexity will not be less than or greater than**



More about asymptotic notation

- There are complementary notations to address variation in algorithm complexity
  - The notation **Ω** defines a lower bound: **the complexity will not be less than** (or better than)
  - The notation **Θ** defines both *lower* and *upper bounds*: **the complexity will not be less than or greater than**
- Formal definition:
  - $f(N) \in \Omega(g(N))$ if there exist positive constants c_1 and N_0 such that
    $f(N) \geq c_1 g(N)$ for all $N > N_0$
  - $f(N) \in \Theta(g(N))$ if there exist positive constants c_1, c_2 y N_0 such that
    $c_1 g(N) \leq f(N) \leq c_2 g(N)$ for all $N > N_0$



$f(N) \in \Theta(g(N))$

Excercices

- Note that the following hold:
  1. $f(N) \in \Omega(g(N)) \iff g(N) \in O(f(N))$
  2. $f(N) \in \Omega(g(N)) \wedge f(N) \in O(g(N)) \iff f(N) \in \Theta(g(N))$
- Prove that: $\frac{1}{2} N \lg N \in \Omega(N)$.
- Prove that: $2N^3 + 4N^2 \in \Theta(N^3)$.

Excercices

- Note that the following hold:
  1. $f(\mathrm{N}) \in \Omega(g(\mathrm{N})) \iff g(\mathrm{N}) \in \mathrm{O}(f(\mathrm{N}))$
  2. $f(\mathrm{N}) \in \Omega(g(\mathrm{N})) \wedge f(\mathrm{N}) \in \mathrm{O}(g(\mathrm{N})) \iff f(\mathrm{N}) \in \Theta(g(\mathrm{N}))$
- Prove that: $\frac{1}{2}$ N lg N $\in \Omega(\mathrm{N})$.
  - $f(\mathrm{N}) = \frac{1}{2}\mathrm{NlgN}$, $g(\mathrm{N}) = \mathrm{N}$, $c_1 = \frac{1}{2}$, $N_0 = 1$.
  - $\frac{1}{2}\mathrm{NlgN} \geq \frac{1}{2}\mathrm{N}$ holds for all $\mathrm{N} > 1$ because: $\mathrm{lgN} \geq 1$.
  - [N = 2] 1 ≥ 1, [N = 4] 4 > 2, [N = 8] 12 > 4…
  - ∴ $\frac{1}{2}$ N lg N $\in \Omega(\mathrm{N})$.

Ejercicios

- Note that the following hold:
  1. $f(\mathrm{N}) \in \Omega(g(\mathrm{N})) \iff g(\mathrm{N}) \in \mathrm{O}(f(\mathrm{N}))$
  2. $f(\mathrm{N}) \in \Omega(g(\mathrm{N})) \wedge f(\mathrm{N}) \in \mathrm{O}(g(\mathrm{N})) \iff f(\mathrm{N}) \in \Theta(g(\mathrm{N}))$
- Prove that: $2N^3 + 4N^2 \in \Theta(N^3)$.
  - $f(N) = 2N^3 + 4N^2$, $g(N) = N^3$, $c_1 = 2$, $c_2 = 3$, $N_0 = 3$.
  - $2N^3 + 4N^2 \leq 3N^3$ holds for all $N > 3$:
  - [N = 4] $128 + 64 = 192$, [N = 5] $250 + 100 < 375$, [N = 6] $432 + 144 < 648$…
  - $2N^3 + 4N^2 \geq 2N^3$ holds for all $N > 3$ because: $4N^2 > 0$
  - $\therefore 2N^3 + 4N^2 \in \Theta(N^3)$.

What did we learn today?

- Introduction to the subject
  - Very important for our current context
- Classification of algorithms by their efficiency
- Importance of choosing an efficient algorithm
- Asymptotic notation to describe algorithm complexity
- Complete assignment 1 (it will be uploaded to Canvas tomorrow and must be submitted by Sunday before midnight).