“*Big O Notation*”

UNIVERSITY OF LUXEMBOURG

Dr. Oscar Castro, Dr. Pierrick Pochelu

Summary

- General presentation
- Demo with Jupyter Notebook
- Data Structure and some ML algorithms
- Conclusion & next steps with Big O

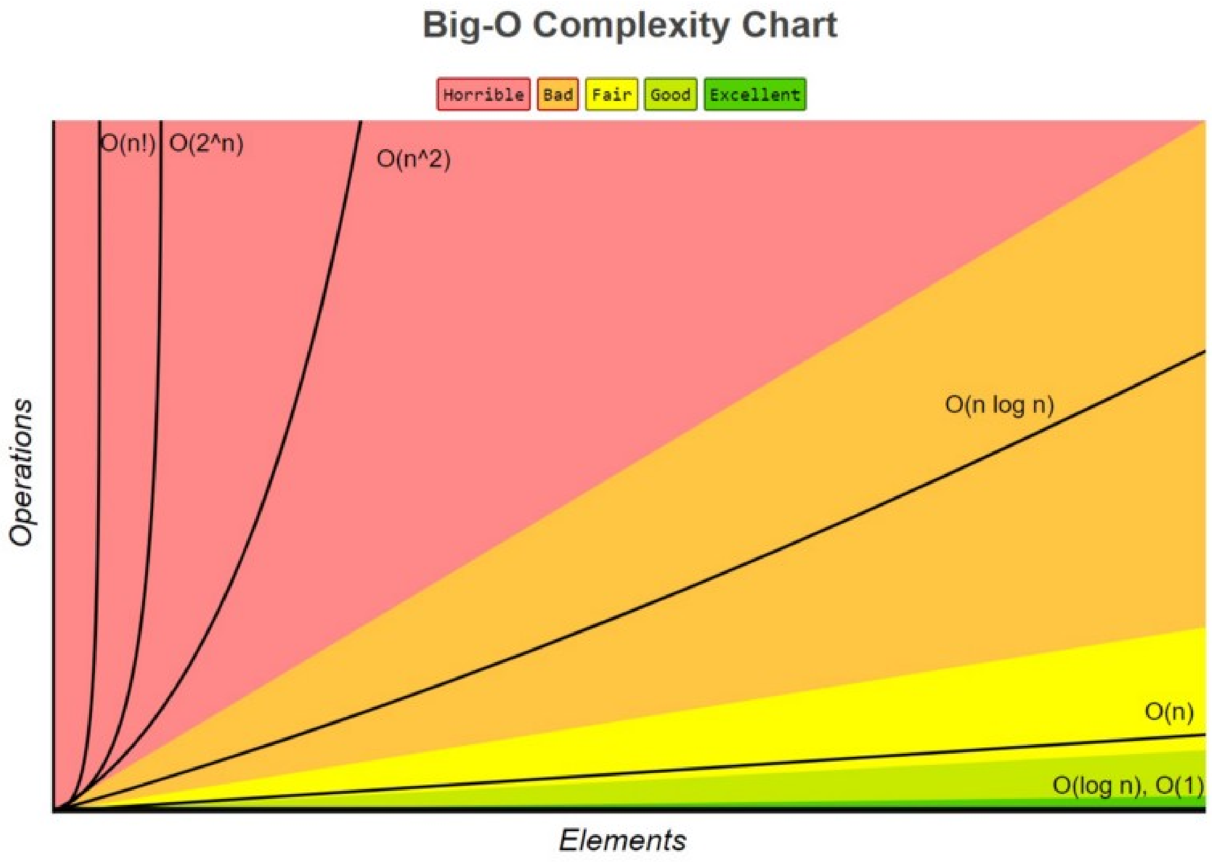
Introduction

What is Big O Notation?

- **"Big"**: Represents the *upper bound* of an algorithm’s growth rate for large input sizes.
- **"O"**: Stands for *Order of Magnitude*, describing how the runtime or space requirements increase with input size.

Why is Big O Important?

- **Analyzes Performance Trends**:
  - *Worst-case performance scenario* as input size grows.
  - Provides a **theoretical analysis**, (does no provide an accurate measurement #iterations, #seconds).
  - May be used for different aspects: computing time, memory, data transfer, …
- **Evaluates Algorithm Efficiency**:
  - **Predicts how algorithms will scale** with larger inputs.
  - Helps in **comparing different algorithms** to choose the best one.
  - Optimizing existing algorithms (bottleneck identification, improvement)
- **Key Questions It Helps Answer**:
  - Is the **algorithm's performance feasible** for large data?
  - Does it justify the **computational cost**?
  - How much **computing power** (e.g., # of cores) is required for reasonable performance?



In this order:

O(1) constant (or “O(C)”)

O(log(n)) logarithmic

O(n) linear

O(n^C) polynomial (“quadratic” when C=2, “cubic when C=3)

O(C^n) exponential

O(n!) factorial

⇒ O(n*m)

Putting in perspective Big O trends

We vary `n` and observe the effect on complexity estimation

Constant (C)	n	$\log_2(n)$	n	$n * \log_2(n)$	n^2	2^n	n!
1	10	3.3219	10	33.2193	100	1024	3,628,800
1	20	4.3219	20	86.4386	400	1,048,576	243,290,200,817,664,000

Big O Notation in ML projects: When and Why ?

1. Large-Scale Datasets

- **What happens if I increase the number of data samples?**
  - Can the model process the dataset within a reasonable time?
  - Should we subsample the data for faster processing?
- **What happens if I increase the number of features?**
  - Analyzes the impact of high dimensionality on computational cost.
  - Guides the need for dimensionality reduction (e.g., PCA) or feature selection.

2. Large-Scale Models

- **What happens if I increase the model size?**
  - Should we use a more efficient model or opt for distributed training?
  - Is the computational cost of increasing model parameters justifiable?

3. Specific Performance Needs

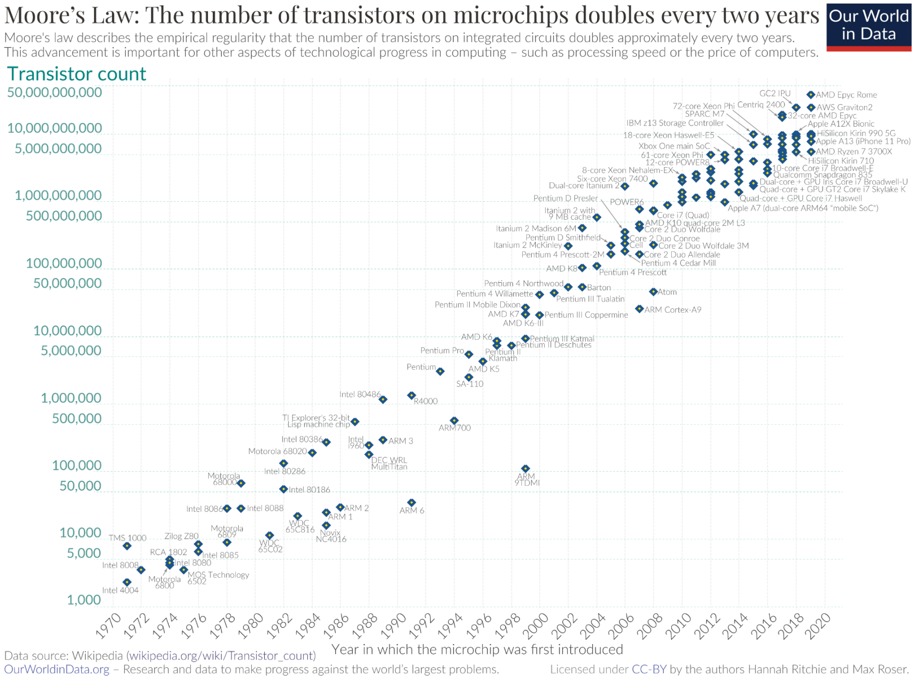
- **Real-time applications** (e.g., real-time inference for video streaming, online decision-making):
  - Can the algorithm meet latency requirements?
- **Low-capacity hardware** (e.g., drones, mobile devices):
  - Is the data and model lightweight enough to run efficiently on limited hardware?

Big O notation a timely topic

2 current trends:

- **End of Moore's Law**: CPU speeds are no longer increasing at the exponential rate they once did.
- **Modern applications (Deep Learning, Big Data)** require massive computational resources.

Conclusion: Optimizing and Parallelizing **algorithms** is the only way to advance modern applications.



/* demo */

Built-in Python data structures

List

Copy	O(n)
Append[1]	O(1)
Pop last	O(1)
Pop intermediate[2]	O(n)
Insert	O(n)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(n)
Iteration	O(n)
Get Slice	O(k)
Del Slice	O(n)
Set Slice	O(k+n)
Extend[1]	O(k)
Sort	O(n log n)
Multiply	O(nk)
x in s	O(n)
min(s), max(s)	O(n)
Get Length	O(1)

Double-Ended queue

(“from collections import deque”)

Copy	O(n)
append	O(1)
appendleft	O(1)
pop	O(1)
popleft	O(1)
extend	O(k)
extendleft	O(k)
rotate	O(k)
remove	O(n)
Get Length	O(1)

Set

x in s	O(1)
Union s|t	O(len(s)+len(t))
Intersection s&t	O(min(len(s), len(t)))
Multiple intersection s1&s2&..&sn	
Difference s-t	O(len(s))
s.difference_update(t)	O(len(t))
Symmetric Difference s^t	O(len(s))
s.symmetric_difference_update(t)	O(len(t))

Dictionary

(“dict” keyword)

k in d	O(1)
Copy[3]	O(n)
Get Item	O(1)
Set Item[1]	O(1)
Delete Item	O(1)
Iteration[3]	O(n)

source: https://wiki.python.org/moin/TimeComplexity

ML complexity

Time Complexity of 10 Most Popular ML Algorithms blog.DailyDoseofDS.com

Algorithm	Training	Inference
Linear Regression (OLS)	$O(nm^2 + m^3)$	$O(m)$
Linear Regression (SGD)	$O(n_{epoch}nm)$	$O(m)$
Logistic Regression (Binary)	$O(n_{epoch}nm)$	$O(m)$
Logistic Regression (Multiclass OvR)	$O(n_{epoch}nmc)$	$O(mc)$
Decision Tree	$O(n \cdot log(n) \cdot m)$ $O(n^2 \cdot m)^*$ Worst case	$O(d_{tree})$
Random Forest Classifier	$O(n_{trees} \cdot n \cdot log(n) \cdot m)$	$O(n_{trees} \cdot d_{tree})$
Support Vector Machines (SVMs)	$O(n^2m + n^3)$	$O(m \cdot n_{SV})$
k-Nearest Neighbors	—	$O(nm)$

n: samples | m: dimensions | n_{epoch}: epochs | c: classes | d_{tree}: depth
n_{SV}: Support vectors | k: clusters | i: iterations

Memory
O(m²)
O(m²)
O(m²)
O(m²)
O(n)
O(ntrees * n)
O(m²)
O(n*m)

Conclusion: Different algorithms have different time & memory complexity
⇒ Use it at your advantage

Conclusion & next steps with Big O

1. Useful for describing ML algorithms

- Describes how an algorithm's performance scales with input size
- Useful to take quick decisions on data structure, ML algorithm
- Useful also to describe complex cases where different factor

2. When a code is well optimized, Big O notation is not enough:

- Ignoring constant factors and lower-order terms
- Ignoring language performance , and individual operations (example: FP64 operations slower than INT32)
- The performance of algorithms can be affected by various CPU architectures, memory hierarchies, and parallel processing capabilities
- Big O notation with HPC methodologies
  - Big O notation of #messages in addition of time and memory consumption
  - Code Profiling: Computing time, Memory, …
  - Strong Scalability Analysis (Fixing the input size, increase the #cores)
  - Weak Scalability Analysis (Increase the input size, increase the #cores)