Data Structures and Algorithms

Lecture 1

© Fall 2025- **Dr. Hesham Sakr**
Building, office No. 12
Hesham.sakr@sut.edu.eg

Why Data Structure & Algorithms:

Students gain a solid foundation in designing, implementing, and analyzing algorithms and data structures, essential for efficient problem-solving in programming and software development.

Course Syllabus

- 1. Introduction to Data Structures and Algorithms
  - Overview of fundamental concepts.
  - Importance of data structures in software development.
  - Basics of algorithmic thinking.

- 2. Arrays and Linked Lists
  - Implementation and manipulation of arrays.
  - Singly and doubly linked lists.
  - Time and space complexity analysis.
- 3. Stacks and Queues
  - Understanding stack and queue data structures.
  - Applications and practical implementation.
  - Evaluating efficiency in different scenarios.

Course Syllabus (Cont.)

- 4. Trees and Graphs
  - Binary trees and their variants.
  - Graph representation and traversal.
  - Tree and graph algorithms.
- 5. Sorting and Searching Algorithms
  - Comparison of sorting algorithms (e.g., Bubble Sort, Quick Sort, Merge Sort).
  - Searching techniques (e.g., Linear Search, Binary Search).
  - Analysis of algorithmic complexity.
- 6. Algorithmic Paradigms
  - Divide and conquer strategies.
  - Dynamic programming principles.
  - Greedy algorithms and their applications.
- 7. Practical Application
  - Coding exercises applying learned concepts.

Our Roadmap

Type	Marks	Week	Discussion
Mini Project	10	Week 5	Week 7
Final Project	10	Week 8	Week 13
Midterm	15	-	-
Final Exam	25	-	-
Lab Participation	15	-	-
Assessments (Lab)	15	-	-
Assessments (Lecture)	10	-	-

Teaching Assistants

1- Aya Abd Elnabi
2-Hagar Sobeh

Textbooks:

"Introduction to Algorithms" by Thomas H. Cormen et al.

"Data Structures and Algorithms" by Michael T. Goodrich et al.

What is an algorithm?

Definition:

- An algorithm is a Step By Step process to solve a problem, where each step indicates an intermediate task.
- Algorithm contains finite number of steps that leads to the solution of the problem.

What is an algorithm?

- It's **a set of well-defined rules for solving some computational problem**.
  - a bunch of numbers and you want to **rearrange them** so that they're **in sorted order**.
  - you have a road map and you want to **compute the shortest path** from some origin to some destination.
  - you need to complete several tasks before certain deadlines, and you want to **know in what order you should finish the tasks** so that you complete them all by their respective deadlines.

Why Study Algorithms?

Why Study Algorithms?

- Important for all other branches of computer science
    1. Routing protocols in communication networks take credit of **classical shortest path algorithms**.
    2. Public-key cryptography relies on **efficient number-theoretic algorithms**.
    3. Computer graphics requires the **computational primitives supplied by geometric algorithms**.
    4. Database indices rely on **balanced search tree data structures**.
    5. Computational biology uses **dynamic programming algorithms to measure genome similarity**.

Challenging (Good for the brain)

Properties /Characteristics of an Algorithm

Algorithm has the following basic properties:

1. Input-Output: Algorithm takes input and produces the required output. This is the basic characteristic of an algorithm.
2. Finiteness: An algorithm must terminate in countable number of steps.
3. Definiteness: Each step of an algorithm must be stated clearly and unambiguously.
4. Effectiveness: Each and every step in an algorithm can be converted in to programming language statement.
5. Generality: Algorithm is generalized one. It works on all set of inputs and provides the required output. In other words it is not restricted to a single input value.

Categories of Algorithm:

- Based on the different types of steps in an Algorithm, it can be **divided into three categories**, namely The written program is buggy.

    - Sequence
    - Selection and
    - Iteration

Sequence:

- The steps described in an algorithm are **performed successively one by one without skipping any step**.
- The sequence of steps defined in an algorithm **should be simple and easy to understand**.
- **Each instruction of such an algorithm is executed**, because no selection procedure or conditional branching exists in a sequence algorithm.

Sequence Example:

// adding two numbers

Step 1: start

Step 2: read a,b

Step 3: Sum=a+b

Step 4: write Sum

Step 5: stop

Selection:

- The sequence type of algorithms are not sufficient to solve the problems, which **involves decision and conditions**.

- In order **to solve the problem which involve decision making or option selection**, we go for **Selection type of algorithm**.

- The general format of Selection type of statement is as shown below:

```
if(condition)
    Statement 1;
else
    Statement 2;
```

- The above syntax specifies that if the condition is true, statement 1 will be executed otherwise statement 2 will be executed.

Selection Examples:

Example1:

```
// Person eligibility for vote
Step 1 : start
Step 2 : read age
Step 3 :  if age > = 18 then step_4 else step_5
Step 4 : write "person is eligible for vote"
Step 5 : write " person is not eligible for vote"
Step 6 : stop
```

Example2:

```
// biggest among two numbers
Step 1 : start
Step 2 : read a,b
Step 3 :  if a > b then
Step 4 : write "a is greater than b"
Step 5 : else
Step 6 : write "b is greater than a"
Step 7 : stop
```

Iteration:

- Iteration type algorithms are used in solving the problems which involves repetition of statement.
- In this type of algorithms, a particular number of statements are repeated 'n' no. of times.

Iteration Example:

Step 1 : start

Step 2 : counter =1

Step 3 : sum=0

Step 2 : read grade

Step 3 : repeat step 4 until counter > 5

Step 4 : (a) sum = grade +sum
(b) Average =sum/5
(c) counter = counter + 1

Step 5 : write Average

Step 6 : stop

The input: 10, 25, 30, 40, 50

The output: 215/5 = 43

Algorithm analysis Importance

- To detect
  - The written program is buggy.
  - The program may be inefficient
  - If the program is running on a large data set, then the running time becomes an issue

Analysis Aspects

- Correctness
  - Does the input/output relation match algorithm requirement?
- Amount of work done (complexity)
  - Basic operations to do task in a finite amount of time
- Amount of space used e.g. Memory used

Performance Analysis of Algorithm

Performance Analysis an Algorithm:

- The Efficiency of an Algorithm can be measured by the following metrics:

i. Time Complexity and

ii. Space Complexity.

Performance Analysis an Algorithm:

i. Time Complexity:

- The amount of time required for an algorithm to complete its execution is its time complexity.
- An algorithm is said to be efficient if it takes the minimum (reasonable) amount of time to complete its execution.

ii. Space Complexity:

- The amount of space occupied by an algorithm is known as Space Complexity.
- An algorithm is said to be efficient if it occupies less space and required the minimum amount of time to complete its execution

Asymptotic analysis

Asymptotic analysis:

- Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation.

- For example, the running time of one operation is computed as $O(n)$ and may be for another operation it is computed as $O(n^2)$.

- This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

ASYMPTOTIC ANALYSIS (High Level Idea)

We'll express the asymptotic runtime of an algorithm using

BIG-O NOTATION

THE POINT OF ASYMPTOTIC NOTATION

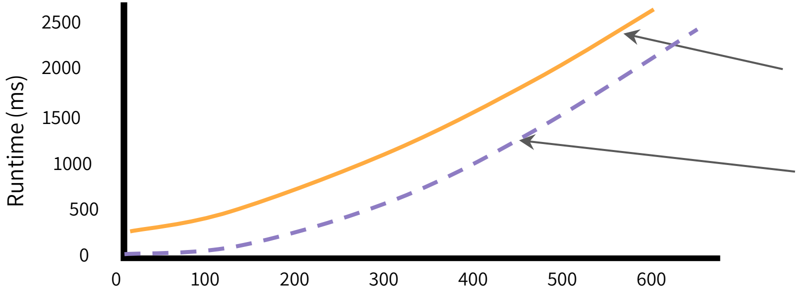
suppress $\underbrace{\text{constant factors}}_{\textit{too system dependent}}$ and $\underbrace{\text{lower-order terms}}_{\textit{irrelevant for large inputs}}$

ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent ***irrelevant for large inputs***



$0.1n^{1.6} + 300$

$.008n^2$

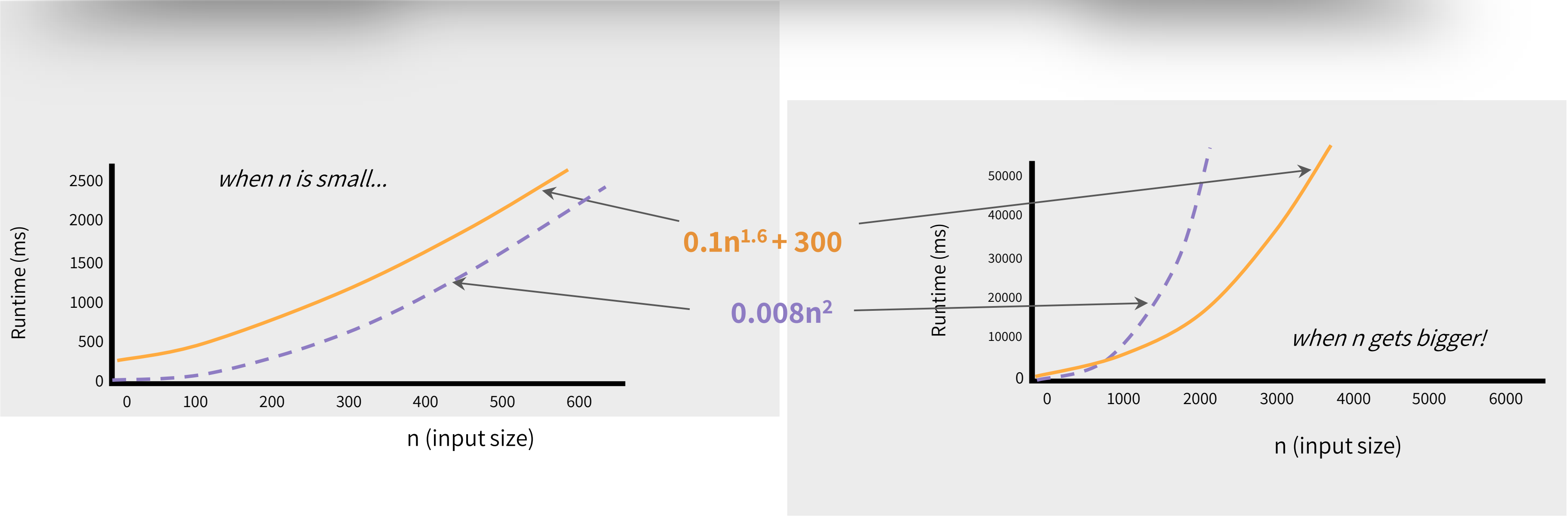
n (input size)

ASYMPTOTIC ANALYSIS (High Level Idea)

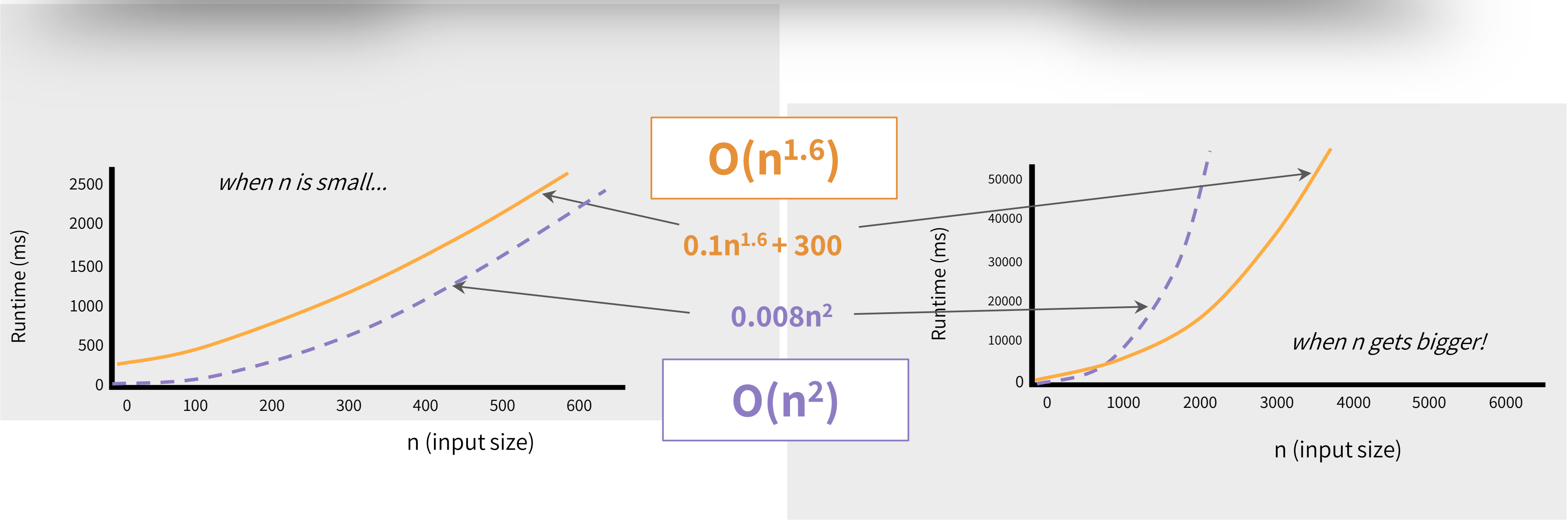
THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

$\underbrace{\text{suppress constant factors}}_{\textit{too system dependent}}$ and $\underbrace{\text{lower-order terms}}_{\textit{irrelevant for large inputs}}$



ASYMPTOTIC ANALYSIS (High Level Idea)



ASYMPTOTIC ANALYSIS (High Level Idea)

- To compare algorithm runtimes in this class, we compare their Big-O runtimes
  - Ex: a runtime of $O(n^2)$ is considered “better” than a runtime of $O(n^3)$
  - Ex: a runtime of $O(n^{1.6})$ is considered “better” than a runtime of $O(n^2)$
  - Ex: a runtime of $O(1/n)$ is considered “better” than $O(1)$

$O(1/n) = O(1 * n^{-1})$, $O(1) = (1 * n^0)$

The time required by an algorithm falls under three types:

- **Best Case:**
- Minimum time required for program execution.

- **Average Case:**
- Average time required for program execution.

- **Worst Case:**
- Maximum time required for program execution.

ASYMPTOTIC NOTATIONS

Asymptotic Notations:

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

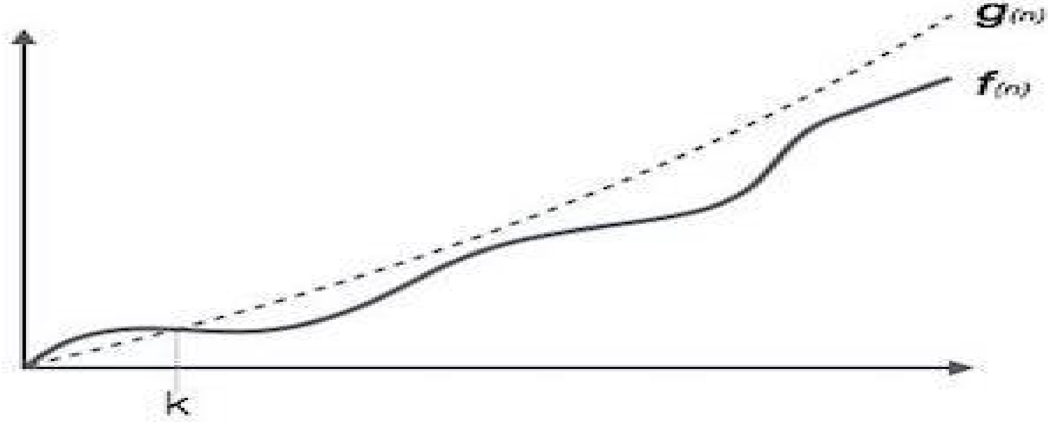
Worst Case

Big Oh Notation, O

Big Oh Notation, O:

- The notation O(n) is the formal way to express the upper bound of an algorithm's running time.
- It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

Big Oh Notation, O:



For example, for a function f(n)

O(f(n)) = { g(n) : there exists c > 0 and n_0 such that f(n) ≤ c.g(n) for all n > n_0. }.

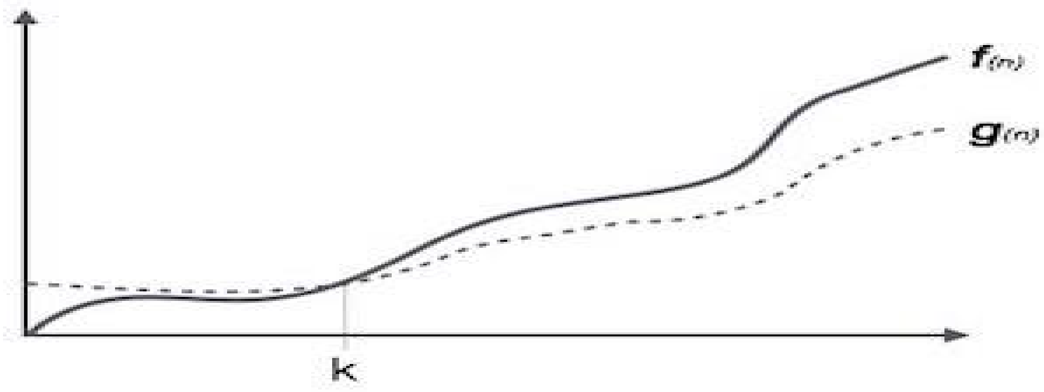
Best Case

Omega Notation, Ω

Omega Notation, Ω:

- The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time.
- It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

Omega Notation, Ω:



For example, for a function f(n)

$\Omega(f(n)) = \{ g(n)$: there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0$. }.

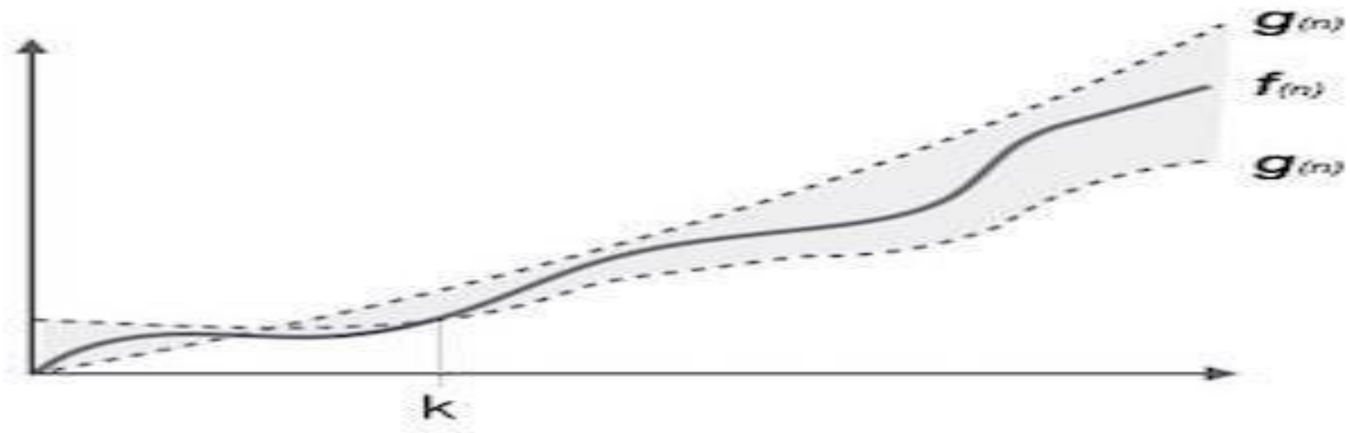
Average Case

Theta Notation, θ

Theta Notation, θ:

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

Big Oh Notation, O:



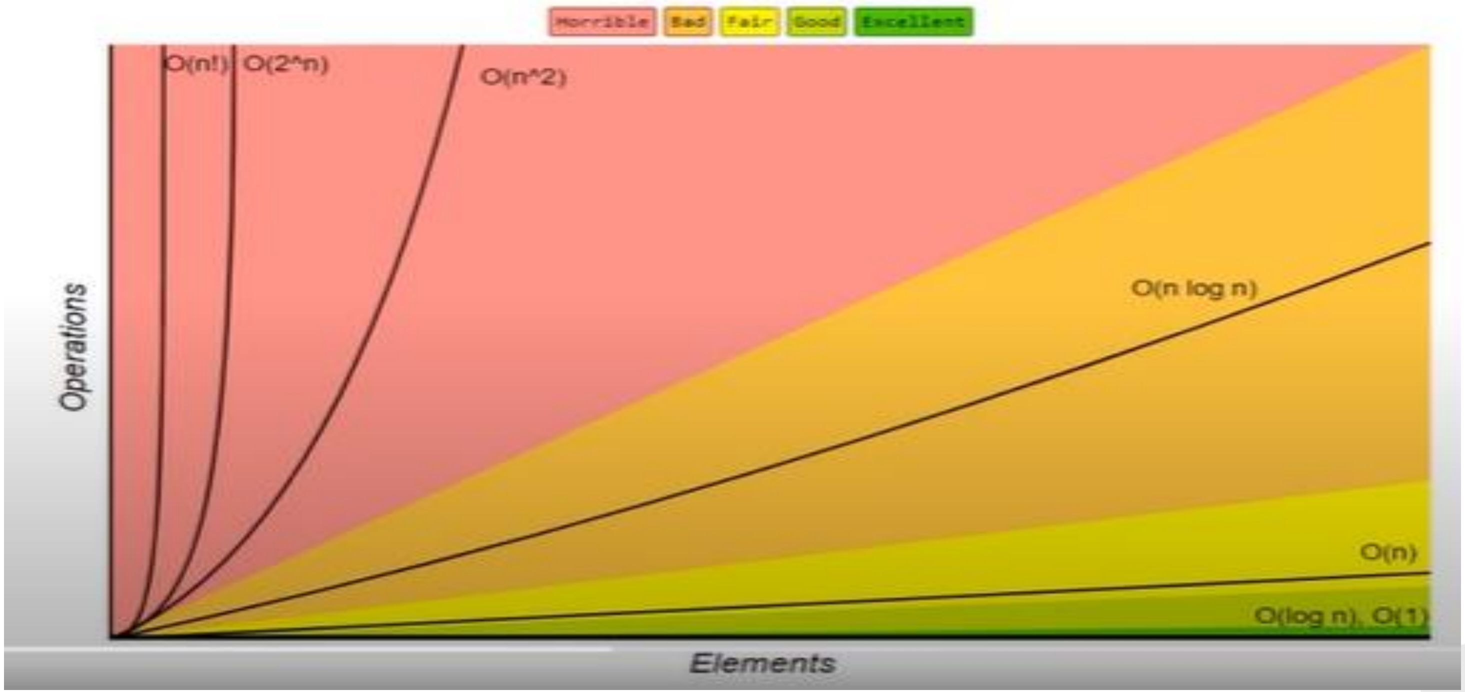
It is represented as follows

θ(f(n)) = { g(n) if and only if g(n) = O(f(n)) and g(n) = Ω(f(n)) for all n > n_0. }.

Some common Big O run times

- O(log n), also known as log time. Example: Binary search.
- O(n), also known as linear time. Example: Simple search.
- O(n * log n). Example: A fast sorting algorithm, like quicksort.
- O(n^2). Example: A slow sorting algorithm, like selection sort.
- O(n!). Example: A really slow algorithm, like the traveling salesperson.

Growth Rates of Common Functions



Growth Rates of Common Functions

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Growth Rates of Common Functions

n \ $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Thank You