Data Structures and Algorithms

LECTURE 04: ALGORITHM ANALYSIS

UNIVERSITY of GREENWICH

Alliance with FPT Education

Pearson BTEC

Contents

- Algorithmic Complexity
- Time Complexity
- Asymptotic notations
- Brute Force

Algorithm Analysis

- Why should we analyze algorithms?
  - Predict the **resources** the algorithm will need
    - Computational time (**CPU** consumption)
    - Memory space (**RAM** consumption)
    - Communication **bandwidth** consumption
    - **Hard disk** operations

Problem: Get Number of Steps

- Calculate maximum steps to find the result

```
long getOperationsCount(int n) {
  long counter = 0;
  for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
      counter++;
  return counter;
}
```

Solution:
T(n) = 3(n ^ 2) + 3n + 3

- The input(**n**) of the function is the main source of steps growth

Simplifying Step Count

- Some parts of the equation **grow much faster** than others
  - – $T(n) = \mathbf{3(n^2)} + 3n + 3$
  - – We can **ignore** some part of this equation
  - – Higher terms **dominate** lower terms – $\mathbf{n > 2}$, $\mathbf{n^2 > n}$, $\mathbf{n^3 > n^2}$
  - – Multiplicative constants can be **omitted** – $\mathbf{12n \rightarrow n}$, $\mathbf{2n^2 \rightarrow n^2}$
- The previous solution becomes $\mathbf{\approx n^2}$

Time Complexity

- Worst-case
  – An **upper** bound on the running time
- Average-case
  – **Average** running time
- Best-case
  – The **lower** bound on the running time (the optimal case)

Time Complexity

- Therefore, we need to measure **all** the possibilities:

Time Complexity

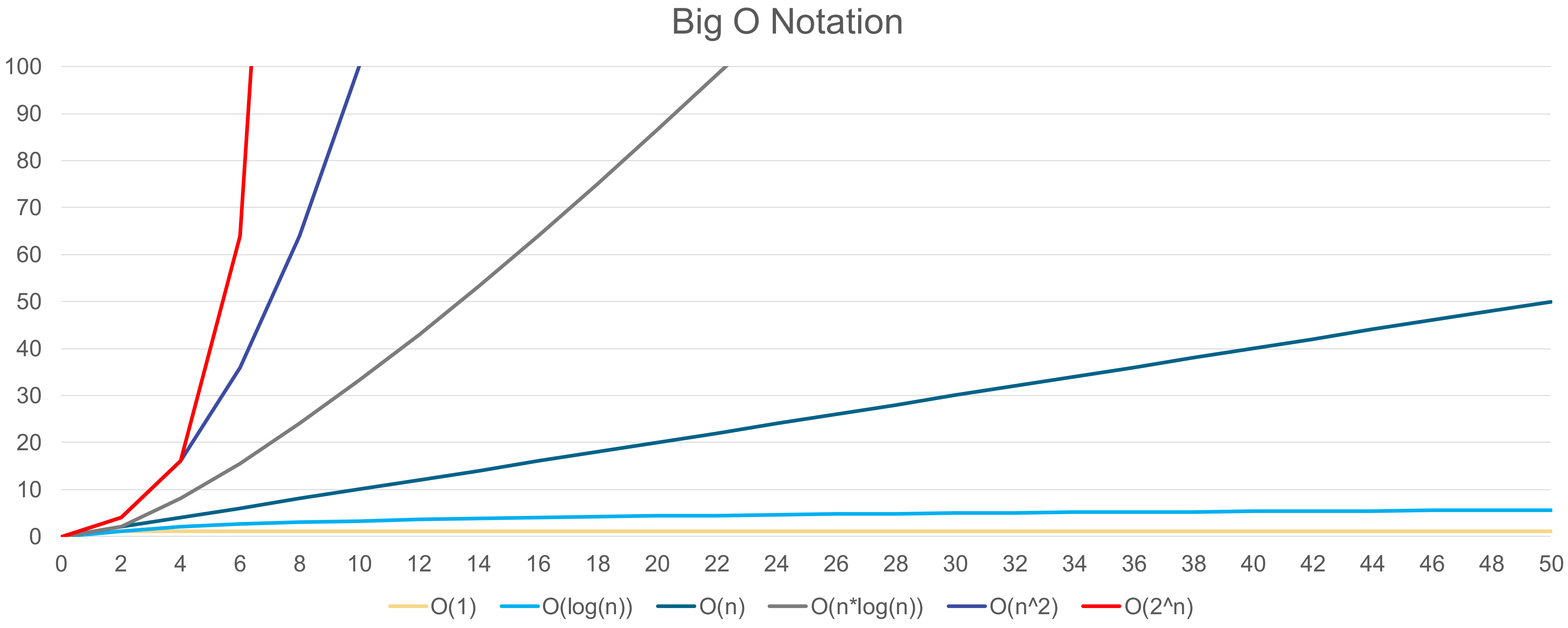
- From the previous chart we can deduce:
  - For smaller size of the input (**n**) we **don't care much for the runtime**. So we measure the time as **n** approaches **infinity**
  - If an algorithm **has to scale**, it **should compute** the result within a **finite and practical time**
  - We're concerned about the **order of an algorithm's complexity**, not the actual time in terms of **milliseconds**

Asymptotic notations

- **Asymptotic notations** are descriptions that allow us to examine an algorithm's running time by expressing its **performance** as the input size, **n**, of an algorithm or a function **f increases**. There are **three** common asymptotic notations:
  - Big **O – O(f(n))**
  - Big **Theta – Θ(f(n))**
  - Big **Omega – Ω(f(n))**

Asymptotic Functions

- Below are some examples of **common algorithmic** grow:



Typical Complexities

Complexity	Notation	Description
constant	O(1)	n = 1 000 → 1-2 operations
logarithmic	O(log n)	n = 1 000 → 10 operations
linear	O(n)	n = 1 000 → 1 000 operations
linearithmic	O(n*log n)	n = 1 000 → 10 000 operations
quadratic	O(n2)	n = 1 000 → 1 000 000 operations
cubic	O(n3)	n = 1 000 → 1 000 000 000 operations
exponential	O(n^n)	n = 10 → 10 000 000 000 operations

Time Complexity and Program Speed

Complexity	10	20	50	100	1 000	10 000	100 000
O(1)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(log n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n*log n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n^2)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	2 s	3-4 min
O(n^3)	< 1 s	< 1 s	< 1 s	< 1 s	20 s	5 hours	231 days
O(2^n)	< 1 s	< 1 s	260 days	hangs	hangs	hangs	hangs
O(n!)	< 1 s	hangs	hangs	hangs	hangs	hangs	hangs
O(n^n)	3-4 min	hangs	hangs	hangs	hangs	hangs	hangs

Brute-Force Algorithms

- Trying all possible combinations
- Picking the best solution
- Usually slow and inefficient

0 0 0 0 0

Brute-Force Algorithms

0	0	0	0	0

Brute-Force Algorithms

0	0	0	0	1

Brute-Force Algorithms

0 0 0 0 2

Brute-Force Algorithms

9 9 9 9 9

10 x 10 x 10 x 10 x 10 = 100,000 combinations

Summary

- Algorithmic **Complexity**
- Time Complexity