Algorithms and Data Structures

Jonah Benedicto

September 2025

Introduction

Contents

Chapter 1

Introduction

1.1 Algorithm

An algorithm is a step-by-step set of instructions used to solve a problem or perform a computation.

1.2 Data Structure

A data structure is a format for organising, storing and accessing data to be used for efficient processing, retrieval, and manipulation.

1.3 Abstract Data Type

An abstract data type is a model that defines the way data is processed, retrieved and manipulated without specifying the way it is organised, stored and accessed.

Chapter 2

Algorithms and Analysis

2.1 Experimental (Empirical) Analysis

Experimental (empirical) analysis is the process of evaluating an algorithm's efficiency by designing, implementing, and testing it in practice. This approach empirically measures time complexity (execution time) and space complexity (memory usage) in relation to the size of its inputs.

2.2 Theoretical Analysis

Theoretical analysis is the process of evaluating an algorithm's efficiency by mathematically determining its time complexity and space complexity in relation to input size, without implementation.

2.3 Theoretical Analysis Steps

1. Write the algorithm in pseudocode.
2. Determine the number of primitive operations executed.
3. Express the algorithm as a function of the input size. f(n)
4. Express the function in asymptotic notation through asymptotic analysis.

2.4 Asymptotic Analysis

Asymptotic analysis is the process of evaluating an algorithm's efficiency by classifying the asymptotic bound of its time complexity and space complexity functions.

2.5 Asymptotic Notation

There are three primary types of asymptotic notation:

- Big-O Notation - Describes an asymptotic upper bound on a function. Given functions $f(n)$ and $g(n)$. A function $f(n)$ is in $O(g(n))$ if there exists a positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.
- Big-Omega Notation - Describes an asymptotic lower bound on a function. Given functions $f(n)$ and $g(n)$. A function $f(n)$ is in $\Omega(g(n))$ if there exists a positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.
- Big-Theta Notation - Describes an asymptotic tight bound on a function. Given functions $f(n)$ and $g(n)$. A function $f(n)$ is in $\Theta(g(n))$ if there exists a positive constant c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

2.6 Fundamental Functions

There are eight fundamental functions in asymptotic analysis:

- Constant: 1
- Logarithmic: $\log_2 n$
- Linear: n
- N-Log-N: $n \log_2 n$
- Quadratic: n^2
- Cubic: n^3
- Exponential: 2^n
- Factorial: $n!$

2.7 Best-Case, Worst-Case, Average-Case

There are three types of cases:

- Worst-case - The minimum time complexity or space complexity an algorithm takes to complete.
- Average-case - The average time complexity or space complexity an algorithm takes to complete.
- Best-case - The maximum time complexity or space complexity an algorithm takes to complete.

2.8 Problems

1. Discuss the difference between worst-case and best-case behaviour, and describe this difference using one or more concrete examples.

2. Given two functions $f(n)$ and $g(n)$, discuss the difference between the following statements:

   - $f(n) \in O(g(n))$;
   - $f(n) \in \Omega(g(n))$;
   - $f(n) \in \Theta(g(n))$.

3. A polynomial of degree k is a function given by $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$, where $a_0, \cdots, a_k$ are constants. Assume $a_k > 0$ for the following questions:

   (a) Give a tight Big-O bound on f(n) using the simplification rules given in the lectures.

   (b) Give a tight Big-Omega bound on f(n) using the simplification rules given in the lectures.

   (c) Does a Big-Theta bound exist for f(n)? If so, explain why it exists, then provide it.

4. Recall that $f(n)$ is $O(g(n))$ (f is bounded from above by g) if there exists positive c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. Using this definition, find and prove big-O bounds for the following functions:

$$3 + 2\log_2 n, \quad (n+1)(n-1), \quad \sqrt{9n^5 - 5}$$

5. Using previous question or otherwise, find big-O bounds for these functions (you are not required to prove these):

$$2^n + n^2, \quad \log_2 2^n \cdot n^2$$

6. In algorithm analysis, we often omit the base of logarithms for convenience. This question will prove this is a mathematically valid thing to do. Using the fact that $\log_a(x) = \log_b(x)/\log_b(a)$ for all $a, b > 1$, prove that $\log_a(n)$ is $O(\log_b(n))$

7. Show that $f(n) = n$ is not $O(\log_2(n))$ by proving no c and n_0 can satisfy the definition.

8. Recall that $f(n)$ is $\Omega(g(n))$ (f is bounded from below by g) if there exist c and n_0 such that $f(n) \geq cg(n)$ for $n \geq n_0$. Prove that for any strictly positive and increasing function is $\Omega(1)$. Why is this not useful in algorithm analysis?

2.9 Solutions

1. The worst-case is the maximum time complexity or space complexity an algorithm takes to complete and the best-case is the minimum time complexity or space complexity an algorithm takes to complete. For example, consider a linear search algorithm through an unordered list. The worst-case would be the case that the target does not exist and the best-case would be the case that the target is the first element.

2. - The statement $f(n) \in O(g(n))$ means that the function $f(n)$ has an asymptotic upper bound $g(n)$.
   - The statement $f(n) \in \Omega(g(n))$ means that the function $f(n)$ has an asymptotic lower bound $g(n)$.
   - The statement $f(n) \in \Theta(g(n))$ means that the function $f(n)$ has an asymptotic tight bound $g(n)$.

3. (a) $O(n^k)$

   (b) $\Omega(n^k)$

   (c) Yes, the Big-Theta bound for f(n) exists since the Big-O bound for f(n) and the Big-Omega bound for f(n) are the same. $\Theta(n^k)$

4. $f(n) = 3 + 2\log_2 n$ is $O(\log_2 n)$.

$$\begin{aligned} f(n) &<= cg(n) \\ 3 + 2\log_2 n &<= 3\log_2 n + 2\log_2 n \\ 3 + 2\log_2 n &<= 5\log_2 n \\ c = 5, \quad n_0 &= 2 \quad \square \end{aligned}$$

   $f(n) = (n+1)(n-1)$ is $O(n^2)$

$$\begin{aligned} f(n) &<= cg(n) \\ (n+1)(n-1) &<= n^2 \\ n^2 - 1 &<= n^2 \\ c = 1, \quad n_0 &= 1 \quad \square \end{aligned}$$

   $f(n) = \sqrt{9n^5 - 5}$ is $O(\sqrt{n^5})$

$$\begin{aligned} f(n) &<= cg(n) \\ \sqrt{9n^5 - 5} &<= \sqrt{9n^5} \\ \sqrt{9n^5 - 5} &<= 3\sqrt{n^5} \\ c = 3, \quad n_0 &= 1 \quad \square \end{aligned}$$

5. $2^n + n^2$ is $O(2^n)$

$$\begin{aligned} log_2(2^n \cdot n^2) &= \log_2 2^n + \log_2 n^2 \\ &= n + 2 \end{aligned}$$

$\log_2(2^n \cdot n^2)$ is $O(n)$

6. $f(n) = log_a(n)$ is $O(log_b(n))$

$$\begin{aligned} f(n) &<= cg(n) \\ log_a n = \log_b(n)/\log_b(a) &<= \log_b(n) \\ c = 1, \quad n_0 = b & \quad \square \end{aligned}$$

7. $f(n) = n$ is not in $O(\log_2(n))$

$$\begin{aligned} f(n) &<= cg(n) \\ n &<= c\log_2(n) \\ \frac{n}{\log_2 n} &<= c \end{aligned}$$

There does not exist a c and n_0.

8.

$$\begin{aligned} f(n) &\geq cg(n) \\ f(n) &\geq f(1) \\ c = 1, \quad n_0 = 1 & \quad \square \end{aligned}$$

It is not useful because stating that an algorithm is $\Omega(1)$ is essentially equivalent to saying that any algorithm requires at least one operation to run. While this is true, it provides no meaningful information about the algorithm's time complexity or space complexity. In other words, the bound $\Omega(n)$ is trivial, since it holds for virtually all algorithms, and therefore does not help us distinguish between different time complexities.