Chpater 3. Characterizing Running Times

Joon Soo Yoo

March 18, 2025

Assignment

- Read §3.1, §3.2
- Problems:
  - §3.1 - #2
  - §3.2 - #1, #3, #4

Chapter 3: Characterizing Running Times

Overview of Asymptotic Notation

- **Chapter 3.1:** $O-$**notation,** $\Omega-$**notation, and** $\theta-$**notation**
- Chapter 3.2: Asymptotic notation – formal definitions
- Chapter 3.3: Standard notations and common functions

Motivation: Why Asymptotic Notation?

Analyzing Insertion Sort (Worst-Case)

- In Chapter 2, we analyzed the worst-case running time of **Insertion Sort**.
- The exact running time expression was:

$$T(n) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n - (c_2 + c_4 + c_5 + c_8)$$

- We discarded:
  - Lower-order terms: $\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n - (c_2 + c_4 + c_5 + c_8)$
  - Coefficient of the leading term: $\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}$
- This left us with just the dominant term: $\Theta(n^2)$.

Why is this important?

- We focus on **growth rate** rather than exact constants.
- Helps compare different algorithms effectively.

Big-O Notation: Upper Bound

Intuition:

- Big-O notation describes the **upper bound** on how fast a function can grow.
- It tells us that a function grows **no faster** than a certain rate, based on the highest-order term.

Example:

- Consider $f(n) = 7n^3 + 100n^2 - 20n + 6$.
- The highest-order term is $7n^3$, meaning $f(n)$ grows **at most** as fast as n^3.
- So, we write:

$$f(n) = O(n^3).$$

Omega (Ω) Notation: Lower Bound

Intuition:

- Omega notation describes the **lower bound** on how fast a function grows.
- It tells us that a function grows **at least as fast** as a certain rate, based on the highest-order term.

Example:

- Consider $f(n) = 7n^3 + 100n^2 - 20n + 6$.
- Since $f(n)$ grows **at least** as fast as n^3, we write:

$$f(n) = \Omega(n^3).$$

Omega (Ω) Notation: Lower Bound

Intuition:

- Omega notation describes the **lower bound** on how fast a function grows.
- It tells us that a function grows **at least as fast** as a certain rate, based on the highest-order term.

Example:

- Consider $f(n) = 7n^3 + 100n^2 - 20n + 6$.
- Since $f(n)$ grows **at least** as fast as n^3, we write:

$$f(n) = \Omega(n^3).$$

Theta (θ) Notation: Tight Bound

Intuition:

- Theta notation describes a **tight bound** on how fast a function grows.
- It tells us that a function grows **precisely** at a certain rate, based on the highest-order term.

Example:

- Since $f(n) = 7n^3 + 100n^2 - 20n + 6$ is both $O(n^3)$ and $\Omega(n^3)$,
- We conclude:

$$f(n) = \Theta(n^3).$$

Example: Insertion Sort

```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i - 1].
    j = i - 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = key
```

How Insertion Sort Works:

- Elements are inserted one by one into their correct position.
- The left part remains sorted while the right part is unsorted.

Big-O of Insertion Sort: Worst Case

Step-by-Step Deduction of $O(n^2)$

- **The running time is dominated by the inner loop.**
- **The outer loop** runs $n-1$ times (from $i=2$ to n).
- **The inner loop** iterates at most $i-1$ times for each i.
- Since i is at most n, the total number of inner loop iterations is:
$$(n-1)(n-1) = (n-1)^2$$
- This is **less than** n^2, so the total time spent in the inner loop is at most:
$$O(n^2)$$
- Since each iteration takes **constant time**, the overall worst-case running time is:
$$O(n^2)$$

Omega (Ω) of Insertion Sort: Worst Case

$A[1:n/3]$	$A[n/3+1:2n/3]$	$A[2n/3+1:n]$
each of the $n/3$ largest values moves	through each of these $n/3$ positions	to somewhere in these $n/3$ positions

Omega (Ω) of Insertion Sort: Worst Case

Understanding the Lower Bound

- To claim that insertion sort has a worst-case running time of $\Omega(n^2)$, we must show that there exists at least one input that requires at least cn^2 time for some constant $c > 0$.
- Consider an input where the largest $n/3$ elements are in the first $n/3$ positions.
- These values must be moved to the last $n/3$ positions.
- Each of these $n/3$ elements must pass through the middle $n/3$ positions one step at a time, requiring at least:

$$(n/3) \times (n/3) = n^2/9$$

- Since $n^2/9 = \Omega(n^2)$, the worst-case time complexity is:

$$\Omega(n^2)$$

Conclusion: Worst-Case Complexity

Final Observation

- We have shown that insertion sort runs in:

$$O(n^2) \quad \text{(upper bound)} \quad \text{and} \quad \Omega(n^2) \quad \text{(lower bound)}$$

- Since both bounds match up to constant factors, the worst-case running time is:

$$\Theta(n^2)$$

- This means that insertion sort **always requires** $\Theta(n^2)$ operations in the worst case.
- However, in Chapter 2, we saw that the **best case** running time is:

$$\Theta(n)$$

Chapter 3: Characterizing Running Times

Overview of Asymptotic Notation

- Chapter 3.1: $O-$notation, $\Omega-$notation, and $\theta-$notation
- **Chapter 3.2: Asymptotic notation – formal definitions**
- Chapter 3.3: Standard notations and common functions

Big-O Notation: Formal Definition

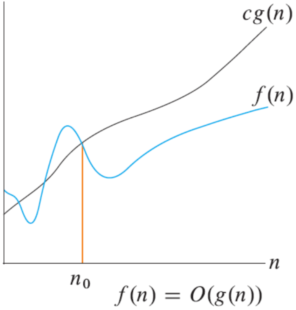
What is Big-O Notation?

- As seen in Section 3.1, O-notation provides an **asymptotic upper bound**.
- It describes how a function grows **at most** at the rate of another function, up to a constant factor.
- Formally, for a given function $g(n)$, the set $O(g(n))$ is defined as:

$$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

- This definition ensures that $f(n)$ is bounded above by $g(n)$ for sufficiently large n.

Visualizing Big-O Notation



Interpretation:

- The function $f(n)$ is always below or equal to $cg(n)$ for $n \geq n_0$.
- This means $g(n)$ serves as an **upper bound** for $f(n)$ in the long run.

Key Assumptions in Big-O Notation

- The definition requires $f(n)$ to be **asymptotically nonnegative** for sufficiently large n.
- That means $f(n) \geq 0$ for all $n \geq n_0$.
- Likewise, the function $g(n)$ must also be **asymptotically nonnegative**.
- Otherwise, the set $O(g(n))$ would be empty.

Big-O Notation as a Set

- Mathematically, we define Big-O notation using **set notation**.
- A function $f(n)$ belongs to $O(g(n))$:

$$f(n) \in O(g(n))$$

- However, in common usage, we often write:

$$f(n) = O(g(n))$$

- Even though this is an **abuse of equality**, it is widely accepted for simplicity.

Example: Showing $4n^2 + 100n + 500 = O(n^2)$

- Consider the function:

$$f(n) = 4n^2 + 100n + 500$$

- We need to find constants c and n_0 such that:

$$f(n) \leq cn^2, \quad \forall n \geq n_0$$

- Dividing by n^2:

$$4 + \frac{100}{n} + \frac{500}{n^2} \leq c$$

- Choosing $n_0 = 1$, $c = 604$ works.
- Choosing $n_0 = 10$, $c = 19$ also works.
- Thus, $f(n) \in O(n^2)$.

Big-Omega (Ω) Notation: Formal Definition

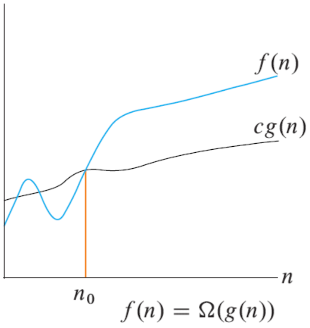
What is Big-Omega Notation?

- Just as O-notation provides an **asymptotic upper bound**, Ω-notation provides an **asymptotic lower bound**.
- It describes how a function grows **at least as fast** as another function, up to a constant factor.
- Formally, for a given function $g(n)$, the set $\Omega(g(n))$ is defined as:

$$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$

- This definition ensures that $f(n)$ is bounded below by $g(n)$ for sufficiently large n.

Visualizing Big-Omega (Ω)



Interpretation:

- The function $f(n)$ is always above or equal to $cg(n)$ for $n \geq n_0$.
- This means $g(n)$ serves as a **lower bound** for $f(n)$ in the long run.

Example: Showing $4n^2 + 100n + 500 = \Omega(n^2)$

- We have the function:

$$f(n) = 4n^2 + 100n + 500$$

- We need to find constants c and n_0 such that:

$$cn^2 \le f(n), \quad \forall n \ge n_0$$

- Dividing by n^2:

$$c \le 4 + \frac{100}{n} + \frac{500}{n^2}$$

- Choosing $n_0 = 1$, $c = 4$ works.
- Thus, $f(n) \in \Omega(n^2)$.

Example: Showing $\frac{n^2}{100} - 100n - 500 = \Omega(n^2)$

- Consider:

$$f(n) = \frac{n^2}{100} - 100n - 500$$

- We divide by n^2:

$$\frac{1}{100} - \frac{100}{n} - \frac{500}{n^2} \leq c$$

- Choosing $n_0 = 10,005$, we can set $c = 2.49 \times 10^{-9}$.
- If we choose a larger n_0, we can increase c closer to $\frac{1}{100}$.
- This proves that $f(n) \in \Omega(n^2)$.

Theta (Θ) Notation: Formal Definition

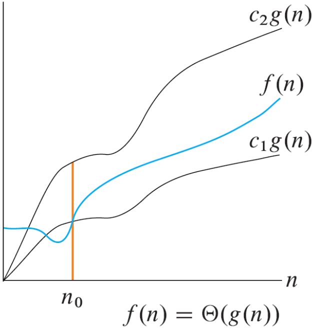
What is Theta Notation?

- We use Θ-notation to represent an **asymptotically tight bound**.
- It describes how a function grows **exactly at the rate** of another function, up to constant factors.
- Formally, for a given function $g(n)$, the set $\Theta(g(n))$ is defined as:

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 > 0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$$

- This means $f(n)$ is bounded both **above and below** by $g(n)$ within constant factors.

Visualizing Theta (Θ)



Interpretation:

- The function $f(n)$ is sandwiched between two constant multiples of $g(n)$.
- This means $g(n)$ provides an **exact rate of growth** for $f(n)$.

Theorem: Relationship Between $O(g(n)), \Omega(g(n))$, and $\Theta(g(n))$

Theorem 3.1:

- A function $f(n) = \Theta(g(n))$ if and only if

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

- This means if $f(n)$ is both upper and lower bounded by $g(n)$, then $f(n)$ is tightly bound.

Proof of Theorem 3.1: If $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$, then $f(n) = \Theta(g(n))$

Proof:

Backward direction: Assume $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- Since $f(n) = O(g(n))$, there exist constants $c_2 > 0$ and $n_2 > 0$ such that:

$$f(n) \leq c_2 g(n), \quad \forall n \geq n_2$$

- Since $f(n) = \Omega(g(n))$, there exist constants $c_1 > 0$ and $n_1 > 0$ such that:

$$f(n) \geq c_1 g(n), \quad \forall n \geq n_1$$

- Let $n_0 = \max(n_1, n_2)$. Then, for $n \geq n_0$, we have:

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

- This matches the definition of $\Theta(g(n))$, so $f(n) = \Theta(g(n))$.

Proof of Theorem 3.1: If $f(n) = \Theta(g(n))$, then $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Proof:

Forward direction: Assume $f(n) = \Theta(g(n))$.

- By definition of $\Theta(g(n))$, there exist positive constants c_1, c_2 and n_0 such that:

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \forall n \geq n_0$$

- **To show $f(n) = O(g(n))$:**
  - From the upper bound $f(n) \leq c_2 g(n)$, we see that $f(n)$ is at most a constant multiple of $g(n)$ for sufficiently large n.
  - This satisfies the definition of $O(g(n))$, so $f(n) \in O(g(n))$.
- **To show $f(n) = \Omega(g(n))$:**
  - From the lower bound $c_1 g(n) \leq f(n)$, we see that $f(n)$ is at least a constant multiple of $g(n)$ for sufficiently large n.
  - This satisfies the definition of $\Omega(g(n))$, so $f(n) \in \Omega(g(n))$.
- Since we have shown both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, it follows that:

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n)) \Rightarrow f(n) = \Theta(g(n))$$

Question?