Review: Algorithm Analysis

Viet Tung Hoang
1. Overview

2. Asymptotic order of growth
Running Time Matters

Time

Centuries

Years

Days

Hours

Minutes

Seconds

3

Seconds

Minutes

Hours

Days

Years

Centuries

$10$

$10^2$

$10^3$

$10^4$

$10^5$

$10^6$

$n$

$n!$

$n^2$

$n^3$

$n \log(n)$

$n$

Practical

Prohibitive
Type of Analysis: Worst-case

Running time guarantees for any input of size $n$ - Draconian view, but generally captures efficiency in practice

Exception: Some exponential-time algorithms are used widely in practice because the worst-case instances seem to be rare

Simplex algorithm  Linux grep  K-means algorithm
Digression: Amortized Analysis

Worst-case running time of a **sequence** of \( n \) tasks

**Example:** Implementing a \( k \)-bit counter, starting from 0. Cost is measured by the number of bit flips

<table>
<thead>
<tr>
<th># increments, ( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total flips</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

But for a sequence of \( n \) increments from 0, cost at most \( 2n \) flips for any choice of \( k \)

An increment may cost \( k \) flips \( \Rightarrow \) \( nk \) flips for \( n \) increments
Amortized Analysis on Counters

Total number of flips is at most:

\[ n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots \leq 2n \]

n/4 flips  n/2 flips  n flips
Agenda

1. Overview

2. Asymptotic order of growth
Asymptotic Behavior

How to analyze running time?

- **Want**: a *simple* analysis, independent of programming language, architecture, etc

**Main idea**: focus on how running time scale with $n$ (input size)

Informally, only focus on the behavior when $n$ is *large enough*
An Illustration: Insertion Sort vs Merge Sort

A slick implementation of **Insertion Sort**: $n^2$ steps

A poor implementation of **Merge Sort**: $100n \log_2(n)$ steps

Merge Sort still wins in the long run
**Big-Oh Notation**

**Upper bounds.** \( T(n) \in O(f(n)) \) if there exist constants \( c > 0 \) and \( n_0 \geq 0 \) such that \( T(n) \leq c \cdot f(n) \) for all \( n \geq n_0 \)

**Example:** \( T(n) = 32n^2 + 17n + 1 \)

\[
\begin{align*}
    n_0 &= 1 \\
    c &= 50 \\
    T(n) &= O(n^2)
\end{align*}
\]

**Theorem:** If \( \lim_{n \to \infty} \frac{T(n)}{f(n)} \to a \geq 0 \) then \( T(n) \in O(f(n)) \)
A Usage Example: Selection Sort

- Maintain a sorted left half and a right unsorted one
- Loop: Swap the min of right half with its first element, and extend the left half

Round 1: 36 13 20 25 15 4 51 33 64 22 84 8

Round 2: 4 13 20 25 15 36 51 33 64 22 84 8

... 4 8 20 25 15 36 51 33 64 22 84 13

Total $n$ rounds, and $O(n^2)$ time
**Big-Omega Notation**

**Lower bounds.** \( T(n) \in \Omega(f(n)) \) if there exist constants \( c > 0 \) and \( n_0 \geq 0 \) such that \( T(n) \geq c \cdot f(n) \) for all \( n \geq n_0 \)

Example: \( T(n) = 32n^3 + 17n + 1 \)

\[ n_0 = 1 \quad c = 32 \]

\( T(n) = \Omega(n^2) \)

**Theorem:** If \( \lim_{n \to \infty} \frac{f(n)}{T(n)} \to a \geq 0 \) then \( T(n) \in \Omega(f(n)) \)
A Usage Example: Comparison-based Sort

**Want**: Sort these coins based on their weights using a scale

**Rules:**
- Can only weigh two coins $A$ and $B$ at a time
- Can only tell whether $A$ is heavier than $B$, but not how much
A Usage Example: Comparison-based Sort

Comparison-based Sort:
-Determine order based on **pairwise comparison** of elements, not actual value

<table>
<thead>
<tr>
<th>Comparison-based</th>
<th>Not comparison-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge Sort, Quick Sort</td>
<td>Counting Sort</td>
</tr>
<tr>
<td>Heap Sort, Insertion Sort</td>
<td>Radix Sort</td>
</tr>
<tr>
<td>Selection Sort, Bubble Sort</td>
<td>Bucket Sort</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Need extra assumption on the elements

**Theorem:** Any comparison-based sort algorithm must use \( \Omega(n \log_2(n)) \) comparisons
**Big-Theta Notation**

**Tight bounds.** $T(n) \in \Theta(f(n))$ if there exist constants $c_1, c_2 > 0$ and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$

**Example:** $T(n) = 32n^2 + 17n + 1$

\[
\begin{align*}
n_0 &= 1 \\
c_1 &= 32, c_2 = 50 \\
T(n) &= \Theta(n^2)
\end{align*}
\]

**Theorem:** If $\lim_{n \to \infty} \frac{T(n)}{f(n)} \to a > 0$ then $T(n) \in \Theta(f(n))$
A Usage Example: Selection Sort, Again

- Recall: Selection Sort takes $O(n^2)$ time for any input.
- Selection Sort takes $\Omega(n^2)$ time for finding the min elements alone.

<table>
<thead>
<tr>
<th>Round 1:</th>
<th>36</th>
<th>33</th>
<th>40</th>
<th>15</th>
<th>22</th>
<th>9</th>
<th>7</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

$\geq n$ steps

<table>
<thead>
<tr>
<th>Round 2:</th>
<th>1</th>
<th>33</th>
<th>40</th>
<th>15</th>
<th>22</th>
<th>9</th>
<th>7</th>
<th>2</th>
<th>36</th>
</tr>
</thead>
</table>

$\geq n - 1$ steps

... | 1 | 2 | 40 | 15 | 22 | 9 | 7 | 33 | 36 |

Running time $\geq n + (n - 1) + \cdots + 1 = \frac{n(n + 1)}{2} = \Omega(n^2)$

**Conclusion:** Selection Sort takes $\Theta(n^2)$ time.
Asymptotic Bounds for Common Functions

Polynomials:

\[ f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \text{ with } a_d > 0 \]

\[ f(n) = \Theta(n^d) \]

Rule: Pick the term of highest order

Log:

\[ \log_a(n) \in \Theta\left(\log_b(n)\right) \] for any constants \( a, b > 0 \)

Write \( O(\log(n)) \) without specifying the base
Asymptotic Bounds for Common Functions

**Exponential**

\[ a^n \text{ with } a > 1 \]

\[ a^n \in \Omega(n^c) \]

**Polynomial**

\[ n^c \text{ with } c > 0 \]

**Log**

\[ \log(n) \]

\[ \log(n) \in O(n^c) \]