Algorithms
(Algorithm Analysis)

Pramod Ganapathhi
Department of Computer Science
State University of New York at Stony Brook

February 15, 2021
Contents

- Complexity Analysis
- Worst-case, Best-case, and Average-case
- Asymptotic Notations and Complexity Classes
Complexity Analysis
Complexity analysis

• Framework for analyzing the efficiency/complexity of algorithms e.g.: time complexity and space complexity
• Running time of a computer program depends on:
  • Algorithm
  • Input size
  • Input data distribution
  • Machine or computing system
  • Operating system
  • Compiler
  • Programming language
  • Coding
• We analyze the running time of algorithms using asymptotic analysis
Complexity analysis

• Running time of an algorithm can be considered as a function of the algorithm’s input size
## Complexity analysis

<table>
<thead>
<tr>
<th>Problem</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search in a sorted array</td>
<td>$\mathcal{O}(\log n)$</td>
</tr>
<tr>
<td>Search in an unsorted array, Integer addition</td>
<td>$\mathcal{O}(n)$</td>
</tr>
<tr>
<td>Generate primes</td>
<td>$\mathcal{O}(n \log \log n)$</td>
</tr>
<tr>
<td>Sorting, Fast Fourier transform</td>
<td>$\mathcal{O}(n \log n)$</td>
</tr>
<tr>
<td>Integer multiplication</td>
<td>$\mathcal{O}(n^2)$</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>$\mathcal{O}(n^3)$</td>
</tr>
<tr>
<td>Linear programming</td>
<td>$\mathcal{O}(n^{3.5})$</td>
</tr>
<tr>
<td>Primality test</td>
<td>$\mathcal{O}(n^{10})$</td>
</tr>
<tr>
<td>Satisfiability problem</td>
<td>$\mathcal{O}(2^n)$</td>
</tr>
<tr>
<td>Traveling salesperson problem</td>
<td>$\mathcal{O}((n - 1)!)$</td>
</tr>
<tr>
<td>Sudoku, Chess, Checkers, Go</td>
<td>expo. class</td>
</tr>
<tr>
<td>Simulate problem, Halting problem</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Program correctness, Program equivalence</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Integral roots of a polynomial</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
Polynomial and exponential functions

The graph shows the comparison of different functions:

- $n!$ (factorial)
- $2^n$ (exponential)
- $n^2$ (quadratic)
- $n\log(n)$
- $\log(n)$
- $\sqrt{n}$
- $n$

The graph plots these functions on the y-axis against $n$ on the x-axis, with $n$ ranging from 20 to 100.
Units for measuring running time

- **Basic operation** is the most important operation of the algorithm. Each basic operation takes constant time.
  - Arithmetic operation ($\times, \div, +, -$)
  - Comparison operation ($<, \leq, =, \neq, >, \geq$)
  - Memory operation ($a \leftarrow b, C[i]$)
  - Function invocation and return
Units for measuring running time

Array-Sum($A[0..n-1]$)

1. $sum \leftarrow 0$ \quad \triangleright \quad 1 \text{ op}
2. $\textbf{for } i \leftarrow 0 \textbf{ to } n-1 \textbf{ do}$ \quad $\triangleright \quad n \times (\text{cmp} + \text{inc}) = 2n \text{ ops}$
3. $sum \leftarrow sum + A[i]$ \quad $\triangleright \quad n \times (\text{index} + \text{add} + \text{assign}) = 3n \text{ ops}$
4. $\textbf{return } sum$ \quad $\triangleright \quad 1 \text{ op}

- Runtime $= 1 + n(2 + 3) + 1 = 5n + 2$ operations
  ($n$ comparisons, $n$ increments, $n$ memory index accesses, $n + 1$ assignments, $n$ additions, 1 function return)
Worst-case, best-case, and average-case analysis

- **Worst-case complexity** $T_{\text{worst}}(n)$ of an algorithm. Complexity for the worst-case input of size $n$ for which the algorithm runs the longest among all possible inputs of that size.
- **Best-case complexity** $T_{\text{best}}(n)$ of an algorithm. Complexity for the best-case input of size $n$ for which the algorithm runs the shortest among all possible inputs of that size.
- **Average-case complexity** $T_{\text{avg}}(n)$ of an algorithm. Complexity for a typical or random input of size $n$.
- **Amortized complexity** $T_{\text{amortized}}(n)$ of an algorithm. Average complexity for a sequence of operations.
Worst-case, best-case, and average-case analysis

Problem
What are the worst-case, best-case, and average-case analyses for the sequential search algorithm?

**Sequential-Search**($A[0..n-1], key$)

**Input:** An array $A$ and search key $key$

**Output:** The index of the first element in $A$ that matches $key$ or $-1$ if there are no matching elements

1. $i \leftarrow 0$
2. while $i < n$ and $A[i] \neq key$ do
3.   $i \leftarrow i + 1$
4. if $i < n$ then return $i$
5. else return $-1$
Worst-case, best-case, and average-case analysis

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Solution" /></td>
</tr>
</tbody>
</table>

Let $p \in [0, 1]$ be the probability of successful search
The prob. of first match occurring at any position be the same
$T_{\text{avg}}(n) = (1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}) + n \cdot (1 - p)$
Simplifying, $T_{\text{avg}}(n) = \frac{p(n+1)}{2} + n(1 - p)$.

What do you get when you set $p = 1$ or $p = 0$?
Asymptotic Notations and Complexity Classes
## Asymptotic notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O\left(g(n)\right)$ at most $g(n)$</td>
<td>Set of all functions with the same or lower order of growth as $g(n)$</td>
</tr>
<tr>
<td></td>
<td>$3n^2 \in O\left(n^2\right)$, $n^2/17 + n \in O\left(n^2\right)$, $n(n-1)/2 \in O\left(n^2\right)$</td>
</tr>
<tr>
<td></td>
<td>$n \in O\left(n^2\right)$, $4\sqrt{n} + 3\log^2 n \in O\left(n^2\right)$, $2000 \in O\left(n^2\right)$</td>
</tr>
<tr>
<td></td>
<td>$n^3 \notin O\left(n^2\right)$, $0.001n^{\pi-1} \notin O\left(n^2\right)$, $n^4 + n + 1 \notin O\left(n^2\right)$</td>
</tr>
<tr>
<td>$\Omega\left(g(n)\right)$ at least $g(n)$</td>
<td>Set of all functions with the same or higher order of growth as $g(n)$</td>
</tr>
<tr>
<td></td>
<td>$3n^2 \in \Omega\left(n^2\right)$, $n^2/17 + n \in \Omega\left(n^2\right)$, $n(n-1)/2 \in \Omega\left(n^2\right)$</td>
</tr>
<tr>
<td></td>
<td>$n^3 \in \Omega\left(n^2\right)$, $0.001n^{\pi-1} \in \Omega\left(n^2\right)$, $n^4 + n + 1 \in \Omega\left(n^2\right)$</td>
</tr>
<tr>
<td></td>
<td>$n \notin \Omega\left(n^2\right)$, $4\sqrt{n} + 3\log^2 n \notin \Omega\left(n^2\right)$, $2000 \notin \Omega\left(n^2\right)$</td>
</tr>
<tr>
<td>$\Theta\left(g(n)\right)$ same as $g(n)$</td>
<td>Set of all functions with the same order of growth as $g(n)$</td>
</tr>
<tr>
<td></td>
<td>$3n^2 \in \Theta\left(n^2\right)$, $n^2/17 + n \in \Theta\left(n^2\right)$, $n(n-1)/2 \in \Theta\left(n^2\right)$</td>
</tr>
</tbody>
</table>
Definition

A function $T(n)$ is said to be in $\mathcal{O}(g(n))$, denoted $T(n) \in \mathcal{O}(g(n))$, if $T(n)$ is bounded above by some constant multiple of $g(n)$ for all large $n$, i.e., if there exist some positive constant $c$ and nonnegative integer $n_0$ such that

$$T(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0$$
$O()$ notation

$\begin{align*}
T(n) & \in O(g(n)) \\
c \cdot g(n) &
\end{align*}$
### Problem

Show that if $f(n)$ is a polynomial of degree $d$, that is, 
$$T(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$$
and $a_d > 0$, then $T(n) \in \mathcal{O}(n^d)$.

### Solution

- For $n \geq 1$, we have $1 \leq n \leq n^2 \leq \cdots \leq n^d$.
- So, $a_d n^d + \cdots + a_1 n + a_0 \leq (|a_d| + \cdots + |a_1| + |a_0|) n^d$
- By choosing $c = (|a_d| + \cdots + |a_1| + |a_0|)$ and $n_0 = 1$, we get $T(n) \in \mathcal{O}(n^d)$.
$\mathcal{O}()$ notation
## Definition

A function $T(n)$ is said to be in $\Omega(g(n))$, denoted $T(n) \in \Omega(g(n))$, if $T(n)$ is bounded below by some constant multiple of $g(n)$ for all large $n$, i.e., if there exist some positive constant $c$ and nonnegative integer $n_0$ such that

$$T(n) \geq c \cdot g(n) \quad \text{for all } n \geq n_0$$
Ω() notation

\[ T(n) \in \Omega(g(n)) \]
**Ω () notation**

- $Ω(n!)$
- $Ω(2^n)$
- $Ω(n^2)$
- $Ω(n \log n)$
- $Ω(n)$
- $Ω(√n)$
- $Ω(\log n)$
- $Ω(1)$
\( \Theta () \) notation

**Definition**

- A function \( T(n) \) is said to be in \( \Theta (g(n)) \), denoted \( T(n) \in \Theta (g(n)) \), if \( T(n) \in O(g(n)) \) and \( T(n) \in \Omega (g(n)) \).
- A function \( T(n) \) is said to be in \( \Theta (g(n)) \), denoted \( T(n) \in \Theta (g(n)) \), if \( T(n) \) is bounded both above and below by some constant multiples of \( g(n) \) for all large \( n \), i.e., if there exist some positive constants \( c_1, c_2 \) and nonnegative integer \( n_0 \) such that

\[
T(n) \in [c_2 \cdot g(n), c_1 \cdot g(n)] \quad \text{for all } n \geq n_0
\]
\( \Theta () \) notation

\[ T(n) \in \Theta(g(n)) \]
Problem

Show that \( \frac{1}{2}n(n - 1) \in \Theta(n^2) \).

Solution

- **Step 1.** Show that \( \frac{1}{2}n(n - 1) \in \mathcal{O}(n^2) \)
  \[
  \frac{1}{2}n(n - 1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \text{ for all } n \geq 0
  \]

- **Step 2.** Show that \( \frac{1}{2}n(n - 1) \in \Omega(n^2) \)
  \[
  \frac{1}{2}n(n - 1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n \frac{1}{2}n = \frac{1}{4}n^2 \text{ for all } n \geq 2
  \]

- As \( c_2 = \frac{1}{4}, \ c_1 = \frac{1}{2}, \) and \( n_0 \geq 2 \), we have the result.
\( \Theta() \) notation

\( \Theta(1) \), \( \Theta(\log n) \), \( \Theta(\sqrt{n}) \), \( \Theta(n) \), \( \Theta(n \log n) \), \( \Theta(n^2) \), \( \Theta(2^n) \), \( \Theta(n!) \)
Properties

<table>
<thead>
<tr>
<th>Notation</th>
<th>Reflexivity</th>
<th>Symmetry</th>
<th>Transitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O} ()$</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>$\Omega ()$</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>$\Theta ()$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- $f(n) \in \mathcal{O} (g(n))$ if and only if $g(n) \in \Omega (f(n))$
- If $t_1(n) \in \mathcal{O} (g_1(n))$ and $t_2(n) \in \mathcal{O} (g_2(n))$, then $t_1(n) + t_2(n) \in \mathcal{O} (\max(g_1(n), g_2(n)))$
- How do you formally prove the propositions above?
Comparing orders of growth

\[ \lim_{n \to \infty} \frac{T(n)}{g(n)} = \begin{cases} 
0 & \text{implies } T(n) \text{ has smaller growth rate than } g(n), \\
c & \text{implies } T(n) \text{ has same growth rate as } g(n), \\
\infty & \text{implies } T(n) \text{ has larger growth rate than } g(n). 
\end{cases} \]

\[ \lim_{n \to \infty} \frac{T(n)}{g(n)} = \begin{cases} 
0 & \text{implies } T(n) \in o(g(n)), \\
c & \text{implies } T(n) \in \Theta(g(n)), \\
\infty & \text{implies } T(n) \in \omega(g(n)). 
\end{cases} \]

\[ \lim_{n \to \infty} \frac{T(n)}{g(n)} = \lim_{n \to \infty} \frac{T'(n)}{g'(n)} \quad \text{(L’Hôpital’s rule)} \]
Example

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Prove that $\log(n!) \in \mathcal{O}(n \log n)$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Show that $\log(n!) \leq cn \log n$ for some $c &gt; 0$ and $n \geq n_0$</td>
</tr>
<tr>
<td>• S.T. $\log(n!) \leq \log((n^n)^c)$</td>
</tr>
<tr>
<td>• S.T. $\log(n!) \leq \log(n^n)$</td>
</tr>
<tr>
<td>• S.T. $n! \leq n^n$</td>
</tr>
<tr>
<td>• S.T. $\prod_{i=1}^{n} i \leq \prod_{i=1}^{n} n$</td>
</tr>
<tr>
<td>• S.T. $i \leq n$ for all $i \in [1, n]$</td>
</tr>
<tr>
<td>• This is trivially true from the constraints.</td>
</tr>
<tr>
<td>• Thus, the theorem follows.</td>
</tr>
</tbody>
</table>
Determining complexities from pseudocodes

**SEQUENCE-OF-STATEMENTS**

1. statement $s_1$
2. statement $s_2$
3. statement $s_3$

$$\text{total time} = \text{time}(s_1) + \text{time}(s_2) + \text{time}(s_3)$$

**IF-ELSE-LADDER**

1. **if** condition1 **then**
2. block $b_1$
3. **else if** condition2 **then**
4. block $b_2$
5. **else**
6. block $b_3$

$$\text{total time} = \max( \text{time}(b_1), \text{time}(b_2), \text{time}(b_3) )$$
### LOOPS

1. `for i ← 1 to m do`
2. `for j ← 1 to n do`
3. block $b$

\[
\text{total time} = mn \times \text{time}(b)
\]

(assuming block $b$ takes the same time in every iteration)

### FUNCTIONS

1. `for i ← 1 to m do`
2. `for j ← 1 to n do`
3. $F(i, j)$ \[\triangleright\] Suppose this takes $\Theta(ij)$ time

\[
\text{total time} = \sum_{i=1}^{m} \sum_{j=1}^{n} \text{time}(F(i, j))
\]