Lecture 3:
• Divide and Conquer: Mergesort
• Asymptotic Analysis

Sep 14, 2018
Asymptotic Analysis
Asymptotic Order Of Growth

• Predicting the wall-clock time of an algorithm is nigh impossible.
  • What machine will actually run the algorithm?
  • Impossible to exactly count “operations”?
• Do we really need to worry about this problem?
  • Mostly we want to compare algorithms, so we can select the right one for the job
  • Mostly we don’t care about small inputs, we care about how the algorithm will scale

Asymptotic Order Of Growth

\[ y = n^2 \]
\[ y = 10n + 50 \]
Asymptotic Order Of Growth

- **Asymptotic Analysis:** How does the running time grow as the size of the input grows?

\[ f(n) \implies g(n) \]

- exact running time (messy, dependent on the machine)

\[\text{order of growth}\]
**Asymptotic Order Of Growth**

- **“Big-Oh” Notation:** \( f(n) = O(g(n)) \) if there exists \( c \in (0, \infty) \) and \( n_0 \in \mathbb{N} \) such that \( f(n) \leq c \cdot g(n) \) for every \( n \geq n_0 \).

- Asymptotic version of \( f(n) \leq g(n) \)

- Roughly equivalent to \( \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty \)

\[
f(n) = 3n^2 + n \quad g(n) = n^2
\]

\[
\text{Lim: } f(n) = O(g(n))
\]

\[
\text{Pf: } c = 4 \quad n_0 = 1
\]

\[
\forall \ n > n_0 \quad 3n^2 + n \leq 4n^2
\]

\[
3n^2 + n \leq 3n^2 + n^2 \leq 4n^2 \leq 4n^2 \quad \alpha
\]
Ask the Audience

• **“Big-Oh” Notation:** $f(n) = O(g(n))$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.

• Which of these statements are true?
  - $3n^2 + n = O(n^2)$ ✓
  - $n^3 = O(n^2)$
  - $10n^4 = O(n^5)$
  - $\log_2 n = O(\log_{16} n)$

\[ \lim_{n \to \infty} \frac{n^3}{n^2} = \infty \]

\[ \lim_{n \to \infty} \frac{n^3}{n^2} = \infty \]

\[ c = 1 \quad n_0 = 10 \]

\[ \forall n > n_0 \quad 10n^4 \leq n^5 \]

\[ \log_{16} n = \frac{\log_2 n}{\log_2 16} = \frac{1}{4} \log_2 n \]
Big-Oh Rules

- Constant factors can be ignored
  \[ \forall C > 0 \quad Cn = O(n) \quad \text{and} \quad f(n) = C \cdot g(n) \Rightarrow f(n) = O(g(n)) \]

- Smaller exponents are Big-Oh of larger exponents
  \[ \forall a > b \quad n^b = O(n^a) \quad \text{and} \quad n^2 = O(n^{2.0001}) \]

- Any logarithm is Big-Oh of any polynomial
  \[ \forall a, \varepsilon > 0 \quad \log_a n = O(n^{\varepsilon}) \quad \text{and} \quad \log_2 n = O(n^{0.0001}) \]

- Any polynomial is Big-Oh of any exponential
  \[ \forall a > 0, b > 1 \quad n^a = O(b^n) \quad \text{and} \quad n^{1000} = O(1.0001^n) \]

- Lower order terms can be dropped
  \[ n^2 + n^{3/2} + n = O(n^2) \quad \Rightarrow \quad f_1(n) + f_2(n) \quad \text{and} \quad f_1(n) = O(g(n)), f_2(n) = O(g(n)) \quad \Rightarrow \quad f_1 + f_2 = O(g) \]
A Word of Caution

• The notation $f(n) = O(g(n))$ is weird—do not take it too literally

\[ n = O(n^2) \quad n = O(n^3) \quad \text{(Not really an “=” sign)} \]

Claim: $n = O(1)$

\[
\begin{align*}
    n &= \sum_{i=1}^{n} 1 = \sum_{i=1}^{n} O(1) \\
    &= \sum_{i=2}^{n} O(1) \\
    &\vdots \\
    &= \sum_{i=n}^{n} O(1) = O(1)
\end{align*}
\]
Asymptotic Order Of Growth

• **“Big-Omega” Notation:** \( f(n) = \Omega(g(n)) \) if there exists \( c \in (0, \infty) \) and \( n_0 \in \mathbb{N} \) s.t. \( f(n) \geq c \cdot g(n) \) for every \( n \geq n_0 \).
  
  - Asymptotic version of \( f(n) \geq g(n) \)
  - Roughly equivalent to \( \lim_{n \to \infty} \frac{f(n)}{g(n)} > 0 \)

• **“Big-Theta” Notation:** \( f(n) = \Theta(g(n)) \) if there exists \( c_1 \leq c_2 \in (0, \infty) \) and \( n_0 \in \mathbb{N} \) such that \( c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \) for every \( n \geq n_0 \).
  
  - Asymptotic version of \( f(n) = g(n) \)
  - Roughly equivalent to \( \lim_{n \to \infty} \frac{f(n)}{g(n)} \in (0, \infty) \)
Asymptotic Running Times

• **We usually write running time as a Big-Theta**
  - Exact time per operation doesn’t appear
  - Constant factors do not appear
  - Lower order terms do not appear

• **Examples:**
  - $30 \log_2 n + 45 = \Theta(\log n)$
  - $Cn \log_2 2n = \Theta(n \log n)$
  - $\sum_{i=1}^{n} i = \Theta(n^2)$
Asymptotic Order Of Growth

- **“Little-Oh” Notation:** \( f(n) = o(g(n)) \) if for every \( c > 0 \) there exists \( n_0 \in \mathbb{N} \) s.t. \( f(n) < c \cdot g(n) \) for every \( n \geq n_0 \).
  - Asymptotic version of \( f(n) < g(n) \)
  - Roughly equivalent to \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)

- **“Little-Omega” Notation:** \( f(n) = \omega(g(n)) \) if for every \( c > 0 \) there exists \( n_0 \in \mathbb{N} \) such that \( f(n) > c \cdot g(n) \) for every \( n \geq n_0 \).
  - Asymptotic version of \( f(n) > g(n) \)
  - Roughly equivalent to \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \)
Ask the Audience!

- Rank the following functions in increasing order of growth (i.e. $f_1, f_2, f_3, f_4$ so that $f_i = O(f_{i+1})$)
  - $n \log_2 n$
  - $n^2$
  - $100n$
  - $3^{\log_2 n}$

Correct Order: $100n, n \log_2 n, 3^{\log_2 n}, 100n, n^2, 3^{\log_2 n}, n^2, 3^{\log_2 n}$
\[100n \text{ vs. } n \log_2 n\]

\[100n = O(n \log_2 n), \quad c = 100, \quad n_0 = 2\]

\[100n \leq 100n \log_2 n = O(n \log_2 n)\]

\[n \log_2 n \text{ vs. } n^2\]

\[n \cdot \log_2 n \text{ vs. } n \cdot n\]

\[O(n) \cdot O(\log n) \text{ vs. } O(n) \cdot O(n)\]

\[2^{\log_2 n} = n, \quad 3^{\log_2 n} = \left(2^{\log_2 3}\right)^{\log_2 n} = (2^{\log_2 n})^{\log_2 3} = n^{\log_2 3} \approx 1.59\]

\[3^{\log_2 n} = O(n^2)\]

\[n \log_2 n = O(3^{\log_2 n})\]
## Why Asymptotics Matter

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$n \log_2 n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$1.5^n$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 10$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>4 sec</td>
</tr>
<tr>
<td>$n = 30$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>18 min</td>
<td>$10^{25}$ years</td>
</tr>
<tr>
<td>$n = 50$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>11 min</td>
<td>36 years</td>
<td>very long</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>1 sec</td>
<td>12,892 years</td>
<td>10$^{17}$ years</td>
<td>very long</td>
</tr>
<tr>
<td>$n = 1,000$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>1 sec</td>
<td>18 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>$n = 10,000$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>2 min</td>
<td>12 days</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>$n = 100,000$</td>
<td>$&lt; 1 \text{ sec}$</td>
<td>2 sec</td>
<td>3 hours</td>
<td>32 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>$n = 1,000,000$</td>
<td>1 sec</td>
<td>20 sec</td>
<td>12 days</td>
<td>31,710 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
</tbody>
</table>

- **polynomials** good / **exponentials** bad
- **logarithms** good / **polynomials** bad
- different polynomials make a big difference
Divide and Conquer Algorithms
Divide and Conquer Algorithms

- Split your problem into smaller subproblems
- Recursively solve each subproblem
- Combine the solutions to the subproblems

Useful when combining solutions is easier than solving from scratch

Divide et impera!
-Philip II of Macedon
Divide and Conquer Algorithms

• **Examples:**
  - Mergesort: sorting a list
  - Binary Search: search in a sorted list
  - Karatsuba’s Algorithm: integer multiplication
  - Fast Fourier Transform
  - ...

• **Key Tools:**
  - Correctness: proof by induction
  - Running Time Analysis: recurrences
  - Asymptotic Analysis
Sorting

Given a list of $n$ numbers, put them in ascending order.
A Simple Algorithm: Insertion Sort

Find the maximum

Put it at the end

11 3 42 28 17 8 2 15

↑

11 3 15 28 17 8 2 42
A Simple Algorithm: Insertion Sort

<table>
<thead>
<tr>
<th>11</th>
<th>3</th>
<th>42</th>
<th>28</th>
<th>17</th>
<th>8</th>
<th>2</th>
<th>15</th>
</tr>
</thead>
</table>

Find the maximum

<table>
<thead>
<tr>
<th>11</th>
<th>3</th>
<th>15</th>
<th>28</th>
<th>17</th>
<th>8</th>
<th>2</th>
<th>42</th>
</tr>
</thead>
</table>

Swap it into place, repeat on the rest

<table>
<thead>
<tr>
<th>11</th>
<th>3</th>
<th>15</th>
<th>2</th>
<th>17</th>
<th>8</th>
<th>28</th>
<th>42</th>
</tr>
</thead>
</table>

Repeat \( n - 1 \) times.

| 2 | 3 | 8 | 11 | 15 | 17 | 28 | 42 |
A Simple Algorithm: Insertion Sort

Find the maximum

Swap it into place, repeat on the rest

7.

Running Time:

\[
\begin{align*}
\sum_{i=1}^{n-1} (n-i+1) &= \frac{n(n+1)}{2} - 1 = \Theta(n^2)
\end{align*}
\]
Divide and Conquer: Mergesort

Split

| 11 | 3  | 42 | 28 | 17 | 8  | 2  | 15 |

Recursively Sort

<table>
<thead>
<tr>
<th>11</th>
<th>3</th>
<th>42</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11</td>
<td>28</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17</th>
<th>8</th>
<th>2</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>15</td>
<td>17</td>
</tr>
</tbody>
</table>

Merge

| 2  | 3  | 8  | 11 | 15 | 17 | 28 | 42 |
Divide and Conquer: Mergesort

- **Key Idea:** If $L$, $R$ are sorted lists of length $n$, then we can merge them into a sorted list $A$ of length $2n$ in time $O(n)$
  - Merging two sorted lists is faster than sorting from scratch
Merging

Merge(L,R):

Let n ← len(L) + len(R)
Let A be an array of length n
j ← 1, k ← 1,

For i = 1,...,2n:
    If (j > len(L)):                      // L is empty
        A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):                 // R is empty
        A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):               // L is smallest
        A[i] ← L[j], j ← j+1
    Else:                                // R is smallest
        A[i] ← R[k], k ← k+1

Return A
Merging

MergeSort(A):
    If (len(A) = 1): Return A    // Base Case

    Let \( m \leftarrow \lceil \text{len}(A)/2 \rceil \)    // Split
    Let L ← A[1:m], R ← A[m+1:n]

    Let L ← MergeSort(L)    // Recurse
    Let R ← MergeSort(R)

    Let A ← Merge(L,R)    // Merge

    Return A
Correctness of Mergesort

- **Claim:** The algorithm **Mergesort** is correct

  \( \forall n \in \mathbb{N} \quad \forall \text{ list } A \text{ with } n \text{ numbers} \quad \text{Mergesort returns } A \text{ in sorted order} \)

  **Inductive Hypothesis:** \( H(n) = \forall A \text{ of size } n \text{ Mergesort is correct} \)

  **Base Case:** \( H(1) \) is true, obviously

  **Inductive Step:** Assume \( H(1), \ldots, H(n) \) are all true. We’ll prove \( H(n+1) \).
Running Time of Mergesort

**Inductive Step:**

Assume that Mergesort is correct for all $A$ of size $\leq n$.

1. $\left\lceil \frac{n+1}{2} \right\rceil, \left\lfloor \frac{n+1}{2} \right\rfloor \leq n$
2. $L, R$ are correctly sorted by Mergesort
3. $L, R$ are sorted $\Rightarrow A$ is sorted
4. Mergesort is correct for lists of size $n+1$

**Correctness**

**Mergesort**($A$):

- If $(n = 1)$: Return $A$

- Let $m \leftarrow \lceil n/2 \rceil$
  - Let $L \leftarrow A[1:m]$  
    - Let $R \leftarrow A[m+1:n]$ 
  - Let $L \leftarrow \text{Mergesort}(L)$
  - Let $R \leftarrow \text{Mergesort}(R)$
  - Let $A \leftarrow \text{Merge}(L, R)$

Return $A$

**H(n)**

\[ H(1) \quad \ldots \quad H(n) \]

\[ \downarrow \]

\[ H(n+1) \]
Running Time of Mergesort

\[ T(n) = \text{time to sort a list of size } n \]

\[ T(n) = 2 \times T\left(\frac{n}{2}\right) + C_n \]

\[ T(1) = C \]

\[ T(n) = O(n \log n) \]

\[
\text{MergeSort}(A):
\begin{align*}
\text{If } (n = 1): & \text{ Return } A \\
\text{Let } m & \leftarrow \lceil n/2 \rceil \\
\text{Let } L & \leftarrow A[1:m] \\
\text{Let } R & \leftarrow A[m+1:n] \\
\text{Let } L & \leftarrow \text{MergeSort}(L) \\
\text{Let } R & \leftarrow \text{MergeSort}(R) \\
\text{Let } A & \leftarrow \text{Merge}(L,R) \\
\text{Return } A
\end{align*}
\]
Mergesort Summary

• Sort a list of \( n \) numbers in \( Cn \log_2 2n \) time
  • Can actually sort anything that allows comparisons
  • No comparison based algorithm can be (much) faster

• Divide-and-conquer
  • Break the list into two halves, sort each one and merge
  • Key Fact: Merging is easier than sorting

• Proof of correctness
  • Proof by induction

• Analysis of running time
  • Recurrences