Lecture 2:
• Divide and Conquer: Mergesort
• Asymptotic Analysis

Jan 8, 2020
Divide and Conquer Algorithms
Divide and Conquer Algorithms

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

διαιρει και βασίλευε!
-Philip II of Macedon
Divide and Conquer Algorithms

• **Examples:**
  • Mergesort: sorting a list
  • Binary Search: searching in a sorted list
  • Karatsuba’s Algorithm: multiplying integers
  • Finding the median of a list
  • Fast Fourier Transform
  • ...

• **Key Tools:**
  • Correctness: proof by induction
  • Running Time Analysis: recurrences
  • Asymptotic Analysis
Given a list of \( n \) numbers, put them in ascending order.
A Simple Algorithm

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

1. Find the maximum
2. Swap it into place, repeat on the rest

Repeat $n - 1$ times.
A Simple Algorithm: Insertion Sort

11 3 42 28 17 8 2 15

Find the maximum

11 3 15 28 17 8 2 42

Swap it into place, repeat on the rest

Running Time:

\[ \sum_{i=1}^{n-1} n - i + 1 = \sum_{i=1}^{n-1} i \approx n^2 \]
Divide and Conquer: Mergesort

Split

11 3 42 28 17 8 2 15

11 3 42 28

3 11 28 42

Recursively Sort

17 8 2 15

2 8 15 17

Recursively Sort

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 $Cn$.
  - Merging two sorted lists is faster than sorting from scratch.

- $A$ is sorted
- $L$ is sorted
- $R$ is sorted
- All of $A$ is less than all of $L+R$
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,...,n:
    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):

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

Let m ← [len(A)/2] // 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

  (a) Argue that **Merge** works. If \( L, R \) are sorted lists then **Merge\( (L,R) \)** are sorted.

  \[ \ldots \]

  (b) Prove that **Mergesort** works. For every list \( A \), **Mergesort\( (A) \)** returns \( A \) in sorted order.
Stmt: For every $n \in \mathbb{N}$, for every list $A$ of size $n$, Mergesort($A$) works.

Inductive Hyp.: $H(n):$ for every list $A$ of size $\leq n$, Mergesort($A$) works.

Base Case: $H(1)$ is true “obviously.”

Inductive Step: $\forall n, H(n) \Rightarrow H(n+1)$

Let $A$ be a list of size $n+1$

$L, R$ are lists of size $\leq n$

$L = \text{MS}(L)$ is sorted, $R = \text{MS}(R)$ is sorted ($IH$)

$A = \text{Merge}(L, R)$ is sorted ($Part (a)$)
(a) We will argue that throughout the execution of Merge the following are all true:

- $A, L, R$ are sorted
- $A \leq$ everything in $L + R$

At the start these are true because ...

At each step they remain true because ...
Running Time of Mergesort

\[ T(n) : \text{ the running time on inputs of size } n \]

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

\[ T(1) = C \]

MergeSort(A):

If \( n = 1 \): Return A

Let \( m \leftarrow \left\lceil \frac{n}{2} \right\rceil \)

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
Recursion Trees

\[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + Cn \]
\[ T(1) = C \]

Diagram:
- \( \log_2(n) \) levels
- \( n, n/2, n/4, \ldots \)
- \( Cn \)

\[ 2 \times C\left(\frac{n}{2}\right) = Cn \]
\[ 4 \times C\left(\frac{n}{4}\right) = Cn \]
\[ n \times C = Cn \]
\[ C \cdot n \cdot \log_2(n) \]
Recursion Trees

\[ T(n) = 2 \cdot T(n/2) + Cn \]
\[ T(1) = C \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Input Size at this Level</th>
<th>Work at this Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( Cn )</td>
</tr>
<tr>
<td>1</td>
<td>( n/2 )</td>
<td>( 2 \cdot \left( \frac{Cn}{2} \right) = Cn )</td>
</tr>
<tr>
<td>2</td>
<td>( n/4 ) \hspace{1cm} ( n/4 )</td>
<td>( 4 \cdot \left( \frac{Cn}{4} \right) = Cn )</td>
</tr>
<tr>
<td>( i )</td>
<td>( \ldots ) \hspace{1cm} ( \ldots )</td>
<td>( 2^i \cdot \left( \frac{Cn}{2^i} \right) = Cn )</td>
</tr>
<tr>
<td>( \log_2 n )</td>
<td>( 1 ) \hspace{1cm} ( 1 )</td>
<td>( 2^{\log_2 n} \cdot C = Cn )</td>
</tr>
</tbody>
</table>
Proof by Induction

- **Claim**: \( T(n) = Cn \log_2 2n \)

\[
\begin{align*}
T(n) &= 2 \cdot T(n/2) + Cn \\
T(1) &= C
\end{align*}
\]
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 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
Asymptotic Analysis
Analyzing Running Time

- We don’t have enough information to precisely predict the running time of an algorithm
  - The running time will depend on the size of the input
  - The running time might depend on the input itself
  - Don’t know what machine will run the algorithm
  - Impractical to precisely count operations
Asymptotic Order Of Growth

• 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

- **Asymptotic Analysis**: How does the running time grow as the size of the input grows?
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 \( f(n) \leq g(n) \)

\[ n+1 = O(n) \]

\[ 10x + 5O = O(x^2) \]

\[ n_0 = 15 \quad c = 1 \]
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 versions:

  • \( \exists a, b \geq 0 \quad \forall n \in \mathbb{N} \quad f(n) \leq a \cdot g(n) + b \)

  • \( \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty \quad \begin{align*}
  f(n) &= 3n + 6 \\
  g(n) &= n
  \end{align*} \)

  \[
  \frac{3n+6}{n} = 3 + \frac{6}{n}
  \]
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?
  
  • $3n^2 + n = O(n^2)$
  • $n^3 = O(n^2)$
  • $10n^4 = O(n^5)$
  • $\log_2 n = O(\log_2(\sqrt{n}))$
Big-Oh Rules

- Constant factors can be ignored
  \[ \forall C > 0 \quad Cn = O(n) \]

- Smaller exponents are Big-Oh of larger exponents
  \[ \forall a < b \quad n^a = O(n^b) \]

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

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

- Lower order terms can be dropped
  \[ n^2 + n^{3/2} + n = O(n^2) \]
  \[ n \cdot \log n = n \cdot O(n) \]
  \[ = O(n^2) \]
A Word of Caution

• The notation $f(n) = O(g(n))$ is weird—do not take it too literally
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 $0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$
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:
  • $30 \log_2 n + 45 = \Theta(\log n)$
  • $Cn \log_2 2n = \Theta(n \log n)$
  • $\sum_{i=1}^{n} Ci = \Theta(n^2)$

• We usually “think asymptotically”
  • We don’t even discuss constants when they won’t affect the asymptotic running time
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

• **“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 \( n \geq n_0 \).

• Which of these statements are true?
  - \( 3n^2 + n = o(n^2) \)
  - \( n^3 = o(n^2) \)
  - \( 10n^4 = o(n^5) \)
  - \( \log_2 n = o(\log_2(\sqrt{n})) \)
Ask the Audience!

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

<table>
<thead>
<tr>
<th>$n$</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>10</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 sec</td>
</tr>
<tr>
<td>30</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>10$^{25}$ years</td>
</tr>
<tr>
<td>50</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>11 min</td>
<td>36 years</td>
<td>very long</td>
</tr>
<tr>
<td>100</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>12,892 years</td>
<td>10$^{17}$ years</td>
<td>very long</td>
</tr>
<tr>
<td>1,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 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>10,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 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>100,000</td>
<td>&lt; 1 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>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>