LECTURE 3

Asymptotic Notation
• \( O, \Omega, \Theta, o, \omega \)-notation

Divide and Conquer
• Merge Sort
• Binary Search

Sofya Raskhodnikova and Adam Smith
Review Questions

• If input length doubles, by roughly how much does the worst-case running time of Insertion Sort increase?

  (Answer: 4)

• How long does Insertion Sort run on the input 2,1,4,3,6,5,…,n, n–1 (as a function of n)?

  (Answer: Θ(n). Inner “insertion” loop never has to look more than one position down the list to find the right spot.)
Asymptotic notation

**O-notation (upper bounds):**

We write \( f(n) = O(g(n)) \) if there exist constants \( c > 0, n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**Example:** \( 2n^2 = O(n^3) \) \((c = 1, n_0 = 2)\)

functions, not values

funny, “one-way” equality
Set definition of O-notation

\[ O(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \]

**Example:** \(2n^2 \in O(n^3)\)

(Logicians: \(\lambda n.2n^2 \in O(\lambda n.n^3)\), but it’s convenient to be sloppy, as long as we understand what’s really going on.)
Macro substitution

Convention: A set in a formula represents an anonymous function in the set.

Example: \( f(n) = n^3 + O(n^2) \)

(right-hand side)

means

\( f(n) = n^3 + h(n) \)

for some \( h(n) \in O(n^2) \).
Macro substitution

**Convention:** A set in a formula represents an anonymous function in the set.

**Example:** \( n^2 + O(n) = O(n^2) \)

(left-hand side)

means

for any \( f(n) \in O(n) \):

\[ n^2 + f(n) = h(n) \]

for some \( h(n) \in O(n^2) \).
**Ω-notation (lower bounds)**

*O*-notation is an *upper-bound* notation. It makes no sense to say \( f(n) \) is at least \( O(n^2) \).

\[
\Omega(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}
\]

**Example:** \( \sqrt{n} = \Omega(\lg n) \) \((c = 1, n_0 = 16)\)
**Θ-notation (tight bounds)**

\[ \Theta(g(n)) = O(g(n)) \cap \Omega(g(n)) \]

**Example:** \[ \frac{1}{2} n^2 - 2n = \Theta(n^2) \]

Polynomials are simple:

\[ a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d) \]
0-notation and $\omega$-notation

$O$-notation and $\Omega$-notation are like $\leq$ and $\geq$.

$o$-notation and $\omega$-notation are like $<$ and $>$. 

$$ o(g(n)) = \{ f(n) : \text{for any constant } c > 0, \text{there is a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0 \} $$

**Example:** $2n^2 = o(n^3)$ \quad (n_0 = 2/c)
**O-notation and \( \omega \)-notation**

*O*-notation and \( \Omega \)-notation are like \( \leq \) and \( \geq \).

\( o \)-notation and \( \omega \)-notation are like \( < \) and \( > \).

\[ \omega(g(n)) = \{ f(n) : \text{for any constant } c > 0, \text{ there is a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0 \} \]

**Example:**  \( \sqrt{n} = \omega(\log n) \quad (n_0 = 1 + 1/c) \)
## Summary

<table>
<thead>
<tr>
<th>Notation</th>
<th>… means …</th>
<th>Think…</th>
<th>E.g.</th>
<th>Lim $f(n)/g(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n)=O(n)$</td>
<td>$\exists c&gt;0, n_0&gt;0, \forall n&gt;n_0: 0 \leq f(n) &lt; cg(n)$</td>
<td>Upper bound “≤”</td>
<td>$100n^2$</td>
<td>If it exists, it is $&lt; \infty$</td>
</tr>
<tr>
<td>$f(n)=\Omega(g(n))$</td>
<td>$\exists c&gt;0, n_0&gt;0, \forall n&gt;n_0: 0 \leq cg(n) &lt; f(n)$</td>
<td>Lower bound “≥”</td>
<td>$n^{100}$</td>
<td>If it exists, it is $&gt; 0$</td>
</tr>
<tr>
<td>$f(n)=\Theta(g(n))$</td>
<td>both of the above: $f=\Omega(g)$ and $f=O(g)$</td>
<td>Tight bound “=”</td>
<td>$\log(n!)$</td>
<td>If it exists, it is $&gt; 0$ and $&lt; \infty$</td>
</tr>
<tr>
<td>$f(n)=o(g(n))$</td>
<td>$\forall c&gt;0, n_0&gt;0, \forall n&gt;n_0: 0 \leq f(n) &lt; cg(n)$</td>
<td>“&lt;”</td>
<td>$n^2 = o(2^n)$</td>
<td>Limit exists, =0</td>
</tr>
<tr>
<td>$f(n)=\omega(g(n))$</td>
<td>$\forall c&gt;0, n_0&gt;0, \forall n&gt;n_0: 0 \leq cg(n) &lt; f(n)$</td>
<td>“&gt;”</td>
<td>$n^2 = \omega(\log n)$</td>
<td>Limit exists, =∞</td>
</tr>
</tbody>
</table>

---

1/25/2007

*S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson*
Some functions sorted by asymptotic growth

- $\log(n)$
- $\sqrt{n}$
- $n$
- $n \log(n)$
- $n^2$
- $n^{1,000,000}$
- $2^n$ (beats $n^k$ for any fixed $k$)
- $n!$
The divide-and-conquer design paradigm

1. *Divide* the problem (instance) into subproblems.

2. *Conquer* the subproblems by solving them recursively.

3. *Combine* subproblem solutions.

- We’ll see lots of examples
A faster sort: Merge Sort

**MERGE-SORT** \(A[1 \ldots n]\)

1. If \(n = 1\), done.

2. Recursively sort \(A[1 \ldots \lceil n/2 \rceil]\) and \(A[\lfloor n/2 \rfloor + 1 \ldots n]\).

3. “Merge” the 2 sorted lists.

*Key subroutine: MERGE*
A faster sort: Merge Sort

input $A[1 \ldots n]$

$A[1 \ldots \lceil n/2 \rceil ]$

$A[\lfloor n/2 \rfloor +1 \ldots n]$

MERGE SORT

Sorted $A[1 \ldots \lceil n/2 \rceil ]$

Sorted $A[\lfloor n/2 \rfloor +1 \ldots n]$

MERGE

output

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Merging two sorted arrays

20 12
13 11
7 9
2 1
Merging two sorted arrays

20  12
13  11
7   9
2   1
1

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Merging two sorted arrays
Merging two sorted arrays

20 12 || 20 12
13 11 || 13 11
7 9 || 7 9
2 1 || 2 2
1 || 2

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Merging two sorted arrays

20 12 || 20 12 || 20 12

13 11 || 13 11 || 13 11

7 9 || 7 9 || 7 9

2 1 -> 1 2

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Merging two sorted arrays

20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11
7 9  || 7 9  || 7 9 

2 1 || 2  || 7
1  || 2  || 7
Merging two sorted arrays

20 12 || 20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11 || 13 11
7  9 || 7  9 || 7  9 || 7  9
2  1 || 2  2 || 7  7 || 9  9
1  2 || 2  7 || 7  9 || 9  9

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Merging two sorted arrays
Merging two sorted arrays

20 12
13 11
7 9
2 1

20 12
13 11
7 9
2 2

20 12
13 11
7 9
7 9

20 12
13 11
9 9
9 9

20 12
13 11
13 11
13 11
Merging two sorted arrays
Merging two sorted arrays
Merging two sorted arrays
Merging two sorted arrays

Time = one pass through each array
= $\Theta(n)$ to merge a total of $n$ elements (linear time).

1/25/2007

S. Raskhodnikova and A. Smith. Based on notes by E. Demaine and C. Leiserson
Analyzing Merge Sort

\[
\begin{align*}
T(n) & \quad \textbf{MERGE-SORT} A[1 \ldots n] \\
\Theta(1) & \quad 1. \text{ If } n = 1, \text{ done.} \\
2T(n/2) & \quad 2. \text{ Recursively sort } A[1 \ldots \lfloor n/2 \rfloor] \\
\Theta(n) & \quad \text{ and } A[\lceil n/2 \rceil + 1 \ldots n]. \\
\end{align*}
\]

\textbf{Abuse}: Should be \( T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \), but it turns out not to matter asymptotically.

\textbf{Sloppiness}: Should be \( T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \), but it turns out not to matter asymptotically.
Recurrence for Merge Sort

\[ T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1; \\
2T(n/2) + \Theta(n) & \text{if } n > 1. 
\end{cases} \]

- We usually omit stating the base case because our algorithms always run in time \( \Theta(1) \) when \( n \) is a small constant.
- CLRS and next lectures provide several ways to find a good upper bound on \( T(n) \).
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{align*}
T(n) &= 2T(n/2) + cn \\
&= 2(2T(n/4) + cn/2) + cn \\
&= 4T(n/4) + 2cn/2 + cn \\
&= 4(2T(n/8) + cn/4) + 2cn/2 + cn \\
&= 8T(n/8) + 4cn/4 + 2cn/2 + cn \\
&\vdots \\
&= 2^hT(n/2^h) + c(2^h - 1)n \\
&\leq 2^hT(1) + c(2^h - 1)n \\
&= \Theta(1) + c(2^h - 1)n \\
&= \Theta(1) + c(2^{\lg n} - 1)n \\
&= \Theta(1) + c(n - 1)n \\
&= \Theta(n^2) \\
&\leq \Theta(n^2) \\
\end{align*}
\]

Total = \( \Theta(n \lg n) \)
Merge Sort vs Insertion Sort

• $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.

• Therefore, Merge Sort asymptotically beats Insertion Sort in the worst case.
  – In practice, Merge Sort beats Insertion Sort for $n > 30$ or so…
  – But how does Merge Sort do on nearly sorted inputs?

• Go test it out for yourself!