IT501: Algorithms

Lectures 1 and 2: Introduction, Asymptotic notations
Some slides courtesy from Jeff Edmonds
The course

• Instructor: Dr. Anish Mathuria
  – Office: Faculty Block 1, 1105

• Textbook: *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein (CLRS)
  – An excellent reference you should own
Course Topics

• Algorithm analysis
  – Asymptotic notation, solving recurrence relation

• Classical problems
  – Sorting, knapsack problem, scheduling, graph-related problems

• Meta-algorithms (classes of algorithms)
  – Divide-and-conquer, dynamic programming, greedy approaches

• NP-completeness
Course Format

• 3 lectures / week
  – Mon – 11 am
  – Tue – 10 am
  – Thu – 9 am

• Two tests + final exam

• Tests/Exam cannot be made up, and must be taken in class at the scheduled time.
Grading policy

• Test 1 – 30%
  – slot in 2\textsuperscript{nd} in-sem week
• Final – 50%
• Test 2 – 20%
  – 10\textsuperscript{th} Nov
• Cheating on an exam/test will result in failing the course.
Example: sorting

• Input: A sequence of N numbers \( a_1 \ldots a_n \)
• Output: the permutation (reordering) of the input sequence such that \( a_1 \leq a_2 \ldots \leq a_n \).
• Many possible algorithms
  – Insertion, selection, bubble, quick, merge, …
Brute force algorithm

• To sort \( n \) numbers, we can enumerate all permutations of these numbers and test which permutation has the correct order.

• Why cannot we do this?
  – Too slow!
  – By what standard?
Comparison of functions

<table>
<thead>
<tr>
<th></th>
<th>log₂n</th>
<th>n</th>
<th>nlog₂n</th>
<th>n²</th>
<th>n³</th>
<th>2ⁿ</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>10</td>
<td>33</td>
<td>10²</td>
<td>10³</td>
<td>10³</td>
<td>10⁶</td>
</tr>
<tr>
<td>10²</td>
<td>6.6</td>
<td>10²</td>
<td>660</td>
<td>10⁴</td>
<td>10⁶</td>
<td>10³⁰</td>
<td>10¹⁵⁸</td>
</tr>
<tr>
<td>10³</td>
<td>10</td>
<td>10³</td>
<td>10⁴</td>
<td>10⁶</td>
<td>10⁹</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10⁴</td>
<td>13</td>
<td>10⁴</td>
<td>10⁵</td>
<td>10⁸</td>
<td>10¹²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10⁵</td>
<td>17</td>
<td>10⁵</td>
<td>10⁶</td>
<td>10¹⁰</td>
<td>10¹⁵</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10⁶</td>
<td>20</td>
<td>10⁶</td>
<td>10⁷</td>
<td>10¹²</td>
<td>10¹⁸</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For a super computer that does 1 trillion operations per second, it will be longer than 1 billion years.
Insertion Sort

InsertionSort(A, n) {
    for j = 2 to n {
        ▷ Pre condition: A[1..j-1] is sorted
        ▷ Post condition: A[1..j] is sorted
    }
}

1

sorted

j
Insertion Sort

InsertionSort(A, n) {
    for j = 2 to n {
        key = A[j];
        i = j - 1;
        while (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i - 1;
        }
        A[i+1] = key
    }
}
Example of insertion sort

Done!
Q

Insertion sort is applied to the array $A = \{31, 41, 59, 26, 41, 58\}$. What is the content of this array after 3 iterations of the outermost loop of insertion sort?

A. $\{26, 31, 41, 41, 58, 59\}$
B. $\{26, 41, 59, 31, 41, 58\}$
C. $\{26, 31, 41, 59, 41, 58\}$
D. $\{26, 31, 41, 41, 59, 58\}$
Running time of insertion sort

• The running time depends on the input: an already sorted sequence is easier to sort.
• Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
• Generally, we seek upper bounds on the running time, because everybody likes a guarantee.
Kinds of analyses

• Worst case
  – Provides an upper bound on running time
  – An absolute guarantee

• Best case – not very useful

• Average case
  – Provides the expected running time
  – Very useful, but treat with care: what is “average”?  
    • Random (equally likely) inputs
    • Real-life inputs
Analysis of insertion Sort

InsertionSort(A, n) {
    for j = 2 to n {
        key = A[j]
        i = j - 1;
        while (i > 0) and (A[i] > key) {
            A[i+1] = A[i]
            i = i - 1
        }
        A[i+1] = key
    }
}

How many times will this line execute?
Analysis of insertion Sort

InsertionSort(A, n) {
    for j = 2 to n {
        key = A[j]
        i = j - 1;
        while (i > 0) and (A[i] > key) {
            A[i+1] = A[i]
            i = i - 1
        }
        A[i+1] = key
    }
}

How many times will this line execute?
Analysis of insertion Sort

<table>
<thead>
<tr>
<th>Statement</th>
<th>cost</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertionSort(A, n) {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for j = 2 to n {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>key = A[j]</td>
<td>$c_1$</td>
<td>$n$</td>
</tr>
<tr>
<td>i = j - 1;</td>
<td>$c_2$</td>
<td>$(n-1)$</td>
</tr>
<tr>
<td>while (i &gt; 0) and (A[i] &gt; key) {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = i - 1</td>
<td>$c_5$</td>
<td>$(S-(n-1))$</td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A[i+1] = key</td>
<td>$c_6$</td>
<td>$(S-(n-1))$</td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$S = t_2 + t_3 + \ldots + t_n$ where $t_j$ is number of while expression evaluations for the $j^{th}$ for loop iteration.
Analyzing Insertion Sort

- $T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4S + c_5(S - (n-1)) + c_6(S - (n-1)) + c_7(n-1)$
  
  $= c_8S + c_9n + c_{10}$

- What can $S$ be?
  - Best case -- inner loop body never executed
    - $t_j = 1 \implies S = n - 1$
    - $T(n) = an + b$ is a linear function

The diagram shows a segment of a sorted list, indicating the insertion of a key at position $i$.
Analyzing Insertion Sort

- \( T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4S + c_5(S - (n-1)) + c_6(S - (n-1)) + c_7(n-1) \)
  \[ = c_8S + c_9n + c_{10} \]
- What can \( S \) be?
  - Worst case -- inner loop body executed for all previous elements
    - \( t_j = j \quad \Rightarrow \quad S = 1 + 2 + 3 + \ldots + n - 1 \)
    - \( T(n) = an^2 + bn + c \) is a quadratic function
Asymptotic Analysis

• $T(n)$: the time taken on input with size $n$

• Look at **growth** of $T(n)$ as $n \rightarrow \infty$.
  – Give more importance to highest order terms

• Classifying functions
  – E.g. the functions $10n^3 + 5n^2 + 17$ and $2n^3 + 3n + 79$ should go into the same class
  – Ignore constant multipliers
Asymptotic notations

• Formal notations to speak about functions and classify them

• \( \Theta \): Theta
• \( O \): Big-Oh
• \( \Omega \): Omega
Class \( \Theta \)

- \( f, g \): non-negative functions of non-negative arguments
- **Definition:**
  - \( \Theta(g(n)) = \{ f(n): \exists \) positive constants \( c_1, c_2, \) and \( n_0 \) such that \( c_1 g(n) \leq f(n) \leq c_2 g(n), \ \forall \ n \geq n_0 \} \)
  - The condition \( n \geq n_0 \) captures the requirement that we do not care about small values of \( n \)
Class Θ

• Claim:
  – $f_1(n) = 10n^3 + 5n^2 + 17 \in \Theta(n^3)$

• Proof by definition:
  – Need to find three constants $c_1$, $c_2$ and $n_0$
    such that $c_1n^3 \leq f_1(n) \leq c_2n^3$ for all $n \geq n_0$
  – A simple solution is $c_1 = 10$, $c_2 = 32$ and $n_0 = 1$

• Prove $2n^3 + 3n + 79 \in \Theta(n^3)$
More examples

• $f_3(n) = 10n^3 + n \log n \in \Theta(n^3)$
  – not a cubic polynomial

• $f_4(n) = 5n \log n + 10n \in \Theta(n \log n)$
On Writing Style

• Proper style: $f \in \Theta(g)$
• More common style: $f = \Theta(g)$
  • Abuse of = sign
• Meaningless: $\Theta(g) = f$
Another Example

• Prove $f_5(n) = 2 + 1/n \in \Theta(1)$

• What is $\Theta(1)$?
More classes

• \( \Theta(g(n)) = \{ f(n): \exists \text{ positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall \ n \geq n_0 \} \)

• \( O(g(n)) = \{ f(n): \exists \text{ positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq cg(n) \ \forall \ n \geq n_0 \} \)

• \( \Omega(g(n)) = \{ f(n): \exists \text{ positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \ \forall \ n \geq n_0 \} \)
Examples of O-notation

• $3n^2 \in O(n^2)$
• $10n^3 + 5n + 7 \in O(n^3)$
• $10n^3 + 5n + 7 \in O(n^4)$
Using $O$ and $\Omega$

- Suppose we know
  \[ T(n) \leq 15n^3 + 7n^2 + 35 \leq 57n^3 \]
  - So, $T(n) \in O(n^3)$
- Suppose we also know
  \[ T(n) \geq 2n^3 + 37 \geq 2n^3 \]
  - So, $T(n) \in \Omega(n^3)$
Fact

• $\Theta(g) = O(g) \cap \Omega(g)$
Short-cut

• $S(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
• Prove $S(n) \in \Theta(n^2)$, without finding $S(n)$
Short-cut

• HW:
  – \( T(n) = \sum_{i=1}^{n} i^2 \)
  – Prove \( T(n) \in \Theta(n^3) \), without finding \( T(n) \)
Asymptotic notations

- \( \Theta: = \)
- \( O: \leq \)
- \( \Omega: \geq \)