CS141: Intermediate Data Structures and Algorithms

Ahmed Eldawy
Welcome back to UCR!
Class information

- Classes: Monday, Wednesday, and Friday
  4:10 PM – 5:00 PM
- Instructor: Ahmed Eldawy
- Office hours:
  Monday and Wednesday 3:00 PM – 4:00 PM
  @357 WCH. Conflicts?
- Email: eldawy@ucr.edu
- Subject line: “[CS141] …”
- Website
  http://www.cs.ucr.edu/~eldawy/19WCS141
Textbook

- Introduction to Algorithms. Third Edition
- Cormen, Leiserson, Rivest, and Stein
Course goals

- What are your goals?
- Analysis of algorithms
- Design of algorithms
- How to compare and choose different algorithms and data structures
- How to improve existing algorithms
Covered topics

- Analysis of algorithms
- Big-O notation
- Divide and conquer algorithms
- Greedy algorithms
- Dynamic programming
- Graph algorithms
- Computational geometry
Course work

- Class participation (5%)
- Five assignments (20%)
  - Prepared on Latex or any other word processor
  - Late policy: 20% per calendar day (up to four days)
- Two quizzes (15% + 20% = 35%)
- Final exam (40%)

Final Exam
Thursday, March 21st
3:00 PM – 6:00 PM
Background

- Basic data structures (Lists, stacks, and queues)
- Sorting algorithms
- Binary search tree
- Heap data structure (Priority queue)
- Hashtables
- Graphs
- Test your background
Analysis of Algorithms
Criteria of Analysis

- Which criteria should be taken into account?
  - Running time
  - Memory footprint
  - Disk IO
  - Network bandwidth
  - Power consumption
  - Lines of code
  - …
Average Case Vs Worst Case

Running Time

1/7/2019
Case Study: Insertion Sort

**Insertion-Sort** $(A, n)$

```plaintext
for $j = 2$ to $n$
  $key = A[j]$
  $i = j - 1$
  while $i > 0$ and $A[i] > key$
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = key$
```

<table>
<thead>
<tr>
<th>cost</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$n$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$\sum_{j=2}^{n} t_j$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>
Insertion Sort

(a) 1 2 3 4 5 6
    5 2 4 6 1 3

(b) 1 2 3 4 5 6
    2 5 4 6 1 3

(c) 1 2 3 4 5 6
    2 4 5 6 1 3

(d) 1 2 3 4 5 6
    2 4 5 6 1 3

(e) 1 2 3 4 5 6
    1 2 4 5 6 3

(f) 1 2 3 4 5 6
    1 2 3 4 5 6
Growth of Functions

- It is hard to compute the actual running time.
- The cost of the worst-case is a good measure.
- The *growth* of the function is what interests us (Big Data).
- We are more concerned with comparing two functions, i.e., two algorithms.
Growth of Functions
$O$-notation

\[ f(n) = \Theta(g(n)) \quad \exists c > 0, n_0 > 0 \quad 0 \leq f(n) \leq c g(n) \quad n \geq n_0 \]

$g(n)$ is an asymptotic upper bound for $f(n)$.
$\exists c > 0, n_0 > 0$
$0 \leq cg(n) \leq f(n)$
$n \geq n_0$

$g(n)$ is an asymptotic lower-bound for $f(n)$
\[ \exists c_1, c_2 > 0, n_0 > 0 \]
\[ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \]
\[ n \geq n_0 \]

\text{g(n) is an asymptotic tight-bound for f(n)}
\[ f(n) = o(g(n)) \]

\[ \forall c > 0 \]
\[ \exists n_0 > 0 \]
\[ 0 \leq f(n) \leq cg(n) \]
\[ n \geq n_0 \]

\textit{g(n) is a non-tight asymptotic upper-bound for f(n)}
g(n) is a non-tight asymptotic lower-bound for f(n)
Compare two functions

- \( \lim_{n \to \infty} \frac{f(n)}{g(n)} \)
- \( 0: \) \( f(n) = o(g(n)) \)
- \( c > 0: \) \( f(n) = \Theta(g(n)) \)
- \( \infty: \) \( f(n) = \omega(g(n)) \)
### Analogy to real numbers

<table>
<thead>
<tr>
<th>Functions</th>
<th>Real numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n) = O(g(n))$</td>
<td>$a \leq b$</td>
</tr>
<tr>
<td>$f(n) = \Omega(g(n))$</td>
<td>$a \geq b$</td>
</tr>
<tr>
<td>$f(n) = \Theta(g(n))$</td>
<td>$a = b$</td>
</tr>
<tr>
<td>$f(n) = o(g(n))$</td>
<td>$a &lt; b$</td>
</tr>
<tr>
<td>$f(n) = \omega(g(n))$</td>
<td>$a &gt; b$</td>
</tr>
</tbody>
</table>
Simple Rules

- We can omit constants
- We can omit lower order terms
- $\Theta(an^2 + bn + c)$ becomes $\Theta(n^2)$
- $\Theta(c_1)$ and $\Theta(c_2)$ become $\Theta(1)$
- $\Theta(\log_{k_1} n)$ and $\Theta(\log_{k_2} n)$ become $\Theta(\lg n)$
- $\Theta(\lg(n^k))$ becomes $\Theta(\lg n)$
- $\lg^{k_1}(n) = o\left(n^{k_2}\right)$ for any positive constants $k_1$ and $k_2$
Popular Classes of Functions

- Constant: \( f(n) = \Theta(1) \)
- Logarithmic: \( f(n) = \Theta(\lg(n)) \)
- Sublinear: \( f(n) = o(n) \)
- Linear: \( f(n) = \Theta(n) \)
- Super-linear: \( f(n) = \omega(n) \)
- Quadratic: \( f(n) = \Theta(n^2) \)
- Polynomial: \( f(n) = \Theta(n^k) \); \( k \) is a constant
- Exponential: \( f(n) = \Theta(k^n) \); \( k \) is a constant
**Insertion Sort (Revisit)**

**INSERTION-SORT**\((A, n)\)

\[
\textbf{for } j = 2 \textbf{ to } n \\
\quad \text{key} = A[j] \\
\quad \text{// Insert } A[j] \text{ into the sorted sequence } A[1 \ldots j - 1] \\
\quad i = j - 1 \\
\quad \textbf{while } i > 0 \text{ and } A[i] > \text{key} \\
\quad \quad A[i + 1] = A[i] \\
\quad \quad i = i - 1 \\
\quad A[i + 1] = \text{key}
\]

\[\Theta(n^2)\]
Analysis of Recursive Algorithms
Divide-and-Conquer

Big Dataset

Divide

Smaller Dataset

Conquer

Partial Answer

Smaller Dataset

Conquer

Partial Answer

Combine

Final Answer
Merge Sort

**Merge-Sort**(A, p, r)

if \( p < r \)

\[ q = \lfloor (p + r)/2 \rfloor \]

\[ \text{merge-sort}(A, p, q) \]

\[ \text{merge-sort}(A, q + 1, r) \]

\[ \text{merge}(A, p, q, r) \]

// check for base case

// divide

// conquer

// conquer

// combine

\[ T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1, \\
2T(n/2) + \Theta(n) & \text{otherwise} 
\end{cases} \]
Recursion Tree