Books


Grading

<table>
<thead>
<tr>
<th>Description</th>
<th>Percentage of Final Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class participation &amp; quizzes</td>
<td>15%</td>
</tr>
<tr>
<td>4-5 Programming Projects</td>
<td>30%</td>
</tr>
<tr>
<td>Homework</td>
<td>10%</td>
</tr>
<tr>
<td>Midterm</td>
<td>15%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>30%</td>
</tr>
<tr>
<td>TOTAL</td>
<td>100%</td>
</tr>
</tbody>
</table>

Grading Scale

<table>
<thead>
<tr>
<th>Grade</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>93-100%</td>
</tr>
<tr>
<td>A-</td>
<td>90-92%</td>
</tr>
<tr>
<td>B+</td>
<td>86-89%</td>
</tr>
<tr>
<td>B</td>
<td>83-85%</td>
</tr>
<tr>
<td>B-</td>
<td>80-82%</td>
</tr>
<tr>
<td>C+</td>
<td>76-79%</td>
</tr>
<tr>
<td>C</td>
<td>70-72%</td>
</tr>
<tr>
<td>F</td>
<td>BELOW 60%</td>
</tr>
</tbody>
</table>

Program Submissions

- All programs will be submitted using Springboard. A drop box will be created for each program. You will also need to submit a printed copy of your source code and a readme file. DO NOT submit programs that are not reasonably correct. To be considered reasonably correct, a program must be completely documented and work correctly for sample data provided with the assignment. Programs failing to meet these minimum standards will be returned ungraded and a 30% penalty assessed. Late points, 10% per day, will be added on top of this penalty.

- Ethics: All programming assignments must be your own work. Duplicate or similar programs/reports, plagiarism, cheating, undue collaboration, or other forms of academic dishonesty will be reported to the Student Disciplinary Office as a violation of the Student Honor Code.

Algorithm

A procedure for solving a computational problem (ex: sorting a set of integers) in a finite number of steps.

More specifically: a step-by-step procedure for solving a problem or accomplishing something (especially using a computer).

Solving a Computational Problem

- Problem definition & specification
  - specify input, output and constraints
- Algorithm analysis & design
  - devise a correct & efficient algorithm
- Implementation planning
- Coding, testing and verification

Input → Algorithm → Output

Examples

- RSA
- Cryptography
- Quicksort
- Databases
- FFT
- Signal processing
- Huffman codes
- Data compression
- Network flow
- Routing Internet packets
- Linear programming
- Planning, decision-making
What is CS435/535?
- Learn well-known algorithms and the design and analysis of algorithms.
- Examine interesting problems.
- Devise algorithms for solving them.
- Data structures and core algorithms.
- Analyze running time of programs.
- Critical thinking.

Ch 0: Big-O notation

Asymptotic Complexity
- Running time of an algorithm as a function of input size \( n \) for large \( n \).
- Expressed using only the highest-order term in the expression for the exact running time.
- Instead of exact running time, say \( \Theta(n^2) \).
- Describes behavior of function in the limit.
- Written using Asymptotic Notation.

Asymptotic Notation
- \( \Theta, O, \Omega, o, \omega \)
- Defined for functions over the natural numbers.
- \( f(n) = \Theta(n^2) \): Describes how \( f(n) \) grows in comparison to \( n^2 \).
- Define a set of functions; in practice used to compare two function sizes.
- The notations (\( \Theta, O, \Omega, o, \omega \)) describe different rate-of-growth relations between the defining function and the defined set of functions.

\( O \)-notation

For function \( g(n) \), we define \( O(g(n)) \), big-O of \( n \), as the set:

\[
O(g(n)) = \{ f(n) : \exists \text{ positive constants } c \text{ and } n_0 \text{ such that } \forall n \geq n_0 \text{, we have } 0 \leq f(n) \leq cg(n) \}
\]

Intuitively: Set of all functions whose rate of growth is the same as or lower than that of \( g(n) \).

\( g(n) \) is an asymptotic upper bound for \( f(n) \).

Project 1 and final project

Chapter 0: Prologue
Chapter 1: Algorithms with numbers
Chapter 2: Divide-and-conquer algorithms
Chapter 3: Decompositions of graphs
Chapter 4: Paths in graphs
Chapter 5: Greedy algorithms
Chapter 6: Dynamic programming
Chapter 7: Linear programming
Chapter 8: NP-complete problems
Chapter 9: Coping with NP-completeness
Examples

\( O(g(n)) = \{ f(n) : \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq f(n) \leq cg(n) \} \)

- \( O(n^2) \)
- \( n^2 + 1 \)
- \( n^2 + n \)
- \( 1000n^2 + 1000n \)
- \( n^{1.999} \)
- \( n^2 / \log n \)
- \( n^2 / \log \log \log n \)

Intuitively: Set of all functions whose rate of growth is the same as or higher than that of \( g(n) \).

\( \Omega(g(n)) = \{ f(n) : \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq cg(n) \leq f(n) \} \)

\( \Theta(g(n)) = \{ f(n) : \exists \text{ positive constants } c_1, c_2, \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \} \)

Relations Between \( O, \Omega, \Theta \)

\( \Omega - notation \)

For function \( g(n) \), we define \( \Omega(g(n)) \), big-Omega of \( n \), as the set:

\( \Omega(g(n)) = \{ f(n) : \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq cg(n) \leq f(n) \} \)

\( \Theta - notation \)

For function \( g(n) \), we define \( \Theta(g(n)) \), big-Theta of \( n \), as the set:

\( \Theta(g(n)) = \{ f(n) : \exists \text{ positive constants } c_1, c_2, \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \} \)

To compare orders of growth, look at the leading term.

Exercise: Prove that \( n^2 / 2 - 3n = \Theta(n^2) \)
Exercise

\( n-10^6 = \Omega(n) \)

Example

- Is \( 3n^3 = \Theta(n^4) \)?
- How about \( 2^{2n} = \Theta(2^n) \)?
- How about \( \log_2 n = \Theta(\log_{10} n) \)?

Relations Between \( \Theta, \Omega, O \)

**Theorem:** For any two functions \( g(n) \) and \( f(n) \),

\( f(n) = \Theta(g(n)) \) iff

\( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \).

- i.e., \( \Theta(g(n)) = O(g(n)) \cap \Omega(g(n)) \)

\( f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)). \quad f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n)). \)

\( \Theta(g(n)) \subset O(g(n)). \quad \Theta(g(n)) \subset \Omega(g(n)). \)

- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Comparison of Functions

\( f \leftrightarrow g \sim a \leftrightarrow b \)

- \( f(n) = O(g(n)) \sim a \leq b \)
- \( f(n) = \Omega(g(n)) \sim a \geq b \)
- \( f(n) = \Theta(g(n)) \sim a = b \)

Limits

- \( \lim_{x \to \infty} \left[ \frac{f(x)}{g(x)} \right] < \infty \Leftrightarrow f(x) \in O(g(x)) \)
- \( 0 < \lim_{x \to \infty} \left[ \frac{f(x)}{g(x)} \right] < \infty \Leftrightarrow f(x) \in \Theta(g(x)) \)
- \( 0 < \lim_{x \to \infty} \left[ \frac{f(x)}{g(x)} \right] \Leftrightarrow f(x) \in \Omega(g(x)) \)
- \( \lim_{x \to \infty} \left[ \frac{f(x)}{g(x)} \right] \) undefined \( \Rightarrow \) can’t say, need special attention

Complexity Classes

<table>
<thead>
<tr>
<th>Adjective</th>
<th>( \Theta )-Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>( \Theta(\log n) )</td>
</tr>
<tr>
<td>Linear</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>nlog n</td>
<td>( \Theta(n \log n) )</td>
</tr>
<tr>
<td>Quadratic</td>
<td>( \Theta(n^2) )</td>
</tr>
<tr>
<td>Cubic</td>
<td>( \Theta(n^3) )</td>
</tr>
<tr>
<td>Exponential</td>
<td>( \Theta(2^n) )</td>
</tr>
<tr>
<td>Exponential</td>
<td>( \Theta(n^p) )</td>
</tr>
</tbody>
</table>
eq \Theta(10^n)
Fibonacci Numbers

```python
function fib1(n)
    if n=0: return 0
    if n=1: return 1
    return fib1(n-1) + fib1(n-2)

function fib2(n)
    if n=0: return 0
    create an array f[0...n]
    f[0] = 0, f[1] = 1
    for i=2...n:
        f[i] = f[i-1] + f[i-2]
    return f[n]
```

Which one is better?