Course Essentials

- Algorithms – Design & Practice
  - How to design
  - Learn some good ones
  - How to implement – practical considerations

- How to write about algorithms
- How to present algorithms
- How to discuss algorithms
Five Key Elements of Problem Solving

• What to compute – problem-spec

• How to compute – known solutions

• How to implement – what language and implementation

• Is it correct? – correctness, and testing

• How to measure performance? – efficiency, empirical measurements

An Algorithm for Algorithm Development

def algorithm-development (problem-spec):

    correct = false
    while not correct or not fastEnough(runningTime)
        algorithm = deviseAlgorithm(problem-spec)
        correct = analyzeCorrectness(algorithm)
        runningTime = analyzeEfficiency(algorithm)

    return algorithm
## Topics & Algorithms

<table>
<thead>
<tr>
<th>Correctness</th>
<th>Complexity</th>
<th>C/C++</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>Recursion</td>
<td>Fisher-Yates</td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>Insertion Sort</td>
<td>Merge Sort</td>
<td>Heap Sort</td>
<td>Quick Sort</td>
</tr>
<tr>
<td>Hybrid Quicksort</td>
<td>Dual Pivot QS</td>
<td>Anagrams</td>
<td>Hash Tables</td>
<td>Hash Functions</td>
</tr>
<tr>
<td>Graphs</td>
<td>Directed Acyclic Graphs</td>
<td>Breadth-First Search</td>
<td>Depth-First Search</td>
<td>The W-O-M-A-N Puzzle</td>
</tr>
<tr>
<td>Critical Path</td>
<td>Topological Sort</td>
<td>Shortest Paths</td>
<td>Dijkstra's</td>
<td>Bellman-Ford Floyd-Warshall</td>
</tr>
<tr>
<td>Soundex</td>
<td>Metaphone</td>
<td>Regular Expressions</td>
<td>Python Library</td>
<td>Java Library</td>
</tr>
<tr>
<td>K-Nearest Neighbors</td>
<td>Decision Trees</td>
<td>Neural Networks</td>
<td>Data Compression</td>
<td>Huffman Encoding</td>
</tr>
<tr>
<td>LZ Compression</td>
<td>LZW Compression</td>
<td>Databases</td>
<td>Digital Signatures</td>
<td>Cryptographic Hash Functions</td>
</tr>
<tr>
<td>Fibonacci Numbers</td>
<td>Higher-order functions</td>
<td>Advanced Python (Iterators, generators, decorators)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional Topics

- Decision Trees
- Neural Networks
- Data Compression
- Digital Signatures
- Huffman Encoding
- Cryptographic Hash Functions
- Soundex
- Metaphone
- Regular Expressions
- Python Library
- Java Library
- Bellman-Ford Floyd-Warshall
- The W-O-M-A-N Puzzle
- Advanced Python (Iterators, generators, decorators)
- Linux Shell & Resource Allocation
- Enhanced Hashing
- Advanced Data Structures
- Asynchronous Programming
- Distributed Systems
- Cloud Computing
- Cybersecurity
- Artificial Intelligence
- Machine Learning
- Deep Learning
- Natural Language Processing
- Blockchain Technology
What we didn’t do?

What we didn’t do? Strings!!

• Longest Common Subsequence

• Matching – Knuth-Morris Pratt, Boyer Moore, etc.

• String Transformation (Dynamic Programming)
What you **liked**, **liked to program**, and **hated**

![Bar chart showing the distribution of likes, preferred programming, and dislikes.](chart1)

What **THEY** **Liked**, **Liked to Program**, and **Hated**. (2017)

![Bar chart showing the distribution of likes, preferred programming, and dislikes.](chart2)
What you liked, liked to program, and hated

PROGRAMMING LANGUAGES

Java 33%
Python 64%
C 9%
C++ 0%
Core Foundations – Primitive Types

- Machine representations of int, char, float, double, etc.
- What operations are available on them

<table>
<thead>
<tr>
<th>Type</th>
<th>Contains</th>
<th>Default</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>false</td>
<td>1 bit</td>
<td>NA</td>
</tr>
<tr>
<td>char</td>
<td>Unicode character</td>
<td>\u0000</td>
<td>16 bits</td>
<td>\u0000 to \uFFFF</td>
</tr>
<tr>
<td>byte</td>
<td>Signed integer</td>
<td>0</td>
<td>8 bits</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>Signed integer</td>
<td>0</td>
<td>16 bits</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>Signed integer</td>
<td>0</td>
<td>32 bits</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>Signed integer</td>
<td>0</td>
<td>64 bits</td>
<td>-9223372036854775808 to 9223372036854775808</td>
</tr>
<tr>
<td>float</td>
<td>IEEE 754 floating point</td>
<td>0.0</td>
<td>32 bits</td>
<td>\pm 1.0e-38 to \pm 3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>IEEE 754 floating point</td>
<td>0.0</td>
<td>64 bits</td>
<td>\pm 1.7976931348623157E+38</td>
</tr>
</tbody>
</table>
Core Foundations – Arrays

- Machine representations: 1-D, 2-D, etc.
- What operations are available on them
- Know how to iterate, resize, partition, merge, insert, delete, shuffle, initialize
- Library facilities available

Core Foundations – Strings

- Machine representations
- What operations are available on them
- How to create, compare, copy, match, join, split, etc.
- Algorithms for string matching, transformation, etc.
Core Data Structures - Lists

• Arrays versus linked representations (single-, doubly-linked)

• How to define, create, iterate, insert, delete, search, etc.

• How dynamic allocation works?

Core Data Structures – Stacks & Queues

• Arrays versus linked representations (single-, doubly-linked)

• How to define, create, iterate, insert, delete, search, etc

• How dynamic allocation works?

• Last-in First out, First-in First out. Dequeues

• Provided in library?
Core Data Structures – Heaps

• Machine representations (arrays)

• What operations are available on them

• How to create, insert, delete.

• $O(1)$ lookup (max, min), $O(\log n)$ insert/delete

• Priority Queues

Core Data Structures – Hash Tables

• Hash Functions, collisions, collision resolution (linear, quadratic probing, lists, etc.)

• Library?

• $O(1)$ insertion/delete/lookup (mostly!)

• Load factor affects performance
Core Data Structures – Trees

• Binary, Binary Search, Quad, etc.

• Properties: full, complete, balanced, etc.

• How algorithms are impacted (e.g. $O(\log n)$ insert, delete, search in BSTs)

Core Data Structures – Graphs

• Kinds of graphs (undirected, directed, weighted)

• Graph representations (adjacency matrix, array of linked lists, etc.)

• Graph algorithms (Topological sort, Dijkstra’s, Bellman Ford, Floyd Warshall, Minimum Spanning Tree, etc.)
Tips for Technical Interviews

• **Listen carefully**
  - Given two arrays that are sorted...
  - Design an algorithm to be run repeatedly...

• **Draw an example**
  - should be specific
  - sufficiently large
  - not a special case

• **Offer a brute force solution**
  - don’t worry about it being terrible
  - get its complexity and make improvements

Tips for Technical Interviews

• **Optimize**
  - look for any unused information in problem statement
  - use a fresh example
  - make time vs space tradeoff

• **Solution walkthrough**
  - identify special cases

• **Code**
  - modularize
  - error checks
  - use good programming practices
  - plan tests
Tips for Technical Interviews

• **Use good coding practices**
  - correct
  - efficient
  - simple
  - readable
  - maintainable
  - make generous use of data structures
  - write modular code
  - make it flexible and robust
  - don’t forget error checking (start w/ todo-s)

• **What if you give an incorrect answer?**

...and

• **Lots of other stuff!**
Finally...

RTFM, please.
PLEASE,
Don’t be this person!!!
Thank you.