Data Structures and Algorithms (CS121 and CS160)

* Foundation in computer science, diverse applications, broad impact
  * in problem solving and computational thinking
  * Along with cs 180, 170, 140

* Classical and Fundamental Data Structures and Algorithms
  * Data Structures
    * array, linked lists, stacks, queues, priority queues, trees, hash tables, disjoint sets (union find)
    * concepts, algorithms, implementations and applications
  * Algorithms: big ideas, subtle details, adaptations, applications, general principles
    * Sorting algorithms
    * Searching: binary search, Symbol Table based approaches
  * Graph Algorithms
    * Traversals, and traversal based problems
    * Minimum Spanning Trees, Shortest Paths, Max Flow

* What, how, why (correctness, efficiency under ... conditions), alternatives (pros and cons), adaptations
CS121 and CS160

* Foundational knowledge and skills, diverse applications, broad impact

* Analysis: correctness, complexity (big-Oh notation, tilde notation)

* Design techniques
  * Iteration (can be subtle and powerful, e.g., sorting, binary search, priority queues, DP)
  * Recursion (e.g. trees, dfs on graphs, quick sort)
  * Divide and conquer (e.g. merge sort, quick sort, binary search)
  * Randomized algorithm (e.g. quick sort, quick select)
  * Greedy (be careful about correctness, e.g. MST, Dijkstra’s, coin change)
  * Reduction: reduce a new problem to some known problems, with adaptations when needed (e.g. Baseball elimination $\rightarrow$ max flow)
Computational Thinking and Problem Solving

- Apply knowledge, principles, methods, skills to solve new problems
  - Identify and formulate new problems; or understand given problems
  - Develop a good (correct, efficient) solution
    - Iteratively decompose a big problem into smaller problems
      - Different levels of details: short elevate talk to detailed pseudocode
      - Develop, test, revise ideas/solutions before settling on a good one
  - Implement the solution and produce high-quality programs and docs
    - Systematic, disciplined development
    - Incremental development, thorough testing, professional style
  - Reflect on your work and be better prepared for new problems
    - What were the stumbling blocks? How did you solve them? Any connections to the old problems/methods? New insights, approach...

- Develop and enhance fundamental skills
  - Technical reading, algorithms/code tracing, debugging, testing
  - Critical thinking, deep understanding, disciplined work, and time management...
A Few Points to Wrap up this semester

- Sorting, Hash Tables, Binary Search Trees ...
  - The **system** side: implement the data structures and algorithms (e.g. java system developers at Oracle)
  - The **client** side: use the provided data structures and algorithm in their own code (e.g. entry-level java developers)
    - Our study and textbook focused on the system side. Also be sure to understand the client side and know how to use the system libraries.
    - Example: java system sort and compareTo() method (using 2D point as an example)
A Few Points to Wrap up this semester

Classical, important algorithms and data structures

* know their properties, and
* consider adaptations to achieve better performance.

Examples:

* If your data has some special properties, consider modifying some sorting algorithm to take advantage of the properties

* Prim’s eager MST Algorithm, Dijkstra’s shortest path algorithm
  * the code in the textbook and algs4.jar: based on the adjacency list representation and indexed-priority queue (pretty subtle to understand or implement):
    * Some example review questions: why indexed priority queue? why not just the basic priority queue?
    * the worst case time complexity is $O(E \lg V)$ -- another review question: how to explain this complexity?

* What if your graph is dense, with the number of edges $E$ in the order of $V^2$?
  * The algs4 code will have $O(V^2 \lg V)$ worst-case time complexity.
  * It is possible to achieve better worst case performance $O(V^2)$ with simpler data structure.
    * Check out exercise problem 4.3.19.