Analysis of Algorithms Learning Outcomes
91.404 (Section 201)

(adapted from *ACM Computer Science Computing Curricula 2001, Steelman Draft, August, 2001)*

1) Use big O, omega and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
2) Determine the time complexity of simple algorithms.
3) Express simple algorithms using pseudocode conventions.
4) Justify the correctness of an algorithm
   a) "Mechanical": e.g. no infinite loops or recursion, remaining within array bounds
   b) "As advertised": the algorithm solves the given problem: e.g. sorting algorithm results in correct ordering of elements.
5) Given a computational problem begin to learn how to:
   a) Recognize its underlying structure
   b) Identify aspects of the problem relevant to time & space complexity (e.g. problem size, distribution of inputs)
   c) Develop intuition about it using worst-case, best-case and average-case inputs
   d) Determine if a known solution exists
   e) Adapt a solution to a closely related problem
   f) Design an algorithm from scratch by appropriately selecting an algorithmic paradigm and making of abstract data type and implementation choices.
6) Given a list of functions of a single variable, order them according to asymptotic growth.
7) Perform worst-case, best-case and average case asymptotic analysis.
8) Deduce the recurrence relations that describe the time complexity of recursive algorithms, and solve simple recurrence relations using Master Theorem, recursion trees and induction/ substitution.
9) Design and analyze simple randomized algorithms.
10) Solve problems using sorting algorithms:
   a) Comparison-based sorting algorithms
      i) Insertion Sort
      ii) Merge Sort
      iii) Heap Sort: Execute operations that preserve the heap property
      iv) Quick Sort
   b) Non-comparison and/or hybrid sorting algorithms.
      i) Counting Sort
      ii) Radix Sort
      iii) Bucket Sort
11) Assess the impact of choices of abstract data type and implementation on running time and storage requirements
   a) Stacks
   b) Queues
   c) Linked Lists
   d) Trees: Binary search trees and balanced binary search trees
      i) Apply left and right rotations that preserve the binary search tree property
   e) Hash Tables, including collision avoidance strategies
   f) Graphs, including both adjacency matrix and adjacency list representations
12) Understand the operation of the following graph algorithms
   a) Depth-First Search
   b) Breadth-First Search
   c) Topological Sort
13) Understand how a multi-threaded Merge Sort works (time permitting)