COMP340 - SP22 - Standards

Topic/Skill	Standard
Analysis	
Correctness	Know how to formulate and evaluate a formal problem statement
	Use standard combinatorial objects to model problems and reasoning recursively about those objects
	Be able to generate and use concrete instances of a problem to explore algorithms and evaluate their potential correctness
	Be able to reason about and analyze potential brute-force solutions to an algorithmic problem
	Be able to generate and use a specific counter-example to prove an algorithm is incorrect
	Understand the use of inductive proofs for establishing the correctness of an algorithm
	Be able to generate and reason about loop-invariants and in particular their role in proving algorithm correctness
Big-O	Know the definitions for and differences between Big-O, Omega, and Theta.
	Know how Big-O, Omega, and Theta behave under addition and multiplication and whether it is transitive. Be able to justify these behaviors by using the formal definitions.
	Understand what a dominance relation is and the relations between the eight key functions used in algorithm analysis.
	Be able to relate functions using Big-O, Omega, and Theta
	Be able to use Big-O, Omega, and Theta in the analysis of an algorithm.
	Understand the use of Big-O, Omega, and Theta for classifying and comparing algorithms in terms of standard mathematical functions.
Summations	Know how to interpret a mathematical expression using summation notation.
	Know how to use summation notation to express a sum
	Know the Theta classification for arithmetic, geometric, and harmonic series commonly encountered in algorithm analysis
	Be able to prove the correctness of a closed-form solution to a summation using inductive reasoning and the recursive structure of a sum.
	Know how to express and analyze the running time of standard loops using summation notation.
Recurrence Relations	Know how to interpret a mathematical expression of a recurrence relation.
	Be able to express the running time of an algorithm, especially a recursive or divide and conquer algorithm, using a recurrence relation.
	Be able to solve simple recurrence relations by unrolling the recursion
Data Structures	

Topic/Skill	Standard
Array	Know how to use array's efficiently using techniques such as the shift-less circular array and exponential capacity increases
	Know how to use amortized time analysis in the context of dynamic arrays and exponential capacity increases.
	Be able to carry out basic array traversals and operations using both loops and recursive functions.
Linked-Structures	Know how to use linked structures including those with multiple pointers such as doubly-linked lists and tree nodes
	Be able to carry out basic linked-structure operations using both recursive functions and iterative loops.
(Binary) Trees	Know the vocabulary for talking about binary trees
	Be able to do a pre,post, and in-order traversal of a binary tree
	Identify and analyze the properties of Complete, Full, and balanced binary trees.
	Explain the importance of balanced tree structures and tree structures generally to the design and analysis of O(f(n)*log n) algorithms.
General	Be able to compare and contrast the strengths and weaknesses of array-based implementations vs linked-structure implementations in general and when considering specific ADTs and their needs.
Abstract Data Types	
Stack & Queue	Identify and know how to use the standard interface for each ADT
	Know the expected running time of the standard interface functions
	Understand ways in which arrays and linked-structures can be used to achieve expected running times for stack and queue operations
Binary-Search Trees	Know what is a BST. Explain how it differs from a standard binary tree.
	Understand and explain the importance of balanced BSTs
Dictionaries	Know the standard interface for dictionary structures and how to use them effectively.
	Compare and contrast dictionary implementations based on arrays, sorted arrays, linked-lists, sorted linked-lists, BSTs, and hashing.
	Be able to implement array, list, and BST based dictionaries.
Heap	Know what a min/max heap is, how to use it, and how it differs from standard binary trees and BSTs.
	Be able to implement an efficient, complete-tree based heap using using dynamic arrays. Analyze the performance of that implementation.
	Know how to convert an *n* element array into a heap in *O(n)* time.
Priority Queue	Know the standard interface for priority queues and how to use it.

Topic/Skill	Standard
	Know how to implement a Priority Queue efficiently using a heap
Graphs	Know what a Graph is, the vocabulary used to talk about graphs, and how they compare and relate to Trees.
	Explain the differences between the different flavors of graphs.
	Know what a Directed-Acyclic Graph (DAG) is and how it relates to a standard Graph
	Be able to represent a graph with an adjacency list or an adjacency matrix
	Be able to draw a graph from its adjacency list or matrix
	Be able to program with and reason about graphs in their adjacency list/matrix representation.
	Be able do and program a Depth-First and Breadth-First traversal of a graph identifying when a vertex is discovered and processed and when edges are first encountered.
	Interpret, explain, and generate a search tree from a DFS or BFS
	Identify tree, back, cross, an forward edges in the context of a BFS or DFS search tree
Algorithms	
Sorting	Be able to explain the design principles for Insertion, Selection, Merge, Heap, and Quicksort.
	Be able to implement textbook versions of the sorts listed above.
	Be able to analyze the runtime of the sorts listed above
	Know, be able to implement, and be able to analyze the partition and merge subroutines for quick and merge sort.
	Be able to address and analyze the runtime impact of randomness on the partition algorithm.
Search	Know how to carry out, program, and analyze, a linear search
	Know how to carry out, program (loops and recursion), and analyze a binary search
	Know how to carry out, program, and analyze an efficient one-sided binary search
Divide and Conquer	Know how to use the Master Theorem to analyze divide and conquer algorithms. Be able to recognize when something is not divide and conquer and therefore not subject to the master theorem.
	Explain and prove the key cases of the master theorem in terms of the structure of the computational trees generated by divide and conquer algorithms.
Graphs	Be able to use DFS in order to find paths and cycles and to perform a topological sort
	Be able to use BFS to find paths and detect connected components.
	Compare and Contrast BFS and DFS and identify when one might be more appropriate than the other
Design	
	Be able to analyze a problem and identify appropriate combinatorial structures and ADTs for the problem
	Know when and how to utilize sort and search algorithms in service of the design of an algorithm
	Know how to approach a problem with both iteration and recursion in mind.

Topic/Skill	Standard
	Be able to formulate a general divide and conquer approach to an algorithm
	Know when it's appropriate to try a binary-search based approach to a problem and know how to properly carry out such an approach.
	Be able to map algorithms for trees to the basic structure of pre, post, and in-order traversals.
	Be able to map algorithms for graphs toa BFS or DFS of the graph
	Understand that designing and analyzing efficient algorithms is a process of exploration and discovery. Hurdles and failures along the way are expected. Problems and potential algorithms must be constantly probed for weakness and misunderstandings. It requires is a critical mind, perseverance, and practice.