CS 240 Detailed Course Objectives
Fall 2019

Students completing this course should have a solid understanding of the following key concepts and general skills:

- The distinction between APIs, ADTs, and data structures
- Techniques for formally evaluating and characterizing the performance of an algorithm
- The primary operations and key terminology for each ADT (list, stack, queue, map, priority queue, graph); this involves using correct terms for each (e.g., don't pop from a list or do a topological sort of a priority queue)
- The algorithms for manipulating array-based and linked data structures, including sorting and hash table collision resolution
- When to use a linear structure vs. a tree or graph structure
- Techniques for traversing and manipulating trees and graphs, including self-balancing and topological sorting

Students completing this course should be able to:

1. Distinguish the concepts of APIs and ADTs
2. Explain the purpose of using Java generics
3. Distinguish the Iterator<E> and Iterable<E> interfaces
4. Implement the required functionality for a simple Java Iterator<E> instance
5. Construct and use Java UnaryOperator<E> lambdas to modify a value
6. Construct and use Java Predicate<E> lambdas to perform a boolean test
7. Construct and use Java Function<T,U> lambdas to execute an arbitrary function
8. Distinguish the concepts of algorithms and programs
9. Identify non-algorithm factors that influence the performance of programs
10. Describe the notion and purpose of asymptotic analysis
11. Identify the strengths and weaknesses of asymptotic analysis
12. Distinguish best/worst case analyses from lower and upper bounds on function growth rates
13. Justify the choice of best, worst, every, or average case analysis for evaluating an algorithm
14. Rank the size of given mathematical functions for large values of n
15. Construct a function T(n) that characterizes the time performance for an algorithm
16. Construct a function S(n) that characterizes the space requirements for an algorithm
17. Apply the formal definitions of Big-O, Big-Ω, and Big-Θ precisely
18. Select and justify the choice of appropriate c and n_0 constants to prove an asymptotic analysis claim
19. Apply valid rules to simplify a mathematical function in Big-O analysis
20. Analyze the worst-case asymptotic complexity for a given algorithm
21. Apply the calculus (limits) definition of Big-O, Big-Ω, and Big-Θ to asymptotic analysis of mathematical functions
22. Explain the purpose of amortized analysis

23. Compare and contrast amortized analysis, worst-case analysis, and average-case analysis
24. Compare and contrast the performance of list implementations (array-based lists vs. linked lists) for a variety of operations
25. Summarize the advantages and disadvantages of using arrays or linked lists to implement a stack or a queue
26. Relate stack and queue ADT operations with array and linked list ADT operations
27. Predict the output of a program using list, stack, and queue ADT operations
28. Illustrate the results of applying ADT operations to array lists, linked lists, stacks, and queues
29. Select an appropriate linear data structure (array list, linked list, stack, queue) for a desired application
30. Illustrate the expansion and contraction of a recursive function for a specified input value
31. Solve a given recurrence relation into its closed form
32. Characterize the run-time complexity of a recursive algorithm in terms of Big-O, Big-Θ, and Big-Ω
33. Illustrate the execution of common sorting algorithms on an array
34. Characterize the Big-Θ run-time complexity of common sorting algorithms in terms of the number of comparisons or swaps
35. Characterize the Big-Θ space complexity of common sorting algorithms in terms of the memory overhead
36. Define the notion of stability and explain its relevance to sorting algorithms
37. Distinguish the properties and relative advantages of Θ(n log n) sorting algorithms
38. Show the order of nodes visited using pre-order, in-order, post-order, and breadth-first tree traversals
39. Reconstruct a binary tree given a combination of pre-, in-, and/or post-order traversals
40. Reconstruct a complete binary tree from the array representation created from a breadth-first traversal
41. Distinguish the notions of complete, full, and balanced trees
42. Explain the relationship between tree size and properties of the tree
43. Illustrate changes to binary search trees, AVL trees, 2-3 trees, red-black trees, and heaps when items are inserted
44. Illustrate changes to binary search trees, AVL trees, and heaps when items are inserted
45. Summarize the benefits and drawbacks of AVL trees compared with red-black trees
46. Distinguish between the array implementations of binary search trees and heaps
47. Distinguish the relative merits of binary search trees, AVL trees, 2-3 trees, and red-black trees in relation to the notion of balance
48. Explain the relationship between complete, full, and balanced binary trees and their maximum heights
49. Identify the relationship between nodes in an array implementation of a complete binary tree (e.g., find a node's parent and/or children)
50. Construct a heap from its array representation
51. Characterize the Big-Θ run-time complexity of tree and heap operations in terms of the number of nodes accessed or swapped

52. Summarize operation of heapsort and compare its performance with other sorting algorithms
53. Construct a Huffman tree for a given set of character frequencies
54. Decode a binary sequence given a Huffman tree encoding
55. Explain how tree-based data structures (binary search trees and heaps) can be used to implement dictionary and priority queue ADTs
56. Select an appropriate data structure (linear or tree-based) for a given application
57. Characterize the best, worst, and average-case complexity for hash tables
58. Calculate the hash index value using simple hash functions, including binning (integer division), modulo calculations, mid-square
59. Use string folding to calculate a hash for a string input
60. Determine the probability of at least one collision for a hash table with T slots and N keys
61. Distinguish the implementation of open and closed hashing
62. Illustrate hash collision resolution using linear probing, pseudorandom probing, and double hashing
63. Explain the concept of primary clustering in the context of hash collisions
64. Calculate the load factor α for a hash table with N records and a capacity of M
65. Describe when to re-hash a hash table and why it is needed to maintain $\Theta(1)$ average time
66. Compare and contrast the performance of the Java TreeMap and HashMap classes, including when each is better than the other
67. Classify a graph based on its characteristics (directed, undirected, weighted, connected, complete, acyclic)
68. Explain the concept of a free tree using graph terminology
69. Use a mathematical formula to characterize the relationship between the number of vertices and the maximum number of edges in a graph
70. Identify a clique in a visual representation of a graph
71. Construct an adjacency matrix and an adjacency list given a visual representation of a graph
72. Construct a visual representation of a graph given an adjacency matrix
73. Calculate the space requirements for a graph using either an adjacency matrix or an adjacency list
74. Determine the best- and worst-case time complexity for finding the neighbors of a given vertex in a graph
75. Illustrate the order of visited vertices in a depth-first traversal of a graph
76. Illustrate the order of visited vertices in a breadth-first traversal of a graph
77. Construct a topological sort of a directed acyclic graph (DAG)