Exam 1 Study Guide

This study guide covers the material you are responsible for on the first exam. This guide describes skills you must be able to demonstrate on the exam. You are expected to be able to *implement* C++ flavored pseudocode demonstrating skills covered in this guide. The order, length or depth of a section in this guide is no indication of the relative importance of that topic or its likelihood to appear on the exam.

General

- *Define* an abstract data type (ADT), and *contrast* it with a data structure.
- *Compare and contrast* static and dynamic data structures
- *Identify* whether a data structure implementation is static or dynamic

Pointers and memory management

- *Explain* what pointers are in C++ and in general, what information they store and how it is represented.
- *Compare* pointers and references.
- Coding pointers
    - *Instantiate* pointers to various data types
    - *Delete* the object to which a pointer points
    - *Null* a pointer, and *explain* why it is good practice to do so after deleting the object to which it points
    - *Dereference* a pointer and *explain* the difference between a dereferenced pointer and the pointer itself.
- *Access* member fields and methods of objects referenced via pointer.
- Arrays
    - *Define* the relationship of pointers to arrays
    - *Compute* the memory address of an array element given:
        - the address of the array
        - the size of the objects stored in the array
        - the index of the desired array element
- Memory management
    - *Implement* copy constructors, assignment operators, and destructors for classes using dynamic memory
    - *Define* a memory leak in terms of pointers and the heap
    - *Find* potential memory leaks in supplied C++ code
    - *Explain* potential causes of memory errors
    - *Define* a segmentation fault
    - *Identify* scenarios in which dereferencing a pointer may result in a segmentation fault

Linked Lists and Arrays

- *Compare and contrast* lists with arrays
- Linked lists
    - *Implement*
        - Element addition
        - Element removal
        - Forward traversal
        - Element retrieval
        - Iterators
    - Nodes
        - *Enumerate* what information nodes contain
        - *Explain* how nodes “know” about each other
- Dynamic arrays (vectors)
    - *Implement*
        - Element addition
        - Element removal
        - A forward traversal
        - Element retrieval
        - Iterators
    - *Describe*
        - Dynamic resizing
- *Compare and contrast* dynamic arrays and linked lists with respect to performance of CRUD (Create, Read, Update, Delete) operations.
- *Create* procedures or algorithms using linked list data structures and *justify* the selection of the data structure.

Stacks, Queues, and Dequeues (SQ+Ds)

- *Identify* all operations in the SQ+D abstract data types
- *Implement* all the operations using either a linked list or dynamic array
- *Explain* the performance benefits of using SQ+Ds
- Describe applications using SQ+Ds and *justify* the selection of the ADT.

Trees

- *Define and identify*
    - Tree types: Binary tree, Binary search tree
    - Tree features: Depth, Height.
    - Subtrees
- Tree nodes
    - *Describe* what information is stored in nodes (for binary trees with a linked representation).
    - *Compare* tree nodes with list nodes.
    - *Describe* the relationships of nodes based on family relations.
    - *Identify* the root, internal, and external (leaf) nodes of trees.
    - *Determine* the height and depth of a node; *determine* the height of a tree.
- Tree traversals
    - *Perform* preorder, inorder, and postorder traversals of trees.
    - *Identify* what types of traversals are useful in a given situation.
- *Identify* characteristics of trees that will give best- and worst-case performance of tree operations.
- *Apply* mathematical induction to prove properties of trees.

Binary Search Trees

- *Define* a binary search tree
- *Draw* a binary search tree given a set of input keys and keys to delete
- *Implement*
    - Insertion
    - Traversal
    - Deletion
    - Retrieval

AVL Trees

- *Define* AVL trees and the height-balance property.
- *Apply* trinode restructuring to maintain the height-balance property.
- *Perform* search, insertion, and deletion in an AVL tree.
- *Evaluate* asymptotic performance of find, insert, and erase in an AVL tree.

Balanced Binary Search Trees and Maps

- *Define* the Map abstract data type, and *define* keys and values in the context of maps.
- *Explain* the purpose of a Map; give examples of applications.
- *Define, compare and contrast* ordered and unordered maps, paying particular attention to what requirements these maps put on the types of keys used.
- *Explain* the advantages of balancing a binary search tree in terms of asymptotic performance (upper and lower bounds) of Map operations.

Asymptotic Analysis (A.K.A Big-Oh, Big-Omega)

- *Compare and contrast* **experimental** (also called **empirical**) algorithm analysis with **theoretical** algorithm analysis.
- *Analyze* algorithms asymptotically and give the *tightest* upper and lower bounds.
- *Implement* algorithms that have a performance described as O(f(n)).
- *Order* functions by their asymptotic growth rate, particularly:
    - O(1)
    - O(lg n)
    - O(n)
    - O(n lg n)
    - $O(n^2)$
    - $O(n^k)$ for some constant k
    - $O(2^n)$
- Given an algorithm, *identify* the best and worst cases for its runtime and what inputs would create those cases. E.g. What are the best, average and worst cases for sequential search, and identify inputs for those cases given a list with values.
- *Prove* simple theorems using the definitions of Big-Oh, Big-Omega, and Big-Theta.