

C++ review material (my notes; also, see Sections 2.2-2.4, 3.3, 4.2)

- Very good knowledge of calling a function by value or by reference.
- Very good knowledge of constructors, copy-constructors, and destructors.
- Very good knowledge of operator overloading (especially the assignment operator).
- Very good knowledge of pointers and dynamic array allocation.
- Differences between static and dynamic array allocation.

Section 2.6 (Comparison of Algorithms)

- Why do we need to compare algorithms?
- How do we compare algorithms?
- Asymptotic Analysis
- Big-O notation

Sections 5.1, 5.3, 6.1 (Array-based Stacks, Queues, and Templates)

- Very good knowledge of the Stack specification and implementation using arrays.
- Very good knowledge of the Queue specification and implementation using arrays.
- Implementation tradeoffs (e.g., using "front" and "rear" to determine whether the queue is empty or full).
- Running times for all the functions of the Stack and Queue implementations.
- Good understanding of the "template" mechanism.

Sections 3.1, 3.2, 4.1, 4.2 (Unsorted Lists, Sorted Lists)

- Very good knowledge of the Unsorted List specification and implementation using arrays.
- Very good knowledge of the Sorted List specification and implementation using arrays.
- Implementation tradeoffs (e.g., in the case of unsorted lists, it is more efficient to insert at the end of the list).
- Binary search algorithm (be careful when "item" is not in the list).
- Running times for all the functions of the Unsorted and Sorted List implementations.

Sections 5.2, 5.4 (Linked Stacks, Linked Queues)

- Very good knowledge of the Stack specification and implementation using linked-lists.
- Very good knowledge of the Queue specification and implementation using linked-lists.
- Running times for all the functions of the linked Stack and Queue implementations.
- Tradeoffs (e.g., memory or time) between array-based and linked-list-based implementations.

Sections 3.4, 3.5, 4.3 (Linked Unsorted List, Linked Sorted List)

- Very good knowledge of the Unsorted List specification and implementation using linked lists.
- Very good knowledge of the Sorted List specification and implementation using linked-lists.
- Running times for all the functions of the linked Unsorted List and Sorted List implementations.
- Tradeoffs (e.g., memory or time) between array-based and linked-list-based implementations.

Sections 3.4, 3.5, 4.3 (Variations of Linked Lists)

- Circular Lists - main idea
- Doubly linked-list - know how to insert/delete
- Singly vs doubly linked-lists - tradeoffs
- "Headers" and "Trailers" - main idea and benefits
- Implementing a linked-list using arrays - main idea and benefits

Chapter7 (Recursion)

- What is recursion?
- How do we implement a recursive function?
- Why are function calls expensive?
- Activation record and run-time stack.
- Iterative solutions vs recursive solutions (tradeoffs).
- Recursive implementation of binary search.
- Inserting items onto a sorted list recursively.

Chapter8 (Binary Search Trees)

- Definitions: binary tree, height, node level, ancestors, descendants, leaves.
- How to compute the height if we know the number of its nodes? Learn the proof.
- How to compute the number of nodes if we know its height? Learn the proof.
- Binary search tree property.
- Very good knowledge of the Binary Search Tree specification and implementation.
- Inserting/Deleting nodes (understand them from an algorithmic point of view - don't memorize code) - Running times important.
- Tree traversals: inorder, preorder, postorder.
- Running times for all the functions of the Binary Search Tree type.
- Binary search (array-based) versus binary search tree.

Sections 8.8, 9.1, and 9.2 (Heaps)

- Full and complete trees (definitions).
- Array-based implementation of trees (why?).
- Very good understanding of the heap data structure (what it is and what kind of applications it is useful for).
- Reheap-up and Reheap-down (understand them from an algorithmic point of view - don't memorize code) - Running times important.
- What is a priority queue and what it is useful for.
- Good knowledge of the priority queue specification and implementation.
- Running time of Enqueue/Dequeue for priority queues.
- Other implementations of priority queues and tradeoffs (e.g., linked-list).

Section 9.3 (Graphs)

- Very good understanding of the graph data structure (what it is and what kind of applications it is useful for).
- Array-based vs linked-list-based implementation of graphs. Tradeoffs between the two implementations.
- Searching a graph using DFS or BFS (understand them from an algorithmic point of view - learn pseudo-code but don't memorize actual code)

Sections 2.4, pp. 294-298, and Section 6.7 (Inheritance)

- What is inheritance?
- Why using inheritance?
- How is it implemented?
- Polymorphism
- Static vs dynamic binding
- Virtual functions
- Slicing problem
- Study very well all the examples

General Comments

The final exam will be closed-books, closed-notes. The format of the final exam will be similar to that of the midterm exam (e., True/False, Short Answer Questions, and Coding). Most of the questions (60%) will be based on material covered after the midterm. Make sure that you go over the examples that we did in class as well as the homework problems. Also, try to do more problems from each chapter if you have the time.

- Have a good understanding of the *trade-offs* between different implementations (e.f., array-based queues versus linked-list-based queues or array-based graphs versus linked-list-based graphs).
- Know how to compute the running-time of an algorithm and how to express it in terms of big-O notation.