Study Guide for Midterm Exam

CSE 30331/34331: Data Structures

The midterm exam will be on Thursday, October 15, 9:30am to 10:45am in 102 DeBartolo. You only need a pen/pencil. (It might be helpful to have a red and a black one.) Books, notes, smart phones, and computers must be closed. You won't need a calculator. The exam is worth 40 points, or about 13% of your grade.

Structure

The exam will have three parts of roughly equal length and value:

- Factual questions. These will test your recall of material in the book, though they will not refer to any specific examples or implementation details in the book.
- Short answers. These require some thought but not much writing.
- Long answers. These may involve writing pseudocode or C++ code, simulating an algorithm or data structure, or discussing how an algorithm/data structure works or why it is good/bad.

Topics

- Complexity analysis. Expect to be given a piece of C++ code and to figure out what the big-O time and/or space complexity are.
- Linked lists.
- Stacks, queues, and deques (implemented as arrays or as linked lists).
- Priority queues and binary heaps, including the handout on `make_heap` (excluding the proof).
- Sorting algorithms: insertion, selection, heap, quick, merge. You do not need to memorize all the quicksort partitioning schemes (Lomuto, Sedgewick, etc). If we ask you to implement quicksort, you can choose the partitioning scheme, or we will describe one to you.
- Binary search.
- Binary search trees (insertion and deletion).
- B-trees and red-black trees. We won't ask you to write C++ code for these, and we won't ask you about deletion. Expect a question that asks you to insert elements into a B-tree or a red-black tree and show the resulting tree.

Sample Questions

Factual Questions

[HW1, Q5] Using Big-O notation, give the worst-case times for each of the following stack operations, for a stack of size *n* that is implemented using a linked list: empty, size, push, pop, and top.

[HW2, Q2] Suppose top is called on a priority queue that has exactly two entries with equal priority. How is the return value of top selected? (a) The implementation gets to choose either one. (b) The one which was inserted first. (c) The one which was inserted most recently. (d) This can never happen (violates the precondition).

[HW2, Q3] Please give the worst-case time complexity for each of the following sorting algorithms: selection sort, insertion sort, and heap sort.

[HW2, Q4] Insertion sort is faster than selection sort when the input array is already in sorted order. True, false, or "it depends"?

Fill in the following table (with "yes" or "no" in each cell) so that it correctly says what kinds of linked lists can be used to implement what kinds of abstract data types efficiently (that is, with constant-time operations).

	stack	queue	deque
singly linked list			
singly linked list with tail pointer			
doubly linked list			

Short Answers

[HW1, Q3] In a few words, summarize the advantages of each of: singly-linked lists, doubly-linked lists, and circularly-linked lists.

[HW3, Q2] The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the longest root-to-leaf path in the resulting tree?

[HW3, Q6] What is the worst-case running time of searching for an element of a red-black tree? Use big-O in terms of *n*, the number of elements in the tree, and briefly justify your answer.

Can a binary tree be both a heap and a binary search tree? If so, give an example; if not, explain why not.

Long Answers

[Fall 2014] What would a B-tree of order 4 (each node has 1–3 values and 2–4 children) look like after inserting the following letters (in the order shown): E, X, A, M. What would a red-black tree look like after inserting the same letters in the same order?

[Fall 2014] You have $k = 1000$ text files that each have $n = 1$ billion lines, in alphabetical order. You need to merge them into a single text file in alphabetical order. How would you use a priority queue (binary heap) to do this efficiently? Hint: the running time should be $O(kn \log k)$, and the memory usage should be $O(k)$.

Fill in the body of the following function to reverse a linked list.

```
template <typename T>
struct node {
      T value;
      node *next;
};

node<T> * reverse_list(node<T> *lst) {
      // Reverses the linked list `lst` destructively
      // and returns the result.









}
```