Name:

CMPS 223 Sample Final

The final is cumulative.
Also review the Sample Midterm and Midterm Solutions.

Concepts

1. Definitions/Short answer.
   (a) Define the term **subtree** for binary trees.
   (b) Define the term **lopsidedness** as it applies to binary trees.
   (c) Define the term **min-heap property** as it applies to heaps.
   (d) What is the average runtime efficiency of the hash table search operation?
   (e) What is the best runtime efficiency of the heapsort operation? (Assume the heap already exists, do not include the runtime for heapify)
2. You have been asked to code an app for sorting and traversing a large dataset. The dataset can fit into the physical memory on the system where the code will run. The user wishes the sort to be as fast as possible, but also wants the ability to print out the data while the sort operation is proceeding. Searching the dataset is not a priority, but the user does want the print traversal to clearly show the largest values in the dataset even while the sort operation is proceeding. Which data structure would you use for this app? Justify your answer by comparing your chosen data structure to data structures you are not using.
3. Hash Tables - What is the effect of collisions on the runtime of the insert, remove and search operations?
4. Heaps - For this question, use max-heaps. You do not need to show all steps, but if you have the wrong answer, any work shown will help you earn partial credit. You may also use the back of the page if you choose to show all steps.
   (a) Show the heap that results from inserting 5, 19, 72, 39, 23 in the order given.
   (b) Using your heap from (a), show the heapsort routine.
5. Binary Search Trees - The following sub-questions concern binary search trees. Use the in-order predecessor method to select the replacement node when deleting. Again, you do not need to show all steps, but you MUST show the steps asked for in the question. If you opt to show all steps, it will help you to receive partial credit for incorrect answers.
   (a) Show the binary search tree that is created by inserting the values 53, 21, 85, 45, 96, 15, 56 one at a time.
   (b) Using the tree created in (a), show the tree after deleting 45. If 45 has two children, first draw the tree with the in-order predecessor circled, then show the tree after deletion.
   (c) Using the tree created in (b), show the tree after deleting 21. If 21 has two children, first draw the tree with the in-order predecessor circled, then show the tree after deletion.

Coding

1. Give the **PSEUDOCODE** (just the pseudocode, not the actual C++ code) for the `void insertionSort(ELEMTYPE *array, int size)` algorithm.
2. Give the **PSEUDOCODE** (just the pseudocode, not the C++ code) for the quicksort `int split(ELEMTYPE *array, int start, int end)` function. This function selects a `pivot` value, partitions the array into "left" and "right" arrays around the pivot and returns the index of the pivot. Be sure to detail how `split` performs each of these operations, particularly partitioning the array.
3. Give the procedural C++ code for the array-based queue functions `bool dequeue(arrayQueue *)` and `ELEMTYPE front(arrayQueue *)`. The `dequeu` function should return false when the queue is empty and true when the first element has been removed. The `front` function should just return the value at the front of the queue (or the "zero" value if the queue is empty) and should not remove anything from the queue. Assume that the structure `arrayQueue` has been defined with the member variables `array`, `frontIndex`, and `backIndex`. Also assume all other supporting functions such as `bool empty(arrayQueue *)` and `advance(x)` have been defined.
4. Give the procedural C++ code for the recursive in-order traversal of a binary search tree, which is accomplished with the functions: `void inorder(BST *tree)` and `void inorderSubtree(TreeNode *subtree)`. The `inorder` function should call `inorderSubtree` if the tree has a root node. The `inorderSubtree` function should recursively traverse the tree using the in-order traversal. This function **MUST** be recursive. Two points will be deducted if the iterative traversal is given. Assume that the structure `BST` has been defined with the member variable `root` and that the structure `TreeNode` has been defined with the member variables `data`, `left`, and `right`.
5. Give the procedural C++ code for the hash table function `bool search(HashTable *h, ELEMTYPE value)` for a hash table that uses double hashing. This function should return true if the value is found in the hash table and false if it is not. Assume that the functions `int hash1(ELEMTYPE value)` and `int hash2(ELEMTYPE value)` have been defined and return the primary and secondary hash keys respectively. Also assume `EMPTY_VALUE` and `DELETED_VALUE` have been defined and that the structure `HashTable` contains the member variables `hashtable` and `count`.