Test Study Guide

This test will be composed of three parts:

1. Demonstrate your knowledge of the functionality and use of various data structures by designing the architecture of a system using multiple structures.
2. Demonstrate your understanding of algorithms by using your architecture to complete a task.
3. Analyze the space/time complexity of your architecture/algorithm.

What to Study

This test will be difficult. Please study. Everything will be included except for trees.

1. Practice Exam
2. Slides
3. Sample problems on slides and homework problems – applications of queues/stacks/etc. – try to implement them (or similar problems) yourself.
4. If you see something you don’t understand, read the info in the book (and/or google it).
5. Review polymorphism and any other course prereqs that you aren't sure about.

C++

This test is designed to test your understanding of data structures concepts, not your ability to code in C++. You have a limited amount of time – if you try to write perfect C++ syntax, you will run out. You probably won’t lose points for writing syntactically incorrect code as long as the semantics are correct.

Example:

Dog theDog = new Dog();

You would not lose points for leaving off the * before theDog, but you would lose points if you needed a pointer and you just wrote “Dog theDog;” Sometimes it is important to distinguish between passing around references and passing around copies. Imagine you had the same object in two different data structures and you wanted to pull the object out of one structure, modify it, and see those changes reflected in the other structure – in this case, you would absolutely need to store pointers in that structure rather than copies of objects.
This exam is worth 100pts and you have 55 minutes to complete it. You can implement the solutions in the language of your choice (or pseudocode), but your representation must adequately convey the correct use of pointers, templates and any other relevant constructs, and use the same level of detail as C++ or Java code. To be safe, you should probably just use C++ or a bastardized version thereof. Do everything as efficiently as possible – you will lose points if your approach is too expensive. Write your name as often as needed.

You will create some combination of data structures that models this system in a space- and time-efficient manner (you can ignore memory nuances like cache misses and contiguous space). You may use these data structures, or any others we’ve covered in class: queue<T> and stack<T> (which both have push, pop/top, empty, and size methods and a constructor that takes no arguments), chain<T>, vector/arrayList<T> (which has get(index), add(element), insert(index, element), indexOf(T element), size and remove(index) methods and a constructor that takes an initial capacity), hashtable<K, E> (or map) (which has put and find methods and a constructor that takes the expected number of elements), pair<K, E> (which has member variables first and second) and the string class. All have iterator methods (begin() and end()) which return pointers to elements or pairs of elements (depending on the structure).

Space requirements:

- integers, Arrays (vectors, queues, stacks), Pointers (chains) 4 bytes
  (note: this is only the overhead – arrays have elements that also take space)
- Hashtables are actually arrays with chains for each bucket
  (assume numBuckets=numKeys)
- Each characters is 1 byte (a string is numCharacters bytes)
Assume the following constants:
  <constants omitted>