212 Test 1 Review Sheet

You will be permitted to write (hand-write or type, not copy) and bring a single, two-sided 8.5x11” sheet of notes. In general, you should be able to use the asymptotic analysis concepts and notation developed in class, analyze algorithms for runtime, and know and be able to trace through algorithms we learned on example data sets. You should also know the runtimes of algorithms learned in class. You won’t have to write any code, though you may be asked to explain an algorithm or even to give a bit of pseudo-code.

Coverage: everything from the sections of chapters 1 and 2 that we covered is fair game. The topics below are those I consider most important:

Chapter 1: Algorithm analysis
- Asymptotic notation
  - definition of $\Omega$, $\Theta$, $O$ (be able to write out formal definitions)
  - examples: e.g. is $3n \log n = \Omega(n)$?
  - polylog, polynomial, and exponential functions -- definition
  - exponentials dominate polynomials; polynomials dominate polylogs
- Runtime analysis
  - worst-case, best case, average case; upper/lower/tight bounds
  - amortized analysis
  - why do we ignore multiplicative constants in analyzing runtimes?
  - Why can’t we compensate for a poor algorithm with a faster computer?
  - what does "for large enough $n$" mean?
  - what is the runtime of the following code examples?
    - e.g. for (i=1; i<n; i++) for (j=1; j<i; j++) cout << "Hello world!\n"
  - estimate runtimes. E.g. if the runtime of an alg is $O(n \log n)$ and $T(1000)=c$, estimate $T(2000)$

Chapter 2: Elementary data structures
- Abstract data types
  - What is an ADT?
  - How do you decide what ADT your program needs?
  - What data structure should you use to implement the ADT you need, given the frequencies of use of each operation?
  - ADTs stack, queue: operations?
- Growable array-based stacks: operation runtimes?
- Trees
  - terminology: height, depth, leaves, ancestors, descendents, parents, children, etc.
  - Binary trees
  - Tree traversals: preorder, inorder, postorder
- ADT priority queue
- Heaps
  - definition,
  - operations (how they work),
  - runtimes of operations
  - Heapsort
- ADT dictionary
- Hash tables
  - hash function,
  - collision resolution: chaining, linear probing, double hashing
  - dynamic hashing
Sample Questions

1. Why do we ignore constants in theoretical analysis of algorithms? How do we atone for this sin?
2. What does it mean to say that $f(n)$ is $O(g(n))$?
3. Suppose an algorithm $A$ has runtime $n^2$ for some problems of size $n$ and $n^3$ for others, when $n \geq 100$. When $n < 100$, the runtime is $n^4$. Which of the following are known to be true about the runtime of $A$?

   Worst case: $O(n)$ $O(n^2)$ $O(n^3)$ $O(n^4)$
   $\Omega(n)$ $\Omega(n^2)$ $\Omega(n^3)$ $\Omega(n^4)$
   $\Theta(n)$ $\Theta(n^2)$ $\Theta(n^3)$ $\Theta(n^4)$

   Average case: $O(n)$ $O(n^2)$ $O(n^3)$ $O(n^4)$
   $\Omega(n)$ $\Omega(n^2)$ $\Omega(n^3)$ $\Omega(n^4)$
   $\Theta(n)$ $\Theta(n^2)$ $\Theta(n^3)$ $\Theta(n^4)$

4. True or false?
   - $n^{1.001} = O((\log n)^{97})$
   - $2^{2n} = O(2^n)$
   - $2^{n^2} = O(2^n)$
   - $n^{2.01} = O(10 n^2)$

5. What is the asymptotic runtime of the following segments of code?
   1. for (i=1; i<n; i++)
      for (j=1; j<n; j++)
      cout << "Hello world!\n";
   2. for (i=1; i<n; i++)
      for (j=1; j<n; j*=2)
      for (k=1; k<n; k++)
      cout << "hello world!\n";
   3. for (i=n; i>1; i/=2)
      cout << "Hello world!\n";
   4. for (i=1; i<n; i *= 3)
      cout << "Hello world!\n"

6. Define ADT ordered dictionary.
7. What is the runtime of the best algorithm studied for building a heap of $n$ numbers?
8. What is the average and worst-case runtime for inserting $n$ items into a hash table using double hashing?
9. What is the worst-case runtime for inserting an item into a binary search tree with containing $n$ items? (2,4)-tree? Red-black-tree?
10. Show how to insert the numbers 10 14 18 31 35 into a 7-element hash table using linear probing for collision resolution. Let the hash function be $n \mod 7$.
11. Show how heapsort would sort the array 1 2 3 4 5 in place. Use a heap with the largest item at the root. Show the contents of the array after each swap.
12. Show the results of a post-order traversal of the following tree: [tree given here]
13. What data structure we studied would be best for implementing a priority queue when we have a very large number of inserts but delete_min operations? How about when we have many min and delete_min operations but few inserts?

Selected answers:
4. F F T F
5: $\Theta(n), \Omega(n^2 \log n), \Theta(\log n), \Theta(\log n)$
6: An ADT that supports insert(), delete(), find(), min(), max(), pred(), succ(), and other basic operations
9: $\Theta(n), \Theta(\log n), \Theta(\log n)$
10: 0: 14, 1:35; 2: nothing, 3: 10, 4:18, 5:31, 6:nothing
11: 1 2 3 4 5; 1 5 3 4 2; 5 1 3 4 2; 5 4 3 1 2; 2 4 3 1 5; 4 2 3 1 5; 1 2 3 4 5; 3 1 2 4 5; 2 1 3 4 5; 1 2 3 4 5