1. Do insertions into an AVL tree.

2. Compare/contrast AVL and red-black trees: What are the efficiencies for insertion/deletion/search? How are these trees similar/different? Is one preferable to the other? If so (not), why (not)?

3. Where are red-black trees used in the C++ STL? ... in Java?

4. Do insertions into a 2-3 tree.

5. What is the minimum/maximum amount of data that can be stored in a 2-3 tree of height \( h \)?

6. Trace bottom-up and top-down (max-) heap construction.

7. What is the worst-case scenario for bottom-up (max-) heap construction of an array of \( n=2^k-1 \) elements? Derive a summation for \( C_w(n) \), the number of key comparisons performed in this worst-case scenario. Solve the summation to get a closed form for \( C_w(n) \).

8. Do the above problem for top-down (max-) heap construction.

9. Trace heap sort and show that its worst-case efficiency is \( O(n \log n) \).

10. Write a recursive function to determine whether \( H[1..n] \) is a max-heap.

11. Given a priority-queue class template with private data members \( H \) and \_size \( (\text{shown on the right}) \) and given the no-argument constructor for the class, show how to implement member functions \textit{empty}, \textit{size}, \textit{top}, \textit{push}, and \textit{pop}.

   ```cpp
   vector<T> H; // a heap
   size_t _size;
   ```

12. Answer questions about hash table insertion, deletion, search for open (chained) hashing; ... for closed hashing using linear probing.

13. Where are hash tables used in the C++ STL? ... in Java?

14. Given a set of characters and their frequencies, show the Huffman tree and Huffman codes that will be produced using the algorithm that we traced in class.

15. Write a recurrence relation for, \( F(n, k) \), the number of function calls performed by the “naive” recursive algorithm for computing the binomial coefficient, \( C(n, k) \). Then, solve this recurrence to write \( F(n, k) \) in terms of \( C(n, k) \).

16. Give the average and worst case time complexities for several implementations of a dictionary.

17. Know the dynamic programming technique for the 0-1 Knapsack problem.