

CSE 100 Final Exam Study Guide,
Fall 2014
CSE Department
University of California, San Diego

1 Final Exam

The final exam will cover all the data-structures and algorithms that we have covered during the quarter, including all the readings, class discussions, and the assignments. Be sure you understand each of the data structures and the algorithms. Make sure you are able to run the algorithms by hand.

Please discuss the questions and answers on Piazza. Find good explanations for the answers. Memorizing answers is counter-productive.

What follows is a list of the subjects that you should review. For additional practice I am providing you with a practice final that is based on Professor Baden's final. I have added to this to more accurately reflect the format of the final. The practice final is available via the course website. Please also go through Professor Kube's midterm and final exams available at this link: <http://cseweb.ucsd.edu/~kube/cls/100/exams.html>. While these can be helpful, understand that we do not necessarily cover all of the same topics nor in the same order. If a question seems off topic for our version of CSE100, ask on Piazza if it is in the scope of what you are expected to know if you are unsure.

Finally, here's a list of potential topics. However, understand that these are *not necessarily complete*. It is your responsibility to review everything we've covered for yourself. This list is just a place to help you get started.

2 Topics

1. Binary Search Trees
 - (a) Insertion and deletion
 - (b) Big-O running time: worst case and average case for successful find
2. Heaps and Treaps
 - (a) Heap properties
 - (b) Treap insert and find
 - (c) Treaps uniqueness (randomized data structure, too)
 - (d) Big-O running time
3. Balanced Trees
 - (a) AVL trees, Red-Black trees and B-trees
 - (b) Insertion and find
 - (c) Running time (Big-O and in practice) Lower bound on size given depth.
 - (d) Comparison with other data structures we've seen, advantages/disadvantages of each
 - (e) B-Trees and memory/disk access (design considerations, why B-Trees are good)
4. Huffman coding
 - (a) Coding trees (tries)
 - (b) Huffman's algorithm
 - (c) Creating trees
 - (d) Using priority queues in Huffman's algorithm
 - (e) Traversing a tree/list using recursion

- (f) Running time analysis of Huffman's algorithm
- 5. Prefix data structures:
 - (a) Multi-way tries, ternary trees
 - (b) Insert and find in each
 - (c) Advantages/disadvantages
 - (d) Running time (Big-O, worst case), space requirements
- 6. Randomized data structures
 - (a) Skip-lists
 - i. Insert, find
 - ii. Construction
 - iii. Expected time for insert/find
 - (b) Randomized search trees (and Treaps, which are not a randomized structure, but used in an RST)
 - i. Big-O running time, comparison to BSTs and Balanced trees
- 7. Hash tables
 - (a) Hash function design, properties of a good hash function, hash table size
 - (b) Collision resolution
 - i. Open addressing (Linear probing, Double hashing, Random hashing)
 - ii. Separate chaining
 - (c) Load factor α
- 8. Graphs
 - (a) Types of graphs: directed vs. undirected, weighted vs. unweighted, Acyclic Directed Graph (DAG)
 - (b) Adjacency list vs Adjacency matrix representation, advantage and disadvantages
 - (c) Traversal for shortest path
 - (d) Breadth first search (BFS), depth first search (DFS), Dijkstra's algorithm
 - (e) Spanning tree and minimal spanning tree: Kruskal's and Prim's algorithm
 - (f) Handle the various cases of the above algorithms (weighted vs unweighted, directed vs undirected)
 - (g) Running time (Big-O) analysis for each of the above algorithms
- 9. Disjoint Set, Union-find
 - (a) Array representation for up trees
 - (b) Union-by-size, union-by-height
 - (c) Path compression
 - (d) Worst case running times under various assumptions
- 10. Algorithms and analysis
 - (a) Amortized cost analysis
 - (b) Worst case vs "average" case
- 11. C++, Bitwise I/O and GitLab
 - (a) Classes and subclasses, abstract base classes, inheritance, templates
 - (b) Constructors and Destructors
 - (c) Pointer Arithmetic
 - (d) Iterator pattern

- (e) Pointers vs. references
- (f) Pass by value and pass by reference
- (g) Const; const pointer vs. pointer to const
- (h) Standard Template library
- (i) C++ array and string vs. STL::Vector and STL::Strings
- (j) Storage allocation and reclamation
- (k) Bit manipulation
- (l) command line parameters, argc and argv
- (m) Stream I/O and File I/O
- (n) Bitwise I/O
- (o) Version control – basic ideas, data and workflow
- (p) Distributed vs centralized Version control system

12. Code

- (a) C++ code for implementing pieces of the insert and find algorithms and data structure construction. In general, I expect you to be able to write relatively simple C++ functions to implement the algorithms for manipulating any of the data structures we've seen in class or in the PAs. I won't make you do anything really complex, like implement search tree insert from scratch, but you should be able to, for example, write code to find an element in the various trees, manipulate a Huffman tree, given the representation of the data structure or implement simple graph algorithms like BFS and DFS .
- (b) All the Programming Assignments including relevant analysis, e.g. explain where your speed up came from vs. reference solutions.