csce350 — Data Structures and Algorithms
Fall 2019 — Review Sheet for Final Exam

Test date: Tuesday, December 10, 9:00am

This review sheet is a general outline of the topics we covered in this class.

- ✓ The test will cover everything from the course. Material covered since Test 2 will comprise more than one third of the exam.
- ✓ You'll have 150 minutes to complete the test.
- ✓ Notes are permitted subject to the following conditions, about which I am quite serious.
  - The notes must be on a single note card measuring 3 inches by 5 inches or less.
  - Yes, you may write on both sides.
  - You must write the notes yourself, using your hand and a pencil or pen.
  - You must write your name in the top left corner.
  - You must submit your note card with your test, but I will return it later at your request.
  - You must not make any modifications that change the surface area of the card, nor use magnifiers, colored films, nor any other objects to assist with reading the card.
  - If you prefer, you may choose not to use any notes at all.
- ✓ No calculators nor other reference material are permitted.
- ✓ The test packet will include copies of pages 475, 476, and 477 of the textbook. No other reference material will be included.
- ✓ Possible question types include, but are not limited to:
  - Problems created by mutation of the homework. (. . . but be prepared to think!)
  - Problems similar in spirit to the homework, but related to different concepts from the lectures.
  - Questions evaluating your mastery of the new terminology and concepts introduced in the course. Example formats: Multiple choice, fill-in-the-blank, short answer.

General

1. For each algorithm we covered, be prepared to: Explain the problem it solves, including the inputs and outputs. Describe the basic idea of how the algorithm works, and why each part of its pseudocode is important. Classify the underlying algorithm design strategy. Execute the algorithm on appropriately-sized inputs. Analyze the run time of the algorithm.

Review

1. Homework 0

Chapter 1

1. Definition of an algorithm. Unambiguous. Correct for all valid inputs. Be prepared to identify 'impostors' that are not unambiguous, or that do not work correctly for all valid inputs.
2. Choices for expressing algorithms: Flow charts, structured prose, 'real code'. Pseudocode. Why do we write pseudocode? Dos and Don'ts for writing pseudocode.
3. Euclid's algorithm for $\gcd(m, n)$. What is $\gcd$? Be able to execute Euclid's algorithm, showing each step of the computation.
4. Basic data structures: Arrays, linked list, queues, stacks. Operations available for each. Advantages and disadvantages.

Chapter 2

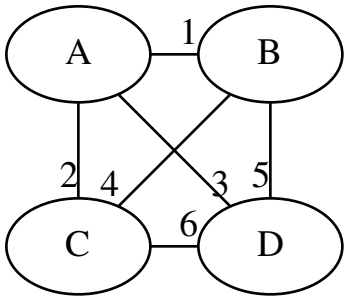
1. Four steps for analyzing an algorithm. Why do we focus only on basic operations? $T(n) \approx c_{\text{op}}C(n)$.
2. Best-case vs. worst-case vs. average-case analysis. Why do we usually not bother with best-case analysis? Why do we usually not bother with average-case analysis? What does the phrase "worst-case" mean?
3. Θ, O, and Ω notations: Why do these exist? Why do we ignore multiplicative constants? Basic intuition of what each one means. Proving from the definitions (must show c (or c_1 and c_2) and n_0). Proving using limits (with and without L'Hôpital's rule). Understand Figures 2.1, 2.2, and 2.3 from the book. Four "shortcuts".
4. Analyzing nonrecursive algorithms: Write a summation formula for the number of basic operations. Simplify such a summation to determine the *exact* count of basic operations. Classify the order of growth using Θ notation. Be careful not to confuse the output of the algorithm with its runtime. When does this method fail?
5. Analyzing recursive algorithms: Write a recurrence for the number of basic operations (including the base case). Solve such a recurrence by backward substitution. Be careful not to confuse the output of the algorithm with its runtime. Change of variables, for recurrences like $A(n) = A(\lfloor (n/2 \rfloor) + \ldots$. Towers of Hanoi example is not important in itself, but being able to analyze that algorithm *is* important.

Chapter 3 (Brute force)

1. Definition of brute force.
2. Integer powers by brute force.
3. Selection sort.
4. Brute force string matching.
5. Exhaustive search: General idea (try all possibilities, check each one against the constraints, pick the best). Traveling salesman problem. Knapsack problem. How many candidates need to be considered for each of these?

6. Not covering: Bubble sort, Closest-pair, convex hull, assignment problem, depth-first and breadth-first search.

7. Example problem: Use exhaustive search to solve this instance of the traveling salesman problem:



8. Example problem: Use exhaustive search to solve this instance of the Knapsack problem:

i	1	2	3
w_i	10	12	16
v_i	10	8	16

$W = 24$

Chapter 4 (Decrease and Conquer)

1. Definition of decrease and conquer.

2. Integer powers, decrease and conquer version.

3. Insertion sort

4. Binary search

5. QuickSelect ("Hoare's selection algorithm") What is the role of the partition function? How does partitioning work? How does the choice of pivot impact the algorithm's run time?

6. Not covering: topological sorting, generating permutations, generating subsets, fake-coin problem, Russian peasant multiplication, Josephus problem, interpolation search, game of Nim

Chapter 5 (Divide and Conquer)

1. Definition of divide and conquer. How does it differ from decrease and conquer?

2. Master theorem. Be able to identify recurrences to which the Theorem applies. Be able to simplify the answers where appropriate.

3. Mergesort. What is the role of the merge function?

4. Quicksort. What is the role of the partition function? How does the choice of pivot impact the algorithm's run time?

5. Multiplication of large integers. Brute force ("pencil-and-paper") algorithm. Karatsuba algorithm.

6. Not covering: binary tree traversals, Strassen's algorithm, closest pair, convex hull

Chapter 6 (Transform and Conquer)

1. Definition of transform and conquer.

2. Presorting. When is this a good idea? Why?

3. AVL trees. Background: Inserting and search in standard binary search trees. An AVL tree *is* a binary search tree. What problem with BSTs do AVL trees solve? Definition of balance factor. What balance factors are allowed in an AVL tree? Rotations: Left and right. Where do the subtrees go during a rotation? Algorithm to rebalance a tree. How many rotations are needed in the worst case? **Practice problem for AVL trees: 6.3.4**

4. **Topics below this appeared after Test 2.**

5. 2-3 trees. Difference from binary search trees. How to insert, including splits and promotions. How to search. How does a 2-3 tree grow taller?

6. Heaps. What is a heap? What operations can be done on a heap? Requirements for a binary tree to be a heap. Insertion and deletion algorithms. Be able to list compares and swaps for insert and delete operations. Heapsort.

7. Skip: Gaussian elimination, Horner's rule, binary exponentiation, problem reductions.

Chapter 8 (Dynamic Programming)

1. Describe the dynamic programming approach. How does it differ from divide-and-conquer? How do the recurrences differ from the recurrences we wrote in Chapter 2?

2. Basic steps for designing a DP algorithm.

3. Binomial coefficients.

4. Knapsack problem with DP. How to fill in table using recurrence. How to extract optimal selection of items from the table. Understand the purpose of each part of the solution recurrence. Understand what each value in the DP table means.

5. Top down versus bottom up. Advantages and disadvantages of each.

Chapter 9 (Greedy algorithms)

1. Describe the greedy method. Each step must be feasible, locally optimal, and irrevocable.
2. Minimum spanning trees. What is this trying to compute? Prim's algorithm. Kruskal's algorithm.
3. Example problem: Use Prim's algorithm to find a minimum spanning tree of the graph below. Use node A as the root. List the vertices added to the MST in order the order in which they are added.
4. Example problem: Use Kruskal's algorithm to find a minimum spanning tree of the graph below. List the edges considered by the algorithm in order, and indicate whether each one is selected or rejected by the algorithm. Show the disjoint set forest after each step.