Computer Science Department - Rutgers University **Fall 2017**

CS 512: Final Notes

16:198:512
Instructor: Wes Cowan

Chapter 0: Prologue

- The most important thing to get from this chapter is probably the review of asymptotic / big-O notation, both formal definitions and practical usage. Be familiar with the basic orders (n, $\ln n$, n^2, $n \ln n$, 2^n, $n!$), how they compare, how they related to each other. What are O vs Ω vs Θ? What are some good rules of thumb for being able to compute / compare the dominant or leading terms?
- Know your rules of logs and rules of exponents.
- If $f = O(g)$ and $g = O(h)$, show formally that $f = O(h)$.
- Why are bubble sort and insertion sort $O(n^2)$? With or without the Master Theorem, why is merge sort $O(n \ln n)$? What about quicksort?
- Express the complexity of bubble sort, both as a recurrence relation, and explicitly in terms of the operations in the individual loops.

Chapter 1: Algorithms with Numbers

- Complexity of basic arithmetic operations - addition, subtraction, multiplication, division.
- Can you put any lower bounds on the complexity of these operations? How much work would addition and multiplication need to do, at least?
- Why do we do arithmetic in binary? Must we?
- How do the 'naive' algorithms compare to the more advanced ones (this ties into material from future chapters)?
- Be familiar with the definition and use of modular arithmetic. Importantly, if something is congruent to something else mod N, i.e., $a \equiv b (\text{mod} N)$, this does *not* mean that $a = b$. What does it mean?
- What is the practical importance of modular arithmetic? (Hint: why do we talk about 64-bit computers?)
- It is frequently useful to exchange from modular relations to statements about the structure or form of numbers - for instance, replacing the statement that $a \equiv b (\text{mod} N)$ is equivalent to $a = b + kN$ for some integer k.
- Prove that if $a \equiv a' (\text{mod } N)$ and $b \equiv b' (\text{mod } N)$, then $a + b \equiv a' + b' (\text{mod } N)$ and $ab \equiv a'b' (\text{mod } N)$.
- How does the complexity of integer arithmetic compare to the complexity of modular arithmetic? How does modular exponentiation compare in complexity to exponentiation?
- Important Algorithm: Euclid's Algorithm for the GCD. What is the GCD, why is it important?
- Given integers a, b, c, d, how could you use Euclid's (extended) algorithm to find integers x, y, z such that $ax + by + cz = d$? When would these integers exist, and what is the complexity of finding them? Relate this to the complexity of solving certain integer linear programming problems with equality constraints.
- Connect the Extended Euclidean Algorithm with division (multiplicative inverses) mod N.

- If the multiplicative inverse of a exists mod N, is it unique?
- Definition of primality, definition of factoring. What are some advantages to doing arithmetic mod a prime p?
- Can Fermat's Little Theorem prove that a number is prime? Can it prove that a number is not prime? What is the value, practically speaking, of probabilistic algorithms?
- It is worth spending some time percolating on the importance of modular inverses to Fermat's Little Theorem, both the statement of it and the proof of it.
- Use Fermat's Little Theorem to prove Fermat's Last Theorem.

Chapter 2: Divide and Conquer

- The basic divide and conquer approach: reduce a problem into a set of subproblems that are all some fraction of the whole, solve each subproblem, and then merge the results to solve the main problem.
- Review how to construct a recurrence, even an approximate recurrence, relating solving the whole problem to the complexity of solving the individual parts. What determines the complexity of merging sub-problems?
- Review the Master Theorem: how is the total work of solving the problem distributed between the splitting, the solving, and merging operations? *What is the dependence on a, b, d?* Recall the notion of the branching tree indicating the work distribution, and measuring its depth.
- Write the recurrence relation for merge sort and solve it using the Master Theorem. What is the over all complexity? What if you divided the list to sort into three sections, rather than two?
- Why is the complexity of naive matrix multiplication $O(n^3)$? Argue via the Master Theorem that matrix multiplication via blocking gives the same complexity. The book gives an algorithm of complexity $O(n^{2.81})$ - in terms of the splitting / merging breakdown, where are the savings achieved?
- Why is finding the median of a list (on average) easier than sorting the list?
- How does the problem of *expected* complexity (as Median is a randomized algorithm) factor into the use of the Master theorem?
- Given a list of n elements, what is the complexity of verifying that a specific value is within the middle 50%?
- Review the quiz questions on analyzing structured or sorted lists as practice, utilizing known structure to break down a problem into sub-problems.
- **Sorting:**
  - Know the complexity of the various sorting algorithms. What is the worst case for quicksort? Why? How can that worst case be avoided? What is the worst case for merge sort?
  - Your grad student tells you that they have a sorting algorithm that is $O(n\sqrt{\ln n})$. Do you believe them? Why? Another grad student tells you that they have a sorting algorithm that performs in $O(\sqrt{n})$. Do you believe them? Why?
  - Formulate sorting as a search problem in NP - i.e., what is an instance of the problem, what does a proposed solution look like? What is the complexity of verifying a proposed solution? Argue that sorting is in P.

Chapter 3: Graph Decomposition

- It cannot be overly stressed: be comfortable taking a graph (in some form) and performing a DFS with pre/post labeling, and analyzing the structure of the graph. What are the taxonomy of edges in and related to the DFS tree? Be able to decompose a graph into connected, biconnected, and strongly connected components (review relevant quiz, exam, and homework problem).
- How can you identify sources and sinks in directed acyclic graphs? How can you identify source and sink components in directed graphs?
- Claim: deciding whether a graph is connected is in P. True or false?
- What does it mean to topologically sort a directed acyclic graph? Why would you want to (what could a DAG represent)? How could you do it, and what is the complexity of that operation?

Chapter 7: Linear Programming and Reductions

- Review applications of linear programming - what can be represented as linear programs? (Max flow, min cut, etc.) How can linear programs and their solutions be interpreted?
- In the reduction of matching to max flow, what does the solution of the LP look like if there is no matching?
- Practice solving simple linear programs by sketching the feasible set and identifying the vertices. How does the dual correspond to the primal, geometrically?
- Be able to write the dual. What does the dual of a max flow LP look like, and how can the variables be interpreted?
- The dual can be incredibly useful. Why? What are potential limitations / pitfalls to watch out for?

Chapter 8: NP-Complete Problems

- What does it mean for a problem to be in NP? What does it mean for a problem to be in P?
- Can you provide some quick and dirty upper and lower bounds for TSP?
- Search Problems vs Optimization Problems vs Decision Problems: How are they connected? How can one let you solve the others?
- What would the search formulation of finding minimum spanning trees be?
- Is finding a minimum spanning tree in P? Is finding a minimum spanning tree in NP?
- Know common (hard) NP problems, common P problems. What connects the easy version of a problem and the hard version? Why is MST easy but TSP hard?
- Formulate a dynamic programming solution to the longest path problem.
- 3-SAT is NP-Complete. Does this mean that **every** instance of 3-SAT is hard to solve? Generate instances of 3-SAT in n variables that are easy to solve.

- Your grad student tells you that they've built an oracle that, given an instance of 3-SAT, can tell you in polynomial time whether it is hard or easy to solve. Do you believe them?
- Familiarize yourself with the classes of NP-complete problems presented in this chapter, and the connections between them. What does it mean that a problem reduces to another problem? Finding shortest distances can be reduced to CircuitSAT - does this mean that finding shortest distances is NP-complete?
- Familiarize yourself with some of the important reductions. What are the connections between finding certain paths and certain cycles in graphs? (Recall the DP formulation of TSP, which reduced the problem to finding certain paths rather than cycles.) Why is SAT no harder than 3-SAT? Why is 3-SAT no harder than Independent Set? What is the connection between Independent Set, Vertex Cover, and Clique?
- Suppose you were searching for the largest independent set of a graph G. An oracle can tell you, given a graph and a vertex, whether that vertex is in the largest independent set in G (or a largest independent set, if there are multiple ones). How could you use this oracle to efficiently construct the largest independent set? What if the oracle were only right 50% of the time?
- Does P = NP?
- Why is CircuitSAT top of the NP food chain?

Chapter 9: Coping with NP-Completeness

- Intelligent Backtracking: Sometimes searching for a solution is the only thing to be done, but at least do it intelligently.
- How can SAT be broken up into sub problems to solve, how can you tell when a partial solution is not worth pursuing?
- How could a system of ZOE be broken up into sub problems to solve, how can you tell when a partial solution is not worth pursuing?
- How could finding a Rudrata Path be broken up into sub problems to solve, how can you tell when a partial solution is not worth pursuing?
- Try to formulate a branch and bound approach to the knapsack problem. How can you generate upper and lower bounds?
- Try to formulate a branch and bound approach to the longest path problem.
- Be familiar with the TSP - MST bounding and approximation.
- Be familiar with the Vertex Cover - Maximal Matching bounding and approximation.
- Your grad student tells you they have discovered a polynomial time approximation algorithm for all traveling salesman problems. Do you believe them?
- What is clustering, and why is it hard?
- For any given NP-Complete problem, try to formulate a local search type solution, that modifies proposed solutions in the hope of reaching an optimal solution. What are some weaknesses and strengths of your approach?