

SOLUTIONS: Midterm, CS520, McConnell

1. Big-O analysis and identities involving logs and exponents.

Compare the running times of the first versus the second. Show how you got your answer. Let the first be $f(n)$ and the second be $g(n)$. Write your answer as $f(n) = o(g(n))$, $f(n) = \Theta(g(n))$, or $f(n) = \omega(g(n))$. Prove your answer.

- (a) n^2 versus $5^{\log_2 n}$.

Solution: $4^{\log_2 n} = (2^2)^{\log_2 n} = (2^{\log_2 n})^2 = n^2$. Therefore, $g(n)/f(n) = 5^{\log_2 n}/n^2 = 5^{\log_2 n}/4^{\log_2 n} = (5/4)^{\log_2 n}$, which grows without bound so $f(n) = o(g(n))$.

- (b) $(\log_2 n)^{(\log_2 n)}$ vs. n^2

Solution: When $n \geq 256$, $\log_2 n \geq 8 = 2^3$, so $(\log_2 n)^{\log_2 n} \geq (2^3)^{\log_2 n} = (2^{\log_2 n})^3 = n^3$. $f(n) = \omega(g(n))$.

- (c) Hurting your case: $\sum_{i=1}^n i^3$ vs. n^4

Solution: $f(n) = O(g(n))$, since each term is at most n^3 and there are n of them. $f(n) = \Omega(g(n))$, since the largest $\lfloor n/2 \rfloor$ terms are at least $n^3/8 = \Theta(n^3)$ and there are $\Theta(n)$ of these. It follows that $f(n) = \Theta(g(n))$.

- (d) $\sum_{i=1}^{\lceil \log_2 n \rceil} 4^i$ vs. n^2 .

Solution: The summation is a geometric series with largest term $\Theta(4^{\log_2 n}) = \Theta((2^2)^{\log_2 n}) = \Theta((2^{\log_2 n})^2) = \Theta(n^2)$, so $f(n) = \Theta(g(n))$.

- (e) $\log_2^* n$ vs. $\log_2^*(\log_2 n)$. **Solution:** $f(n)$ is just the number of times you can iterate the \log_2 until you get down to 2; $g(n)$ has a head start of one iteration, so $g(n) = f(n) - 1$ and $f(n) = \Theta(g(n))$.

2. Get a big- Θ bound on $T(n) = 2T(n/2) + n/\log_2 n$.

Solution: $T(n/2) = 2T(n/4) + (n/2)/\log_2(n/2) = 2T(n/4) + (n/2)/(\log_2 n - 1)$.
 $T(n) = 2[2T(n/4) + (n/2)/(\log_2 n - 1)] + n/\log_2 n = 4T(n/4) + n/(\log_2 n - 1) + n/\log_2 n$.
 It's easy to see by induction that after i iterations, the expression is $2^{i+1}T(n/2^{i+1}) + n/(\log_2 n - i) + n/(\log_2 n - i + 1) + \dots + n/\log_2 n$.

Therefore, the full expansion is $2^{\log_2 n}T(n/(2^{\log_2 n})) + n/1 + n/2 + \dots + n/\log_2 n = n + n(1/1 + 1/2 + 1/3 + \dots + 1/\log_2 n) = \Theta(n \log \log n)$ by the bound for the harmonic series.

3. We gave an $O(n \log n)$ algorithm for finding a longest increasing subsequence of a permutation of $(1, 2, \dots, n)$. It maintained an array $A[]$ that recorded, for each i , the smallest ending value of an increasing subsequence of length i . Let's run it on the sequence $(3, 2, 7, 1, 9, 6, 8, 4, 5, 10)$.

- (a) Draw the state of the array after each of the ten updates.

Solution:

3
 2
 2 7
 1 7
 1 7 9

1 6 9
 1 6 8
 1 4 8
 1 4 5
 1 4 5 10

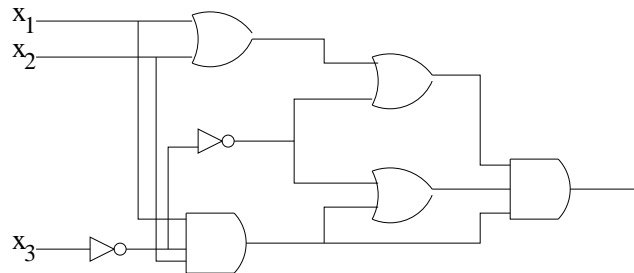
- (b) On top of the sequence (3, 2, 7, 1, 9, 6, 8, 4, 5, 10), draw the pointers the algorithm generates for allowing us to reconstruct a longest increasing sequence from a chain of pointers. Give the longest increasing subsequence it produces.

Solution: 3 points to null, 2 points to null, 7 points to 2, 1 points to null, 9 points to 7, 6 points to 1, 8 points to 6, 4 points to 1, 5 points to 4, 10 points to 5. The longest chain is 10 points to 5 points to 4 points to 1. The solution it gives is (1, 4, 5, 10).

4. Briefly describe a greedy 2-approximation algorithm for finding a vertex cover. Be sure that it specifies which vertices you select. Then explain why this can be no more than twice the number of vertices in an optimum vertex cover.

Solution: Iteratively select an edge and include its two endpoints in the vertex cover, and remove the two endpoints from the graph. Since the selected edges share no endpoint, an optimum vertex cover must have at least one vertex for each selected edge. This algorithm selects two for each selected edge.

5. Show that you understand the reduction from CIRCUIT-SAT to SAT by illustrating the reduction on the following example:



Solution: See Figure 34.10 from the book.

6. Show that if m is the largest key in a binary search tree whose n elements were inserted in random order (with no balancing operations), then the expected number of ancestors of m is $O(\log n)$.

Solution: The smallest element can only be an ancestor if it was inserted first; otherwise it's sent into the left subtree, which doesn't have m in it. The chances of this are $1/n$. This is what it contributes to the expected number of ancestors. The smallest element has no effect on the chances of other elements become ancestors of m , so we can recurse on the remaining $n - 1$ elements to get the rest of the expected value. This gives $1/n + 1/(n - 1) + \dots + 1/1$, which is the harmonic series, $\Theta(\log n)$.

7. Prove the $O(n \log n)$ expected bound for randomized **Quicksort**. Your proof should be in the style of the one you did for randomized **Select** on a homework. Assume all elements are unique. Let X_{ij} denote the random variable that has value 0 if i^{th} and j^{th} largest elements are not compared, and 1 if they are compared. These X_{ij} 's are

like biased coins. Group them into piles of coins with equal bias. Think about the expected sum of the coins in each pile.

For your written answer, write a summation that has one term for each pile, bounding its expected sum. Then use an identity we've studied to get an $O(n \log n)$ bound on the sum.

Solution: X_{ij} has probability $2/(j - i + 1)$ of being 1, so this is its expected value. There are at most n X_{ij} 's with expected value $2/2$, at most n with expected value $2/3$, at most n with expected value $2/4$. The sum of expected values is $(2n/2 + 2n/3 + 2n/4 + \dots + 2n/n)$, which is $2n(1/2 + 1/3 + 1/4 + \dots + 1/n) = 2n\Theta(\log n) = \Theta(n \log n)$.

8. For the *weighted vertex cover* problem, the vertices have weights. We want to find a vertex cover whose total weight is minimum.

(a) State a corresponding decision problem. (Make sure that it's in NP and that it reduces to this optimization problem in polynomial time.)

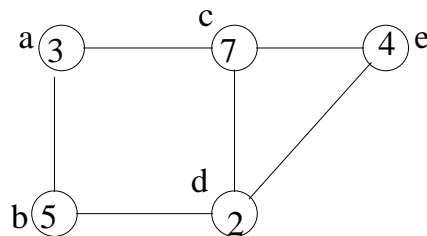
Solution: Given a graph with vertex weights and a target weight W , determine whether there is a vertex cover of weight at most W .

(b) Show that the decision problem is NP-complete. You may make use of the NP-completeness of any other problem we've shown to be NP-complete.

Solution: The standard vertex-cover decision problem is a special case where the weights are equal to 1.

(c) The book shows how to get within a factor of two by setting up a linear program that relaxes the constraint that a vertex either be entirely present or entirely absent.

Rather than describing how this is done, illustrate it on the following example by writing down the objective function and the inequality constraints:



(The numbers in the vertices are their weights.)

Solution: Minimize $3x_a + 5x_b + 7x_c + 2x_d + 4x_e$, subject to $x_a + x_c \geq 1$, $x_a + x_b \geq 1$, $x_b + x_d \geq 1$, $x_c + x_d \geq 1$, $x_c + x_e \geq 1$, $x_d + x_e \geq 1$, $x_a, x_b, x_c, x_d, x_e \geq 0$.

(d) Tell why a minimum-weight vertex cover has at least the weight of a solution returned by the linear program.

Solution: An optimum vertex cover satisfies the constraints of the linear program, and the linear program returns the solution that satisfies the constraints and minimizes the total weight. (The vertex cover satisfies the additional constraint that the variables are 0 or 1.)

(e) Tell how to select the approximate solution from the solution to the linear program, and why its weight is no worse than twice the optimum weight.

Solution: Select each vertex y such that $x_y \geq 1/2$. This at most doubles the weight contributed by y over what it contributes in the linear program.

- (f) Explain why this gives a solution that's a valid vertex cover.

Solution: It's a vertex cover because if (y, z) is an arbitrary edge, the constraint $x_y + x_z \geq 1$ guarantees that at least one of x_y and x_z will be selected, hence the edge will be covered.

9. *Integer linear programming* is just like linear programming except that you have the additional constraint on your solution that all variables must take on integer values. You seek to maximize the objective function, subject to $A\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$, and all coefficients of \mathbf{x} are integers.

Show that it's NP-hard by showing that even finding whether a feasible solution exists is NP-complete:

- (a) Show that determining whether there is a feasible solution is in NP.

Solution: If the answer is yes that it's feasible, then there exists a feasible solution that can be verified in polynomial time.

- (b) Find a polynomial-time reduction from 3-CNF-SAT. Rather than describing the reduction, show what instance of integer linear programming feasibility the following instance of 3-CNF-SAT reduces to:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

Solution: $x_1 + \overline{x_1} \leq 1$, $x_2 + \overline{x_2} \leq 1$, $x_3 + \overline{x_3} \leq 1$,
 $x_1 + \overline{x_2} + \overline{x_3} \geq 1$, $\overline{x_1} + x_2 + x_3 \geq 1$, $x_1 + x_2 + x_3 \geq 1$
 $x_1, x_2, x_3 \geq 0$.