Basic Algorithms (Section 7) Practice Midterm Solution
Instructor: Jiaxin Guan

October 14, 2024

1. This is a practice exam to help you prepare. You do not need to turn it in.
2. You have 75 minutes to complete the real midterm, this exam is intended to be comparable in length and difficulty.
3. You may use a double-sided, letter-size "cheatsheet". The use of phones, computers, or other reference material during the exam is not permitted.
4. You may use any algorithm or theorem we saw in class (or homework) without proof, as long as you state it correctly. For all questions asking you to give algorithms, you do not have to give detailed pseudo-code, or even any pseudo-code. It is enough to give a clear description of your algorithm. You should additionally give a brief justification of the correctness and the claimed run time of your proposed algorithm.
5. The structure of the real midterm will be somewhat similar to what is below, but I might adjust the number of multiple choice/algorithm design questions. In general, you should expect questions of the following types:
   (a) Questions that can be solved if you understood what was presented in class (definitions, how the algorithms work).
   (b) Questions that can be solved by reducing to something we did in class (e.g. use some algorithm from class to solve a new problem).
   (c) Questions that can be solved by adapting an idea from class (e.g. Question 2 in HW5 asks you to modify the rod cutting DP algorithm to handle unique lengths).
6. This exam contains 5 pages (including this cover page).

Question	Points	Score
Multiple Choices	50	
k Largest Elements	20	
Longest Increasing Subsequence	30	
Total:	100	

1 Multiple Choices (50 points)

Choose the best answer for each of the questions below. No justification is needed.

1. Which of the following is **not** $O(n \log n)$? . . . . (A)
   (A) $\frac{9n^2}{(\log n)^3}$
   (B) $\sqrt{58n} \cdot (\log n)^2$
   (C) $87n^{\frac{\log_5 n}{\log_3 n}}$
   (D) $n \cdot \log(n^3)$
2. Let $f(n), g(n), h(n)$ be positive functions. Which one of the following statements is true? . . . . (C)
   (A) If $f(n) \neq o(g(n))$, then $f(n) = \Omega(g(n))$
   (B) If $f(n) = o(g(n))$ and $g(n) = \omega(h(n))$, then $f(n) = \Theta(g(n))$
   (C) If $f(n) = O(g(n))$, then $(f(n))^2 = O((g(n))^2)$
   (D) None of the above
3. Suppose we have $T(n) = 2T(n/3) + n$ and $T(0) = T(1) = 1$, which of the following statements is false? . . . . (B)
   (A) $T(n) = \omega(n^{2/3})$
   (B) $T(n) = \Omega(n \log n)$
   (C) $T(n) = O(n \log n)$
   (D) $T(n) = o(n^2)$
4. Which of the following is **not** an example of a divide-and-conquer algorithm? . (D)
   (A) QuickSort algorithm for sorting an array
   (B) Karatsuba's algorithm for fast integer multiplication
   (C) Deterministic Selection algorithm for finding the median
   (D) Gale-Shapley algorithm for stable matching
5. Consider the QuickSort algorithm where we always pick the last element as the pivot and arrays $A = [1, 2, 3, \ldots, n]$ and $B = [n, n-1, n-2, \ldots, 1]$. Let C_A be the number of comparisons made when running QuickSort on A, and C_B be the number of comparisons made when running QuickSort on B. Then we have . . . . (B)
   (A) $C_A > C_B$
   (B) $C_A = C_B$
   (C) $C_A < C_B$
   (D) Cannot say anything for arbitrary n
6. Alice comes up with a comparison-based sorting algorithm whose best-case run time is $O(n)$, and Bob comes up with a comparison-based sorting algorithm whose worst-case run time is given by the recurrence $T(n) = 4T(n/5) + 5n \log \log n$. Which of these two algorithms can **possibly** be correct? . . . . (B)

(A) Neither Alice's nor Bob's
(B) Only Alice's
(C) Only Bob's
(D) Both Alice's and Bob's

7. The subset-sum problem is defined as follows. Given a set of n positive integers $A = \{a_1, a_2, a_3, \dots, a_n\}$ and positive integer s, is there a subset of A whose elements sum to s?

   A DP algorithm for solving this problem uses a 2-dimensional Boolean array X, with n rows and $s+1$ columns. The entry $X[i,j]$ for $1 \le i \le n, 0 \le j \le s$ is TRUE if and only if there is a subset of $\{a_1, a_2, \dots, a_i\}$ whose elements sum to j.

   With that in mind, which of the following is valid for $2 \le i \le n$ and $a_i \le j \le s$? ($\wedge$ is the logical AND, and $\vee$ is the logical OR) . . . . (B)

   (A) $X[i,j] = X[i-1,j] \vee X[i,j-a_i]$
   (B) $X[i,j] = X[i-1,j] \vee X[i-1,j-a_i]$
   (C) $X[i,j] = X[i-1,j] \wedge X[i,j-a_i]$
   (D) $X[i,j] = X[i-1,j] \wedge X[i-1,j-a_i]$

8. Which of the following statements is true? . . . . (C)

   (A) For the same problem, the DP algorithm with memoizaiton/bottom-up always performs better than a Divide-and-Conquer algorithm
   (B) To justify the correctness of a DP algorithm, it is sufficient to justify the correctness of the optimal substructure used in the algorithm
   (C) There can be more than one possible optimal substructure for a DP problem
   (D) None of the above

2 k **Largest Elements** (20 points)

Describe an $O(n)$ time algorithm that, given an unordered array of n arbitrary integers, and an integer $k \in \{1, \ldots, n\}$, outputs the k largest integers in the array (not necessarily in order). Justify the correctness and run-time of your proposed algorithm.

The problem is very simple if we assume all elements are distinct: we just use the deterministic select algorithm with "median of medians" to select the k-th largest integer, denoted as x, from the array, and then output all elements $\geq x$. It is slightly more complicated if we allow repetitions.
First, we use the deterministic select algorithm with "median of medians" to select the k-th largest integer, denoted as x, from the array. Then, we iterate through the array and output all elements that are strictly greater than x, and keep track of the number of elements we have output, denoted as m. Lastly, we output $k - m$ copies of the integer x.
Correctness: By the correctness of the deterministic select algorithm, there must be at least k elements greater than or equal to x in the array. Furthermore, there must be at most $k - 1$ elements that are strictly greater than x in the array (otherwise, the k-th largest element will be something greater than x). Notice that these elements are among the k largest numbers, and they are what we output in the second step. Therefore, before the last step, we have successfully output the $m \leq k - 1$ largest elements from the array. The remaining $k - m$ largest elements must all be x, which is both the largest element remaining and the $(k - m)$-th largest element remaining. Therefore, this algorithm correctly outputs the k largest elements. (To get full credit, your proof of correctness can be way more handwavy than this. In this particular problem, I would give full credit even for a one-liner.)
Run-time: The deterministic select algorithm from lecture takes linear time. Iterating through the array takes linear time. Outputting $k - m$ copies of x takes $O(k - m) = O(n)$ time. Therefore, the run-time is $O(n)$.

3 Longest Increasing Subsequence (30 points)

Given a sequence of integers of length n, we want to find the length of the longest increasing subsequence, where the elements monotonically increases. For example, if the input sequence is $1, 7, 4, 5, 8, 3, 9, 6, 2$, then $1, 4, 8$ is such a sequence, as well as $7, 8, 9$ and $4, 5, 6$.

(a) What is the length of the longest increasing subsequence for the example $1, 7, 4, 5, 8, 3, 9, 6, 2$? And what is the subsequence that yields this length? (5 points)

(b) Design an $O(n^2)$ algorithm that finds the length of the longest increasing subsequence. You only need to output the length, not the sequence itself. For simplicity, assume all input numbers are distinct, i.e. no repetitions. Justify the correctness and run-time of your proposed algorithm. (25 points)

(a) 5. 1, 4, 5, 8, 9.
(b) Let the sequence be $X = x_1, x_2, \dots, x_n$. For $i = 0, 1, 2, \dots, n$, $j = 1, 2, \dots, n+1$, we define the subproblem $LIS(i, j)$ as the longest increasing subsequence in $x_1, x_2, \dots, x_i$ where the entire subsequence is less than x_j. We have

$$LIS(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ LIS(i-1, j) & \text{if } i \geq 1 \text{ and } x_i \geq x_j \\ \max(LIS(i-1,j), LIS(i-1,i)+1) & \text{if } i \geq 1 \text{ and } x_i < x_j \end{cases}$$

The answer is given by $LIS(n, n+1)$, where we hard-code $x_{n+1} = \infty$. The algorithm is then a memoization/bottom-up implementation of the above DP algorithm.
Correctness: We show optimal substructure and correct base cases. We argue the base case first. If we don't have any elements in the sequence, the LIS for sure is going to be 0. Furthermore, $LIS(i, j)$ depends on $LIS(i-1, j)$ and $LIS(i-1, i)$, so we will eventually reach the base case of $i = 0$. Therefore, we have the correct base case.
Next we argue optimal substructure for $LIS(i, j)$, which should give us the LIS in $x_1, x_2, \dots, x_i$ where the entire LIS is less than x_j. Notice that if $x_i \geq x_j$, then x_i must not be in the LIS. So the LIS must be the LIS in $x_1, x_2, \dots, x_{i-1}$. This corresponds to the second case.
On the other hand, if $x_i < x_j$, there are two possibilities: either x_i is in the LIS, or not.

- If it is not in the LIS, this is the same as before: the LIS must be the LIS in $x_1, x_2, \dots, x_{i-1}$ and gives $LIS(i-1, j)$.
- If x_i is in the LIS, then the remainder of the LIS must be $< x_i$ in order for it to be an increasing subsequence. And since x_i is used, we can only use elements in $x_1, \dots, x_{i-1}$, which gives us $LIS(i-1, i)$, and we add one to it to account for x_i being selected.

Since we want the longest subsequence, we take the max of these two, and this corresponds exactly to the third case. (To get full credit, your proof of correctness can be way more handwavy than this, but it should contain a brief argument about optimal substructure and correct base cases.)
Run-time: Number of subproblems is $(n+1)^2$, and the time per subproblem is $O(1)$. Therefore, with memoization/bottom-up, the run-time of this DP algorithm is $O(n^2)$ as desired.