Dalhousie University
CSCI 3110 — Design and Analysis of Algorithms I
Summer 2016
Midterm Examination
June 23
16:37PM-17:52PM

Please Note: These solutions are for students in the Summer 2016 version of CSCI 3110 only. They may not be photocopied or distributed in any way, including electronically, to any other person without permission of the instructor.

Instructions (Read Carefully):

1. Aids allowed: one 8.5" by 11" piece of paper with anything written or printed on it (both sides). No textbooks, computers, calculators, or other aids.
2. This exam has 8 pages, including this page. Ensure that you have a complete paper.
3. Understanding the exam questions is part of the exam. Therefore, questions will **not** be interpreted. Proctors will confirm or deny any error or ambiguities only. If you are unsure of your own understanding, clearly state reasonable assumptions that will not trivialize the questions.

Problem	**Marks**	**Score**	**Marker**
1	10		
2	15		
3	25		
4	20		
5	15		
6	15		
Total	100		

1. [10 marks] True-false: 5 marks each. No justification necessary.

   (a) The intersection of the set $O(f(n))$ and the set $o(f(n))$ is $o(f(n))$.

   True

   (b) If $f_1(n) = O(f_2(n))$ and $f_2(n) = \Omega(f_3(n))$, then $f_1(n) = \Theta(f_3(n))$.

   False

2. [15 marks] For each of the following recurrences, use the "master theorem" and give the solution using Θ-notation. Simply write down your answer and no justification is necessary.

   If the "master theorem" does not apply to a recurrence, indicate this, but you need not show your reasoning or give a solution.

   (a) $T(n) = T(n/2) + \Theta(1)$

   $\Theta(\lg n)$

   (b) $T(n) = 3T(n/2) + n^2$
   $\Theta(n^2)$

   (c) $T(n) = 3T(n/3) + n/\lg n$

   The master theorem does not apply to this question.

3. [25 marks](Order of growth)

(a) [6 marks] Complete the formal definition of $\omega(g(n))$ by filling in the blank below:

$\omega(g(n)) = \{f(n)$: for any positive constant c, there exists a constant n_0 such that

$$\underline{0 \leq cg(n) < f(n) \text{ for all } n \geq n_0}$$

(b) [9 marks] Place the following functions in the three blanks below so that $f_1(n) = o(f_2(n))$ and $f_2(n) = o(f_3(n))$ (Recall that $\lg n = \log_2 n$):

$$\sqrt{\lg n} \qquad n \qquad \lg \lg n$$

$f_1(n) = \underline{\lg \lg n}$ $f_2(n) = \underline{\sqrt{\lg n}}$ $f_3(n) = \underline{n}$

(c) [10 marks] Prove that $f_2(n) = o(f_3(n))$ using the limit approach.

Hint: l'Hôpital's rule may be helpful.

$$\begin{aligned} \lim_{n\to\infty} \frac{\sqrt{\lg n}}{n} &= \lim_{n\to\infty} \frac{1}{2n(\ln 2)\sqrt{\lg n}} \quad \text{(l'Hôpital)} \\ &= 0. \end{aligned}$$

4. [20 marks] In class we wrote down the pseudocode of insertion sort as an iterative procedure, and analyzed its running time.

   (a) [6 marks] What is the running time of the insertion sort algorithm shown in class? Express your answer using the big-Oh notation and no justification is needed.

   $$O(n^2)$$

   (b) [6 marks] There is a different way of implementing insertion sort: We can express insertion sort as a recursive procedure. To sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Write down a recurrence for the running time of this recursive version of insertion sort, and make sure to cover the base case. No justification is needed.

   $$T(n) = \begin{cases} T(n-1) + n & \text{if } n \geq 2; \\ 1 & \text{Otherwise.} \end{cases} \tag{1}$$

   (c) [8 marks] Solve the recurrence you have written down for question (b). Express the result, i.e., the running time of the algorithm, using big-Oh.

   Hint: the iteration method might be the easiest approach.

   By the recurrence, we have:

   $$\begin{aligned} T(n) &= T(n-1) + n \\ T(n-1) &= T(n-2) + n - 1 \\ &\cdots \\ T(2) &= T(1) + 2 \end{aligned}$$

   Summing up, we have $T(n) = T(1)+2+3+\ldots+n = 1+2+\ldots+n = n(n+1)/2 = O(n^2)$.

5. [15 marks] Let $A[1..n]$ be an array of n distinct integers, and let k be a given integer. Your goal is to find two distinct elements of A whose sum is equal to k, or report that no such two elements exist.

   (a) [7 marks] One $O(n \lg n)$-time solution to this problem is based on reducing to the sorting problem. In this algorithm, we first sort A in increasing order, and then perform additional algorithmic steps that require $O(n)$ time to compute the answer. Describe, in English, how to use $O(n)$ time to compute the answer after the array has been sorted.

   There's no need to give pseudocode, but you can if it makes your explanation clearer.

   Note: For all sub-questions, to receive full marks, the additional computational steps performed after sorting must use $O(n)$ time only. If your solution requires more time after sorting, then up to **9 marks** will be awarded to these sub-questions. If you are unable to make use of sorting to solve this problem at all, then design a brute-force solution, and you may still be awarded up to **half of the total marks**. This policy of awarding partial marks applies to this question only and does not apply to the rest of the exam.

   Solution: After sorting, we keep two indices i and j, initialized to be 1 and n, respectively. If $A[i] + A[j] = k$, then we return $A[i]$ and $A[j]$ as the answer. If $A[i] + A[j] < k$, increase i by 1. If $A[i] + A[j] > k$, decrease j by 1. Repeat this process until either two elements whose sum is equal to k are found, or $i = j$. In the latter case, return that no such two elements exist.

   One solution based on sorting that can also achieve $O(n \lg n)$ overall running time is to look for $k - A[i]$ for each $i \in [1, n]$ using binary search in the array A. This however requires $\Theta(n \lg n)$ time after sorting, so it is worth at most 9 marks as stated in the policy of awarding partial marks.

(b) [5 marks] Justify the correctness of your algorithm.

We first show that when there are no two elements whose sum is equal to k, the algorithm generates the correct output. From the description of the algorithm, we can see that it only returns two elements if their sum is indeed equal to k, so it will not return two elements if no such elements exist.

We next show that when there are pairs of elements whose sum is equal to k, the algorithm correctly computes one such pair. We give a proof by contradiction. Assume to the contrary that there exist i and j, where $i < j$ and $A[i] + A[j] = k$, but the algorithm never evaluates $A[i] + A[j]$ and erroneously reports that no two elements of A sum to k. When this happens, we have $i = j$, and by this time, i and j, collectively, have scanned the entire array. Therefore, at some point, either $i = x$, or $j = y$. Assume without loss of generality that $i = x$ occurs before $j = y$ (the other case is symmetric). At that point, $j > y$ and thus $A[j] > A[y]$. Therefore, at that point, $A[i] + A[j] > A[x] + A[y] = k$. The algorithm will decrease j by 1, and this is repeated until $j = y$. When this happens, $A[i] + A[j] = A[x] + A[y] = k$, and the algorithm will return $A[x]$ and $A[y]$, which contradicts the assumption that the algorithm never returns any pairs of elements.

(c) [3 marks] Analyze the running time briefly and give the result using big-O notation, in terms of n.

Sorting requires $O(n \lg n)$ time. After sorting, the scan requires $O(n)$ steps, as at each step, $j - i$ is decreased by 1. Therefore, the total running time is $O(n \lg n)$.

Note: it is also OK to analyze the $O(n)$-time scan only, as this question does not explicitly state whether you should include the step of sorting.

6. [15 marks] NASA plans to find out the chemical composition of an asteroid. They have already determined a list of n possible constituent substances, and the next step is to shoot lasers at certain frequencies at the asteroid to determine whether each of these substances can possibly exist in the asteroid. Each possible constituent substance, s_i, on this list has a range $[l_i, h_i]$, such that, if the laser is shot at a frequency within this range, substance s_i will react to the laser.

   Your tasks is to choose the smallest set of frequencies $\{f_1, f_2, \ldots, f_k\}$, so that each substance s_i will react to at least one of these frequencies. That is, for each i, there exists at least one j such that $l_i \leq f_j \leq h_i$.

   For example, if there are 4 possible substances on the list and their corresponding ranges are $[5, 10], [27, 35], [20, 30], [25, 40]$, then one optimal solution is choose the set of frequencies $\{5, 28\}$.

   (a) ([3 marks]) For the example given in this question, write down a different optimal solution by giving a minimum set of frequencies.

   One possible solution is $\{5, 27\}$.

   (b) ([4 marks]) The following is one possible greedy strategy: Pick a frequency f that the maximum number of substances react to. Remove the substances that react to f and then repeat, until all substances on the list are covered. The set of frequencies picked in this way is the solution.

   Show that this strategy does not always give an optimal solution by constructing a counterexample. Explain why the solution is not optimal.

   Solution: In the counterexample, the ranges for substances s_1, s_2, s_3, s_4, s_5 and s_6 are $[1, 4]$, $[2, 6]$, $[3, 7]$, $[5, 11]$, $[5, 12]$ and $[10, 15]$, respectively. The greedy algorithm will first pick one frequency from $[5, 6]$, as any value in this range will cover the maximum number of elements (s_2, s_3, s_4 and s_5). After that, two more frequencies are needed to cover s_1 and s_6, respectively. Therefore, 3 frequencies are needed.
   A better solution that uses two frequencies is $\{3, 10\}$.

   (c) [4 marks] Describe, in English, a greedy algorithm that can always find an optimal solution. There is no need to give pseudocode, but you can if it makes your explanation easier. Make your algorithm as efficient as possible. There is some white space on the top of the next page which you can use to write down your answer in addition to using the white space below.

   Let s_i be the substance with the smallest h_i value. Choose frequency $f = h_i$ and remove the substances that react to f. Repeat this process until all substances have been covered.
   To perform this process efficiently, we first sort the list of substances in increasing order of their h_i values. Let substance s_i be the first substance in the sorted list. We choose frequency $f = s_i$. To remove the substances that react to f, we scan

the sorted list starting from s_i and remove substances whose l_i values are less than f, and stop until we encounter the first element whose l_i value is greater than f. We report this process until no substance is left in the list.

(d) [3 marks] Justify the correctness of your algorithm.

Let $G = \{f_1, f_2, \ldots, f_g\}$ be the greedy solution where $f_1 < f_2 < \ldots < f_g$, and $S = \{f'_1, f'_2, \ldots, f'_s\}$ be an optimal solution where $f'_1 < f'_2 < \ldots < f'_s$. Then $s \leq g$. We first show that, if we can prove that for $i = 1, 2, \ldots, g$, $f_i \geq f'_i$ always holds, then we have $g = s$, which means that G is also optimal. Assume to the contrary that $g > s$. Then by the greedy algorithm, f_{s+1} covers at least one substance that is not covered by f_s. Supposed that one such substance has range $[a, b]$. Then, since it is covered by f_{s+1}, we have $b \geq f_{s+1} > f_s \geq f'_s$. Furthermore, as it is not covered by f_s, then $a > f_s \geq f'_s$. Therefore, both a and b are strictly greater than f'_s, which means that this substance does not react to any frequency in S, which contradicts the fact that S is a solution.

We now prove that for $i = 1, 2, \ldots, g$, $f_i \geq f'_i$ always holds, by induction on i. For the base case, f_1 is the minimum h_i value among all substances. As the substance with the minimum h_i value must react to at least one frequency in S, there is at least one frequency in S that is no greater than f_1. As f'_1 is the smallest frequency in S, we have $f'_1 \leq f_1$.

In the inductive case, assume that the claim holds for i, and we now prove it for $i + 1$. By the greedy algorithm, f_{i+1} is the h_i value of a substance, s, that is not covered by $f_1, \ldots, f_i$. Let $[c, d]$ be the range of s. Then $d = f_{i+1}$ and $c > f_i \geq f'_i$. Therefore, s does not react to frequencies $f'_1, f'_2, \ldots, f'_i$. Let f'_j be a frequency in S that s reacts to. Then $j \geq i + 1$, so $f'_{i+1} \leq f'_j \leq d = f_{i+1}$, and the induction goes through.

(e) [1 marks] Analyze the running time briefly and give the result using big-O notation, in terms of n.

Sorting requires $O(n \lg n)$ time. The remaining steps require only a linear scan through the sorted sequence, which requires $O(n)$ time. Therefore, the total running time is $O(n \lg n)$.