Midterm 1 solutions

Please do not read or discuss these solutions in the exam room while others are still taking the exam.
Problem 1. [True or false] (16 points)

Circle TRUE or FALSE. Do not justify your answers on this problem.

(a) TRUE or FALSE: If \( f(n) = (n + 1)/2 \), then \( f(n) \in O(n^2) \).

(b) TRUE or FALSE: If \( f(n) = (n + 1)/2 \), then \( f(n) \in \Theta(n^2) \).

(c) TRUE or FALSE: If \( f(n) = (n + 1)/2 \), then \( f(n) \in \Theta(n^3) \).

(d) TRUE or FALSE: \( n^{1.1} \in O(n \log n^2) \).

(e) TRUE or FALSE: It’s possible to multiply two \( n \)-bit integers in \( O(n^{1.9}) \) time.

Comment: We saw that it was possible to multiply two \( n \)-bit integers in \( O(n^{1.59}) \) time, using a divide-and-conquer algorithm. So it’s certainly possible to multiply them in \( O(n^{1.9}) \) time: just multiply using the divide-and-conquer algorithm, then execute no-op instructions to mark time until the total running time is \( O(n^{1.9}) \).

(f) TRUE or FALSE: If vertices \( u, v \) are in the same strongly connected component of a directed graph \( G \), then it is necessarily the case that \( v \) is reachable from \( u \) in \( G \).

(g) TRUE or FALSE: If vertices \( u, v \) are not in the same strongly connected component of a directed graph \( G \), then it is necessarily the case that \( v \) is not reachable from \( u \) in \( G \).

Comment: It’s possible that \( u, v \) are in two different SCCs and there is an edge from \( u \) to \( v \).

(h) TRUE or FALSE: If we run breadth-first search on a directed graph \( G \) starting from vertex \( s \), then depending on the graph, it might visit some vertices that a depth-first search starting from \( s \) would not visit. (Assume that we use exactly the breadth-first search algorithm specified in the book.)

Problem 2. [Recurrences] (16 points)

Write the solution to the following recurrences. Express your answer using \( O(\cdot) \) notation. Do not justify your answers on this problem. Do not show your work.

(a) Solve the recurrence \( F(n) = F(\lceil n/2 \rceil) + O(1) \).

Answer: \( F(n) = O(\log n) \).

(b) Solve the recurrence \( F(n) = 4F(\lceil n/2 \rceil) + O(1) \).

Answer: \( F(n) = O(n^2) \).

(c) Solve the recurrence \( F(n) = 4F(\lceil n/2 \rceil) + O(n) \).

Answer: \( F(n) = O(n^2) \).

(d) Solve the recurrence \( F(n) = 4F(\lceil n/2 \rceil) + O(n^2) \).

Answer: \( F(n) = O(n^2 \log n) \).

Comment: All of these can be solved using the Master theorem, or by drawing a tree.
Problem 3. [Three-way mergesort] (18 points)

Alice suggests the following variant on mergesort: instead of splitting the list into two halves, we split it into three thirds. Then we recursively sort each third and merge them.

Mergesort3(A[0..n−1]):
1. If \( n \leq 1 \), then return \( A[0..n−1] \).
2. Let \( k := \lceil n/3 \rceil \) and \( m := \lceil 2n/3 \rceil \).
3. Return Merge3(Mergesort3(A[0..k−1]),Mergesort3(A[k..m−1]),Mergesort3(A[m..n−1])).

Merge3(L0,L1,L2):
1. Return Merge(L0,Merge(L1,L2)).

Assume that you have a subroutine Merge that merges two sorted lists of lengths \( \ell, \ell' \) in time \( O(\ell + \ell') \). You may assume that \( n \) is a power of three, if you wish. Do not justify your answers on this problem. Do not show your work.

(a) What is the asymptotic running time for executing \( \text{Merge3}(L_0, L_1, L_2) \), if \( L_0, L_1, \) and \( L_2 \) are three sorted lists each of length \( n/3 \)? Express your answer using \( O(\cdot) \) notation.

**Answer:** \( O(n) \).

**Comment:** The running time is \( \frac{n}{3} + \frac{n}{3} \) for the call to \( \text{Merge}(L_1, L_2) \) and \( \frac{n}{3} + \frac{2n}{3} \) for the outer call, for a total of \( \frac{5n}{3} \), which is in \( O(n) \).

(b) Let \( T(n) \) denote the running time of Mergesort3 on an array of size \( n \). Write a recurrence relation for \( T(n) \).

**Answer:** \( T(n) = 3T(n/3) + O(n) \).

**Comment:** There are three recursive calls to Mergesort3, each on a list of size \( n/3 \), followed by a call to \( \text{Merge3} \), which takes \( O(n) \) time by part (a).

(c) Solve the recurrence relation in part (b). Express your answer using \( O(\cdot) \) notation.

**Answer:** \( T(n) = O(n \log n) \).

(d) Is the Mergesort3 algorithm asymptotically faster than insertion sort? Circle \[YES\] or \[NO\].

**Comment:** Insertion sort runs in \( O(n^2) \) time, which is asymptotically slower.

(e) Is the Mergesort3 algorithm asymptotically faster than the ordinary mergesort? Circle \[YES\] or \[NO\].

**Comment:** Ordinary mergesort runs in \( O(n \log n) \) time, which is asymptotically no faster or slower than Mergesort3.

Problem 4. [Algorithm design] (12 points)

Suppose we have \( t \) sorted arrays, each with \( n \) elements, and we want to merge them to get a single sorted array with \( tn \) elements. Your task is to fill in the blanks below to get an algorithm, ManyMerge, that solves this problem and has a running time of \( O(nt \log t) \).

You may assume that you are given an algorithm, \( \text{Merge}(L, L') \), that merges two sorted lists of size \( \ell, \ell' \) into a single sorted list of size \( \ell + \ell' \). You may assume that it runs in \( O(\ell + \ell') \) time.

Fill in the empty boxes below, to get a correct algorithm whose running time is \( O(nt \log t) \).
ManyMerge($L_1[0..n-1], L_2[0..n-1], \ldots, L_t[0..n-1]$):
1. If $t = 1$, then return $L_1$.
2. If $t = 2$, then return Merge($L_1, L_2$).
3. Set $L := \text{ManyMerge}(L_1, L_2, \ldots, L_{\lfloor t/2 \rfloor})$.
4. Set $L' := \text{ManyMerge}(L_{\lfloor t/2 \rfloor+1}, \ldots, L_{t-1}, L_t)$.
5. Return Merge($L, L'$).

Comment: The running time of this solution satisfies the recurrence $T(t) = 2T(t/2) + O(nt)$: we recursively invoke ManyMerge twice, in each case with $t/2$ sorted lists of size $n$; and then we merge two lists of size $nt/2$, which takes $O(nt)$ time. The solution to this recurrence is $T(t) = O(nt \log t)$, so the algorithm shown here does indeed achieve the necessary time bound. It is not hard to see why it is correct.

Problem 5. [Depth-first search] (16 points)

Run depth-first search on the directed graph below, starting at vertex A. Whenever there is a choice of the order to explore vertices, use alphabetical order (so A is chosen before B, and B before C, etc.).

(a) Draw the DFS tree that results, in the space provided below. Use solid lines for tree edges, and dotted lines for non-tree edges.

(b) Label each non-tree edge in the graph above with “forward”, “back”, or “cross”, according to whether the edge is a forward edge, back edge, or cross edge.

(c) Can the above graph be topologically sorted? Circle YES or NO. Do not justify your answer.
(d) How many strongly connected components does this graph have? Do not justify your answer.

Answer: 3.

Comment: The three strongly connected components are: \{A\}, \{E\}, and \{B,C,D\}.

Problem 6. [Short answer] (22 points)

Answer each question below concisely (one short sentence or a number should suffice). Do not justify your answer. Do not show your work.

(a) Suppose we are given a directed graph \( G = (V,E) \) represented in adjacency list format, and we want to test whether \( G \) is a dag or not, using a method that is as asymptotically efficient as possible. In a sentence, what approach would you use?

Answer 1: Use DFS, check for back-edges.

Answer 2: Decompose into strongly connected components, check for a SCC with more than one vertex.

Comment: There is a cycle (and hence \( G \) fails to be a dag) if and only if DFS finds a back edge. There is a cycle (and hence \( G \) fails to be a dag) if and only if there is a strongly connected component with more than one vertex.

(b) What’s the running time of your solution in (a), using \( O(\cdot) \) notation?

Answer: \( O(\lvert V \rvert + \lvert E \rvert) \).

(c) Let \( G = (V,E) \) be a directed graph with \( \lvert V \rvert = 1000 \) vertices, \( \lvert E \rvert = 5000 \) edges, and 700 strongly connected components. How many vertices does the metagraph have?

Answer: 700.

Comment: Each vertex in the metagraph corresponds to a strongly connected component in \( G \), so the number of vertices in the metagraph is the same as the number of SCCs in \( G \).

(d) Let \( G = (V,E) \) be a dag, where each edge is annotated with some positive length. Let \( s \) be a source vertex in \( G \). Suppose we run Dijkstra’s algorithm to compute the distance from \( s \) to each vertex \( v \in V \), and then order the vertices in increasing order of their distance from \( s \). Are we guaranteed that this is a valid topological sort of \( G \)? Circle YES or NO.

Answer: C

Comment: Sorting by increasing distance gives \( S,C,B \); but \( B \) must precede \( C \) in any valid topological sort.
(f) Suppose we run Dijkstra’s algorithm on a graph with \( n \) vertices and \( O(n \lg n) \) edges. Assume the graph is represented in adjacency list representation. What’s the asymptotic running time of Dijkstra’s algorithm, in this case, if we use a binary heap for our priority queue? Express your answer as a function of \( n \), and use \( O(\cdot) \) notation.

Answer: \( O(n \lg n)^2 \).

Comment: \(|V| = n, |E| = O(n \lg n)\), and Dijkstra’s runs in \( O((|V| + |E|) \lg |V|) \) time, which is \( O((n + O(n \lg n)) \lg n) = O((n \lg n) \times \lg n) = O(n(\lg n)^2) \).