EECS 336: Introduction to Algorithms **Sample Midterm**
80 minutes

Advice: Skip problems that might take a while and come back to them later. If an algorithm is requested, be as succinct as possible. Often a simple description in English is more effective than pseudocode. You do not need to supply any proofs unless explicitly asked. All runtimes should be given using big-oh notation.

1. Simplify the following expressions if possible. [Points: ______/12]

   (a) $O(n^2 + 500n)$. $O(n^2)$
   (b) $O(n \log n + n^{1.01})$. $O(n^{1.01})$
   (c) $O(3n^3 - 2n^2 + n)$. $O(n^3)$
   (d) $O(\sqrt{n} + \log n)$. $O(\sqrt{n})$
   (e) $O(1 + 1/n)$. $O(1)$
   (f) $O(\log(n^2))$. $O(\log n)$

2. The Mergesort algorithm described in class works as follows on an unordered list U of n numbers:

   - If $n = 1$, return U (it is already sorted).
   - Break U into two lists U_1 and U_2 (of roughly the same length).
   - Recursively sort: $S_1 = \text{Mergesort}(U_1)$ and $S_2 = \text{Mergesort}(U_2)$.
   - Output merger: $S = \text{Merge}(S_1, S_2)$

   Answer the following questions: [Points: ______/2]

   (a) Give the recurrence relationship that describes the runtime of Mergesort.

   $T(n) = 2T(n/2) + O(n)$

   (b) Give the runtime of Mergesort using big-oh notation.

   $O(n \log n)$

3. Define 3-Mergesort to break the list into three sublists of length roughly $n/3$, recursively sort these sublists, and then 3-Merge them together. [Points: ______/8]

   (a) Give an algorithm for 3-Merge.

   Return Merge$(S_1,$ *Merge*$(S_2, S_3))$

   (b) Give the runtime for 3-Merge using big-oh notation.

   $O(n)$

   (c) Give the recurrence relationship that describes the runtime of 3-Mergesort.

   $T(n) = 3T(n/3) + O(n)$

   (d) Give the runtime for 3-Mergesort using big-oh notation.

   $O(n \log n)$

4. Define k-Mergesort to break the list into k sublists of length roughly n/k, recursively sort these sublists, and then k-Merge them together. [Points: ______/6]

   (a) Assume that k-Merge (on lists of size n/k) can be implemented in $\Theta(n \log k)$ and give the recurrence relationship for the runtime of k-Mergesort.

   $T(n) = kT(n/k) + \Theta(n \log k)$

   (b) Give the runtime for k-Mergesort using big-oh notation.

   - $T(n) = \log_k n \times n \times \log k$
   - $\log_k n = \frac{\log n}{\log k}$
   - $\Rightarrow O(n \log n)$

   (c) Give an algorithm for k-Merge that runs in $O(n \log k)$ time. (Hint: use priority queues.)

   i. *$S = \emptyset$.*
   ii. *Insert each list into priority queue Q with first element as key.*
   iii. *$L = DeleteMin(Q)$.*
   iv. *Append first element of L to S.*
   v. *Insert list of remaining elements of L into Q with first element as key.*
   vi. *Repeat iii.*

5. You are given a graph $G(V, E)$ with edge costs $c(e)$ for $e \in E$. You may assume that the edge costs $c(\cdot)$ are distinct. Let $T \subset E$ be the minimum spanning tree. Consider what happens to the tree if we change the cost of an edge e'. Formally, the the new edge costs are $c'(\cdot)$ given by $c'(e') \neq c(e')$ and $c'(e) = c(e)$ if $e \neq e'$. You may assume that the edge costs $c'(\cdot)$ are each distinct. Let T' be the minimum spanning tree with respect to costs $c'(\cdot)$. [Points: ______/10]

   (a) Consider decreasing the cost of an edge in T (i.e., $e' \in T$ and $c'(e') < c(e')$). Is $T' = T$ always?

   Yes.

   (b) Consider increasing the cost of an edge in T (i.e., $e' \in T$ and $c'(e') > c(e')$). Is $T' = T$ always?

   Not always.

   (c) Consider decreasing the cost of an edge not in T (i.e., $e' \notin T$ and $c'(e') < c(e')$). Is $T' = T$ always? If not, give an illustrative example that shows why and give a simple algorithm for computing T' from T without solving the MST problem over from scratch.

   Not always. To compute T', consider adding edge e' to T. Since T is a spanning tree, adding e' creates a cycle. The most expensive edge in any cycle is not in any spanning tree. Therefore, we can break this cycle by removing the most expensive edge. The resulting graph is a spanning tree.

   (d) Consider increasing the cost of an edge not in T (i.e., $e' \notin T$ and $c'(e') > c(e')$). Is $T' = T$ always? If not, give an illustrative example that shows why and give a simple algorithm for computing T' from T without solving the MST problem over from scratch.

   Yes.

6. You've just started consulting for a startup company, DigiDyne, that is doing dynamic pricing of digital music downloads. They are considering two business models. In the *subscription* model customers are asked to pay a fixed price p and can download as many songs as they please. In the *a-la-carte* model a customer is asked to pay a fixed price q per song. The price for d downloads in the a-la-carte model is $d \cdot q$.

   DigiDyne has done some market research that suggests that each consumer behaves in the following way. Consumer i wishes to download d_i songs and pay at most v_i for the privilege. If the total price consumer i is asked to pay for d_i downloads is at most v_i, they will pay the asked price. If the total price consumer i is asked to pay is more than v_i, they will not pay for any service (Perhaps they will use a competing service instead). Thus, the input to DigiDyne's pricing problem is completely specified by two n-dimensional vectors, $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{d} = (d_1, \ldots, d_n)$. [Points: ______/8]

   Example:

   - $S = \{1, 2, 3\}$, $\mathbf{v} = (4, 5, 6)$, $\mathbf{d} = (1, 2, 3)$.
   - For subscription price $p = 5$: consumers 2 and 3 buy (v_2 and v_3 are greater than $p = 5$), consumer 1 does not buy ($v_1 < p = 5$). The total revenue is 10.
   - For a-la-carte price $q = 3$: consumer 1 buys ($v_1/d_1 \geq q = 3$), consumers 2 and 3 do not buy (v_2/d_2 and $v_3/d_3 < q = 3$). The total revenue is 3 (consumer 1 buys one song for $q = 3$).

   (a) Show that neither of these business models is always better than the other. To do so, give an input $(\mathbf{v}, \mathbf{d})$ where the revenue from the optimal subscription price, p^*, is less than the revenue from the optimal a-la-carte price, q^*. Then given an input $(\mathbf{v}', \mathbf{d}')$ where the revenue from the optimal subscription price, p^*, is more than the revenue from the optimal a-la-carte price, q^*.

   - *For $\mathbf{v} = (1, 3)$ and $\mathbf{d} = (1, 3)$, the optimal subscription price is $p^* = 3$ which sells to consumer 2 for a revenue of 3. The optimal a-la-carte price is $q^* = 1$ that sells to both consumers for a revenue of 4. The a-la-carte revenue is higher.*
   - *For $\mathbf{v}' = (3, 3)$ and $\mathbf{d}' = (1, 3)$, the optimal subscription price is $p^* = 3$ which sells to both consumers for a revenue of 6. The optimal a-la-carte price is $q^* = 1$ that sells to both consumers for a revenue of 4. The subscription revenue is higher.*

   (b) Give an algorithm that on input $(\mathbf{v}, \mathbf{d})$ computes the a-la-carte price, q^*, with the highest total revenue. What is the runtime of your algorithm?

   Algorithm:

   i. *Sort the consumers by decreasing value-per-song, v_i/d_i.*
   ii. *Notice that at a-la-carte price $q = v_i/d_i$, the first i consumers will buy.*
   iii. *Compute $D_i = \sum_{j=1}^{i} d_i$ for all i.*
   iv. *Compute $P_i = D_i v_i/d_i$ for all i.*
   v. *For $i^* = \operatorname{argmax}_i P_i$ output price $q^* = v_{i^*}/d_{i^*}$.*

   Analysis: Step 1: $O(n \log n)$; Steps 3-5: $O(n)$ each; Total: $O(n \log n)$.