Sample exam (with answers)

- Do not open this quiz booklet until you are directed to do so.
- Quiz ends at 4:30pm. It has 4 problems with multiple parts. You have 180 minutes to earn 90 points. Plan your time wisely.
- This final exam is open-book, open-notes. You may not use other books, bibles, or photocopies.
- When the quiz begins, write your name on the top of every page in this quiz booklet, because the pages will be separated for grading.
- When describing an algorithm, **please** describe the main idea in English. Use pseudocode only to the extent that it helps clarify the main ideas.
- Good luck!

Problem	Points	Grades	Total
1	??		
2	??		
3	??		
4	??		

Name: ______________________________

Please circle your TA's name and recitation:

Brian Jon Josh Rachel Yoav

10am 11am 12pm 1pm 2pm

Problem 1. Short questions

In the following questions, fill out the blank boxes. In case more than one answer is correct, you should provide the **best known** correct answer. E.g., for a question "Sorting of n elements in the comparison-based model can be done in $\boxed{}$ time", the best correct answer is $O(n \log n)$. No partial credit will be given for correct but suboptimal answers. However, you **do not** have to provide any justification.

(a) Consider a modification to QUICKSORT, such that each time PARTITION is called, the median of the partitioned array is found (using the SELECT algorithm) and used as a pivot.

The worst-case running time of this algorithm is: $\boxed{}$

Answer: $\Theta(n \log n)$. Reason: the median can be found in $O(n)$ time, so we have the running time recurrence $T(n) = 2T(n/2) + O(n)$.

(b) If a data structure supports an operation `foo` such that a sequence of n `foo`'s takes $\Theta(n \log n)$ time to perform in the worst case, then the amortized time of a `foo` operation is $\Theta\left(\boxed{}\right)$, while the actual time of a single `foo` operation could be as high as $\Theta\left(\boxed{}\right)$.

Answer: Amortized time $\Theta(\log n)$, worst-case time $\Theta(n \log n)$.

(c) Does there exist a polynomial time algorithm that finds the value of an $s-t$ minimum cut in a directed graph? $\boxed{}$ (Yes or No)

Answer: Yes. Reason: The value of an $s-t$ minimum cut is equal to the value of the $s-t$ maximum flow, and there are many algorithms for computing $s-t$ maximum flows in polynomial time (e.g. the Edmonds-Karp algorithm).

Problem 2. Typesetting

Suppose we would like to neatly typeset a text. The input is a sequence of n words of lengths $l_1, l_2, \cdots, l_n$ (measured in the number of fixed-size characters they take). Each line can hold at most P characters, the text is left-aligned, and words cannot be split between lines. If a line contains words from i to j (inclusive) then the number of spaces at the end of the line is $s = P - \sum_{k=i}^{j} l_k - (j-i)$. We would like to typeset the text so as to avoid large white spaces at the end of lines; formally, we would like to minimize the sum over all lines of the square of the number of white spaces at the end of the line. Give an efficient algorithm for this problem. What is its running time?

Answer:

We solve the problem with dynamic programming. Let $A[j]$ denote the optimal "cost" (that is, the sum of the square of number of trailing white space characters over all lines) one may achieve by typesetting only the words $1 \ldots j$ (ignoring the remaining words). We can express $A[j]$ recursively as follows:

$$A[j] = \min_{i<j : T[j]-T[i] \leq P} A[i] + (P - (T[j] - T[i]))^2,$$

where $T[j] = \sum_{i=1}^{j} l_i$. A table of the values $T[1 \ldots n]$ can be initially computed in $O(n)$ time. The equation above says the following: in order to optimally typeset words $1 \ldots j$, we must first optimally typeset words $1 \ldots i$ for some $i < j$, and then place the remaining words $i+1 \ldots j$ on the final line.

Using dynamic programming, we compute each value $A[j]$ in sequence for all $j = 1 \ldots n$. Each value is requires $O(n)$ time to compute, for a total running time of $O(n^2)$. After termination, $A[n]$ will contain the value of the optimal solution; we can reconstruct the solution itself (that is, the locations where we should insert line breaks) by maintaining "backpointers" as is usually done with dynamic programming.

Problem 3. Flows

Let G be a flow network with integer capacities, and let f be an integer maximum flow in G. Suppose that we increase the capacity of an arbitrary edge in G by one unit. Describe an efficient algorithm to find a maximum flow in the modified flow network. Analyze the worst-case asymptotic running time. Explain why your algorithm is correct.

Answer:

I. Algorithm:

1. Compute the residual network G'_f from the new graph G' and flow f.
2. Run DFS (or BFS) on the residual network G'_f to find an augmenting path.
3. If an augmenting path is found, increase the flow on the augmenting path by one unit to obtain a new flow f' which is the maximum flow for G'.

II. Analysis

step 1	$O(E+V)$
step 2	$O(E+V)$
step 3	$O(V)$
Running time =	$O(E+V)$

III. Correctness

Since G' has integer capacities and f is an integer max flow, we can augment at most one unit of flow along some path in G'_f, so one pass of BFS is enough. The rest of the correctness argument is similar to the one for the Ford-Fulkerson algorithm.

Problem 4. Princess of Algorithmia (2 parts)

Political upheaval in Algorithmia has forced Princess Michelle X. Goewomans to vacate her royal palace; she plans to relocate to a farm in Nebraska. The princess wants to move all of her possessions from the palace in Algorithmia to the Nebraskan farm using as few trips as possible in her Ferrari. For simplicity, let us assume that the princess's Ferrari has size 1, and that her n possessions $x_1, x_2, \ldots, x_n$ have real number sizes between 0 and 1. The problem is to divide the princess's possessions into as few carloads as possible, so that all her possessions will be transported from Algorithmia to Nebraska and so that her Ferrari will never be overpacked. It turns out this problem is NP-hard.

Consider the following **first-fit** approximation algorithm. Put item x_1 in the first carload. Then, for $i = 2, 3, \ldots, n$, put x_i in the first carload that has room for it (or start a new carload if necessary). For example, if $x_1 = 0.2$, $x_2 = 0.4$, $x_3 = 0.6$, and $x_4 = 0.3$, the first-fit algorithm would place x_1 and x_2 in the first carload, x_3 in the second carload, and then x_4 in the first carload (where there is still enough room). Note that all decisions are made offline; we divide the princess's n possessions into carloads before any trips have actually been made.

(a) The princess's charming husband, Craig, has asserted, "The first-fit algorithm will always minimize the total number of car trips needed." Give a counterexample to Craig's claim.

Answer:

Let $n = 4$ and let $x_1 = 0.3$, $x_2 = 0.8$, $x_3 = 0.2$, and $x_4 = 0.7$. The optimal number of car trips needed is 2, while the first-fit algorithm produces 3 car trips.

(b) Prove that the first-fit algorithm has a ratio bound of 2. (*Hint:* How many carloads can be less than half full?)

Answer:

Consider a carload u of size p which is less than half full, i.e., $p < 0.5$. Let x_i be the first item to go into the next carload v. Then, due to the nature of the first-fit algorithm, it follows that $p + x_i > 1$ and $x_i > 0.5$. Hence, carload v is more than half full. For calculation purposes, we can transfer a portion of the load of item x_i from carload v to carload u to make carload u exactly half full while keeping carload v at least half full. This is because $0.5 - p < x_i - 0.5$. After we perform this operation, every carload is at least half full. Thus, if h is the number of carloads produced by the first-fit algorithm, then $\sum_{i=1}^{n} x_i \geq 0.5h$. Note that the optimal number of carloads OPT is at least $\sum_{i=1}^{n} x_i$. Hence, $h/OPT \leq 2$, which proves that the first-fit algorithm has a ratio bound of 2.