Please read the entire exam carefully before you start. Plan your time, and prepare any questions for when the teacher comes to the exam room.

* * *

Make assumptions where necessary. Write your answers where indicated, and keep them short and concise, if possible. Long explanations that do not directly answer the questions are given little or no weight.

Unless a given problem states otherwise, you may describe your algorithms in prose, pseudocode or program code, according to preference, as long as it is clear how the algorithm works. Short, abstract explanations may be just as good as extensive pseudocode, as long as they are clear and precise.

Your algorithms should be as efficient as possible, unless otherwise stated. Running times are to be given in asymptotic notation, as precisely as possible.

The maximum score attainable is indicated for each problem. The maximum total score attainable is 100 points.

Note: In Problem 2b, the notation $v.d$ follows the most recent edition of the textbook. This is equivalent to the notation $d[v]$ used in previous editions.

Problem 1

a) Which problem does Dijkstra's algorithm solve?

Answer (5 pts): Shortest path.
Note: "Single-source shortest path with non-negative weights" is more precise, but not required here.

b) Which traversal algorithm do we use for topological sorting?

Answer (5 pts): Depth-first search (or simply DFS).
Note: BFS will not work. It is possible to implement topological sorting without traversal, however, iteratively removing nodes without in-edges; using a FIFO queue for this algorithm could be seen as a "relative" to BFS, and so an answer of BFS without further explanation will give 1 point.

c) Which design method would you use to design an algorithm to fully parenthesize a matrix-chain multiplication?

Answer (5 pts): Dynamic programming (or simply DP).
Note: Some students asked what a "design method" was and were given general hints about there being just a handful in the curriculum. (Note also that "divide and conquer" is described as a design method in problem e, below.) The term in the textbook is "design technique," but if one knows the material on matrix-chain multiplication, it should be clear that it is based on dynamic programming.

d) What does Prim's algorithm keep in its min-priority queue?

Answer (5 pts): Nodes.
Note: It may be natural to answer edges here, if one does not remember the details from the curriculum. This would lead to an alternate implementation, without the use of *decrease-key*, but with duplicate nodes and associated edges, and with duplicate detection on removal from the queue. This is not precisely what we're asking, but it's rather close, and is a variation of the algorithm that behaves correctly, so this merits 4 points.

e) What is the typical average-case running time for sorting algorithm based on the divide-and-conquer design method?

Answer (5 pts): $\Theta(n \log n)$
Note: Though precision is generally required in the use of asymptotic notation, for this problem, the less precise $O(n \log n)$ will also give full marks.

Problem 2

a) Which design method would you use in designing an algorithm for the fractional knapsack problem?

Answer (5 pts): Greed (or "the greedy strategy" or the like).
Note: See earlier note about the use of the term "design method." Technically, one could solve this problem using (redundant, slow) dynamic programming, but the point of this problem was to test whether the student had mastered the relevant part of the curriculum, where a greedy solution to this problem is discussed. Thus, an answer of dynamic programming will not give any points.

b) During the execution of some shortest-path algorithm, $u.d = 5$, $v.d = 7$, $w(u, v) = 1$ and $w(v, u) = -2$. After one call to $\textsc{Relax}(u, v, w)$ and a subsequent call to $\textsc{Relax}(v, u, w)$, what is the value of $u.d$?

Answer (5 pts): 4
Note: A possible misunderstanding here is that of u being "finished" after the first call, and thus not being modified by the second. This will result in the answer 5, which will be marked with 1 point.

c) If $a \cdot \varphi(i) \leq b \cdot \psi(i)$ for every $i \geq k$, what can you say about the asymptotic relationship between φ and ψ? (Here φ and ψ are non-negative real-valued functions, defined on the positive integers, a and b are positive real-valued constants and k is a positive integer constant.)

Answer (5 pts): $\varphi(i) = O(\psi(i))$ or, equivalently, $\psi(i) = \Omega(\varphi(i))$.
Note: For the first version, note that we can let the constants from the definition of O be $c = b/c$ and $n_0 = k$.

d) Very briefly, what characterizes a good hash function?

Answer (5 pts): It looks "uniformly random": Each key is approximately equally likely to hash to any slot, independently of the other keys.
Note: Here other similar answers can also give full marks, as long as they describe what characterizes a good hash function.

e) You are examining an unfamiliar problem A, and you wish to relate it to problem B, which you know has a running time of $\Omega(n^2)$ in the worst case. You wish to show that the same bound holds for A, using a reduction. Very briefly, what can you say about this reduction?

Answer (5 pts): It must be a reduction from B to A, and it must have a running time of less than $\Omega(n^2)$ in the worst case.
Note: Simply noting that the reduction must be from B to A gives 3 points, as this is not enough to prove the desired result. Additionally pointing out that it should be "easy" (without specifying what that means) give 4 points. Stating that it should be polynomial does not affect the score. Stating that the reduction must be $O(1)$ or $O(n)$ or some other specific running time lower than quadratic, while technically too strict, will still give full marks.

Problem 3

You wish to sort the sequence $A = (a_1, a_2, \ldots, a_n)$. It is well-known that for comparison-based sorting, the best running time achievable, in the expected and worst cases, is $\Omega(n \log n)$.

a) Assume that the elements are real numbers, distributed according some given probability distribution, which can be computed in constant time for any element. What is the best running time you could get, in the average case, and how would you get it? Explain your reasoning briefly.

Answer (5 pts):
This is Exercise 8.4-5 from Cormen, from the chapter on bucket sort.
Linear time, using bucket sort, creating buckets for the probabilities. The width of the intervals will then be proportional to the number of elements that land in them, i.e., constant.
Note: The idea here is based on a *cumulative* probability distribution function. For a probability density function, setting up a general bucket sort based algorithm would not necessarily be feasible, without knowing which distribution is used (which would let you find the appropriate buckets). Because this may not have been entirely clear, any answer indicating the use of bucket sort, even pointing out that it cannot be used because the distribution is not linear, will get full marks.

b) Now assume that the elements are positive integers, and that $a_i \leq p(n)$ for $i = 1 \ldots n$, where p is some polynomial. What is the best running time you could get, in the worst case, and how would you get it? Explain your reasoning briefly.

Answer (5 pts):
C.f., Exercise 8.3-4 in Cormen (for the example where $p(n) = n^3 - 1$).
Linear time, using radix sort. Representing the numbers in the base-n system will give us at least $p(n)$ possible values with k digits, for some constant k, where $p(n) \leq n^k$. The running time is then $\Theta(kn) = \Theta(n)$.
Note: Using counting sort will give a running time of $\Theta(p(n))$, which for powers greater than two is significantly worse, in the general case, than any of the other sorting algorithms in the curriculum. This, therefore, will not be given any points.

Problem 4

a) Which problem does Floyd-Warshall solve? What assumption must you or can you make, and how do they affect the running time? Explain briefly.

Answer (5 pts):
The all-pairs shortest path problem. Its running time is $\Theta(n^3)$, for n nodes. We assume there are no negative cycles.
Note: There is no requirement here to provide the running time. We assume that there are no negative cycles because otherwise the algorithm won't work. Stating the problem it solves and this assumption, and the reason for it, is enough for full marks. Stating only the problem yields 3 points.

b) Which problem does Ford-Fulkerson solve? What assumption must you or can you make, and how do they affect the running time? Explain briefly.

Answer (5 pts):
The maximum flow problem. If we assume integral (or scaled rational) capacities, the running time is $O(mf)$, for m edges and a maximum flow of f. If we have irrational capacities, it might never terminate. If we assume the augmenting paths are chosen according to breadth-first search (the Edmonds-Karp algorithm), the running time is $\Theta(nm^2)$.
Note: For full marks, one should state which problem it solves and either indicate the integral capacity assumption (with appropriate running time) or the breadth-first search assumption (with appropriate running time) or both. Stating only the problem yields 3 points.

Problem 5

a) Solve the recurrence $T(n) = 2T(n/2) + T(n)/2 + n$, where $T(1) = 1$. Give your answer in asymptotic notation.

Answer (5 pts):
$T(n) = \Theta(n^2)$
Explanation (not required as part of the answer):
$2T(n) = 4T(n/2) + T(n) + 2n$
$T(n) = 4T(n/2) + 2n$
$[n^{\log_b a} = n^2; f(n) = 2n = O(n^{2-\epsilon})]$
$T(n) = \Theta(n^2)$

b) An algorithm processes an image in the form of an $n \times n$ matrix of pixels, by splitting it up into non-overlapping squares of size $(n/2) \times (n/2)$, processing these recursively, and then combining the results in linear time, as a function of the number of pixels involved. What is the total running time, as a function of n?

Answer (5 pts):
$T(n) = \Theta(n^2 \log n)$
This follows from the recurrence $T(n) = 4T(n/2) + n^2$ (not required as part of the answer).
Note: Some students asked what "linear time, as a function of the number of pixels involved" meant. I.e., if it should be $\Theta(n)$ as a function of n^2. As was explained to those who asked, this was not the intended meaning. "Linear as a function of" is here to be taken to mean "an asymptotically linear function of" – i.e., if there are m pixels, it should be $\Theta(m)$. In this case, this means $\Theta(n^2)$. Being able to understand the distinction between being linear in n and in the number of pixels was part of the problem. Thus misunderstanding this, and getting a quadratic running time, will yield only 1 point. Because the problem does not explicitly state that asymptotic notation should be used (even though this is said on the first page of the exam), some students have supplied exact answers. Even though this makes little sense (as the time unit and constants are not known), such answers will still be accepted.

Problem 6

You are planning to plant gardens and dig irrigation canals from some given water source. You can decide the layout yourself, and have decided on the following. Starting at the source, the canals form a binary tree structure, with the gardens acting as leaves. The distancing is completely regular. The distance along a canal from the source to a fork, or from a fork to a leaf or another fork is the same, and is set to $\ell(n)$ for a given number of gardens, n.

a) What is the total length of canal that must be dug, as a function of n?

Answer (5 pts):
The number of leaves is n, so the number of internal nodes is $n-1$. All of these except the water source will have a unique incoming canal segment of length $l(n)$, so the total length is $L(n) = 2\ell(n) \cdot (n-1)$.
Note: Several students asked whether one could have an internal node with a single child. As this would be a "fork" with no actual forking, this is not possible. However, there is no automatic reason why the source itself should necessarily have two children (e.g., if there is a single garden). Thus, an answer stating that the result will be either $2\ell(n) \cdot (n-1)$ or $2\ell(n) \cdot (n-1) + 1$, depending on the arity of the root (or that we can't know which, in the general case) will be given full marks. An answer giving only the latter formula, implicitly assuming a single canal emerging from the water source, will also be given full marks. An answer dropping $\ell(n)$, and only providing the number of edges, will be given 4 points.

Each garden i requires an amount of water equal to w_i per day. To accommodate this, some of the canal segments will need to be wider than others, leading to more digging. If garden i is at a distance of d_i segments from the water source, its contribution to the digging needed is proportional to $d_i w_i$. The total amount of digging will thus be proportional to $\sum_{i=1}^{n} d_i w_i$.

b) Describe an algorithm for deciding how to construct a tree of canals so that the total amount of digging is minimized.

Answer (5 pts):
Use Huffman's algorithm. (No further explanation is required, but will not detract from the score, unless it is wrong.)
Note: The use of flow algorithms is completely unnecessary and irrelevant here. If the discussion of a flow algorithm elucidates the problem somewhat, illustrating some understanding of its workings, some points may be given. One suggested solution is using a heap structure. This would imply having gardens also as internal nodes, and would still only be a heuristic solution; even so, this solution makes some sense, and so will be given 2 points. Suggesting building an optimal binary search tree is on the right track, as we *are* optimizing the same thing – but with fewer constraints. This solution, and other correct (but inefficient) ones based on dynamic programming, will be given 3 points.

Problem 7

Consider the *balanced partition problem*, where we are given n integers $a_1, \ldots, a_n$ in the range $0 \ldots k$, and we want to partition them into two sets S_1 and S_2 so that $\left| \sum_{x \in S_1} x - \sum_{y \in S_2} y \right|$ is minimized.

a) Show that the problem is NP-hard.

Answer (5 pts):
E.g., by reduction from *subset-sum*: Do the numbers $b_1, \ldots, b_m$ contain a subset of sum s? Let t be their total sum. We now want to add a single element so that an exact partition gives us the desired subset, with the new element on "either side." We let $b_{m+1} = |2s - t|$. If $2s > t$, this means that b_{m+1} will end up on the "non-subset" side of the partition; otherwise, it will end up together with the subset.
Note: Other reductions from specific problems will be given full marks as well. If the problem is not in the curriculum, it should be defined as part of the answer. Just pointing out that we must reduce from an NP-hard problem but not showing how will yield 1 point. (Stating that we must reduce *to* an NP-hard problem is given 0 points.) Simply stating that the decision version is NP-complete gives 2 points. Showing the reduction from the decision version gives 4 points.

b) Sketch an algorithm for solving the problem in pseudopolynomial time.

Answer (5 pts):
E.g., by reduction the *0–1 knapsack problem*: Let each item i have weight and value given by a_i, and let the backpack capacity be $\lfloor \sum_{i=1}^{n} a_i/2 \rfloor$.
Note: Other pseudopolynomial solutions will give full marks, regardless of their exact running times.