Practice Final Solutions

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- When the exam begins, write your name on every page of this exam booklet.
- The exam contains seven multi-part problems. You have 180 minutes to earn 180 points.
- This exam booklet contains **17** pages, including this one. An extra sheet of scratch paper is attached. Please detach it before turning in your exam.
- This exam is closed book. You may use three handwritten A4 or $8\frac{1}{2}'' \times 11''$ crib sheets. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Points	Grade	Initials
1	12		
2	56		
3	20		
4	25		
5	27		
6	20		
7	20		
Total	180		

Name: Solutions _____

Circle your recitation letter and the name of your recitation instructor:

David A B Steve C D Hanson E F

Problem 1. Algorithm Design Techniques [12 points]

The following are a few of the design strategies we followed in class to solve several problems.

1. Dynamic programming.
2. Greedy strategy.
3. Divide-and-conquer.

For each of the following problems, mention which of the above design strategies was used (in class) in the following algorithms.

1. Longest common subsequence algorithm
   Solution: Dynamic Programming

2. Minimum spanning tree algorithm (Prim's algorithm)
   Solution: Greedy

3.Select

Solution: Divide-and-Conquer

4.Fast Fourier Transform

Solution: Divide-and-Conquer

Problem 2. True or False, and Justify [56 points]

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation. Each part is 4 points.

(a) T F An inorder traversal of a Min Heap will output the values in sorted order.

Solution: False. Consider the Min Heap with 1 at the root and 3 as left child and 2 as right child.

(b) T F A Monte-Carlo algorithm is a randomized algorithm that always outputs the correct answer and runs in expected polynomial time.

Solution: False. This is definition of a Las Vegas Algorithm.

(c) T F Two distinct degree-d polynomials with integer coefficients can evaluate to the same value in as many as $d + 1$ distinct points.

Solution: False. They have same value on at most d distinct points.

(d) T F The right subtree of an n-node 2-3-4 tree contains $\Omega(n)$ nodes.

Solution: False. A tree with all degree-3 nodes on one subtree and degree-2 nodes on the other will have depth $h = \log_3 n$. There will be $2^{\log_3 n} = n^{\log_3 2} = o(n)$ nodes on the sparse subtree. *Note: The prior version of the solutions incorrectly stated that this problem was true*.

(e) T F If a problem in NP can be solved in polynomial time, then all problems in NP can be solved in polynomial time.

Solution: False. The decision version of MST is in NP, but this doesn't mean that all problems in NP can be solved in polynomial time.

(f) T F If an NP-complete problem can be solved in linear time, then all NP-complete problems can be solved in linear time.

Solution: False. The reductions are *polynomial-time* but not necessarily *linear* time.

(g) T F If single digit multiplication can be done in $O(1)$ time, then multiplying two k-digit numbers can be done in $O(k \log k)$ time.

Solution: True. Use FFT.

(h) T F In a k-bit binary counter (that is initialized to zero and is always non-negative), any sequence of $n < 2^k$ increments followed by $m \leq n$ decrements takes $O(m + n)$ total bit flips in the worst case, where a bit flip changes one bit in the binary counter from 0 to 1 or from 1 to 0.

Solution: True. As shown in lecture the amortized cost of an increment in a sequence of increments is $O(1)$ and by the same arguement, the amortized cost of a decrement in a sequence of decrements is $O(1)$

(i) T F Consider a directed graph G in which every edge has a positive edge weight. Suppose you create a new graph G' by replacing the weight of each edge by the negation of its weight in G. For a given source vertex s, you compute all shortest path from s in G' using Dijkstra's algorithm.

True or false: the resulting paths in G' are the longest (i.e., highest cost) simple paths from s in G.

Solution: False. Dijkstra's algorithm may not necessarily return the minimum weight path because this new graph may contain a negative weight cycle.

(j) T F A spanning tree of a given undirected, connected graph $G = (V, E)$ can be found in $O(E)$ time.

Solution: True. Perform a walk on the graph starting from an arbitrary node.

(k) T F Consider the following algorithm for computing the square root of a number x:

SQUARE-ROOT(x)
For $i = 1, \ldots, x/2$:
    if $i^2 = x$ then output i.

True or False: This algorithm runs in polynomial time.

Solution: False. To run in polynomial-time, this algorithm would have to run in time polynomial in $\lg x$.

(l) T F An efficient max-flow algorithm can be used to efficiently compute a maximum matching of a given bipartite graph.

Solution: True. Add a "supersource" and a "supersink", each connected to a partition of the graph by unit capacity edges.

(m) T F The following recurrence has solution $T(n) = \Theta(n \lg(n^2))$.

$$T(n) = 2T(n/2) + 3 \cdot n$$

Solution: True. $\Theta(n \lg n^2) = \Theta(n \lg n)$.

(n) T F Computing the convolution of two vectors, each with n entries, requires $\Omega(n^2)$ time.

Solution: False. Use the standard FFT algorithm.

Problem 3. Placing Gas Stations Along a Highway [20 points]

Give a dynamic programming algorithm that on input S, where $S = \{s_0 = 0 \leq s_1 \leq \ldots \leq s_n = m\}$ is a finite set of positive integers, determines whether it is possible to place gas stations along an m-mile highway such that:

1. A gas station can only be placed at a distance $s_i \in S$ from the start of the highway.
2. There must be a gas station at the beginning of the highway ($s_0 = 0$) and at the end of the highway ($s_n = m$).
3. The distance between every two consecutive gas stations on the highway is between 15 and 25 miles.

For example, suppose the input is $\{0, 15, 40, 50, 60\}$. Then your algorithm should output "yes", because we can place gas stations at distances $\{0, 15, 40, 60\}$ from the beginning of the highway.

However, if the input is $\{0, 25, 30, 55, 70\}$, then your algorithm should output "no", because there is no subset of the distances that satisfies the conditions listed above.

Remember to analyze the running time of your algorithm.

Solution: Check to see if 0 and m are in S. If not, output "no".
Let $G[0]$ = "yes" if $s_0 = 0$.
For i from 1 to n, let $j = s_i$:
Let $G[j]_{j \in S, j>0}$ = "yes" if $G[k]$ = "yes", for some $k \in S, k < j$, and $15 \leq |s_k - s_j| \leq 25$.
Return $G[m]$.

Problem 4. Independent Set and Vertex Cover [25 points]

For a graph $G = (V, E)$, we say $S \subseteq V$ is an independent set in G if there are no edges between any two vertices in S.

We say a subset $T \subseteq V$ is a vertex cover of G if for every at edge (u, v) in E at least one of u or v is in T.

(a) [10 points] Show S is an independent set in G if and only if $V - S$ is a vertex cover of G.

Solution: $\Rightarrow$ Assume S is an Independent Set, but that $V - S$ is not a Vertex Cover. Then there exists an edge whose endpoints are both not in $V - S$, namely, there is an edge whose endpoints are in S. That is a contradiction, so $V - S$ must be a Vertex Cover.
$\Leftarrow$ Now assume that $V - S$ is a Vertex Cover, but that S is not an Independent Set. Then there exists an edge with both endpoints in S. But then that edge would not be touched by $V - S$, so $V - S$ could not be a Vertex Cover. This contradicts our assumption, so S must be an Independent Set.
Therefore, S is an Independent Set iff $V - S$ is a Vertex Cover.

(b) [5 points] Show that the decision problem *Independent Set* = $\{(G, k) | \ G$ contains an independent set of size at least $k\}$ is in NP.

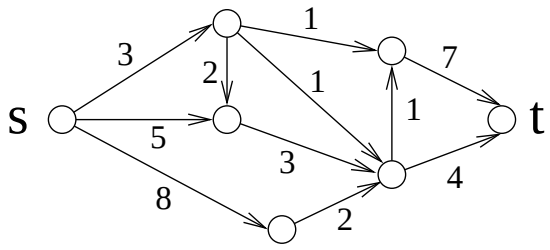
Solution: A set of k vertices forming an independent set is a proof that a given graph is an instance of Independent Set language. This proof can be verified trivially in polynomial time.

(c) [10 points] We showed in class that the decision problem *Vertex Cover* = $\{(G, k) |$ G contains a vertex cover of size at most $k\}$ is NP-complete. Use this to show that *Independent Set* is NP-complete.

Solution: We need to show that Vertex Cover reduces to Independent Set. By part (a), if a graph has a Vertex Cover of size k, then it has an Independent Set of size $n - k$. So, given an instance (G, k), we can trivially make an instance of Independent Set $(G, n - k)$.

Problem 5. Flows [27 points]

Consider the following graph:



(a) [10 points] What is the maximum flow in this graph? Give the actual flow as well as its value. Justify your answer.

Solution: The maximum flow is 6. From S, we route 3 along both the 3-capacity edge and the 5-capacity edge. The resultant flow will inundate the 1, 1 and 4-capacity edges that form a min-cut for the graph. By the Max-Flow/Min-Cut algorithm, this is a Max-Flow.

(b) [5 points] True or false: For any flow network G and any maximum flow on G, there is always an edge e such that increasing the capacity of e increases the maximum flow of the network. Justify your answer.

Solution: False. A counterexample is a graph with two unit capacity edges in a chain. Increasing the capacity of a single edge will not increase the max flow, since the other edge is at capacity.

(c) [5 points] Suppose you have a flow network G with integer capacities, and an integer maximum flow f. Suppose that, for some edge e, we increase the capacity of e by one. Describe an $O(|E|)$-time algorithm to find a maximum flow in the modified graph.

Solution: Add the new flow to the residual flow graph in $O(|E|)$ time. Perform a tree traversal from the source node to detect whether a path now exists to the sink. If so, augment along that path and increase the maximum flow by one.

(d) [7 points] Consider the decision problem: *Flow* = $\{(G, s, t, k) |\ G = (V, E)$ is a flow network, $s, t \in V$, and the value of an optimal flow from s to t in G is $k\}$.

Is *Flow* in NP? Why or why not?

Solution: Yes. We can explicitly compute the max-flow using Edmonds-Karp or another polynomial time max-flow algorithm. Since $P \subseteq NP$, Flow must be in NP.

Problem 6. Comparing Sets [20 points]

Given two sets of integers S_1 and S_2, each of size n, your job is to determine if S_1 and S_2 are identical.

Give a $O(n)$ time randomized algorithm that always outputs "yes" if $S_1 = S_2$ and outputs "no" with probability at least $1 - 1/n$ if $S_1 \neq S_2$.

You may assume that we have a probabilistic model of computation in which generating a random number takes $O(1)$ time and a comparison, a multiplication, and an addition of two numbers each takes $O(1)$ time, regardless of the size of the two numbers involved.

Hint: Reduce the problem of comparing sets to the problem of comparing polynomials.

Solution: Reduce the problem of comparing sets to that of comparing two degree-n polynomials $P_1(x) = \prod_{r_1 \in M_1}(x - r_1)$ and $P_2(x) = \prod_{r_2 \in M_2}(x - r_2)$ with n integer roots, where each polynomial is specified as a list of these n integer roots, i.e. $P_1(x) = \{r_{11}, r_{12}, \ldots r_{1n}\}$. The following randomized algorithm for comparing $P_1(x)$ and $P_2(x)$ which runs in $O(n)$ time.

COMPARE-POLYNOMIALS$(P_1(x), P_2(x))$
1 Choose a random integer $a \in \{1, \ldots n^2\}$
2 Compute $P_1(a) = \prod_{i=1}^{n}(a - r_{1i})$ and $P_2(a) = \prod_{i=1}^{n}(a - r_{2i})$
3 **if** $P_1(a) = P_2(a)$
4 **then** output $M_1 = M_2$
5 **else** output $M_1 \neq M_2$

Line 2 of COMPARE-POLYNOMIALS$(P_1(x), P_2(x))$ runs in $O(n)$ time, since each multiplication takes $O(1)$ time. All other lines take $O(1)$ time under the assumptions stated above. Now we will analyze the probability that COMPARE-POLYNOMIALS$(P_1(x), P_2(x))$ outputs the correct answer.
If $P_1(x) = P_2(x)$, then COMPARE-POLYNOMIALS$(P_1(x), P_2(x))$ outputs the correct answer with probability 1.
A degree-n polynomial $P(x)$ has at most n distinct roots. Furthermore, $P(a) \neq 0$ if a is not a root of $P(x)$. Thus, $P(x)$ has at most n integer zeroes in the range $\{1, \ldots n^2\}$.
Let $P_3(x) = P_1(x) - P_2(x)$. Note that $P_1(a) = P_2(a)$ exactly when $P_3(a) = 0$. Since the maximum degree of $P_3(x)$ is n, $P_3(x)$ has at most n distinct roots. Thus, if we choose a random integer x from the range $\{1, \ldots n^2\}$, the probability that $P_3(x)$ evaluates to 0 is at most $1/n$.
Therefore, if $P_1(x) \neq P_2(x)$, the probability that we choose a such that $P_1(a) - P_2(a) = 0$ is at most $\frac{1}{n}$. Thus, the probability that COMPARE-POLYNOMIALS$(P_1(x), P_2(x))$ outputs the correct answer is at least $1 - \frac{1}{n}$.

Problem 7. Finding the Topological Sort of a Complete Directed Acyclic Graph [20 points]

Let $G = (V, A)$ be a directed acyclic graph that has an edge between every pair of vertices and whose vertices are labeled $1, 2, \ldots n$, where $n = |V|$. To determine the direction of an edge between two vertices in V, you are only allowed to ask a *query*. A query consists of two specified vertices u and v and is answered with:

"from u to v" if (u, v) is in A, or

"from v to u" if (v, u) is in A.

Give matching upper and lower bounds (as functions of n) for the number of queries required to find a topological sort of G.

Solution: This problem reduces to sorting. A query establishes an ordering between two nodes, i.e. (u, v)'s existence can be interpreted as $u > v$ and (v, u) as $v < u$. Since the graph is complete, there exists an ordering between every pair of nodes, in other words, we have a total ordering on the graph.
We can simply run a comparison-based sort to output a topological sort. The "max" element will be a source node with out-going edges to all other nodes. Similarly, the "min" element will be a sink node. Since *query* is effectively a comparison operator, there is a tight $\Omega(n \log n)$ lower bound on performing a topological sort in this model.