CS161 Final Exam

(Do not turn this page until you are instructed to do so!)

Instructions: This is closed-book exam, and you are permitted to refer only to one double-sided sheet of notes, which you should have prepared in advance. You have 3 hours and the exam is worth 150 points. Make sure you print your name legibly and sign the honor code below. All of the intended answers can be written well within the space provided. You can use the back of the preceding page for scratch work. If you want to use the back side of a page to write part of your answer, be sure to mark your answer clearly. Good luck!

The following is a statement of the Stanford University Honor Code:

A. *The Honor Code is an undertaking of the students, individually and collectively:*
   (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
   (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

B. *The faculty on its part manifests its codence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

C. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

I acknowledge and accept the Honor Code.

____

(Signature)

____

(Print your name, legibly!)

Problem	#1	#2	#3	#4	#5	#6	#7	#8	Total
Score									
Maximum	30	15	15	10	15	20	20	25	150

Problem 1. (30 points) [DIFFICULTY LEVEL: Relatively straightforward.] For each of the following 10 statements, circle "T" or "F", depending on whether you think the statement is true or false, respectively. You will receive 3 points for each correct answer *and -2 points for each incorrect answer*. (If you leave your answer blank, you will receive 0 points.) You need not justify your answers.

T F A red-black tree can be used to sort an array of n elements with worst-case running time $O(n \log n)$.

T F A hash table can be used to sort an array of n elements with expected running time $O(n)$.

T F Every recursive algorithm that makes at least 2 recursive calls has worst-case running time $\Omega(n \log n)$ on inputs of size n.

T F In a connected undirected graph $G = (V, E)$ with distinct edge costs, the cheapest edge belongs to every minimum spanning tree.

T F In a connected undirected graph $G = (V, E)$ with distinct edge costs, the most expensive edge is excluded from every minimum spanning tree.

T F Linearity of expectation — that the expectation of a sum of random variables equals the sum of the random variables' expectations — holds even for random variables that are not independent.

T F Let U be a finite set and n a positive integer. Let S be the set of all functions with domain U and range $\{0, 1, 2, \ldots, n-1\}$. Then S is a universal family of hash functions.

T F Every vertex of a directed graph $G = (V, E)$ is a member of exactly one strongly connected component of G.

T F Let G be a directed graph in which edges can have positive or negative edge lengths, but that has no negative cycles. The Bellman-Ford algorithm correctly computes shortest-path lengths from a given origin s to every other vertex v.

T F Let G be a directed graph in which edges can have positive or negative edge lengths, but that has no negative cycles. The Floyd-Warshall algorithm correctly computes shortest-path lengths between every pair of vertices.

Problem 2. (15 points) [DIFFICULTY LEVEL: Relatively straightforward.]

(a) (7 points) Explain, conceptually, why the Master Method has three different cases. Explain the conceptual meaning of each of the cases. Your answer should be 3-6 sentences.

(b) (8 points) Suppose that the worst-case running time $T(n)$ of an algorithm on an input of size n is governed by the following recurrence, where $c > 0$ is some constant:

- $T(1) = T(2) = T(3) = T(4) = 1$; and
- $T(n) \leq T(\lfloor 2n/3 \rfloor) + T(\lfloor n/5 \rfloor) + cn$ for every $n > 4$.

Prove, using the substitution method, that $T(n) = O(n)$. Be sure to state the requirements that your constants need to satisfy for your proof to work.

Problem 3. (15 points) [DIFFICULTY LEVEL: Relatively straightforward.] You are given an undirected connected graph $G = (V, E)$ with n vertices and m edges, in adjacency list representation, and also a distinguished edge $e \in E$. The *length* of a cycle in G is defined as the number of edges that it contains. Design a linear-time ($O(m)$) algorithm that computes the length of the smallest cycle in G that includes the edge e. (If there are no such cycles, your algorithm should detect this.) You should include a succinct high-level description of your algorithm, a brief correctness proof, and a brief analysis of its running time.

[Feel free to use any algorithms from lecture as subroutines; you do not have to re-prove the correctness or re-analyze the running time of such algorithms.]

Problem 4. (10 points) [DIFFICULTY LEVEL: Relatively straightforward.] Recall that in lecture we designed a correct algorithm for the Knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the Knapsack capacity. (Both W and the weights of all items are assumed to be positive integers.)

It is a fact that this version of the Knapsack problem is NP-complete. Why doesn't our algorithm immediately imply that P=NP?

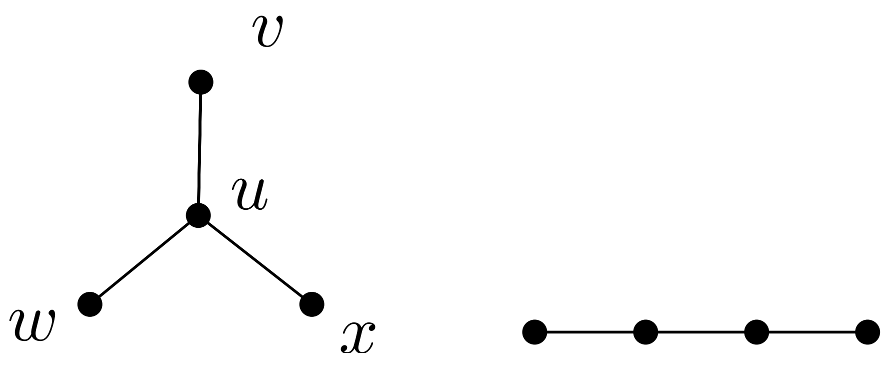
Problem 5. (15 points) [DIFFICULTY LEVEL: Moderately difficult.] You are given a connected undirected graph $G = (V, E)$ with n vertices and m edges, in adjacency list representation. Each edge has a strictly positive real-valued cost c_e. A subset $F \subseteq E$ of edges is called a *feedback edge set (FES)* if deleting F from G yields an acyclic graph. Give an $O(m \log n)$-time algorithm for computing an FES of a graph with the minimum-possible total cost (i.e., the sum of the edge costs in the FES should be as small as possible). Justify the running time of your algorithm, and prove that your algorithm is correct.

[Feel free to use any algorithms from lecture as subroutines; you do not have to reprove the correctness or re-analyze the running time of such algorithms.]

Problem 6. (20 points) [DIFFICULTY LEVEL: Moderately difficult.] You are given a *sorted* (from smallest to largest) array of *distinct* integers $A[1], \ldots, A[n]$, which can be positive, negative, or zero. You want to decide whether or not there is an index $i \in \{1, 2, \ldots, n\}$ such that $A[i] = i$. Design the fastest algorithm that you can for solving this problem. Prove (ideally in about 2-3 sentences) that your algorithm is correct, and provide a formal analysis of its running time.

[Hints: No credit will be given for a linear-time algorithm. Instead, model your algorithm on binary search. You can assume that n is a power of 2.]

Problem 7. (20 points) [DIFFICULTY LEVEL: Moderately difficult.] Let $G = (V, E)$ be an undirected graph. A subset $F \subseteq E$ of edges is a *matching* of G if each vertex is the endpoint of at most one edge of F. (Thus each edge of a matching pairs two vertices together, with each vertex paired to at most one other one.) For example, if G is a star on four vertices u, v, w, x (Figure 1(a)), then the only matchings of G are the subgraphs with at most one edge (since any two edges overlap in the middle vertex). If G is a three-hop path (Figure 1(b)), then the first and third edges form a matching; as does the second edge by itself; as does any subset of one of these matchings.



(a) A Star on Four Vertices (b) A Three-Hop Path

Figure 1: Examples for Problem 7.

Now suppose that each edge e of G has a nonnegative *weight* w_e. We would like to compute the maximum-weight matching — i.e., the matching with the largest sum of edge weights — or at least a good approximation to it. Here is a natural greedy heuristic, in the spirit of Kruskal's algorithm:

- Sort the edges $e_1, \ldots, e_m$ of G from largest to smallest weight.
- $M = \emptyset$
- For $i = 1$ to m:
  - if neither endpoint of edge $e_i = (v_i, w_i)$ is already matched — that is, M currently contains no edges incident to either v_i or w_i — then add e_i to M.

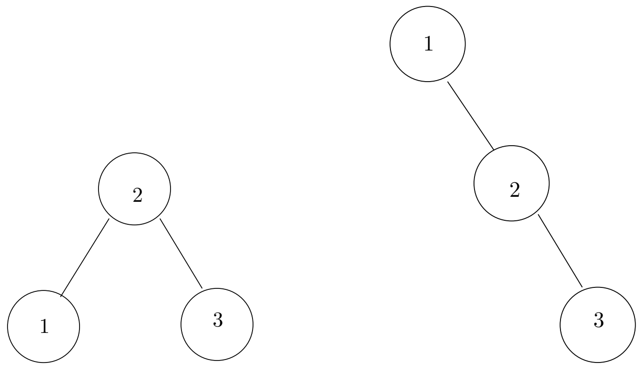
(a) (5 points) Show, by an explicit counterexample, that the algorithm above need not return a maximum-weight matching. For full credit, your counterexample should have the fewest-possible number of edges.

(b) (15 points) Prove that, for every graph and every set of nonnegative edge weights, the algorithm above returns a matching whose total weight is at least 50% of that of a maximum-weight matching. (You can write your solution on the next page, if you like.)

Problem 7 (cont'd). This page is extra space for your solution to Problem 7(b).

Problem 8. (25 points) [DIFFICULTY LEVEL: Difficult.] This problem studies the problem of constructing a binary search tree on the keys $\{1, 2, 3, \ldots, n\}$ that minimizes the weighted search time.

More precisely, you are given as input non-negative weights $w_1, w_2, \ldots, w_n$ for the keys $\{1, 2, \ldots, n\}$. The feasible solutions are binary search trees — i.e., binary trees that store the keys $\{1, 2, \ldots, n\}$ in search tree order. The *search time* $T(i)$ for the key i in the tree T is the depth of i in T, plus one.



(a) One BST T_1 on $\{1, 2, 3\}$

(b) Another BST T_2 on $\{1, 2, 3\}$

Figure 2: Examples for Problem 8.

For example, suppose $n = 3$. Figure 2 shows two binary search trees on $\{1, 2, 3\}$, T_1 and T_2. Observe that $T_1(2) = 1$ while $T_2(2) = 2$.

The *weighted search time* of a tree T is $\sum_{i=1}^{n} w_i \cdot T(i)$. For example, if $w_1 = w_2 = w_3 = 2$, then the weighted search time of T_1 (namely, $2 \cdot 2 + 2 \cdot 1 + 2 \cdot 2 = 10$) is smaller than that of T_2 (namely, 12). On the other hand, if w_1 is huge compared to w_2 and w_3, then the weighted search time of T_2 is smaller than that of T_1.

Give a dynamic programming algorithm that, given non-negative weights $w_1, \ldots, w_n$, returns the minimum-possible weighted search time of a binary search tree on $\{1, 2, \ldots, n\}$. (You do *not* have to construct the tree.) Explain clearly what the subproblems are and what recurrence you use to relate the solutions of different subproblems to one another. Give a succinct proof that your recurrence is correct. Give a brief analysis of the running time of your algorithm, which should be polynomial in n.

[Hints: First, what if someone told you which element is the root of an optimal binary search tree? Second, which distinct subproblems do you really need to solve?]

Problem 8 (cont'd). This page is extra space for your solution to Problem 8.