CS570
Analysis of Algorithms
Fall 2016
Exam I

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	15	
Problem 4	15	
Problem 5	10	
Problem 6	15	
Problem 7	15	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/]
If G is a directed acyclic graph, then G must have a node with no incoming edges.

[TRUE/]
Inserting into a binomial heap of n elements has O(1) amortized cost.

[/FALSE]
In an undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph cannot be in any minimum spanning tree.

[TRUE/]
If $f=O(g)$and $g=O(h)$, then $f=O(h)$.

[/FALSE]
The array [100, 93, 83, 90, 79, 84, 81] corresponds to a binary max-heap.

[TRUE/]
Given a graph, suppose we have calculated the shortest path from a source to all other vertices. If we modify the graph such that weights of all edges are doubled, then the shortest path remains the same, only the total weight of the path changes.

[TRUE/]
Let $d(u, v)$ denote the minimum distance from node u to node v in a weighted graph G. If $d(s, u) + d(u, t) = d(s, t)$, then u is on at least one shortest path from s to t. (All edge weights in G are positive.)

[TRUE/]
Height of a binary heap is O(n)

[/FALSE]
The divide and conquer algorithm to solve the closest pair of points in 2D runs in O(n log n). But if the two lists of points, one sorted by X-coordinate and the other sorted by Y-coordinate are given to us as input, the rest of the algorithm (skipping the sorting steps) runs in O(n) time.

[TRUE/].
The solution of the recurrence $T(n) = 18T(n/3) + O(n^3)$ is $T(n) = O(n^3)$.

2) 10 pts

Prove or disprove the following statement:

For a given stable matching problem, if m and w appear as a pair when men propose and they also appear as a pair when women propose, then m and w must be paired in all possible stable matchings

Solution:

Let S denote the stable matching obtained from the version where men propose and let S_0 be the stable matching obtained from the version where women propose. From (page 11, statement 1.8 in the text), in S, every woman is paired with her worst valid partner. Applying (page 10, statement 1.7) by symmetry to the version of Gale-Shapely where women propose, it follows that in S_0 , every woman is paired with her best valid partner. Since in both the matchings S and S_0, w is paired with m, it follows that for w the best valid partner and the worst valid partner are the same. This implies that w has a unique valid partner (which is m), which implies that m and w are paired together in all the stable matchings and the claim is true.

3) 15 pts

Let G = (V, E) be a connected undirected graph with edge-weight function w, and assume all edge weights are distinct. Consider a cycle $\leq v_1, v_2, \ldots, v_k, v_{k+1} \geq$ in G, where $v_{k+1} = v_1$, and let (v_i, v_{i+1}) be the edge in the cycle with the largest edge weight. Prove that (v_i, v_{i+1}) does not belong to the minimum spanning tree T of G.

Solution:

Proof by contradiction.

Assume that (v_i, v_{i+1}) does belong to the minimum spanning tree T.

Removing (v_i, v_{i+1}) from T, divides T into two connected components A and B, where some nodes of the given cycle are in A and some are in B.

For any cycle, at least two edges must cross this cut, and therefore there is some other edge (v_j , v_{j+1}) on the cycle, such that adding this edge connects A and B again and creates another spanning tree T'.

Since the weight of (v_j, v_{j+1}) is less than (v_i, v_{i+1}), the weight of T' is less than T and T cannot be a minimum spanning tree.

Contradiction.

Therefore, (v_i, v_{i+1}) doesn't belong to any minimum spanning tree T of G.

4) 15 pts

Each of the following problems suggests a greedy algorithm for a specified task. Prove that the greedy algorithm is optimal, or give a counterexample to show that it is not.

(a) **Problem**: You are given a set of n jobs. Job i starts at a fixed time s_i, ends at a fixed time e_i, and results in a profit q_i. Only one job may run at a time. The goal is to choose a subset of the available jobs to maximize total profit.
Algorithm: First, sort the jobs so that $q_1 \geq q_2 \geq \ldots \geq q_n$. Then, try to add each job to the schedule in turn. If job i does not conflict with any jobs schedule so far, add it to the schedule; otherwise, discard it.
This algorithm is not optimal. Consider three jobs with starts {1, 1, 3}, ends {4, 2, 4}, and profits {3, 2, 2}.

(b) **Problem**: You are given a set of n jobs, each of which runs in unit time. Job i has an integer-valued deadline time $d_i \geq 0$ and a real-valued penalty $p_i \geq 0$. Jobs may be scheduled to start at any integer time (0, 1, 2, etc), and only one job may run at a time. If job i completes at or before time d_i, then it incurs no penalty; otherwise, it incurs penalty p_i. The goal is to schedule all jobs so as to minimize the total penalty incurred.
Algorithm: Define slot k in the schedule to run from time $k-1$ to time k. First, sort the jobs so that $p_1 \geq p_2 \geq \ldots \geq p_n$. Then, add each job to the schedule in turn. When adding job i, if any time slot between 1 and d_i is available, then schedule i in the latest such slot. Otherwise, schedule job i in the latest available slot $\leq n$.

Solution:

We prove via exchange argument. Let's start with an arbitrary optimal solution O_0, and prove that we can change this solution to greedy solution without increasing total penalty.

* (Base:) Consider location of job_1, i.e. the job with largest late penalty p_1. Suppose greedy solution places job_1 at slot_x, and an optimal solution O_0 places job_1 at slot_y, and places job_z at slot_x. job_z is from {job_2, ...job_n}

1. If slot_x==slot_y, job_1==job_z, there exist an optimal solution which places job_1 the same slot to greedy solution, and let's fix this position.
2. If slot_x!=slot_y, (must discuss both case)
   2-a. case1, slot_y>slot_x: in optimal solution O_0, job_1 caused penalty p_1, and job_z may cause penalty 0 or p_z (p_z <= p_1). If we switch job_1 and job_z, the total penalty will decrease by p_1 or p_1-p_z. In any case, the total penalty does not increase, so we can swap job_1 and job_z in O_0, and the resultant schedule is still optimal.
   2-b. case2, slot_y<slot_x: if we swap job_1 and job_z, (1) because job_z is moved forward, it won't cause more penalty, (2) because job_1 is moved to the latest slot before d_1, a position with 0 penalty, it won't cause more penalty either. Here we assumed d_1>0. If d_1=0, then job_1 will cause p_1 penalty in any solution, so in this special case, swapping job_1 and job_z in optimal solution won't increase penalty.

From above discussion, we can swap job_1 and job_z in solution O_0 and the result is still an optimal solution. Let's record new optimal solution as O_1.

* (Inductive:) Suppose we have created an optimal solution O_k through swapping, which place {job_1... job_k} in the same position to greedy solution. Use same notation to base case: suppose

greedy solution places job_(k+1) at slot_x, and O_k places it at slot_y, and places job_z at slot_x. Note that because O_k places {job_1...job_k} the same slots to greedy solution, job_z must be one of {job_(k+2), ... job_n}.

1. If slot_x==slot_y, job_(k+1)==job_z, same to base case.
2. If slot_x!=slot_y, (must discuss both case, and must clarify the difference from base case in 2-b)
   2-a. case1, same to base case.
   2-b. case2, slot_y<slot_x: if we swap job_(k+1) and job_z, (1) because job_z is moved forward, it won't cause more penalty, (2) moving job_(k+1) from slot_y to slot_x won't increase penalty: If slot_x <=b_(k+1), job_(k+1) still has 0 penalty; if slot_x>b_(k+1), according to greedy algorithm, the slots from 1 to b_(k+1) must be filled by jobs from {job_1, ... job_k}, in both greedy solution and O_k, so slot_y>b_(k+1) too, and job_(k+1) causes p_k penalty before and after swapping.

Thus we can conclude for any arbitrary k>=1, starting with O_k, we can swap job_(k+1) and job_z and the result is still an optimal solution O_(k+1).

* (conclusion:) by induction, we can move every job from an optimal solution in the order of {job_1, job_2, ... job_n} to the corresponding position in greedy solution one by one, via swapping, without increasing total penalty. Thus greedy solution is an optimal solution.

Here are a list of typical mistakes in 4b).

1. Some students try to prove that in greedy solution, if we swap any pair of jobs in the greedy solution the penalty will not decrease. This is incorrect because optimal solution is not greedy solution add one swapping. Some argues that a sequence of swapping can change greedy to global optimal, but after one swapping, the greedy solution is not long a greedy solution, so the property " if we swap any pair of jobs in the greedy solution the penalty will not decrease" no longer holds.
    1. Note: Exchange argument swaps pairs in optimal solution to match greedy solution. It does not swap greedy solution.
2. Similar to 1, some students think that there must exist one pair of job (i,j) s.t. the position of them in greedy and optimal solution is exchangeable. This is incorrect. e.g. order of (1,2,3) and order of (2,3,1).
3. Some students confused optimal and greedy solution, and attempt to guess property of ordering in optimal solution. e.g. they attempted to declare that two jobs cannot be arranged in certain way if penalty and deadlines of these two jobs satisfies some condition. This is invalid. We know nothing about optimal solution except its late penalty is smallest among all factor(n) combinations.
4. Some students explained the intuition of positioning next job in greedy algorithm but did not explain the optimality.
    1. Some students qualitatively analyze why we start from job with large penalty, and why put the next job in "latest" available slot. e.g. starting with hard (with higher late penalty), we use the best slot resourced to avoid suffering from large penalties; using the "latest" available position is friendly for future jobs, maximizing the probability they have on-time slots.
    2. Many students proved in this way: "<1> provided positions of job with 1st, 2nd, … k-th late penalties, the position of next job (it will have p_(k+1) penalty if late, let's call it (k+1)-th job) assigned by greedy algorithm is optimal because it will minimize the penalty from adding this new job. <2> by induction, claim 1 is true for every k, thus the final solution is optimal". However, they did not understand the difference

between "local/greedy optimality" and "global optimality". We can only declare that, "if we are already fixed position of 1st ~ k-th jobs, and if we only adding one job", the position of job(k+1) according to greedy algorithm is best. They cannot declare that the fixed position of job 1~k are truly optimal (optimal is defined on all n jobs), and they cannot declare the final output of n-jobs' ordering is optimal. In short, this greedy algorithm guarantees one step is best only if we "move only one step, and evaluate only this one step".

5. Many students use induction. They made mistakes in these two ways:
    1. One inductive proof in this way: Provided the total number of jobs is fixed n, (base:) starting with placing first job (which will suffer from most late-penalty if not on-time) into one of n slots, declaring the greedy algorithm provides "optimal first step", then (induction:) suppose we have k jobs have been placed according to greedy algorithm, and their position are optimal, then the position of (k+1)th according to greedy algorithm is optimal. <u>There are at least two mistakes:</u>
        i. Mistake in basis step: We cannot declare position of first job as optimal. Optimality is defined over the order of all n jobs. Knowing position of 1st job, not knowing positions of other (N-1) jobs, we cannot declare if 1st job is optimal or not.
        ii. Induction error: explained in item 3-2.
    2. Another use inductive proof in this way: (base:) suppose we have totally 1 job, then obvious greedy algorithm give optimal solution since only one slot available. Then (induction) suppose for task of arranging n=k jobs, greedy algorithm is optimal, then if there are totally n=(k+1) jobs, greedy algorithm also give optimal solution. In the inductive steps, they declare that the first k of n=k+1 jobs are assigned slots using greedy algorithm, and we have known position of these k jobs are optimal according to inductive assumption.
        i. The declaration that "the first k of n=k+1 jobs are assigned slots using greedy algorithm" is incorrect. Placing k jobs and placing k of k+1 jobs are different problems. Actually placing k of k+1 jobs and place k+1 jobs are same problem, because the position of last job provided position of first k jobs is determined.
    3. Some students mixed the basis condition and inductive arguments in above two cases. This is even worse.
6. Key steps and important statements are not explained correctly: Many students do not provide strong evidence when they declare two jobs (either from optimal solution or from greedy solution) can or cannot be swapped. For example,
    1. They did not explore all possibilities of properties of job pairs.
    2. The calculation of change of penalty after swapping is incorrect.
7. Some students do not know how to prove optimality (or for some other reasons), and they write many statements or random sentence, random expressions without clean or correct logics.

For any of case 1~5, 3 points will be penalized. If most statements and reasoning steps are clean 1~2 points may be rewarded. For case 6, up to 4 points may be penalized. Usually case 6 together with other mistakes from case 1~5 will cause no more than 6 points penalty. Case 7 itself could cause up to 7 points penalty.

5) 10 pts
Arrange the following functions in increasing order of growth rate with g(n) following f(n) in your list if and only if $f(n) = O(g(n))$

$$n^2 \log n,\ \ 2^n,\ \ 2^{2^n},\ \ n^{\log n},\ \ n^2$$

Solution: $n^2,\ n^2 \log n,\ n^{\log n},\ \ 2^n,\ \ 2^{2^n}$

6) 15 pts

Suppose we perform a sequence of n operations on a data structure in which the i^{th} operation costs i if i is an exact power of 2. If it is not, the i^{th} operation costs 1. Use aggregate analysis to determine the amortized cost per operation.

Solution: In a sequence of n operations there are logn+1 exact powers of 2, namely 1,2,4,...,2^{logn} .Thus the total cost of all operations is T(n) ≤ 2n+n = 3n, which means O(3) amortized cost per operation

7) 15 pts

A set of cities V is connected by a network of roads G(V,E). The length of road $e \in E$ is weight of that edge. There is a proposal to add one new road to this network, and a set C contains candidate pairs of cities between which the new road may be built. Each of these potential roads has an associated length. Suppose you want to choose the road that would result in maximum decrease in the driving distance between city s and city t. Give an efficient algorithm for solving this problem, analyze its complexity, and express its complexity as a function of |V|, |E|, and |C|.
For full credit, your algorithm should have a run time of O(|E| log|V| + |C|).
Partial credit (7 pts) will be granted to efficient solutions that don't meet the run time requirement given above.

Solution:

Algorithm:

1. Run Dijkstra's algorithm from s to calculate shortest distances from city s to all other cities
2. Run Dijkstra's algorithm from t to calculate shortest distance from city t to all other cities
   (If graph G is directed, we will first create a reverse graph G^{rev}, which has all of the edges reversed compared to the orientation of the corresponding edges in G, and run Dijkstra's algorithm from city t to all other cities)

3. For every candidate pair of cities {u,v}, the shortest path distance between s and t which covers road $u \rightarrow v$ is **min**{ dist(s, u) + dist(t, v) + length(u, v), dist(s, v) + dist(t, u) + length(u, v) }.
4. Choose the shortest distances from the third step.
   - If the selected distance is longer than original dist(s,t), any candidate road cannot decrease distance between s and t.
   - Else, choose the {u,v} pair that produces shortest new distance. In the case of tie, choose one arbitrarily.

Time Complexity:

Complexity of running Dijkstra's algorithm is $\Theta(2 \times |E| \log|V|)$, the complexity of the third step is $\Theta(2 \times |C|)$. Therefore, the total complexity is $\Theta(|E| \log|V| + |C|)$.

Additional Space

Additional Space