Group A

Problem Set Question: Explain Question 2 on Problem Set 4. Then address the new part (d). The question is restated here, with a new part (d):

A new museum has just opened in Singapore, and everyone wants to go. Alas, they can only let in n people per day. They have hired you as a consultant to help maximize their attendance.

They have the following problem: people arrive at the museum in groups. Sometimes in small groups (e.g., a family) and sometimes a large group (eg., a tour group, or a school class). When a group wants to visit the museum, there must be space for all of the group to enter the museum. If there isn't room for the entire group, then the entire group is turned away. The museum wants your advice on how to let as many people visit as possible.

You are given a set of groups $G[1..m] = [g_1, g_2, \ldots, g_m]$ where g_i is a number representing the size of the group. Each of these groups is trying to make a reservation to enter the museum next week. Assume that you can admit at most n people total.

Part a. Consider the following greedy admission algorithm, where the function `Admit`(i) admits group i and `Reject`(i) rejects group i.

```
1  Algorithm: GroupAdmission(G[1..m], n)
2  Procedure:
3      admitted ← 0
4      remaining ← n
5      G ← Sort(G)
6      for i ← 1 to m do
7          if G[i] ≤ remaining then
8              Admit(i)
9              remaining ← remaining − G[i]
10             admitted ← admitted + G[i]
11         else
12             Reject(i)
13     return admitted
```

The greedy algorithm sorts the groups by size, with the largest groups first. It then iterates through the groups from largest to smallest, admitting any group for which there is room. Show that this algorithm works pretty well, i.e., show it is a 2-approximation of optimal. (That is, if, given G and n, it is possible to admit k people, then the greedy admission algorithm will admit at least $k/2$ people.)

paragraphPart b. Unfortunately, the `GroupAdmission` algorithm does not

work perfectly. Show that the algorithm is not optimal by giving a counterexample in which, asymptotically as n gets large, the ratio between greedy seating and optimal seating approaches 1/2.

Part c. Assume that the groups G are provided in read-only memory. That is, you cannot sort the groups G, nor can you copy the array. (You may only use $O(1)$ extra space.) In this context, give an efficient algorithm, and prove that it is a correct 2-approximation algorithm. (*Hint:* one solution runs in time $O(m \log n)$; however there is a better solution that runs in time $O(m)$.)

Part d. (new) Give a dynamic program for finding an optimal solution to this problem. Your dynamic program should run in time polynomial in m and n (or, alternatively, in m and $\max_i(G[i])$). That is, the running time depends on the values of the input.

Explain your algorithm and give the running time. (Hint: think about the recurrence where $A(i, B)$ that finds the maximum number of people you can admit among the groups $G[1, \ldots, i]$ if the museum has capacity B.)

Part e. (new) Give an integer linear program for solving this problem. (It may be the case that the relaxed linear program version yields a 2-approximation!)

For discussion. In the real version of knapsack problem, each item has to aspects: v_i determines the value of object i and w_i determines the weight of object i. The goal is to choose a subset of the elements with maximum value, subject to the constraint the total weight of all the elements in the subset is at most W. If you sort the elements by (v_i/w_i), then approximately the same approximation algorithm as above gives you a 2-approximation, and approximately the same dynamic program gives you an optimal solution.

Group B

Problem Set Question: Explain Question 2 on Problem Set 3. Then answer the additional question that follows. To restate the question:

Let $G = (V, E)$ be an undirected graph with nonnegative edge weights. Let S, the senders and R, the receivers, be disjoint subsets of V. The problem is to find a minimum cost subgraph of G such that for every receiver $r \in R$, there is at least one sender $s \in S$ such that there is a path connecting r to s in the subgraph. Give a 2-approximation algorithm that runs in polynomial time. *Hint: Consider introducing an additional vertex to the graph, and try building on an approximation algorithm (for a different problem).*

Class Question: Can you solve this problem with network flows? Why or why not? What similar problem can you solve with network flows?

[This page left blank]

Group C

Class Question: In class, we formulated the traveling salesman problem in terms of a set of points V and a distance function d that gives the distance between any two points in V. What if the input to the problem is a graph $G = (V, E)$ with weighted edges? (What if there are no Hamiltonian Cycles in the graph?)

Define a version of TSP for graphs, and explain whether or not it remains approximable using the techniques discussed in class.

Class Question: In class, we formulated the traveling salesman problem in terms of a set of points V and a *symmetric* distance function d that gives the distance between any two points in V. What if the distance function is *not* symmetric, i.e., $d(u, v) \neq d(v, u)$. Does the 2-approximation algorithm from class still work? Why or why not? (How would you even define the 2-approximation algorithm in this case?) I am not asking you here for a new algorithm; simply discuss the issues involved and give examples of how things work or do not work.

Class Question: Think about the Euclidean Traveling Salesman Problem where each point has (x, y)-coordinates to identify it. Imagine that I only want "cycles" the proceed in one direction, e.g., left-to-right. For example, a legal output is a cycle $(v_1, v_2, v_3, v_4, v_1)$ where for each $i < n$, we know that $v_i.x < v_{i+1}.x$. (In the last step of the cycle, going back from v_n to v_1, we are allowed to go to the left.) Give an example where the cycle that is found in this case is very bad compared to the optimal TSP cycle. Is there an efficient algorithm to solve this problem?

[This page left blank]

Group D

Class Question: Given a set of vertices V, a hyperedge is a set of vertices. A classical edge (u, v) is a set of two vertices. A hyperedge can connect more vertices, e.g., (u, v, w, x) all in one edge. A hypergraph is a graph $G = (V, E)$ where E is a set of hyperedges.

We can now consider the problem of vertex cover for a hypergraph: a hyperedge is covered by a set S if any of its vertices are in the set S. A vertex cover for a hypergraph is a set S that covers all the edges. We want to find the minimum sized vertex cover.

Give an algorithm for finding a $\log n$-approximation to the minimum vertex cover on a hypergraph. (Hint: think about reducing this to a problem we already know how to approximate.) Do you think it likely that we can do better than a $\log n$-approximation? Why or why not?

[This page left blank]