The first page of your homework submission must be a cover sheet answering the following questions. Do not leave it until the last minute; it's fine to fill out the cover sheet before you have completely finished the assignment. Assignments submitted without a cover sheet, or with a cover sheet obviously dashed off without much thought at the last minute, will not be graded.

- How many hours would you estimate that you spent on this assignment?

- Explain (in one or two sentences) one thing you learned through doing this assignment.

- What is one thing you think you need to review or study more? What do you plan to do about it?

Question 1 (K&T 6.4)**.** You are running a consulting business, with clients on both east and west coasts. Each month, you can run your business from an office in either New York or San Fransisco. In month i, you will incur an *operating cost* of N_i if you run the business out of New York; you will incur an operating cost of S_i if you run the business out of San Fransisco (the costs can change each month depending on the distribution of client demands).

However, if you run the business out of one city in month i, and then out of the other city in month $i+1$, then you incur a fixed *moving cost* of M to switch cities.

Given a sequence of n months, a *plan* is a sequence of n locations—each one equal to either NY or SF—such that the ith location indicates the city in which you will be based in the ith month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

Given a value for the moving cost M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, the problem is to find a plan with minimum total cost.

(a) Show that the following greedy algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

Algorithm 1: GREEDYPLAN

```
1: for i ← 1 to n :
2:     if N_i < S_i :
3:         Output "NY in month i"
4:     else
5:         Output "SF in month i"
```

In your example, say what the correct answer is and also what the above algorithm finds.

(b) Give an efficient algorithm that takes values for n, M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, and returns the *cost* of an optimal plan. Justify the correctness of your algorithm and analyze its asymptotic running time.

(c) Explain how to extend your algorithm to also find the optimal plan itself, not just its cost.

Question 2. You are building a toy train track out of a sequence of straight pieces laid end-to-end. Some pieces are one unit long, and some are three units long. How many different ways are there to build a track that is five hundred units long?

For example, there are 9 different ways to build a track that is 7 units long, as illustrated below.

Figure 1: The nine ways to build a length-7 train track

Question 3 (K&T 6.3)**.** Let $G = (V, E)$ be a directed, unweighted graph with nodes $v_1, \ldots, v_n$. We say that G is an *ordered graph* if it has the following properties:

(i) Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.

(ii) Every node other than v_n has at least one outgoing edge.

Given an ordered graph G, we want to find the *longest* path from v_1 to v_n.

(a) Consider the following greedy algorithm.

Algorithm 2: GREEDYLONGESTPATH

```
w ← v_1
L ← 0
while w ≠ v_n :
    Choose the outgoing edge (w, v_j) with the smallest j
    w ← v_j
    Increment L
return L
```

Show that this algorithm does *not* correctly solve the problem, by giving an example of an ordered graph for which it does not return the correct answer. Be sure to explain what the correct answer is and what incorrect answer is returned by the algorithm.

(b) Give an efficient algorithm that takes an ordered graph G and returns the length of the longest path from v_1 to v_n. Justify its correctness and analyze its time complexity.

(c) Explain how to modify your algorithm from the previous question to return the longest path itself, rather than only its length.

3(b)

Question 4. This problem is similar to one we did in class, but slightly more general. Suppose we have a set $\{x_1, x_2, \ldots, x_n\}$ of n positive integers, and a target sum S, which is a positive integer. In class, we considered finding a subset of $\{x_1, \ldots, x_n\}$ whose sum is *exactly equal to* S. Instead, let's now consider finding a subset whose sum is *as close to* S *as possible without going over*; that is, the subset with the largest possible sum $\leq S$. For example, given the set $\{1, 2, 7, 12\}$ and the target sum 18, there is no subset which sums exactly to 18, but the one which comes closest without going over is $\{12, 2, 1\}$ which sums to 15. Note that $\{12, 7\}$, with a sum of 19, is closer to the target in an absolute sense, but it is greater than the target; we are interested in the biggest sum which is $\leq$ the target.

(a) Describe a simple brute-force algorithm for solving this problem. What is the running time of the algorithm?

(b) Using dynamic programming, describe an algorithm with running time $\Theta(nS)$. Be sure that you explain how to find not only the maximum possible sum, but also the actual subset which has that sum. Justify the correctness of your algorithm.

4

(c) This is known as a *psuedopolynomial-time* algorithm: the running time is a polynomial in the *value* of S, but actually exponential in terms of the *size* of the input (*i.e.* the number of bits needed to represent S).

Give one example of a set of inputs for which your dynamic programming solution would be faster, and one example of a set of inputs for which the brute force algorithm would be faster.