Question 1. 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?

1

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

1

Question 2 (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 1: X

```
1: w ← v_1
2: L ← 0
3: while w ≠ v_n do
4:     Choose the outgoing edge (w, v_j) with the smallest j
5:     w ← v_j
6:     Increment L
7: end while
8: 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 so that it can also be used to recover the longest path itself, rather than only its length.

Question 3. This problem is similar to one we did in class, but slightly different. 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.

(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.

3

Question 4. Mae T. Ricks has n magnets, with sizes $M_1, M_2, \ldots, M_n$ (each size is a positive real number). At the start, the magnets are all separate; but for Reasons,[1] she wants to stick them all together to make one big "supermagnet". The magnets are heavy and could damage sensitive equipment if they are moved around too much, so they are all mounted on a track where they can slide back and forth. So at each step, she takes two magnets (or clumps of connected magnets which now count as a single "supermagnet") which are next to each other on the track, and slides them toward each other until they stick. She continues this process until all the magnets are stuck together. For example, given magnets $M_1M_2M_3M_4$, she could first stick

[1] Probably because her PhD advisor told her she has to.

M_2 and M_3 together, which we might write as $M_1(M_2M_3)M_4$, indicating that M_2 and M_3 are now stuck together into one "supermagnet". She might then choose to slide (M_2M_3) and M_4 together, resulting in $M_1(M_2M_3M_4)$; finally she slides the two final magnets together, yielding $(M_1M_2M_3M_4)$. Alternatively, she could have first stuck (M_1M_2) together, then (M_3M_4) together, and finally stuck those two resulting supermagnets together. As you can see, although the final product will always be the same, there are many different orders in which Mae could choose to stick the magnets together.

Of course when two magnets with sizes M_i and M_j stick together, they form a new magnet with size $M_i + M_j$. The problem is the noise. Because of very complicated physics that no one understands, when magnets of size M_i and M_j stick together, they make a sound with loudness proportional to the *product* M_iM_j. Naturally, Mae wants to minimize the total noise produced during the process. Design and analyze[2] an efficient algorithm which takes a sequence of magnet sizes $M_1, \ldots, M_n$ as input, and computes the minimum possible noise needed to combine them all.

[2] By "design and analyze" I mean to describe an algorithm, prove its correctness, and analyze its asymptotic running time.