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. 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.

1

(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.

Question 2. Consider the problem of making change for C cents using the fewest possible number of coins. Assume that each coin's value is an integer.

(a) Describe a greedy algorithm to make change for C cents using US quarters, dimes, nickels, and pennies.

It turns out that the greedy algorithm is actually optimal for US coins (and most real-world coin systems), though coming up with a proof of this fact is nontrivial.

(b) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of C.

(c) Design and analyze an algorithm to make change using the fewest number of coins that works for any set of coins. That is, as input your algorithm should take

- n, the number of different coin types;
- a list $c_1, c_2, \ldots, c_n$ giving the values of the different coins (you may assume they are already sorted from smallest to largest); and
- the number of cents C we would like to make change for.

As output your algorithm should either report that it is not possible to make the required amount C using the given coins, or give a multiset[1] of coins which add up to C such that the number of coins in the multiset is as small as possible. For example, if given as input $c_1 = 1$, $c_2 = 5$, $c_3 = 20$ and the target value $C = 47$, your algorithm should output $\{20, 20, 5, 1, 1\}$. Note that we assume there is an unlimited supply of coins of each type. Be sure to justify your algorithm's correctness and analyze its time complexity.

[1] A multiset is like a set that allows duplicate elements.

Question 3 (Derived from K&T 6.6)**.** In a word processor, the goal of loose justification is to take text with a ragged right margin, like this,

```
Call me Ishmael.
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text whose right margin is "as even as possible", like this:

```
Call me Ishmael. Some years ago, never
mind how long precisely, having little
or no money in my purse, and nothing
particular to interest me on shore, I
thought I would sail about a little
and see the watery part of the world.
```

To make this precise enough for us to start thinking about how to write a justifier for text, we need to figure out what it means for the right margins to be "even". Suppose our text consists of a sequence of *words*, $W = \{w_1, w_2, \dots, w_n\}$ where w_i consists of c_i characters. We have a maximum line length of L. We will assume we have a fixed-width font, so we just need to make sure that the number of characters on each line is no more than L.

A *formatting of* W consists of a partition of the words in W into *lines*. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \dots, w_k$ are assigned to one line, then we should have

$$c_k + \sum_{j \le i < k} (c_i + 1) \le L.$$

We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the *slack* of the line—that is, the number of spaces remaining at the right margin. For example, suppose $L = 10$. Then

```
Call me Ishmael.
```

is not valid, since it has length $(4+1)+(2+1)+8$ which is greater than 10. On the other hand,

```
Call me
```

is valid, and has a slack of 3, since it has length only 7, leaving 3 remaining spaces at the end.

We will say that a formatting is optimal when the sum of the *squares* of the slacks of all lines (including the last line) is minimized.

3 Medium hint

(a) Describe a greedy algorithm to find a formatting of a list of words, and give an example where your greedy algorithm does *not* produce an optimal solution.

(b) Using dynamic programming, design and analyze an efficient algorithm to find an optimal formatting of a set of words W into valid lines for a given line length L. (As usual, "analyze" means to prove it is correct, and analyze its asymptotic running time.)

3 Big hint

(c) Why did we use the sum of the *squares* instead of just, say, the sum? That is, what sort of bias does this optimization function create?

Question 4. (**Optional/Extra Credit**)

If you want some practice actually implementing a dynamic programming solution to a problem, write a program that implements your algorithm from Question 3. Your program should take two command-line arguments: (1) an integer representing the maximum line length; and (2) a file name. It should then output a justified version of the file to `stdout` using the algorithm above.

For example, suppose `lorem.txt` contains the text:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque
rhoncus interdum odio, mattis finibus eros imperdiet non. Praesent egestas lectus.
```

Then running your program with the arguments `25` and `lorem.txt` should print

```
Lorem ipsum dolor sit
amet, consectetur
adipiscing elit. Quisque
rhoncus interdum
odio, mattis finibus
eros imperdiet non.
Praesent egestas lectus.
```

which is an optimal formatting of the text into lines of length at most 25.

Also available is a file `neruda.tar` which contains a Pablo Neruda poem together with two outputs: `neruda.50.out` and `neruda.30.out` are the results of running my solution on `neruda` using a line length of 50 and 30, respectively. Note that in both cases, there are multiple correct solutions with the same minimum score. Your program may not produce exactly the same output as mine, but you should ensure that it produces a solution with the same score.