CSC 495 – Problem Set 2 – Due Tuesday, January 24

Problem 2.R1: The Biggest N

Required Problem
Points: 50 points

Background

After analyzing the running time of an algorithm, you can get a formula for the running time that is a function of n, the size of the input. This formula is typically the sum of terms that are either constant, a power of n (like n or n^2), a power of $\log_2 n$ (like $\log_2 n$ or $(\log_2 n)^2$), or – more likely – a product of these (like $1.8n \log_2 n$). For example, the running time of mergesort might be the following (measured in microseconds):

$$1.5n \log_2 n + 4n + 0.2 \log_2 n + 15$$

Note that in an asymptotic algorithm analysis, this would usually just be written as $O(n \log n)$ since that reflects the most significant component of the running time. However, what if you want to use the exact formula[1], and answer the question "what is the largest value of n for which the algorithm runs in less than a second (1,000,000 microseconds)?" Unfortunately, with the formula above, it is impossible to directly solve the resulting equation — you need to find the answer with an algorithm (which is what makes it interesting for us!). The answer to the specific problem given here is $n = 37,337$.

The Problem to Solve

The input will specify a running time formula and a limit on running time. Your program should compute and print the largest input size for which the running time formula evaluate to less than or equal to the running time limit. The running time is the sum of terms, where each term is specified by three values: the constant multiplier, the power of n, and the power of $\log_2 n$. For example, $3n^2$ is given by the three values `3 2 0`. See the sample input for the input representation of the formula given in "Background" above.

There are two guarantees in this problem: first, the running time function is monotonically non-decreasing (that is, for all n the running time for input size $n+1$ is always at least as large as the running time for input size n). Second, you are guaranteed that the answer will fall in the range $1, \ldots, 1,000,000,000$. There will be at most 10 terms in the running time formula. Your program must run in less than 10 seconds to be judged correct (a good solution can easily run in less than a tenth of a second for any input).

[1] Even the more detailed formula is an over-simplification of what happens on real systems these days. Running time is typically not a nice smooth formula like this, and jumps around — primarily due to the effect of caches. See, for example, the actual running times in the graph on page 211 of *Programming Pearls*.

Hints and Techniques

All major programming languages provide a function for computing the natural logarithm, but for this program you need the base 2 logarithm. The important mathematical property to use is that if `log(x)` computes the logarithm of `x` in *any* base, then `log(x)/log(2.0)` is the base 2 logarithm of `x`. You'll also need a powering function — remember that powers are floating point values, so do *not* assume they'll be integers (0.5 is a perfectly valid power).

Input and Output

The input starts with an integer `n` giving the number of terms in the running time formula. Next, for each of the `n` terms there are the 3 numbers that describe the term. Finally, there is the running time limit. The sample input and output given below describe the problem given in the "Background" section.

Sample Input

```
4
1.5 1 1
4 1 0
0.2 0 1
15 0 0
1000000.0
```

Sample Output

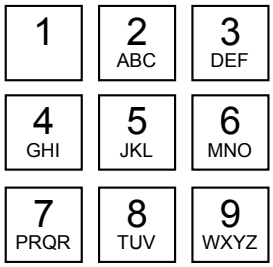
```
37337
```

Problem 2.R2: Phony Words

Required Problem
Points: 50 points

Background

It can be interesting to see what words you can spell using a given phone number. For reference, a standard telephone keypad is shown below (missing the bottom row, which isn't important for this problem).



For example, if your phone number were 266-7453, you could spell "compile." In this problem we're interested in only two ways of making words from phone numbers: either a single 7-letter word, or the first digit of the number followed by a 6-letter word.

The Problem to Solve

Your program should read in a list of 6 and 7 letter words, followed by a list of phone numbers, and print out a report for each number that either says that no words are possible, or a list of possible words. The words don't need to be given in any particular order. The number of both lists can be large: up to 100,000 entries each.

In order to simplify this problem, you are guaranteed that none of the phone numbers contain any 0 or 1 digits, so all digits map to a set of letters. Each number will be given as a 7-digit value, with no dash or other characters. Each dictionary word will be 6 or 7 lower case characters.

Hints and Techniques

For this problem you can either implement some simple data structures yourself, or use data structures that are provided as part of standard libraries. If you use an appropriate library data structure then your coding time could be significantly reduced.

It is a good idea to create some large test cases so you can judge the speed of your solution — you may need to write a test case generation program for this purpose. Note that if you try to match all numbers with all words, you're talking about 100,000 times 100,000 possible matches — think about that!

Input and Output

The sample input and output shown below demonstrate the input and output format, but the samples are much smaller than the actual test data. Test data will include a full dictionary of 6 and 7 letter words (over 68,000 words) and a large list of phone numbers (up to 100,000 numbers). The input starts with a number n, followed by n words, which are followed by another number m and then m phone numbers.

Your output should follow the format below *exactly*. Each number should start with a line that says "`For number xxxxxxx:`," and if no matches are found you should print "`No words found`" *on that same line*. If you do find numbers, print them below the "`For...`" line, indented by exactly one space. For 6 letter words, print the first digit of the number first, followed by a dash, and then the 6 letter word. When I say *exactly* like this, I mean it: Make sure your output doesn't have any extraneous spaces anywhere, like at the ends of lines.

Sample Input

```
9
unsafe
angolar
ninjas
bridger
ridger
sieger
abanet
killer
awesome
4
4867233
2646527
9559799
2743437
```

Sample Output

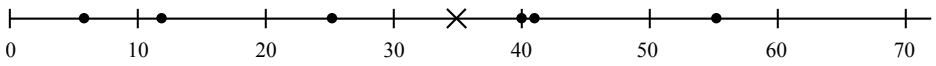
```
For number 4867233:
 4-unsafe
For number 2646527:
 angolar
 2-ninjas
For number 9559799: No words found
For number 2743437:
 bridger
 2-ridger
 2-sieger
```

Problem 2.C1: Danger, Will Robinson!

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Background

In this problem, we consider points on a line, where there are "danger points" and a "test point." The "safety" of a test point from a single danger point is simply the distance to the danger point, and the overall safety measure for a set of danger points is just the sum of all the individual distances. For example, consider the following illustration, where dots indicate danger points (at 6, 12, 25, 40, 41, and 55), and a test point (the "x") at 35.



Point	Distance
6	29
12	23
25	10
40	5
41	6
55	20

Adding up the distances in the righthand column, there is a total "safety measure" of 93 for the test point at location 35.

The Problem to Solve

In this problem, you are given a range to consider, and a set of danger points in that range. You are also given a lower bound on the allowable safety measure, and you need to compute the number of possible integer-coordinate points that are too dangerous (i.e., have safety measure less than the minimum allowable safety measure). The lower bound of the range will be at least 0 (so all coordinates are non-negative), and the upper bound can be as large as one trillion. Danger points are given in no particular order, and can be anywhere in this range. There will be at most 1000 danger points. Your program should run in less than 10 seconds.

For a particularly easy example, consider the range 0..40, two danger points at 10 and 20, and allowable safety value of 14. The safety measure for location 8 is exactly 14, as is the measure at location 22. All points between 9 and 21 (inclusive) have safety measure less than 14, so are too dangerous — therefore the number of points that are too dangerous is 13.

Hints and Techniques

If you picture the numbers on a number line (like in the picture in "Background") think about what happens as the test point moves from left to right through the range — can you predict whether the danger measure will increase, decrease, or remain the same based on where it is?

Input and Output

The first two numbers in the input are the endpoints of the range to consider, where the range is inclusive (so in the sample input below, the range is 0, ..., 80, including 0 and 80). This if followed by a number n that gives the number of danger points, and then n numbers (the danger point locations), and finally a single number that gives the lower bound on acceptable safety.

Your output should consist of a single number: the number of (integer coordinate) points that are unsafe because they have a safety measure less than the safety lower bound.

Sample Input

```
0 80
6
25
41
12
55
6
40
127
```

Sample Output

```
38
```

Problem 2.C2: Scrambled Letters

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Background

Anagram finders can be fun: For example, did you know that "dormitory" and "dirty room" are anagrams? Column 2 in *Programming Pearls* talks about finding anagrams when dealing with single words, but what if you want to consider anagrams that use multiple words (like "dirty room")?

The Problem to Solve

Your program will be given a dictionary (a list of words, one word per line), followed by a list of test phrases (one or more words per test phrase). Your program should find all anagrams of the test phrase that you can make using one or more of the words from the provided dictionary. Spaces and punctuation are not significant, and in order to make things a little easier you only have to consider anagrams that use three or fewer words. There can be as many as 3,000 words in the dictionary, and your program should be able to find all anagrams of any test phrase in under 10 seconds.

Hints and Techniques

The challenge here is making this run in a reasonable amount of time. There are too many three word combinations to allow for trying all combinations, so you need to come up with a method for significantly reducing your search space. Think about "signatures" as described in *Programming Pearls*, although you can't use the exact signature that Bentley gave for one word anagrams (or at least, I don't see how you could...).

Input and Output

Input consists of a number n, followed by n words for your dictionary, and that is followed by a number m of test phrases followed by that many phrases. Dictionary words will be all lower case letters, but test phrases can includes spaces or other non-letter characters (which should be ignored for the purpose of anagrams), but all letters are guaranteed to be lower case.

Output should be formatted exactly as shown below, where the output for each test phrase consists of a header line (which is the only line if no anagrams are found), followed by all possible anagrams, with each one indented by a single space, and multiple words separated by a single space.

Sample Input

```
13
alarms
come
dirty
dots
extra
here
one
plus
room
snooze
twelve
useless
word
6
alas!  no more zs.
dormitory
the morse code
baffling problem
eleven plus two
help me
```

Sample Output

```
Anagrams for "alas!  no more zs.":
 snooze alarms
 alarms snooze
Anagrams for "dormitory":
 dirty room
 room dirty
Anagrams for "the morse code":
 here come dots
 here dots come
 come here dots
 come dots here
 dots here come
 dots come here
Anagrams for "baffling problem": None found
Anagrams for "eleven plus two":
 plus one twelve
 plus twelve one
 one plus twelve
 one twelve plus
 twelve plus one
 twelve one plus
Anagrams for "help me": None found
```