CS 4/512 Exercise 1
XKCD Password Cracker

1. (5 points) (Don't just answer yes or no – provide some reasoning for your answer. Consider some of the ideas in Notes 1.11, 1.14, 1.16, etc.).
    a. How should you compare your concatenated-word-possible password to the actual password? Character by character, using the builtin string compare gadget?
        i. Since we know the words are going to be from a set selection of words, we can compare strings to strings and simply replace the words to compare and contrast rather than compare character to character (i.e. instead of "do a and all match? Well a and a match, so the next character is [ ] and l, so no they don't!" doing "do a and all match? Well the strings are not equivalent so no they do not."). Using this we can change out the words we're comparing to check with (aa to aat to aabout etc.).
    b. Would an array of char be better than a string? What about mapping your char array as an uint array of ¼ length?
        i. Mapping a char array as a uint array of ¼ length is an interesting idea because it can reduce the overall memory footprint and potentially speed up comparisons. By packing multiple char values into a single uint, you can leverage the CPU's ability to compare larger chunks of data at once. However, this would add complexity to the code and is typically only worth it if needing to optimize for very high performance in low-level systems.
    c. Should you sort the list of words by use frequency or length?
        i. Sorting the list of words by frequency of use is generally more effective for password cracking. Most passwords are based on common words or phrases, so starting with the most frequently used words increases the likelihood of quickly finding the correct password. Sorting by length could be useful in some contexts, particularly if the approximate length of the password is known, but in general, frequency is more reliable for prioritizing guesses.
    d. How might you use multiple cores to improve speed and by how much? How else might you improve your cracker's speed?
        i. Using multiple cores can significantly improve the speed of your password cracker by parallelizing the task. You could divide the list of possible word combinations among the cores, allowing each core to work on a subset of the problem simultaneously. The speedup depends on the number of cores and how well the problem can be divided. For example, with 4 cores, you might expect up to a 4x speedup, though in practice, the gain will be slightly less due to overhead from managing the parallel tasks.

2. (5 points) Compare the computational complexity of your dictionary password cracker vs a character password cracker that consist of only lowercase characters between 'a' and 'z'. What independent variable or variables need to be used for determining computational complexity of a dictionary cracker?
    a. **Character Password Cracker:**

For a brute-force character password cracker that tries every possible combination of lowercase letters ('a' to 'z'), the complexity is O(26^L), where L is the length of the password. This is because there are 26 possible characters for each position in the password, and you have to check all possible combinations.

   b. **Dictionary Password Cracker:**
      i. For a dictionary-based password cracker, the complexity depends on two main factors:
         1. The number of words in the dictionary.
         2. The number of words combined to form the password (e.g., 1-word, 2-word combinations).

The complexity is roughly O(N^L), where N is the number of words in the dictionary, and L is the number of words combined. This is because the cracker might need to try every combination of L words from the dictionary.

   c. Independent Variables for Computational Complexity of a Dictionary Cracker:
      i. N (Number of Words in the Dictionary): As the dictionary size increases, the number of combinations grows exponentially.
      ii. L (Number of Words in the Password): The more words combined to form the password, the greater the number of combinations that must be checked.

In summary, a character password cracker has complexity O(26^L), while a dictionary password cracker has complexity O(N^L), with N being the size of the dictionary and L the number of words in the password.

3. How do the runtimes change if you allow for the use of the 500 most commonly used words in English
   a. If you allow for the use of the 500 most commonly used words in English, the runtime complexity of the dictionary password cracker becomes O(500^L), where L is the number of words combined to form the password.
   b. Since N = 500 here, the cracker has fewer combinations to try, so the runtime will be faster compared to using a larger dictionary.
4. How do the runtimes change if you allow the use of the 1000 most commonly used words in English
   a. If you allow for the use of the 1000 most commonly used words in English, the runtime complexity becomes O(1000^L).
   b. Here, N = 1000, meaning the cracker has more combinations to check, so the runtime will be slower compared to using only 500 words.

5. (5 points) If a nefarious individual from Pottsylvania was trying to crack one of <u>your</u> passwords, how might he select a better list of words for his dictionary attack? Given this, how might you select better words for your passwords and otherwise improve them while still making them easy to remember?
   A. A better dictionary may include the top 100 used words in multiple languages, or the top 1000 words instead of just the 100. He could also look into social media, familial trees, ect to get more specific information on what words I use, languages I speak, and things of the like.

B. Use multiple different languages, more words such as a phrase instead of just random words. Like a line you like from a song that isn't well known, for example "tellmehowyouresleepingeasy" (Wolf in Sheep's Clothing by Set It Off) instead of "poursomesugaronme" (Pour Some Sugar On Me by Def Leppard), and then alter some words to be in a different language that you're familiar with. I've studied French for years so using that in the previous example it may become "ditsmoihowyouresleepingtranquillement" which has certain words translated from English to French, so a single language dictionary would be incapable of cracking the password, while for me it would be easy to remember as I think in both English and French regularly because I've been studying it for year.

6. (5 points) Farmer Pete wants to plow his 100-acre field. He has his old tractor, his new tracker, and his plow team of Belgians, The new tractor plows at 10 acres per hour, the old tractor at 4 acres per hour, and the Belgians at 1 acre per hour. He starts plowing at 5:25 am and plows for 1 hour and 14 minutes with the new tractor before it breaks down. He spends 5 minutes trying to fix it, 6 minutes using loud socially unacceptable expletives, and 9 minutes walking back to the barn, firing up the old tractor, and starting to plow again. Meanwhile, Granny Gertrud, on hearing the very loud expletives, takes 10 minutes to hitch up the Belgians and start to plow. Precisely how long, in total from start time to end time, does it take Farmer Peter and Granny Gertrud to plow the field? If the above story problem is missing information or has any ambiguities, in addition to the total plow time, develop precise statements of what clarification assumptions you need to make in order to precisely determine the total plow time.

Problem Breakdown and Reasoning:

1. **Initial Plowing with the New Tractor:**
   - **New Tractor Speed:** 10 acres per hour.
   - **Plowing Time with New Tractor:** 1 hour and 14 minutes (which is 1.2333 hours).
   - **Acres Plowed by New Tractor:**

$$10 \text{ acres/hour} \times 1.2333 \text{ hours} = 12.333 \text{ acres}$$

2. **Time Spent After New Tractor Breaks Down:**
   - **Fix Attempt:** 5 minutes (0.0833 hours).
   - **Expletives:** 6 minutes (0.1 hours).
   - **Walking to Barn and Starting Old Tractor:** 9 minutes (0.15 hours).
   - **Total Time After Breakdown:**

$$0.0833\, hours + 0.1\, hours + 0.15\, hours = 0.3333\, hours$$

3. **Note:** During this 20 minutes, no plowing occurs.
4. **Old Tractor and Belgians Start Plowing:**
   - **Old Tractor Speed:** 4 acres per hour.

- **Belgians Speed:** 1 acre per hour.
- **Combined Speed of Old Tractor and Belgians:**

$$4 \text{ acres/hour}+1 \text{ acre/hour}=5 \text{ acres/hour}$$

5. **Remaining Acres to Plow:**
    - **Total Field Size:** 100 acres.
    - **Acres Already Plowed by New Tractor:** 12.333 acres.
    - **Remaining Acres:**

$$100\, acres\ -\ 12.333\, acres\ =\ 87.667\, acres$$

6. **Time to Plow Remaining Acres:**
    - **Combined Speed (Old Tractor + Belgians):** 5 acres per hour.
    - **Time Required to Plow Remaining Acres:**

$$\frac{87.667 acres}{5\, acres/hour}\ =\ 17.5333\, hours$$

7. **Total Time from Start to Finish:**
    - **Time Spent Plowing with New Tractor:** 1.2333 hours.
    - **Time Spent After Tractor Breakdown (Fixing, Expletives, Walking):** 0.3333 hours.
    - **Time Spent Plowing with Old Tractor and Belgians:** 17.5333 hours.
    - **Total Time:**

$$1.2333\, hours\ +\ 0.3333\, hours\ +\ 17.5333\, hours\ =\ 19.1\, hours$$

Clarification Assumptions:

1. **Simultaneous Plowing:** It is assumed that once Granny Gertrud starts with the Belgians, she and Farmer Pete plow simultaneously, combining their speeds.
2. **Non-overlapping Start Times:** The time taken by Farmer Pete to try fixing the new tractor, use expletives, and switch to the old tractor happens before the Belgians start plowing.
3. **Precision:** All times are considered precise to the nearest minute or fraction of an hour.
4. **Sensitivity:** The Belgians being referred to are the Horses and not the People, in turn making the problem entirely above board and appropriate.

Final Answer:

- **Total Time from Start to Finish: 19.1 hours**.
- This includes all downtime and the combined plowing efforts of the old tractor and the Belgians.