CS 5720 Design and Analysis of Algorithms

Project #1

Submission requirements:

- A .zip file containing your source code. You may use any language you would like.
- A PDF (submitted separately to the Canvas assignment) containing each item below that is listed as a **Deliverable.** For each item contained in your PDF, clearly mark which deliverable it is associated with. Plots should be clearly labeled and have descriptive captions.

Exploring Worst, Best, and Average case. In class, we discussed the concept of Worst, Best, and Average case time complexities from the standpoint of order of growth. In this project, you will explore these concepts empirically. It is my hope that the empirical study and the theoretical work from class and homework will mutually reinforce each other and help you come to a more complete understanding of the concepts.

Your task: Create a search algorithm `Search(A[n],K)` that searches for a character in a character array. The algorithm takes two inputs:

- `A[n]`: A character array of length n, and
- `K`: A character (we call this the "search key").

The algorithm's output is an integer:

- Output: the lowest index in `A[n]` in which `K` appears, or `n` if `K` is not found in `A[n]`.

Use a simple brute-force algorithm for this task which begins at the front of the array and walks down the array until the key is found. I.e., first check if `K` is in `A[0]`; then check `A[1]`, etc.

Assignment:

1. Implement your search algorithm in a language of your choice. Test your implementation thoroughly for correctness (e.g., use your programming language's built-in search algorithm and make sure it agrees with your algorithm on every input). Provide verification that you have tested your algorithm and that it always operates correctly.

   Deliverable 1: Your search algorithm code and your verification of correctness.

2. The speed at which your algorithm searches will depend a lot on the contents of `A[n]` and the likelihood of finding the particular `K`. In this project, you'll use English text and check how long it takes to find the four characters `e, m, Q, %` in various length arrays. To do this, you'll first have to get some English text. There are many ways to do this; one simple way

would be to pick your favorite book(s) from Project Gutenburg and then use a technique like the first answer to this Stackoverflow question to extract the text from the HTML page.

Once you've found a source of English text, you need to extract a set of character arrays from the text which you'll use for the algorithm experiment. Your dataset should have at least 10 values of n, and each value of n should have at least 50 arrays which you've extracted from your English text source. For example, you could construct a dataset which looks like this (for n taking the first 10 multiples of 1000):

- 50 arrays of length 1000,
- 50 arrays of length 2000,
- 50 arrays of length 3000,
- ...
- 50 arrays of length 10000.

You can modify the exact values as much as you like, **as long as each value of n has no fewer than 50 arrays, and at least 10 values of n.**

When generating data for an experiment, it's **critical** that you don't introduce spurious biases in the data by the way you construct your dataset. In this case, it means that you must not "put your finger on the scale" when it comes to which characters appear in which length arrays. One straightforward way to do this is to **ensure that no text from your source ever appears in more than a single array in your dataset.**

Deliverable 2: A precise description of your dataset and the code which you used to generate it. Reproducibility is crucial here: if I try to replicate your results, I should be able to do it with minimal effort.

3. For each character in the test set (that is, `e, m, Q, %`), generate a plot showing the **worst-case** runtime *in your dataset.* For "runtime" in this and following deliverables, simply use the return value of the algorithm (this will correlate well to runtime).

   Following my example above, this means you'll search for `e` in each of your 50 length-1000 arrays, record the worst runtime you found among those (this is your $n = 1000$ datapoint); then you'll search for `e` in each of your 50 length-2000 arrays, record the worst runtime you found (this is your $n = 2000$ datapoint); and so on to create a single plot trace describing the worst-case runtime for finding `e` as a function of n in your dataset. You'll then repeat that procedure for each of the remaining test characters `m, Q, %`.

   Deliverable 3: A plot showing the **worst-case** empirical runtime of your algorithm as a function of n, with a separate trace for each of the test characters `e, m, Q, %` (note: this means that n is on the horizontal axis, and runtime is on the vertical axis). Based on these plots, form a conjecture about the worst-case runtime of your algorithm. If your plots show any unexpected or strange behavior, discuss.

4. For each character in the test set (that is, `e, m, Q, %`), generate a plot showing the **best-case** runtime *in your dataset.*

   Following my example above, this means that you'll search for `e` in each of your 50 length-1000 arrays, record the best runtime you found among those, and so on for each value of n. You'll then repeat that procedure for each of the remaining test characters `m, Q, %`.

Deliverable 4: A plot showing the **best-case** empirical runtime of your algorithm as a function of n, with a separate trace for each of the test characters `e, m, Q, %`. Based on these plots, form a conjecture about the best-case runtime of your algorithm. If your plots show any unexpected or strange behavior, discuss; in particular, compare these results to your Deliverable 3 and note any similarities or discrepancies.

5. For each character in the test set (that is, `e, m, Q, %`), generate a plot showing the **average-case** runtime *in your dataset.*

   Following my example above, this means that you'll search for `e` in each of your 50 length-1000 arrays, record the average runtime you found among those, and so on for each value of n. You'll then repeat that procedure for each of the remaining test characters `m, Q, %`.

   Deliverable 5: A plot showing the **average-case** empirical runtime of your algorithm as a function of n, with a separate trace for each of the test characters `e, m, Q, %`. Based on these plots, form a conjecture about the average-case runtime of your algorithm. If your plots show any unexpected or strange behavior, discuss; in particular, compare these results to your Deliverable 3 and 4 and note any similarities or discrepancies.

Extra credit: If your source of English text is different from everyone else's in the class, I'll give you a few bonus points.

A note: If you would like to organize the plots differently from how I've requested, (for example, place all `e` traces on one plot, then place all `m` traces on one plot, etc), you are welcome to do so **provided you give a convincing argument about why you did.** This is not a free pass to submit plots which make no sense!