CS 5720: Design and Analysis of Algorithms
Project 1 Report

Rachel Koch

October 6, 2024

1 Introduction

The purpose of this project is to empirically analyze the time complexity of a brute-force search algorithm in terms of its worst-case, best-case, and average-case performance. The analysis is conducted using various sizes of character arrays derived from an English text and the algorithm is tested for different search characters (‘e‘, ‘m‘, ‘Q‘, and ‘%‘). We provide insights into the runtime of the algorithm and conjecture about the algorithm's complexity based on the empirical results.

2 Attributions

ChatGPT file changes: *second_code.py*

Overall Code Changes
Changed the name of the file
Imported sys, numpy, and os packages, removed time
Reorganized code so it is in order of deliverable
Removed 'main' if-statement
Changed char array sizes because my text is shorter
Renamed the Search function and the arguments
Used a different book and url
Removed save_text_to_file function, put code direct in body
Added logic to separate new best/worst/average dicts by char
Results are graphed differently based on character

New Functions
Added a newCharArray function to create arrays of random chars
Added a newChar function to generate a random character
Added a verificationTest function to test my search algorithm
Added wrapper for Python '$A[n]$.index' so it returns n for 'not found'
Added a plot_result function to plot only one case on one graph

File Management
VerificationTest, datasets, and experiment results get sent to a file
Added logic to remove results and dataset files if they exist already
Added success messages for data and graphs written to files

download_text() function
Changed Start and End indices to match my book choice
Removed a line that only kept alphabetic and whitespace characters
Added code to remove carriage returns and newlines

generate_datasets() function
Char array fill starts at beginning of text and pulls in chunks

run_experiments() function
Changed time measurement to the index returned by Search()

plot_results() function
Changed the y-axis label to just 'runtime', removed seconds
Changed the color of the lines to be more intuitive

ChatGPT file changes: *second_latex.tex*

Changed the name of the file
Added listings package
Added clearly delineated sections for deliverables
Section for Deliverable 1 shows algorithm and tests
Array sizes and book title changed in Deliverable 2 section
Added a reason for organizing results/plots by character
Result plots were all replaced
Result sections and conclusions were reorganized by character
Added an analysis of the results for each character

3 Deliverable 1: Search Algorithm

3.1 Algorithm

```
def Search(A, K):
    for i in range(len(A)):
        if A[i] == K:
            return i
    return len(A)
```

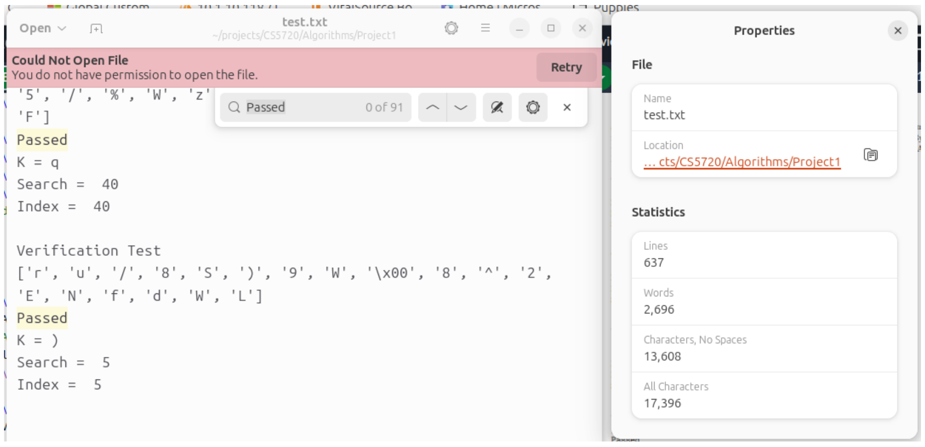
A brute-force search algorithm that takes a character array $A[n]$ and a search key K. The algorithm returns the lowest index in A where K appears or n if K is not found.

3.2 Verification

Verification test results can be found in a file named *test.txt*. Each test generated an array of random characters and compared the search results of both algorithms, indicating a pass/fail. Testing showed that the Search algorithm results and the Python list.index() function results matched in every case. By the end of this project, I had performed over 90 tests and they all passed. A highly curated version of the results are shown below.

```
Verification Test
['S', '*', 'u', '8', 'r']
Passed
K = 8
Search = 3
Index = 3

Verification Test
[ ]
Passed
K = r
Search = 0
Index = 0

Verification Test
['P', 'J', 'w', 'l', 'z', 'Z', 'V', 'm', '?', 'Z', 'h', '!', 'A', 'e',
'b', 'h', 'x', '(', 'q']
Passed
K = !
Search = 11
Index = 11

Verification Test
['q', 'F', 'V', '5', 'h']
Passed
K = X
Search = 5
Index = 5

Verification Test
['\', 'n', 'y', 'g', '/', '6', '^', '^', 'j', 'T', 'g', '(', 'e', 'R',
'r', 'm', 'G', 'Y', 'n']
Passed
K = y
Search = 2
Index = 2
```

Additional test results are appended to the original *test.txt* file every time the program is run, so testing can be independently verified. Screenshots are included to show that the file contained 91 passed tests and no failed tests.



Could Not Open File
You do not have permission to open the file.

```
'5', '/', '%', 'W', 'z'
'F']
Passed
K = q
Search =  40
Index =  40

Verification Test
['r', 'u', '/', '8', 'S', ')', '9', 'W', '\x00', '8', '^', '2',
'E', 'N', 'f', 'd', 'W', 'L']
Passed
K = )
Search =  5
Index =  5
```

Properties

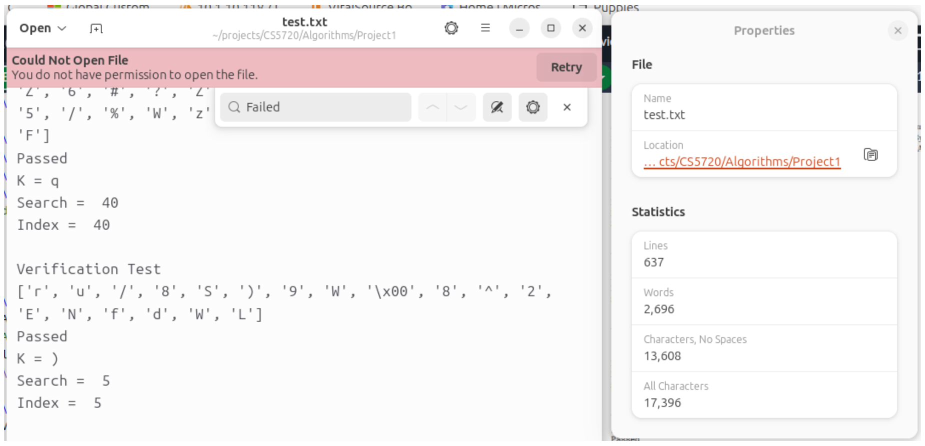
File

Name	test.txt
Location	... cts/CS5720/Algorithms/Project1

Statistics

Lines	637
Words	2,696
Characters, No Spaces	13,608
All Characters	17,396

Figure 1: Screenshot showing 91 passed tests.



Could Not Open File
You do not have permission to open the file.

```
'5', '/', '%', 'W', 'z'
'F']
Passed
K = q
Search =  40
Index =  40

Verification Test
['r', 'u', '/', '8', 'S', ')', '9', 'W', '\x00', '8', '^', '2',
'E', 'N', 'f', 'd', 'W', 'L']
Passed
K = )
Search =  5
Index =  5
```

Properties

File

Name	test.txt
Location	... cts/CS5720/Algorithms/Project1

Statistics

Lines	637
Words	2,696
Characters, No Spaces	13,608
All Characters	17,396

Figure 2: Screenshot showing 0 failed tests.

4 Deliverable 2: Dataset Generation

To generate datasets for the empirical analysis, I used a public domain English text (”Folk Tales Every Child Should Know” by Hamilton Wright Mabie) downloaded from Project Gutenberg. The text was split into ten character arrays of varying lengths (100, 175, 250, ..., 775). For each array of length n, 50 arrays were generated.

5 Deliverables 3, 4, 5: Results

Plots were split by character, and not by the worst/best/average case run-times, because it was easier to see and understand the patterns. For example, the best cases for characters 'e' and 'm' are so close that it was difficult to see their slight differences when graphed together. In another example, characters 'Q' and '%' have the same worst/best/average cases. This led to the appearance of only one line when multiple were graphed together. In fact, the graphs for 'Q' and '%' will be split into three graphs each (worst/best/average), so it can be clearly seen that all cases have the same run-time. The conclusion will include conjectures about the best, worst, and average case runtime of my algorithm.

5.1 Character: *e*

Figure 1 shows the plot for best, worst, and average-case run-times of the test character 'e' in our Search algorithm.

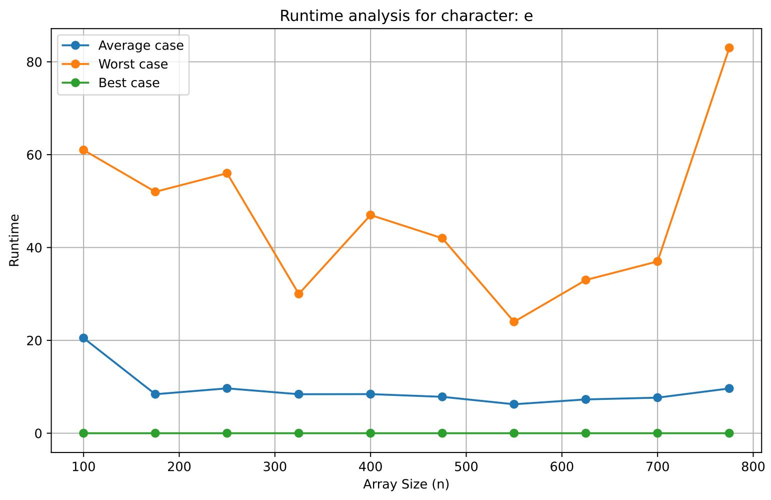


Figure 3: All run-times for character 'e'.

Worst-Case The worst-case line in Figure 3 never reaches the full length of the array, so this means that 'e' was found at least once in every array across all length n arrays. This worst case is $O(n)$ because all the data points are some constant multiplied by the array length n ($\frac{n}{2}$, $\frac{n}{3}$, etc.).

Best-Case As can be seen in Figure 3, the best-case runtime for 'e' was always zero. This means that there was always an array, in every set of length n, where 'e' was the first character in the array. This best case is $O(1)$ -

constant time - which matches what we found in the worst-case: character 'e' is a letter that occurs many times in this text.

Average-Case The average case in Figure 3 is clearly between the best and worst cases, and skews towards the lower end. This is not surprising as 'e' is a very common character on English, and will almost always be found quickly (constant time) for large amounts of text.

5.2 Character: *m*

Below is the plot for the best, worst, and average-case run-times for the test character 'm'.

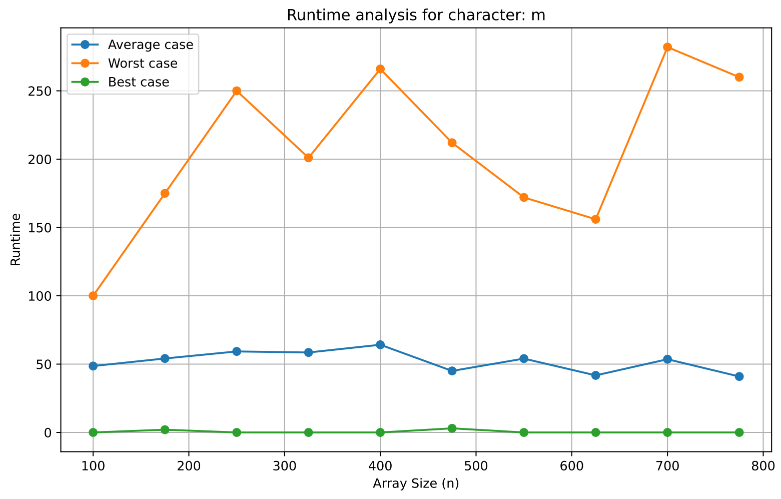


Figure 4: All run-times for character 'm'.

Worst-Case The worst-case line in Figure 4 shows that 'm' was not always found. It was sometimes found - most often in the larger arrays - and sometimes not - less often in the smaller arrays. Given that 'm' is a less common character in the English language, these results are not surprising. This worst case is $O(n)$ because all the data points are some constant multiplied by the array length n (n, $\frac{n}{2}$, $\frac{n}{3}$, etc.).

Best-Case As can be seen in Figure 4, the best-case runtime for 'm' hovered around zero. It is not as common as 'e', but it was still found in the first few characters of at least one length n array per set. This best case is also $O(1)$ - constant time - a strong improvement on the worst-case situation.

Average-Case The average-case line in Figure 4 shows that the character 'm' *was* found in the array, on average. This line falls pretty squarely in between the best and worst cases and seems to remain mostly constant, so the average-case in this situation looks like $O(1)$.

5.3 Character: *Q*

Below are the plots for the best, worst, and average-case run-times for the test character 'Q'.

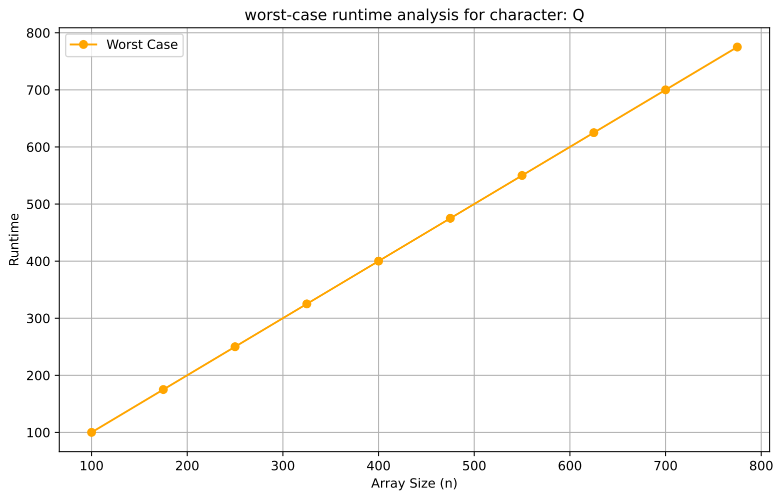


Figure 5: Worst-case runtime for character 'Q'.

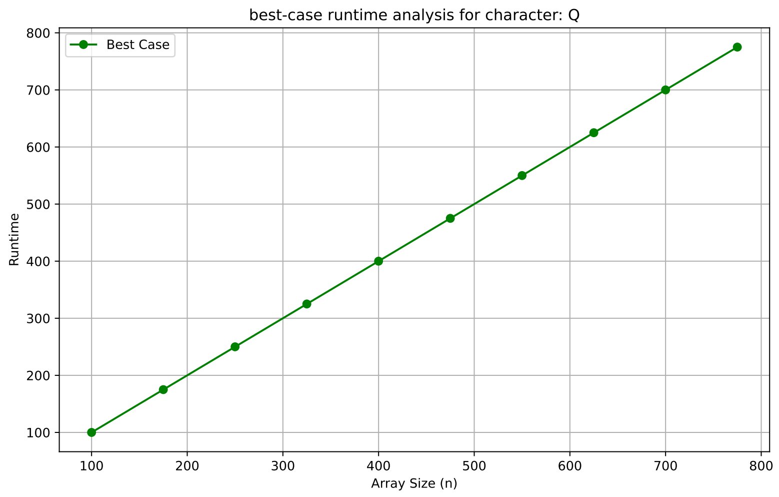


Figure 6: Best-case runtime for character 'Q'.

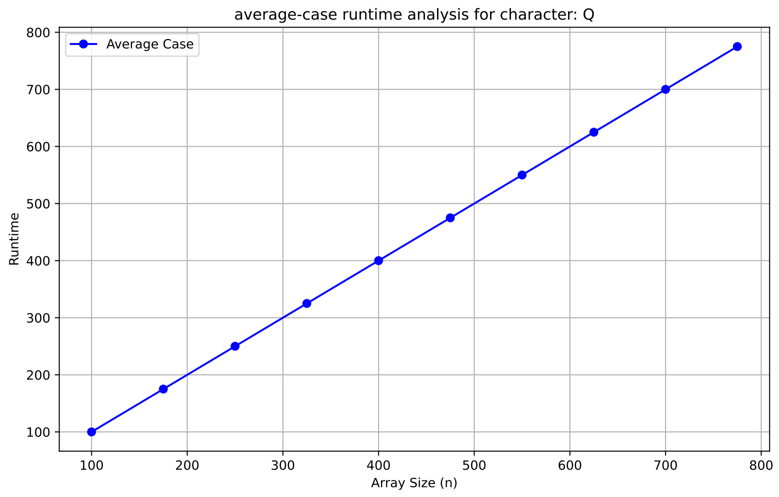


Figure 7: Average-case runtime for character 'Q'.

Worst-Case The worst-case line in Figure 5 shows that 'Q' was not found in at least one array across all sets of arrays of length n. This is also a very clearly linear line, so the order of growth for the worst-case is $O(n)$.

Best-Case As can be seen in Figure 6, the best case in every instance was equal to the length of the array. This means that the character 'Q' was never found in any of the arrays. Therefore, the best case for character 'Q' is $O(n)$. Knowing that the worst-case (Figure 5) is also n in every instance, this leads us to hypothesize that the average-case will also be $O(n)$. These results are not the same as was found for characters 'e' and 'm', which is not surprising because 'Q' is a much less common letter.

Average-Case The average-case line in Figure 7 confirms again that the character 'Q' was never found in any of the arrays. Therefore, the runtime of any algorithm searching for the character 'Q' in this text will equal the size of the text. This means that the average-case is also $O(n)$, as we hypothesized in the previous cases.

5.4 Character: %

Below are the plots for the best, worst, and average-case run-times for the test character '%'.

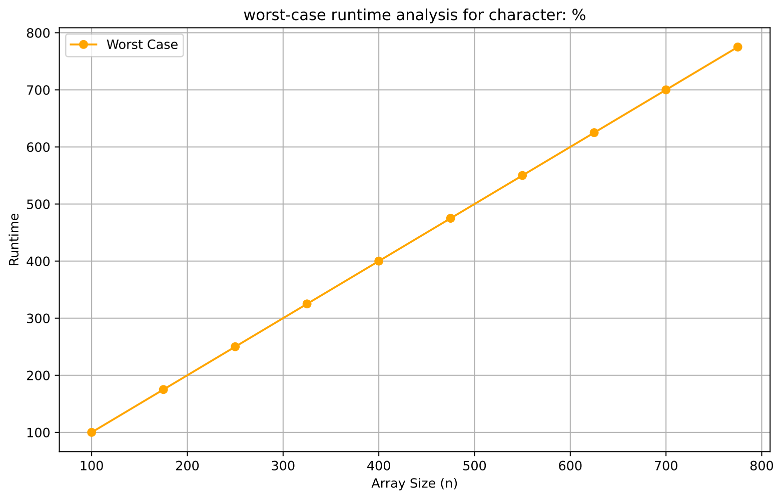


Figure 8: Worst-case runtime for character '%'.

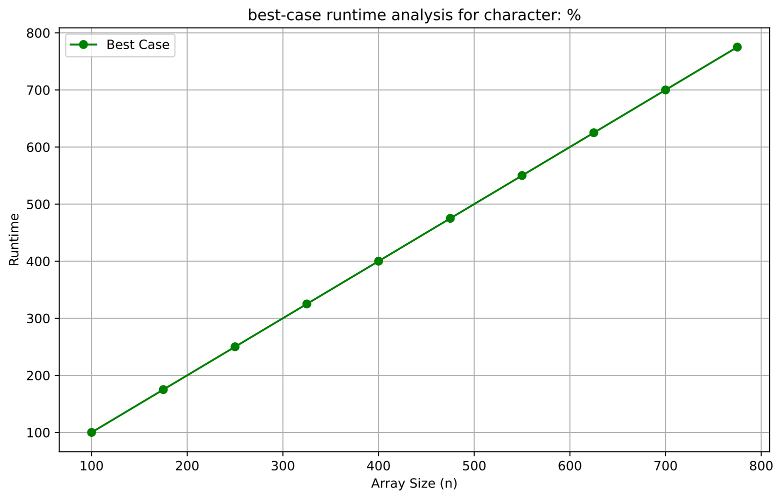


Figure 9: Best-case runtime for character '%'.

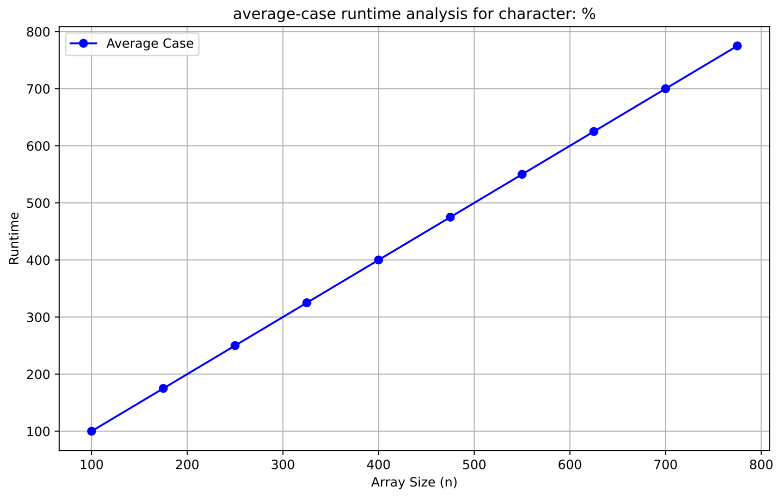


Figure 10: Average-case runtime for character '%'.

Worst-Case The worst-case line in Figure 8 shows that '%' was not found in at least one array across all sets of arrays of length n. This is also a very clearly linear line, so the order of growth for the worst-case is $O(n)$. These results are exactly what was found for character 'Q'.

Best-Case As can be seen in Figure 9, the best case in every instance was equal to the length of the array. This means that the character '%' was never found in any of the arrays. Therefore, the best case for character '%' is $O(n)$. Knowing that the worst-case (Figure 8) is also n in every instance, this leads us to hypothesize that the average-case will also be $O(n)$. These results are, again, exactly what was found for character 'Q'.

Average-Case The average-case line in Figure 10 confirms again that the character '%' was never found in any of the arrays. Therefore, the runtime of any algorithm searching for the character '%' in this text will equal the size of the text. This means that the average-case is also $O(n)$, as we hypothesized in the previous cases. These results match what was found for character 'Q'.

6 Conclusion

This project provided an empirical analysis of a brute-force search algorithm. The worst-case, best-case, and average-case run-times were measured for different search keys across varying array sizes. Given that big-O is the upper bound and big-Ω is the lower bound, our analysis above confirms that the time complexity of our Search algorithm is consistent with the theoretical time complexities:

- Worst-case runtime: $\Omega(n) \leq Search \leq O(n) = \Theta(n)$
- Best-case runtime: $Search \in O(1)$
- Average-case runtime: $O(n)$

The results reinforce our understanding of brute-force search algorithms and their performance characteristics.