Project Analysis

List Heuristics

Gitesh Agarwal

Mohamad Saleh

Jason Benckert

# Worst-Case Runtime

The worst case runtime for each heuristic, in terms of the total number of words is displayed below. For each heuristic, the worst case runtime would be lowest number of words that each one can process in a certain amount of time. The graph displays the total words per millisecond for each. The second graph displays the distinct words per millisecond for each heuristic. The binary search heuristic and the map words heuristic are the best, in terms of runtime, out of the six heuristics we implemented, for both the total words and the distinct words instances. The sort at end heuristic has the worst runtime in both instances, as it processes less words per millisecond than the other heuristics in their worst cases.

**Graph 1: Worst-Case Runtime – Total Words**

**Graph 2: Worst-Case Runtime – Distinct Words**

# Analysis

Below are two linear graphs presenting the runtime of each of the six heuristics, including “Base” and “Naïve” for comparison, with each of our five text files. The first graph displays a comparison between the total number of words in a document and the number of words per millisecond that each heuristic was able to process. The second graph displays a comparison between the total number of unique words in a document and the number of words per millisecond. In both graphs, a higher number of words per millisecond represents a higher processing speed, since the heuristic was able to process more words per millisecond than another.

In both graphs, we can see an interesting trend. The “Map Words” heuristic is by far the best heuristic, as it is able to process more words per millisecond than any other heuristic. Also, it can consistently process more words per millisecond as the number of words in a text file increases, and there is little change in the shape of the line between total words and unique words. The “Binary Search” heuristic is also one of the best, as it can also process a large number of words per millisecond. However, as both of the graphs show, it is inconsistent as the word count increases. According to our data analysis, this heuristic performs better in a setting where there are a small number of unique words. This makes sense ideally since fewer unique words means a smaller database to search through to update the word count. One of the worst heuristic is implemented in this program is the “Sort at End” heuristic, which has the lowest processing speed at low word counts. On the other hand, once we start processing larger files, the “Sort at End” heuristic performs better than the “Move To Front”, “Access Count”, and “Transpose” heuristics. These three heuristics save similar runtimes with regards to how many words each one can process in a limited time, and perform the best with a moderate amount of words (approximately 15000 total words). All of our implemented heuristics perform better than the “Naïve” heuristic, which is a baseline determination of the runtime. “Naïve” basically runs a “straight” search to find the next word in the list, which, according to our data, proves to be highly inefficient. It seems that whether it be to move the next word to the front of the list proceeding the search, moving it one step forward in the list, or dynamically sorting the list by word count, doing something to list is better – in terms of speed – than doing nothing. Each of our six methodologies provides a better runtime, and an increase in processing power, than the “Naïve” heuristic.

|  |  |
| --- | --- |
| The Most Common Words | |
| 5 Text Files | **English Language** |
| the | the |
| of | of |
| and | and |
| in | a |
| a | to |
| to | in |
| is | is |

**Table 1: Most Common Words**

The most common words in our five text files were “the”, “of”, and “and”, by order of commonality. As you can see in the table to the right, the most common words in our text files were a match, almost word for word, to the most common words in the English language. The words displayed in the table are listed by order of commonality.

**Table 2: Unusual Words**

|  |
| --- |
| Unusually Common Words |
| computer |
| Germany |
| Rome |
| city |
| Buddhism |
| s |

We didn’t find many words in our text files that we determined to be unusually common. However, we found common words like “computer”, “germany”, “rome”, and others listed in the table to the left, that were expected as they were directly related to the topic of the text file. One of the most surprising words we found to be common was “s”. It was most likely attached to another word by an apostrophe, which was identified as a word separator in the program, “RunHeuristics”.

**Graph 3: Runtime – Total Words**

**Graph 4: Runtime – Unique Words**

# Python Generated Data

Below is a table detailing the comparisons of all the heuristics including the base and naïve. The numbers highlighted in green represent the unique words in a text file. And the numbers highlighted in blue represent the total words in the text file. By looking at the table it is fairly easy to see that the heuristics “Naïve”, “Move to Front”, “Transpose”, and “Access Count” take almost the same comparisons per lookup. While “Binary Search”, “Sort at End”, and “Map Words” take relatively the same also. However the former takes significantly more comparisons than the latter.

**Table 3: Comparisons per Lookup**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Base | 29 | 211 | 213 | Naïve | 29 | 211 | 213 |
| 213 | 1 | 1 | 1 | 213 | 251 | 901 | 2331 |
| 215 | 1 | 1 | 1 | 215 | 255 | 994 | 3584 |
| 217 | 1 | 1 | 1 | 217 | 256 | 1018 | 3959 |
| Move to Front | 29 | 211 | 213 | Transpose | 29 | 211 | 213 |
| 213 | 249 | 906 | 2338 | 213 | 251 | 901 | 2331 |
| 215 | 255 | 993 | 3587 | 215 | 255 | 994 | 3584 |
| 217 | 256 | 1016 | 3957 | 217 | 256 | 1018 | 3959 |
| Access Count | 29 | 211 | 213 | Binary Search | 29 | 211 | 213 |
| 213 | 250 | 907 | 2336 | 213 | 11 | 14 | 16 |
| 215 | 254 | 990 | 3602 | 215 | 12 | 14 | 17 |
| 217 | 256 | 1018 | 3960 | 217 | 12 | 14 | 17 |
| Sort at End | 29 | 211 | 213 | Map Words | 29 | 211 | 213 |
| 213 | 13 | 13 | 13 | 213 | 17 | 20 | 22 |
| 215 | 15 | 15 | 15 | 215 | 16 | 20 | 24 |
| 217 | 17 | 17 | 17 | 217 | 17 | 21 | 25 |

After analyzing the data in the table above it is much easier to see why the graphs below act the way they do.

Below are three graphs representing the time it took each heuristic to finish sorting randomly generated files. The fewer unique words there are in a file the more skewed the data is. By looking at Graph 5 we can see that most of the heuristics take less than one second to sort a file with 217 total words. At the same time Graph 7 shows a much clearer distinction between the heuristics. The “Map Words”, “Sort at End” and “Binary Search” heuristics take less than a second to sort through a file with 217 total words. “Transpose” takes roughly 15 seconds. And “Move to Front” and “Access Count” take the most time at around 35 seconds.

Having a one to one ratio of unique words to total words also skews the data, as seen in Graph 7. It takes less than one second to sort a file with 213 unique and total words.

\*\*\*sort at end takes lot of time in 2^9 but no time in 2^13\*\*\*

**Graph 5: Sorting 29 Unique Words**

**Graph 6: Sorting 211 Unique Words**

**Graph 7: Sorting 213 Unique Words**