Iva Tomevska
Magdalena Gunkova

A short write-up about the algorithmic/design choices

Our program starts with storing the words from the dictionary into a vector. We decided to use vectors because it is convenient as it is dynamically resizing, it can expands its memory at run time and we don't have to know the number of words in the dictionary to store them.

It continues with a while loop that is prompting the user for input:

1. If the user inputs just a normal word and he wants it to be searched through the dictionary, the program calls the function for searching through the dictionary and prints if the word is found or not, and the number of comparisons it made.
2. If the user inputs a prefix (indicated with '*'), it calls the function that searches through the dictionary for prefixes and returns them (a number of them indicated in the command prompt) and it tells how many comparisons took to find the first one.
3. If the user inputs a query (indicated with '?'), it calls the function that searches through the dictionary for the query and returns them (a number of them indicated in the command prompt). it tells how many comparisons in total took to find them and the total number of matches.
4. If the user inputs 'exit' the program terminates.

The while loop has O(n) complexity.

For the functions that search through the dictionary we use recursive binary search to look for words and iterative to look for partitioned words (when the user inputs a query or a prefix).
We decided to use binary search because it is faster than most of the searching algorithms as it does not search through the whole vector. It patrions it in the middle, it compares if the word we are looking for is smaller, greater than, or equal to the middle word. If it is smaller, it moves the ending to the middle, excluding the half greater than the middle from searching, and it searches only the lower half. If it is greater, it moves the beginning to the middle, excluding the half smaller than the middle from searching, and it searches only the upper half. It goes on until it finds the one that is equal to the word we are searching and it stops.
If it's recursive, when it excludes one half, it will call itself again to search for through the other half, exclude one half from there call itself again and so on.
If it's iterative, it will go on until the beginning is smaller than the end, making them the middle (depending on which half of the vector is needed), excluding one half and changing the boundaries until it finds the word we are searching for.

Iva Tomevska
Magdalena Gunkova

In both cases binary search has O(logn) complexity.

When the function for searching a prefix is called, the program first finds one word which has that prefix (string before the '*') with binary search. Then it checks if there are words with lower indices adjacent to it which match that prefix. Then, with a while-loop it moves the mark to the first word with the required prefix and from there, with another while-loop it appends all the following words which match the prefix in a vector. It only prints as many as we tell it to (through the command prompt) or if the number is smaller than that it prints all of them.
The complexity of the two while-loops is O(n)+O(n)=O(n)

When the function for searching a query is called, the program patrions the word into two other strings: the one before '?' and the one after it. Then, it calls the function that searches a prefix and that function returns a vector of all the words that match the prefix. Then, the program searches that vector with a for-loop and only takes the words which also match she sufix. The complexity of the for-loop is O(n).