Overview:

I'm so sorry about that but I couldn't implement hash table dictionary. I will try in the next week. Also, the AVL Tree.

Here is the array part report:

I. Implementation

With reading file, I couldn't handle with simple array of my defined structure because of terrible running time. If I read file and insert instant to my array, it could take… I don't know but my program goes wrong, it took an hour to read 15000 words to my array. I think it is something wrong when I keep expanding my array allocation each word added to it. The stack raises high and higher in my laptop's ram.

So, my solution to handle it is… using vector to avoid dynamically expand size error using push_back(). Then, I copy the temporary vector into my array.

Sorting the dictionary, using quicksort with custom swap function implemented for defined structure.

In case of look up a word, I use binary search which easy to implements that return position of a word in my array if it is existed. Else, return -1 value for a sign.

In case of adding new word, I don't know why but when I use liked algorithm in reading file, it goes wrong but adding new word, it works perfectly.

At first, I check if the new word is already in the dictionary, if already, I recommend user to edit this word instead of adding. Else, I expand my array size by create new array with size of n+1, after that, I copy all data from the old array into the new array, while copying, I also put the new word into it own valid position to keep my array ordered.

In case of edit meaning of a word, just search the position of the given word, if existed then I replace the old mean with new mean. Then, editing costs no time but searching for its position.

In case of deleting a word, find the position of given word, if existed, ask user to confirm his decision, if so, I create new array with size of n-1 and copy all data from the original to the new one. While copying, when I meet the word which be deleted, let the process ignores this word and keep copying.

In case of saving, nothing to talk about, its just simple as a loop.

II. Experiments

```
Input filename to import data: Oxford English Dictionary
Unable to open file.
```

First, validity of input filename. (Must have .txt at the end)

```
Input filename to import data: Oxford English Dictionary.txt
Loaded Oxford English Dictionary.txt in: 213ms.
Dictionary is sorted in: 162ms.

NOTE: CASE SENSITIVE.
[1] Look up
[2] Edit
[3] Add
[4] Delete
[5] Save
[0] Go back
Your choice:
```

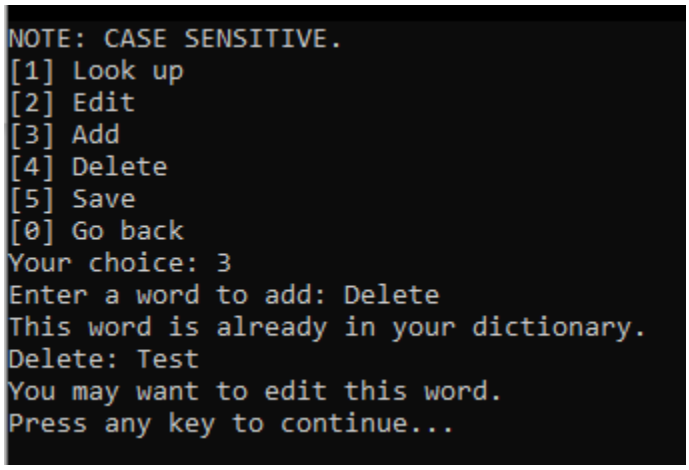
Load and sort 10 times for average running time. It took about 200ms and 160ms each turn.

```
NOTE: CASE SENSITIVE.
[1] Look up
[2] Edit
[3] Add
[4] Delete
[5] Save
[0] Go back
Your choice: 1
Enter a word to lookup: Delete
Delete: v. (-ting) remove (a letter, word, etc.), esp. By striking out. ⌂ deletion n. [latin deleo]
Running time: 0ms.
```

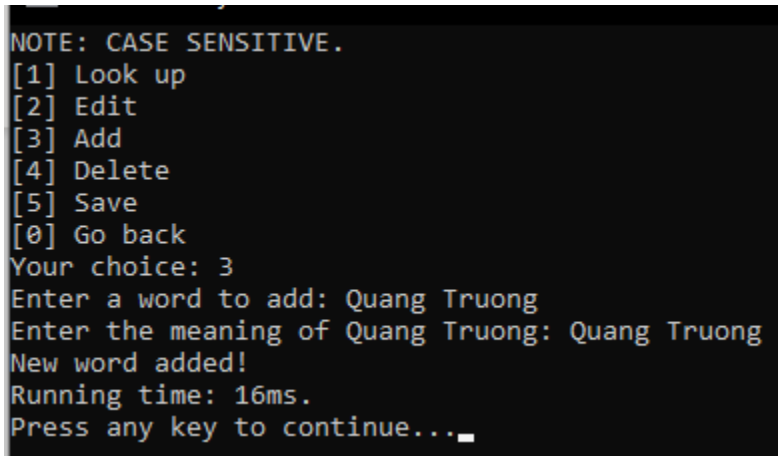
Look up 10 times, and every turn, it took… Oh, well, 0ms. The binary search runs too fast to timing.

```
NOTE: CASE SENSITIVE.
[1] Look up
[2] Edit
[3] Add
[4] Delete
[5] Save
[0] Go back
Your choice: 2
Enter the word you want to edit: Delete
Delete: v. (-ting) remove (a letter, word, etc.), esp. By striking out. ⌂ deletion n. [latin deleo]
Enter new mean of Delete to edit: Test
Word edited.
Delete: Test
```

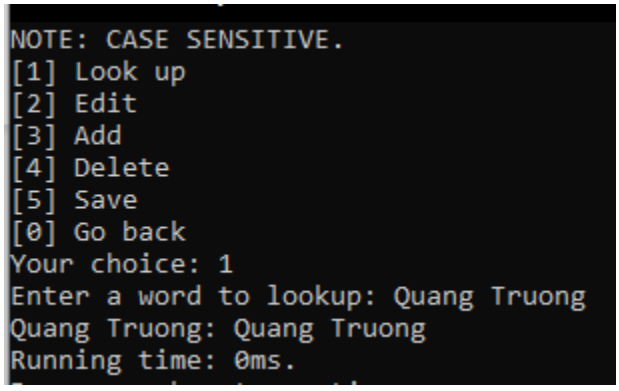
Edit meaning, well, in this case, it took no time to edit. If you want to timing it? It is the time of looking up a word and… look up took 0ms.



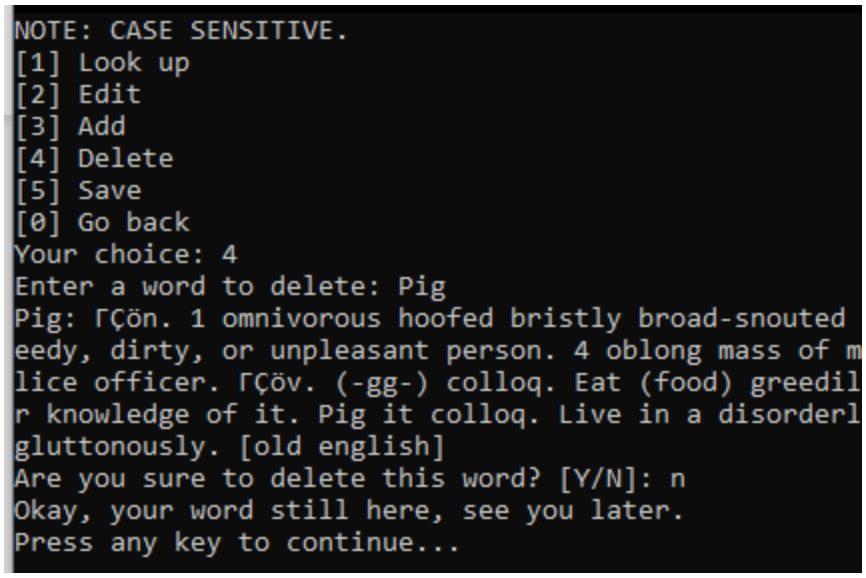
Add an existed word.



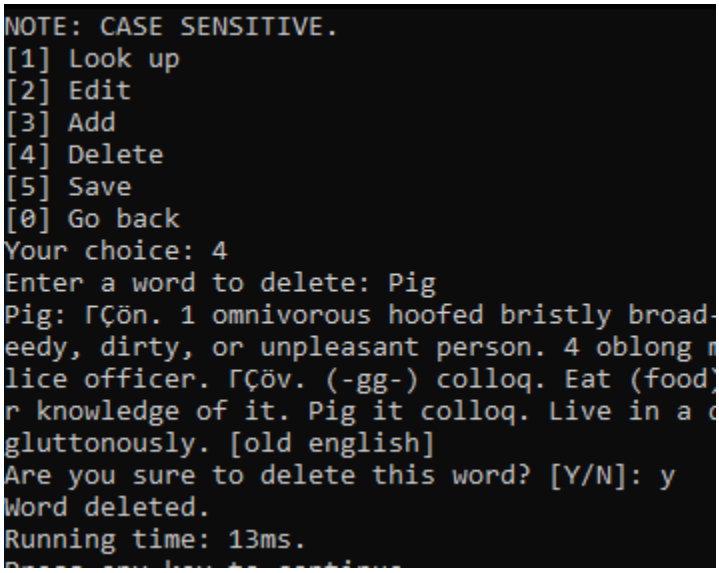
Add 10 times, running time about 10-25 ms.



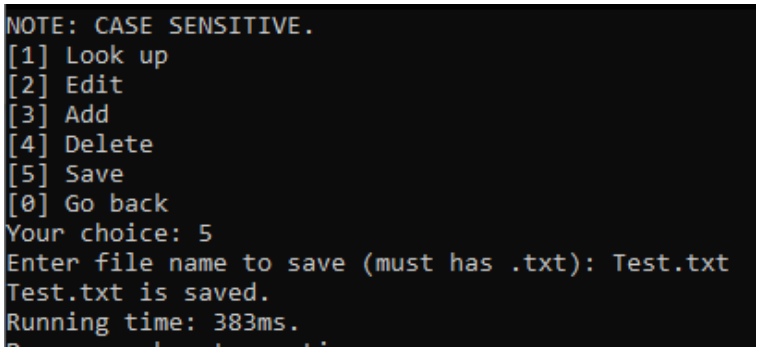
Look up the word added before.



Delete but change my mind after that.



Delete a word, run 10 times to get average time.



Save current dictionary into a text file.

III. Time complexities

Other data structures are not implemented yet, so I can't make a comparison.

IV. Analyze

Strong points:

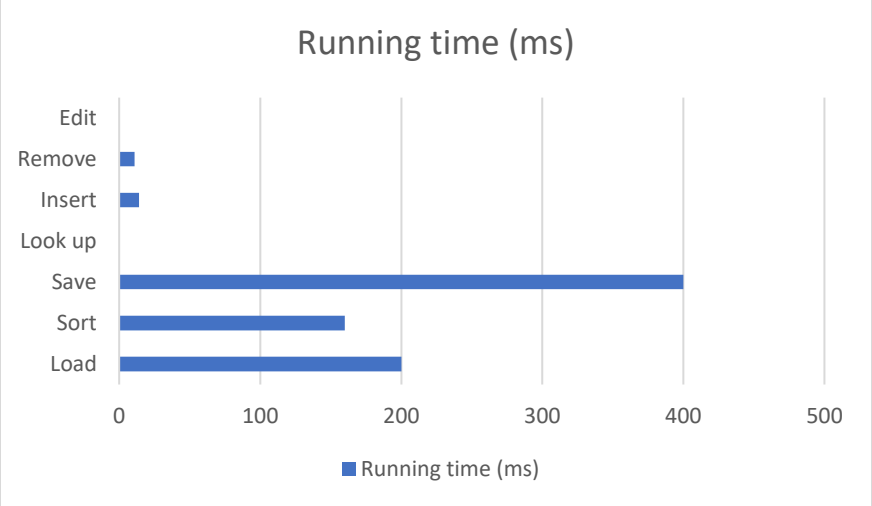
- Ease of implementation.
- Stable time complexity.
- Accepted time performance.

Weakness:

- Not too fast in time complexity.

V. Table and graph

Time	Array	AVL Tree	Hash Table
Load	200		
Sort	160		
Save	400		
Look up	0		
Insert	14		
Remove	11		
Edit	0		



VI. Guide to use

I create a menu that allows user to use my program easier. But there is something I must remind:

File name inputted must has .txt at the end.

While using menu, the inputted choice value must an integer.