

## **Comparison Report, Trie-hards**

Peter Nelson Subrata, Philipus Tandra Adriel, William Jonathan Mulyadi, Wilbert

Wirawan Ichwan

Department of Computer Science, Bina Nusantara University

**COMP6048001**

**Ir. TRI ASIH BUDIONO, M.I.T.**

June 20, 2022

## I. Abstract // Problem description

For our final project, we are trying to make an auto-complete dictionary called Trie. In our final project, we are asked to compare the runtime speeds between different data structures. Hence, we decided to compare it with a Hashtable dictionary.

For the one with the Trie, we made an autocomplete dictionary;( it will show all of the possible words when you type an alphabet). After you choose the word, the definition will be displayed and you can also update the word definition. Besides that, you can also insert words,types, and their definitions. Basically, for the hashtable one, it works almost the same, except it doesn't have the autocomplete features. We made a csv file containing about 1040 words with their definition for both the hashtable dictionary and trie.

A hash table has a time complexity of  $O(1)$  on average and best case and  $O(n)$  on worst case for lookup.

A trie has lookup time  $O(n)$  for the worst and average case but  $O(1)$  on its best case. Time complexity for inserting and deleting should be  $O(n)$  for a trie in all cases. A hash table has  $O(1)$  insertion and deletion on the best and average cases and  $O(n)$  on the worst case.

## II. Comparisons to be made // Result of execution

The comparisons to be made are as such: comparing speeds of inserting, searching, updating, deleting, and sorting.

### Searching

Searching	Hashtable Runtime/ms	Collisions	TRIE Runtime/ms
<b>A. Beginning</b>			
Abbreviate	0.013500	0	0.621800
Allah	0.011900	0	0.712000
Bar	0.009800	0	0.656600
<b>B. Middle</b>			
Mall	0.235000	54	0.621800
Nostalgic	0.218700	37	0.712000
Object	0.287600	87	0.647600
<b>C. Last</b>			
Xenon	0.529700	746	0.735200
Yacht	0.527600	779	0.703800
Zombie	0.496300	771	0.687300
<b>D.Non-existing</b>			
Ayam	0.277900	935	0.001800

### Sorting

Sorting	Hashtable Runtime/ms	Collisions	TRIE Runtime/ms
	107,269000	678	15865,223400

### Insertion

Insertion	Hashtable Runtime/ms	TRIE Runtime/ms
<b>A. Beginning</b>		
Array	0,457400	0,017900
Albino	0,394200	0,010500
Bee	0,012000	0,003800
<b>B. Middle</b>		
Meme	0,577800	0,013100
Nanny	0,477500	0,017000
Ornament	0,429200	0,017100
<b>C. Last</b>		
Xerus	0,445100	0,012000
Yoyo	0,500900	0,004500
Zonk	0,490000	0,005200

### Update

Updating	Hashtable Runtime/ms	TRIE Runtime/ms
<b>A. Beginning</b>		
Abbreviate	0,383300	0.006700
Allah	0,373500	0.008200
Bar	0,362400	0.002900
<b>B. Middle</b>		
Mall	0,377400	0.002800
Nostalgic	0,375600	0.003300
Object	0,373900	0.003700
<b>C. Last</b>		
Xenon	0,378600	0.005600
Yacht	0,367000	0.002700
Zombie	0,380500	0.003200

## Deletion

Deletion	Runtime/ms	Colision	TRIE Runtime/ms
<b>A. Beginning</b>			
Abbreviate	0,489000	0	0.011400
Allah	0,425100	0	0.006000
Bar	0,467200	0	0.003300
<b>B. Middle</b>			
Mall	0,433700	54	0.005500
Nostalgic	0,442400	37	0.007700
Object	0,516400	87	0.007200
<b>C. Last</b>			
Xenon	0,793700	746	0.005800
Yacht	0,865300	779	0.006100
Zombie	0,807300	771	0.007700

### HASHTABLE:

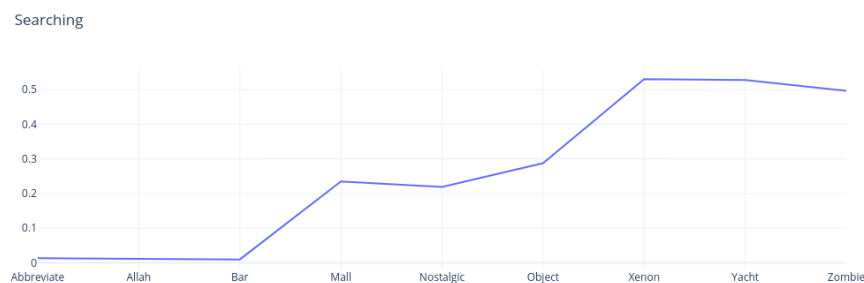
Runtime for searching should be mostly the same at  $O(1)$  in average case or best case, but since there are a lot of collisions the complexity becomes the worst case which is  $O(n)$ .

### TRIE:

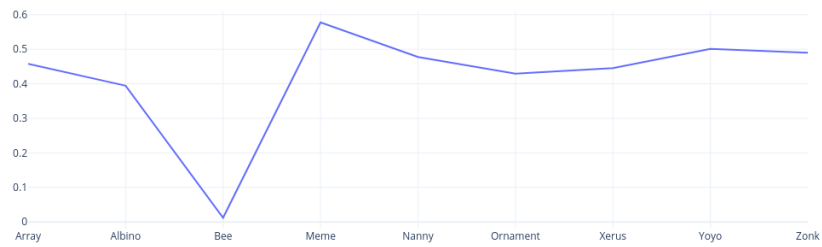
Runtime of the words are based on the length of the words as the trie has to start searching from the left most node to the right and it has to then continue down starting from the left most node again until it finds the leaf node specified. The time complexity of searching in a trie should thus be  $O(n)$ .

## III. The Graph

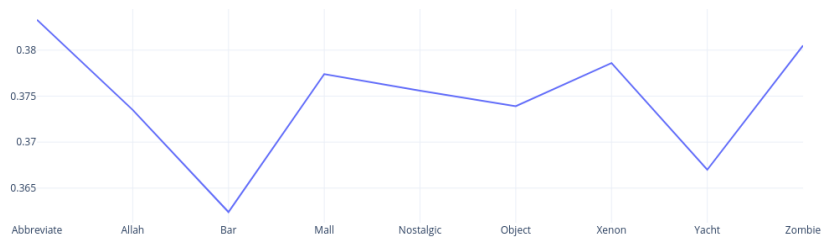
### 1. Hashtable dictionary



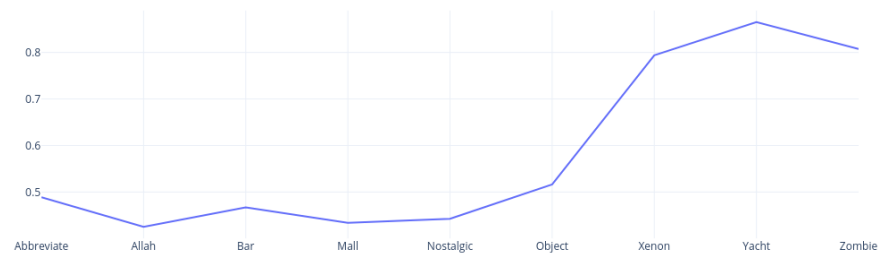
Insertion



Updating

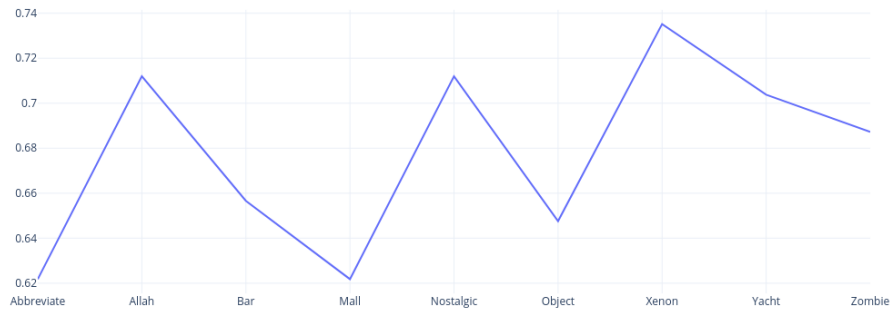


Deletion

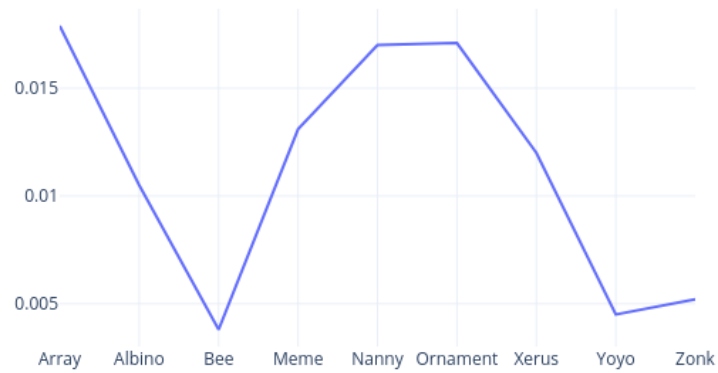


## 2. Trie dictionary

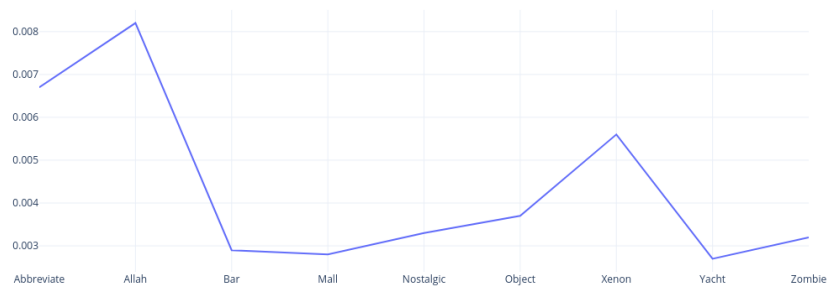
Searching

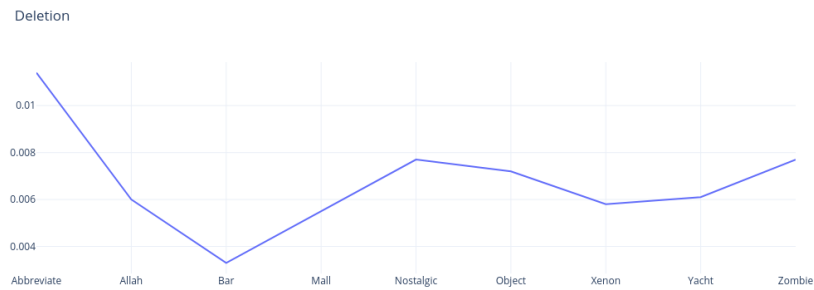


Insertion



Updating





### What can be learned from these reports.

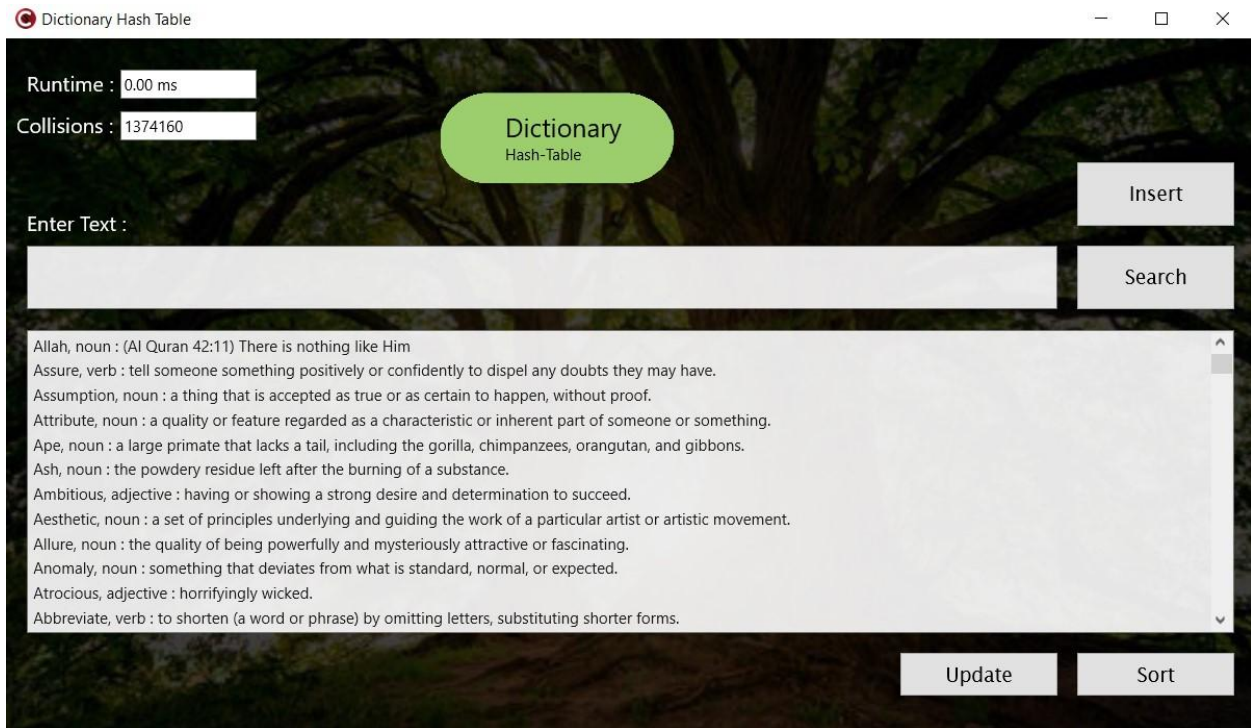
What we can learn from all of these runtime reports is that the hash table and the trie are both very efficient when searching, but a hash table should be faster in most cases and we can also prove that as the time complexity of a hash table on average is  $O(1)$  whereas the time complexity of a trie on average is only  $O(n)$ . A hash table can be slowed down though, when a hash function is not sufficiently good enough to provide minimal collisions, in this case the hash table can be slowed down to a time complexity of about  $O(n)$ . In the end though a hash table should still be faster. When deciding on what data structure to use and when, it is important to take note that a trie can easily be used to search for every word in it perfectly alphabetically in order using a pre-order traversal method as opposed to a hash table. Certain functions such as a prefix search (autocompletion) are also just not possible or at least not as easily implemented using a hash table. Generally speaking though in terms of space complexity both take up a massive amount of memory to run.

## IV. Video Demonstration

Link: <https://drive.google.com/file/d/1d2k989q-nunB0hEb7mAQ1yAlfpAGdsmU/view>

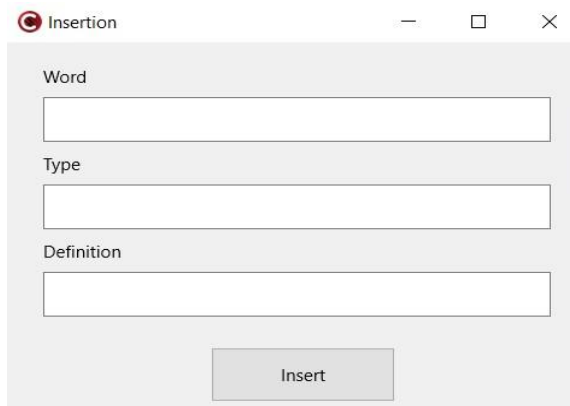
## V. How to execute the program (with screenshots)

### 1. Hashtable dictionary



To sort the words according to the alphabet you can click **the sort button**. As the sort is clicked, it will take some time for the program to process. After that, the runtime text will be updated and also the collisions too. As for searching, you can type the word first in the Enter text box and then click on the **search button**, it'll also update the runtime and collision.

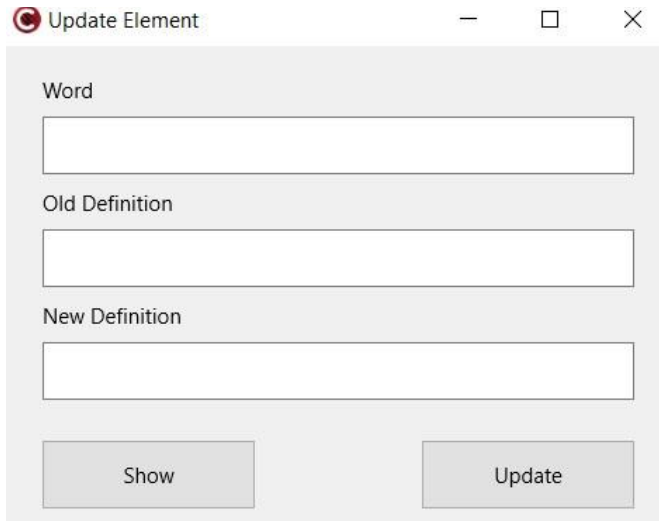
Meanwhile, for inserting, as you clicked on the **insert button**, it will result in showing :



You can type the word, type, and definition. After that, click the insert and the word you typed will be inserted to the dictionary.



As for updating words, you can click the **update button** and it will result in showing :

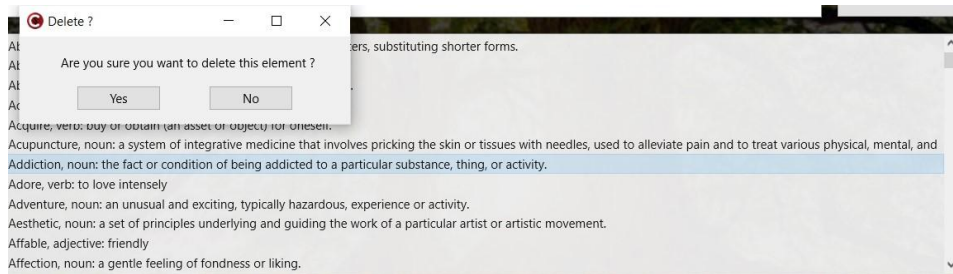


The 'Update Element' dialog box contains three text input fields labeled 'Word', 'Old Definition', and 'New Definition'. Below these fields are two buttons: 'Show' and 'Update'.

You can fill in the word that you want to update, but you don't need to fill in the Old definition. You can update it by only filling the word and new definition, then press the **update button**. The runtime will be recorded in the runtime box and the word will be updated too.

If you click on the **show button**, it will display the definition of the word you want to update in the old definition box.

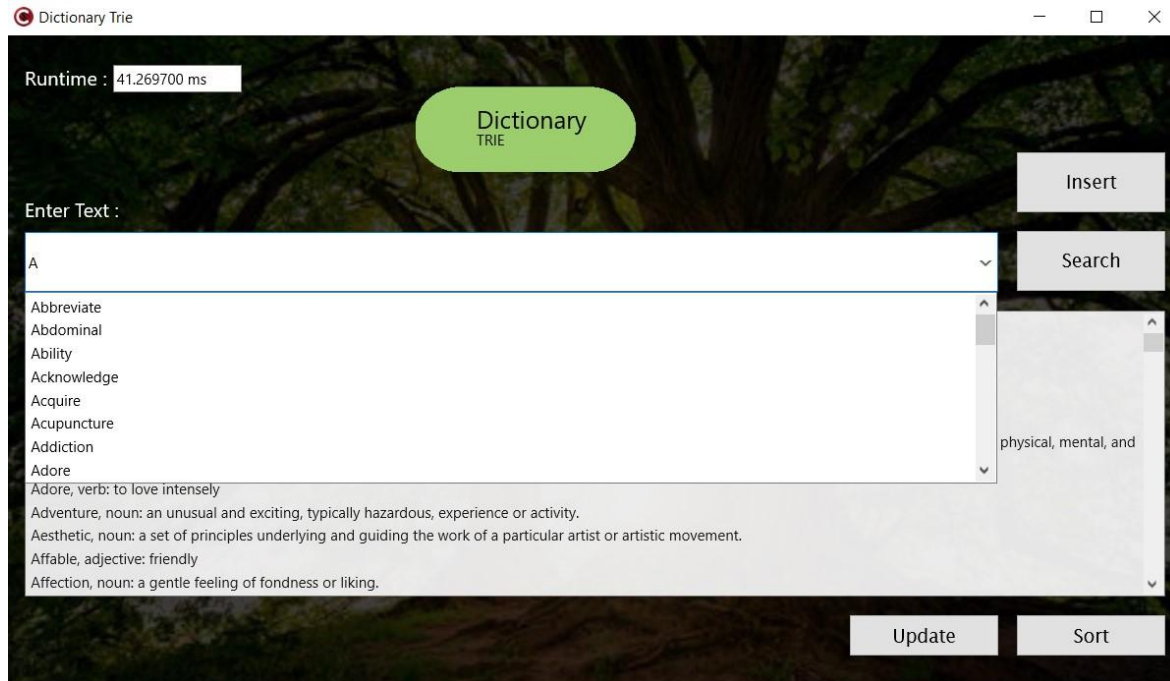
As for **deleting words**, you can directly click on the word that is shown on the box. By clicking on the word, it will show :



You can also search the word first and delete it.

If you press yes, the word in the dictionary will be deleted.

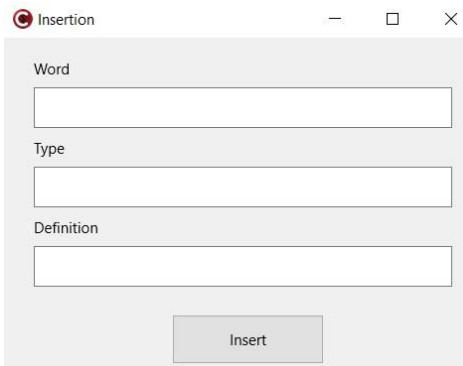
## 2. Trie dictionary:



As for the trie dictionary, by entering an alphabet one by one to make a word in the enter text box, it will show the list of words option (auto-complete dictionary). You can directly click on the shown options, and it will redirect you to search that option. You can also write a full word and click the search button to search for a word. By clicking the search button, the runtime will be updated.

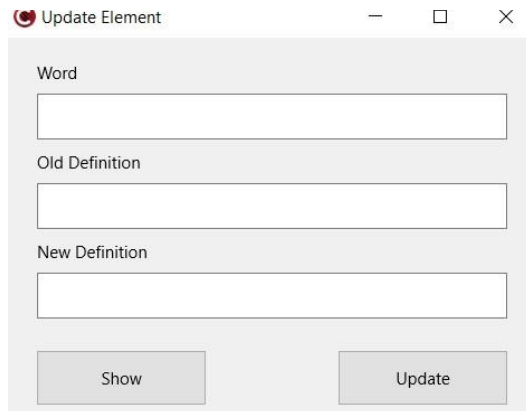
As for the **sort button**, when you click it, it will sort the words in alphabetical order. The runtime will also be updated.

By clicking the **insert button**, it will also show:



You can fill in the word, type, and definition. After that, click the insert button and the word you type with its information will be added to the dictionary. The runtime of the insertion algorithm will also be recorded and shown in the runtime box.

As for the **update button**, when you click it, it will show:



Update Element

Word

Old Definition

New Definition

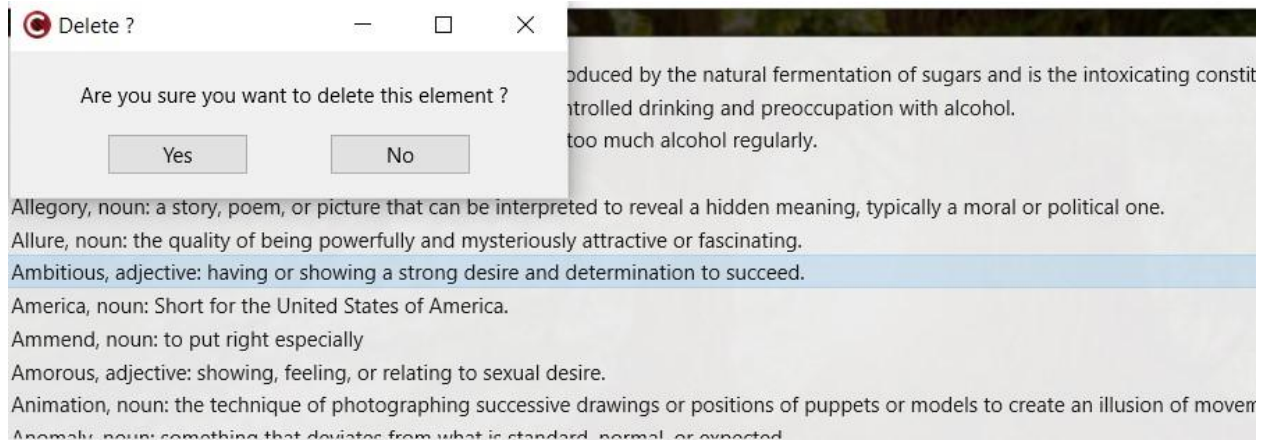
Show Update

This actually works the same with the hashtable dictionary update button.

You should fill in the word and new definition, then click the **update button** and your word is successfully updated. The runtime will be recorded and shown in the runtime box.

If you click on the **show button**, it will display the definition of the word you want to update in the old definition box.

If you want to **delete**, you can directly click on the word in the box. It will then ask you “Are you sure you want to delete this element?”. If you click yes, the word will be deleted, it will show the runtime too in the runtime box.



Delete ?

Are you sure you want to delete this element ?

Yes No

Allegory, noun: a story, poem, or picture that can be interpreted to reveal a hidden meaning, typically a moral or political one.

Allure, noun: the quality of being powerfully and mysteriously attractive or fascinating.

Ambitious, adjective: having or showing a strong desire and determination to succeed.

America, noun: Short for the United States of America.

Ammend, noun: to put right especially

Amorous, adjective: showing, feeling, or relating to sexual desire.

Animation, noun: the technique of photographing successive drawings or positions of puppets or models to create an illusion of movement.

Anomaly, noun: something that deviates from what is standard, normal, or expected.