1. I made use of a hash table which is a vector of pointer to lists of strings

Vector<list<string>\*>

I did this because it allowed easy access to finding a word by just using its bucket number as the index. Moreover, since vector points to a list I was able to store words with same hash values together allowing for quick searching and insertion of anagrams.

In all the execution time for such a hash table resulted in just O(1) time complexity for inserting words in our hash table and O(N) for searching anagrams.

Which looks like the following-

A picture containing diagram, plan, rectangle, square

Description automatically generated



**size\_t DictionaryImpl::hashfunc(const string& word) const**

create a hash object

sort the word and store in separate variable

get the hash value using hash object

get bucket number using hash value divided my size of our hashtable

**void DictionaryImpl::insert(string word)**

remove non letters from word

if word is not empty

sort words and get bucket number

create a list of string pointer to hashtable with the respective bucket number

push the word into the list at that bucket.

**void DictionaryImpl::lookup(string letters, void callback(string)) const**

if callback is nullptr then return

remove non letters from word

if word is empty then return

sort the word in a new variable say a

get list at that bucket number

iterate over the list

sort the word in a new variable say b

if a == b

callback word



Initially I wrote a wrong implementation by not using a pointer to the list of strings due to which the code didn’t execute properly. Second issue I faced was in figuring out how get the list at a defined bucket number. Once I figured that out, code ran properly within the defined time limit. Apart from that I had no other issues per say.