Justin Ngo
jmn4fms
3/5/20
postlab.pdf

The big-theta running speed of the word-search component of my program is n^3. When iterating through the rows and columns with nested for loops in my wordPuzzle.cpp main method, the big theta runtime is n^2. The direction and word checking add an addition worst case runtime of n on top the runtime of rows and columns to produces a big theta of n^3 because it is iterating to call the other functions in my program and we can assume that maximum word size is a small constant so that it does not increase the runtime beyond n.

The runtime of my program with the -O2 flag post-optimizations for words2.txt and the 300x300 grid is 1.513137 seconds. The hash function I used in the hashTable.cpp for my program is:

```
int HashTable::hash(string word){
   int index = 0;
   unsigned long long hash = 7;
   for(int i = 0; i<word.length(); i++){
      hash = hash*31 + ((int)word[i]);
   }
   index = hash % tableSize;
   return index;
}
```

I used prime numbers when multiplying because the only factors of a prime number prime number p and 1. Using prime numbers in multiplication provided better distribution of the hash keys into the table and less clustering. A hash function designed to perform slower would be simply taking the first character of the string word[0] and setting that as the hash value to be mod'ed by the table size:

```
int HashTable::hash(string word){
   int index = 0;
```

```
    unsigned long long hash = 7;
    for(int i = 0; i<word.length(); i++){
        hash = ((int)word[0]);
    }
    index = hash % tableSize;
    return index;
}
```

The runtime of the program with a worse designed hash function with the -O2 flag post-optimizations for words2.txt and the 300x300 grid is 3.268352 seconds and over two times slower. More collisions occur under a hash function of this design because many words have the same first character in the string format. A hash table size designed to make performance worse would be a size of 2 because everything the program searches for would be hashed into two buckets, creating two buckets with lists of n time. I used my MacBook Pro 14,1 with a dual-core intel core i5 processor that has a processor speed of 2.3 GHz, 1 core, 2 L2 Caches, and a 256 KB L3 Cache: 4 MB.

Some of the optimizations I attempted in the experimentation of my post-lab were changing the load factor λ for the hashtable and experimenting with different collision resolution strategies, and looking up better hash functions for strings. Originally I had set the size of my hash table to the next prime number after 100,000 because I believed it had worked best for my pre-lab implementation before it was finalized. After experimenting by adding and subtracting 0s from this value I improved the runtime to be 1.513137 seconds with a table size of 10000. A table size of 1,000,000 increased the runtime to 2.255701 seconds. A table size of 1000 increased the runtime to 1.737675 seconds. I tried this optimization because I realized I had forgotten about the hash table size after I initially set it in the pre-lab and was surprised to find that it worked to improve my runtime. Experimenting and changing the table size worked

because it improved the runtime of my program by over 0.2 seconds. I tried experimenting changing the hash function to multiply by different prime numbers from 7 and 31 but I found that it did not have a significant impact on the runtime and stayed within a margin of +/- 0.1 seconds from my final optimized time such as 1.505285 seconds. Calling the first character in the word string consistently increased the runtime by over a second such as 2.558293 seconds. I worked on and tested my hash function significantly when I worked on the pre-lab in order to make sure I had a strong hash function. I returned to my hash function again in the post-lab to try improving it from my code that utilized prime numbers and each character of the word string without taking too much time however I was not able to find another way to improve it significantly.

I was able to further optimize my original running time of approximately 1.792153 seconds to 1.513137 seconds. I had tested and looked into different methods to optimize and speed up the runtime of my program in the implementation of my pre-lab but found that I overlooked the importance of the hash table size in the optimization process of the post-lab. The speedup after experimenting with different hash functions and implementations of hash table was 1.792153/1.513137 = 1.184396.