Assignment Two

Ethan Ondreicka

Ethan.Ondreicka1@Marist.edu

October 28, 2023

1 Part One: Linear and Binary Search

1.1 Developing the Linear Search Function

```
1  // Linear Search Function
2  bool linearSearch(const std::string target, const std::string lines[], int lineCount, int&
       comparisons) {
3      comparisons = 0; // counter for comparisons
4      for (int i = 0; i < lineCount; i++) {
5          comparisons++; // Increment the comparisons count for each iteration
6          if (lines[i] == target) {
7              return true; // Item found
8          }
9      }
10     return false; // Item not found
11 }
```

Listing 1: Construction of the Linear Search Function

The linear search function takes the list of 666 magic items and sequentially goes through each one until the target value is found. As you can see on in the for loop starting on line 4, it will loop for as long as it takes to find the target value. The asymptotic running time of this function is $O(n)$.

1.2 Developing the Binary Search Function

```
1  // Binary Search Function
2  bool binarySearch(const std::string target, const std::string lines[], int lineCount, int&
       comparisons) {
3      comparisons = 0; // comparisons counter
4      int leftValue = 0;
5      int rightValue = lineCount - 1;
6      while (leftValue <= rightValue) {
7          comparisons++; // count for each iteration
8          int middleValue = leftValue + (rightValue - leftValue) / 2;
9          if (lines[middleValue] == target) {
10             return true; // Item found
11         }
12         if (lines[middleValue] < target) {
```

```
            leftValue = middleValue + 1;
        } else {
            rightValue = middleValue - 1;
        }
    }
    return false; // Item not found
}
```

Listing 2: Creation of the Binary Search Function

The Binary Search function takes the magic items list and will grab the middle value. If the target is not found, it will divide the list in half and grab the middle value again. It will do it until the middle value is the target value (as seen on line 8). If the target is found, the function will return true. The asymptotic running time of this function is $O(logn)$

2 Part Two: Hashing

2.1 Creation of the Hash Table

```
// Class for the hash table itself
class HashTable {
public:
    HashTable() {
        table.resize(TABLE_SIZE);
    }

    void insert(const std::string& key, const std::string& value) {
        unsigned int index = hash(key);
        Node* newNode = new Node(key);
        newNode->setNext(table[index]); // sets next pointer to current head
        table[index] = newNode;
    }

    bool get(const std::string& key, std::string& value, int& comparisons) {
        unsigned int index = hash(key);
        comparisons = 0;
        Node* current = table[index]; // starts at head
        while (current != nullptr) {
            comparisons++;
            if (current->getItem() == key) {
                value = current->getItem();
                return true;      // key is found
            }
            current = current->getNext();
        }
        return false;   // key not found
    }
    std::vector<Node*> table; // linked lists to store table
};
```

Listing 3: Hashing Table Class

The Hashing Class takes all of the magic items and stores them into a hash table of size 250. The get function is used to retrieve and count comparisons while searching for the target value (key). The asymptotic running time for hashing is $O(n)$ while the asymptotic running time of the chaining is $O(\alpha+1)$. The reason chaining is $O(\alpha+1)$ is due to the load factor.

3 Part Three: Results

3.1 Results Table

Search Function	Average Comparisons	Asymptotic Running Time
Linear Search	319.93	$O(n)$
Binary Search	9.64	$O(logn)$
Hashing and Chaining	2.31	$O(n)$ and $O(\alpha + 1)$

The Average Comparisons is based on each function running 42 times for the 42 random items