1 Linear Search

Linear search is a linear algorithm used to look up an item. I use it here in a double for loop which is a n squared algorithm because I want to first get one of the random items and then compare that against each item in the magic items file. The average look up time to find an item is 333.

```
public static void linearSearch(String array[], String searchArray[]) {
        int averageTimeForLinearSearch = 0;
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < searchArray.length; j++) {
                //This compares each element in the magic items text file to the current random item
                if (array[i] == searchArray[j]) {
                    break;
                }
                //global count
                averageTimeForLinearSearch += 1;
            }
        }
        System.out.println("linear search is " +averageTimeForLinearSearch / 42);
    }
```

2 Binary Search

Binary Search is a log n searching algorithm to find something out of a list. This is one of the most efficient ways to look up something without hashing. The average look up time for binary search in a list of 666 items is 9.3. I'm noticing that I'm getting around 5 consistently.

```
public static void binarySearch(String array[], String searchArray[]) {
        float averageTimeForBinarySearch = 0;


        for (int i = 0; i < array.length; i++) {

            int start = 0;
            int end = array.length - 1;

            while (start <= end) {

                int medium = start + (end - start) / 2;

                //if the item is equal to the current mid then it will break out of the while loop
    if (searchArray[medium].replaceAll("\\s", "").compareTo(array[i].replaceAll("\\s", "")) == 0) {
                    averageTimeForBinarySearch += 1;
                    break;
                }
                //if the item is less than the current mid then the start becomes the medium plus 1
    if (searchArray[medium].replaceAll("\\s", "").compareTo(array[i].replaceAll("\\s", "")) < 0) {
                    start = medium + 1;
                //if the item is greater than the current mid then end becomes the medium - 1
                } else {
                    end = medium - 1;

                }
                averageTimeForBinarySearch += 1;
            }

        }
        System.out.println("Binary search is " + averageTimeForBinarySearch /42);
    }
```

3 Creating Hash Table

This function creates a hash table of 250 which will store 666 items. The hash function was provided by Dr.Labouseur to store all of the items in the correct vertices. The first part of the function checks to see if there is a node in the position already. If there is, it will add another node to the next of that previous node. After all of the items are inserted, the look up function is then called below.

```
// This function makes a hash table
    public static void hashTable(String randomItems[], String array[]) {
        // create a new array of nodes size 250 and initializing the indices to null
        Node newArray[] = new Node[250];
        for (int i = 0; i < newArray.length; i++) {
            newArray[i] = null;
        }

        for (int i = 0; i < array.length; i++) {
            Node newNode = new Node();
            int getHashCode = makeHashCode(array[i]);
            // This checks to see if there isn't an item in the indice and then adds the
            // first node if there isn't
            if (newArray[getHashCode] == null) {
                newNode.name = array[i];
                newArray[getHashCode] = newNode;
            } else {
                // This will find the last position
                // in the linked list and will add the new node to the end.
                newNode.name = array[i];
                Node current = newArray[getHashCode];

                while (current.name != null) {
                    if (current.next == null) {
                        current.next = newNode;
                        break;
                    }
                    current = current.next;
                }
            }
        }
        hashTableLookup(randomItems, newArray);
    }
```

4 Looking up in Hash Table

The hash table look up function takes the list of random items and looks up those items in the hash table. This is in one for loop that takes an item from the list and uses the hash function in it. Then the while loop will be executed if there is more than one item in the linked this. After all is searched, the average look up time is 2.

```
public static void hashTableLookup(String[] randomitems, Node[] array) {
        float averageLookUpTime = 0;
        for (int i = 0; i < randomitems.length; i++) {
            int hashCode = makeHashCode(randomitems[i]);
            Node current = array[hashCode];
            // this checks if the current node or the first node
            // in the linked list is the
            // item.
            if (current.name == randomitems[i]) {
                averageLookUpTime += 1;
                continue;
            } else {
                // This goes through every node in the
                // linked list until it finds the correct
                // item.
                while (current.next != null) {
                    averageLookUpTime += 1;
                    if (current.name == randomitems[i]) {
                        break;
                    }
                    current = current.next;
                }
            }

        }
        // This prints out of the total divided by the number of random items in the
        // array.
        System.out.println("Hash look up average is " + averageLookUpTime / 42);
    }
```

5 Number of comparisons

Linear Search	Binary Search	Hash Table
333	5	2
O(n)	O(log n)	O(1)

```
Linear Search is linear algorithm because it's in one for loop going through every item.

Binary Search is a log n algorithm because it cuts the iteration in half every time.

I'm calling the hash table look up time 0(1) because if we had a really good hash function,
then it would be constant time.
```