Assignment 2 - Code Evaluation

Mason Nakamura

October 2021

1 Linear Class

```
1  public class Linear {
2      int comparisons = 0;
3      public int linear_search(String[] A, String target){
4          int n = A.length;
5          for(int i = 0; i < n; i++){
6              comparisons += 1;
7              if(A[i] == target){
8                  // return index of target in array
9                  return i;
10             }
11         }
12         // if target is not in array
13         return -1;
14     }
15 }
```

The method linear_search() from Linear Class searches for a target, in this case a String in an array of Strings. It linearly searches from index 0 to the end of the array, until the target is found, in which case the index is outputted; else it returns index -1. The time complexity is $O(n)$ since you linearly go through the array. However, our result is half of that which is expected since its actually $O(n/2)$, but the $1/2$ coefficient can be pulled out.

2 Binary Class

```
1  public class Binary {
2      int comparisons = 0;
3      public int binary_search(String[] A, int left, int right, String target){
4          if (left <= right){
5              comparisons += 1;
6              // get the middle string between right and left points in A
7              int middle = left + (right - left)/2;
8              if (A[middle] == target){
9                  // found target in A
10                 return middle;
11             }
12             // compare strings A[middle] and target
13             // if target is less than middle go left
14             if (A[middle].compareTo(target) < 0){
15                 return binary_search(A, left, middle-1, target);
16             }
17             else{
18                 // if target is greater than middle go right
19                 return binary_search(A, middle + 1, right, target);
20             }
21         }
22         // if target not in A
23         return -1;
24     }
25 }
```

The binary_search() method from the Binary Class searches through the array of Strings for the target String using intervals rather than linearly. In line 4, we check if the left index is less than or equal to the right index which becomes relevant in our recursion; else the target is not in the array of Strings, outputting -1 as the index in line 23. Next, we declare the middle index between the left and right index (initially, the left index will be 0 and right the length of the array of Strings minus 1). We then check if the string at the middle index will be our target String. If so, we return that index in line 10 (our base case); else, we move to the next if statement. We compare the two strings using .compareTo() to see if we need to use the interval between the left and middle-1 or right and middle+1 recursively. The time complexity of Binary Search is $O(log_2(n))$ since you halve the array in each iteration, until the base case is reached.

3 Table of Comparisons

Linear Search # of Comparisons	Binary Search # of Comparisons
386	9
656	9
393	9
266	10
319	10
415	9
385	9
248	10
380	9
256	10
343	9
663	9
47	10
469	9
473	9
542	9
416	9
126	10
90	10
188	10
319	10
117	10
286	10
345	9
525	9
403	9
349	9
106	10
511	9
523	9
660	9
634	9
325	10
306	10
275	10
445	9
238	10
387	9
146	10
364	9
592	9
589	9

Table 1: Table of Comparisons for Linear and Binary Search

Linear Search Average	Binary Search Average
369.61	9.57

Table 2: Table of Average Comparisons for Linear and Binary Search

4 Hashing Class

```
1  import java.util.Arrays;
2  import java.io.FileReader;
3  import java.io.BufferedReader;
4
5  public class Hashing {
6      public final String FILE_NAME = "Assignment 3/magicitems";
7      public final int LINES_IN_FILE = 666;
8      public final int HASH_TABLE_SIZE = 250;
9      int comparisons = 0;
10
11
12
13     public int doHashCode(String str) {
14         str = str.toUpperCase();
15         int length = str.length();
16         int letterTotal = 0;
17
18         // Iterate over all letters in the string, totalling their ASCII values.
19         for (int i = 0; i < length; i++) {
20             char thisLetter = str.charAt(i);
21             int thisNum = (int)thisLetter;
22             letterTotal += thisNum;
23         }
24
25         // Scale letterTotal to fit in HASH_TABLE_SIZE by taking the letterTotal mod
               HASH_TABLE_SIZE
26         int hashCode = (letterTotal) % HASH_TABLE_SIZE;
27
28         return hashCode;
29     }
30
31     public void analyzeHashValues(int[] hashValues) {
32         System.out.println("\nHash Table Usage:");
33
34         // Sort the hash values.
35         Arrays.sort(hashValues);
36
37         int asteriskCount = 0;
38         int[] bucketCount = new int[HASH_TABLE_SIZE];
39         int totalCount = 0;
40         int arrayIndex = 0;
41
42         for (int i = 0; i < HASH_TABLE_SIZE; i++) {
43             System.out.format("%03d ", i);
44             asteriskCount = 0;
45             while ((arrayIndex < LINES_IN_FILE) && (hashValues[arrayIndex] == i)) {
46                 System.out.print("*");
47                 asteriskCount = asteriskCount + 1;
48                 arrayIndex = arrayIndex + 1;
49             }
50             System.out.print(" ");
51             System.out.println(asteriskCount);
52             bucketCount[i] = asteriskCount;
53             totalCount = totalCount + asteriskCount;
54         }
55     }
56
57     public void populate_retrieveTargets(int[] targets_hashValues, String[] targets,
           int[] hashValues, String[] hashOriginalStrings){
58         // 1) Populate the Hash table first
59         // sort hashValues, then sort hashOriginalStrings the SAME way (RELATIVE
               SORTING)
60         RelativeInsertionSort relativeInsertionSort = new RelativeInsertionSort();
61         relativeInsertionSort.relative_insertionSort(hashValues, hashOriginalStrings)
               ;
62 //        Arrays.sort(hashValues);
63         //convert hashValues to an array of linked lists (hashValues ---->
               hashValues_asNodes)
64         Node[] hashValues_asNodes = new Node[LINES_IN_FILE];
65         for (int i = 0; i < LINES_IN_FILE; i++){
66 //            List list = new List();
67             Node node = new Node();
68             node.index = i;
```

```
69              node.next = null;
70              node.hashValue = hashValues[i];
71              node.name = null;
72  //            list.head = node;
73  //            list.head = node;
74              // linear: 0,1,2,3,4,...,665
75              hashValues_asNodes[i] = node;
76          }
77          int arrayIndex = 0;
78          for (int i=0; i < HASH_TABLE_SIZE; i++) {
79  //            System.out.format("%03d ", i);
80              // This will terminate
81              // go through each index and append a pointer to a node if such a
                    hashValue exists
82              int asteriskCount = 0;
83              while ( (arrayIndex < LINES_IN_FILE) && (hashValues_asNodes[arrayIndex].
                    hashValue == i) ) {
84  //                System.out.print("*");
85  //                asteriskCount = asteriskCount + 1;
86                  Node node = new Node();
87                  node.next = null;
88                  node.hashValue = hashValues[arrayIndex];
89                  //node.name --> String
90                  node.name = hashOriginalStrings[arrayIndex];
91                  hashValues_asNodes[i].next = node;
92                  arrayIndex += 1;
93              }
94  //            System.out.print(" ");
95  //            System.out.println(asteriskCount);
96          }
97  //        System.out.println("passed point A");
98          // 2) Retrieve target values
99          for(int i = 0; i < targets.length; i++){
100 //            String output = "Target not in hashtable";
101             // search the linked list at index=target_hashValue
102 //            if (hashValues_asNodes[targets_hashValues[i]].index ==
        targets_hashValues[i])
103 //                System.out.println("testing");
104 //            System.out.println(hashValues_asNodes[targets_hashValues[i]].index);
105             Node head = hashValues_asNodes[targets_hashValues[i]].next;
106             int output = 0;
107             while(head != null){
108                 // compare here
109 //                System.out.println(targets[i]);
110                 comparisons += 1;
111                 // compare target string to all node.names in the linked list
112                 //NO clue why this does not work, but next if statement does
113                 if(targets[i].equals(head.name)){
114 //                    output += 1;
115                     System.out.println(targets[i]);
116                     break;
117                 }
118 //                if(targets_hashValues[i] == (head.hashValue)){
119 //                    output = "Target Found2";
120 //                    System.out.println(targets[i]);
121 //                    break;
122 //                }
123 //                System.out.println(head.name);
124                 head = head.next;
125             }
126 //            System.out.println(output);
127         }
128     }
129
130 }
```

In line 57, we define our populate_retrieveTargets() method. We first populate our hash table with node objects. In lines 60-61, we change our InsertionSort class to a relativeInsertionSort class which sorts two arrays of the same length in the same order. This is to keep the index of strings in array A with array B after sorting for the purpose of appending elements in A to B as attributes in a Node object. In lines 65-76 we initialize our first node objects with no name or hashValue attributes. In lines 78-96, we populate the hash table with node objects (this has been tested and works).

We now move on to retrieving the target values from the hashtable (this is where the problem is). We first assign head to be hashValuesasNodes[targetshashValues[i]].next which has attributes

hashValue. It then enters the while loop if it's not null. I then start incriminating the comparisons in the while loop (i.e. iterating through the linked list). ****This is where our issue is: the comparisons is 42, but it should be higher. It seems to only compare the first node of every linked list, but line 24 does not seem to work by iterating through the linked lists. The time complexity of hashing with chaining is $O(1+\alpha)$ where α = average chain length. This is expected since finding the hashValue takes $O(1)$ and traversing the linked lists takes the average linked list length, similar to the time complexity of linear search.

5 Test Class

```
1  import java.io.*;
2  import java.util.Arrays;
3  import java.util.Random;
4
5  public class Test {
6
7      public static void main(String[] args) {
8          String[] words = {};
9          // Read line by line the txt file using File reader
10         String fileName = "Assignment 3/magicitems";
11         File file = new File(fileName);
12         try {
13             FileReader fr = new FileReader(file);
14             BufferedReader br = new BufferedReader(fr);
15             String line;
16             while ((line = br.readLine()) != null) {
17                 // add strings from txt file line by line into array words
18                 words = Arrays.copyOf(words, words.length + 1); //extends memory
19                 words[words.length - 1] = line; //adds word to extra memory
20             }
21         } catch (FileNotFoundException e) {
22             System.out.println("An error occurred.");
23             e.printStackTrace();
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27         String[] copy_words0 = words.clone();
28         //Sort the words using insertion sort
29         InserstionSort i_sort = new InserstionSort();
30         i_sort.insertionSort(copy_words0);
31         // check if copy_words0 is sorted
32 //        for (String i: copy_words0){
33 //            System.out.println(i);
34 //        }
35         // rename copy as sorted
36         String[] sorted_words0 = copy_words0;
37         // initialize random generator to generate random index from length of array
38         Random generator = new Random();
39         String[] random_words = {};
40         for(int i = 0; i < 42; i++){
41             int randomIndex = generator.nextInt(sorted_words0.length);
42             random_words = Arrays.copyOf(random_words, random_words.length + 1); //
                   extends memory
43             random_words[random_words.length-1] = sorted_words0[randomIndex];
44         }
45         // test the 42 random words from sorted_words0
46 //        for(String i: random_words){
47 //            System.out.println(i);
48 //        }
49         // getting comparisons for linear search
50         float total_comparisons_linear = 0;
51         for(String i: random_words){
52             Linear linear = new Linear();
53             // search for target i
54             linear.linear_search(sorted_words0, i);
55             System.out.println(linear.comparisons);
56             total_comparisons_linear += linear.comparisons;
57         }
58         System.out.println("Average number of Comparisons for Linear Search:" +
               total_comparisons_linear/42);
59
60         //getting comparisons for binary search
```

```
61          float total_comparisons_binary = 0;
62          for(String i: random_words){
63              Binary binary = new Binary();
64              // start recursion with entire sorted_words0 array of Strings
65              binary.binary_search(sorted_words0, 0, sorted_words0.length-1, i);
66              System.out.println(binary.comparisons);
67              total_comparisons_binary += binary.comparisons;
68          }
69          System.out.println("Average number of Comparisons for Binary Search:" +
                total_comparisons_binary/42);
70
71          // Hash Testing
72          // Print the array and hash values.
73          Hashing hashing = new Hashing();
74          int[] hashValues = new int[hashing.LINES_IN_FILE];
75          int hashCode = 0;
76          for (int i = 0; i < hashing.LINES_IN_FILE; i++) {
77  //              System.out.print(i);
78  //              System.out.print(". " + copy_words0[i] + " - ");
79              hashCode = hashing.doHashCode(copy_words0[i]);
80  //              System.out.format("%03d%n", hashCode);
81              hashValues[i] = hashCode;
82          }
83          // get the hash values for target values
84          int[] target_hashValues = new int[random_words.length];
85          for (int i = 0; i < random_words.length; i++) {
86              // Get the hash code for the target String
87  //              System.out.print(i);
88  //              System.out.print(". " + random_words[i] + " - ");
89              hashCode = hashing.doHashCode(random_words[i]);
90  //              System.out.format("%03d%n", hashCode);
91              target_hashValues[i] = hashCode;
92          }
93          //Testing for relative sorting in Hashing Class
94  //          RelativeInsertionSort relativeInsertionSort = new RelativeInsertionSort();
95  //          relativeInsertionSort.relative_insertionSort(hashValues, copy_words0);
96  //          for (int i: hashValues){
97  //              System.out.println(i);
98  //          }
99  //          for (String j: copy_words0){
100 //              System.out.println(j);
101 //          }
102
103         hashing.populate_retrieveTargets(target_hashValues, random_words, hashValues,
                copy_words0);
104         System.out.println(hashing.comparisons);
105
106         // Analyze the distribution of hash values.
107 //          hashing.analyzeHashValues(hashValues);
108     }
109 }
```

At line 30, I used insertion sort to sort the strings. At line 40, I use a random number generator to get 42 random strings from the sorted array of strings. I then test the linear and binary search in lines 51-69. We now move onto hashing where we first apply the hashing function to the sorted words, then to the random 42 words. We then execute our populate_retrieveTargets() method.