CMPT 435 - Spring 2023 - Prof. Gormanly

Lab Three

Reece Schenck
Reece.Schenck@Marist.edu

April 12, 2023

1 Program

1.1 KnuthShuffle

Same As/Taken From Lab 2
I used ChatGPT to help me trouble shoot some errors I had using the random import in the shuffle method. The error turned out to only be a minor issue related to working with string arrays that was easily fixed. This class takes in a string list and shuffles it in the Knuth Shuffle style. Once the array has been shuffled it can be returned with the getArray() method.

See Lines 8-26 of the KnuthShuffle class.(Find under Appendix/Code/KnuthShuffle).

1.2 InsertionSort

Same As/Taken From Lab 2
I used the psudeo code from class as a skeleton version of my code. I then converted it to java and added the necessary code and requirements outlined by the assignment(adding counters, timers, etc.) and my created methods.

I start with two pointers, i and j. i starts as 1. While traversing the list, j=i+. The strings in the array indexes of j and j-1 are compared while j is greater than 0 and array[j-1] is greater than array[j]. The strings are swapped and the pointer j is decremented by 1 until at least one of the two previous conditions is false, in which i increments up 1. This continues until the entire array is sorted. Once sorted, the time is calculated and the number of comparisons has been recorded. Both are printed out.

I used ChatGPT To help me trouble shoot errors I was having starting and ending the timer.

I used '.compareToIgnoreCase' from:
https://www.w3schools.com/java/ref$_{s}tring_{c}omparetoignorecase.asp$
to compare two strings alphabetically.

See Lines 12-47 of the InsertionSort class.(Find under Appendix/Code/InsertionSort)

1.3 LinearSearch

In linear search, I traverse the array for each target item in the target array going one by one. Once the item is found the time and comparisons are printed and the next item is searched for. After all have been found, the average time and comparisons are computed.

See Lines 3-46 of the LinearSearch class.(Find under Appendix/Code/LinearSearch)

1.4 BinarySearch

In binary search I check the middle index of the array to see if it holds the target item, if not I divide the array based off whether or not the middle index was greater or less than the target. The end and mid points are changed accordingly. once the target is found, the time and comparisons as well as location are printed, the next target is then searched for. After all are found the average time and comparisons are printed. I have some issues with this, they are in the Unresolved Issues and Errors section.

See Lines 3-74 of the BinarySearch class.(Find under Appendix/Code/BinarySearch)

1.5 HashTable

I was honestly stuck on this part for nearly the whole time as I had no clue where to start or how to approach hashing. I used ChatGPT quite a bit from giving me a skeleton code to work off of, to trouble shooting the numerous errors I had along the way of adding to the base code I was given. The hash table takes in values and hashes them as linked lists so that duplicates can be chained. After being added they can be searched and returned with the get method and the comparisons can be retrieved with the getComparisons method. The Entry class was gotten from ChatGPT and is used through the hashTable class.

See Lines 3-109 of the HashTable class.(Find under Appendix/Code/HashTable)

1.6 Main

1.6.1 Imports

These are the imports I needed for my code in Main:
import java.util.ArrayList;
import java.util.Arrays;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Random;
import java.util.List;

These Are used for reading/writing from the text file into and Array, getting a random index used to shuffle a list or pick random values from a list, and basic array / array list manipulation.

See Lines 1-7 of the Main class.(Find under Appendix/Code/Main)

1.6.2 Read/Write Text File

Same from Lab 2
I used the same code from lab 1 for this lab, the only change I mad was removing the hard coded file name

To read/Write the text file into an array I first got the filename of the text file and made an array list. I then use the file reader to to read the file line by line adding them to the array list. I used ChatGPT to help me with this part.

See Lines 9-37 of the Main class.(Find under Appendix/Code/Main)

1.6.3 Searching Hashing

After writing the text file into the array, I then instantiate my shuffler using that array as the parameter. I then sort the array and shuffle it, from this shuffled array I take the first 42 items to be the target items in the searches. I did use an outside recourse to help me with this, that site being: 'https://www.geeksforgeeks.org/java-util-arrays-copyofrange-java/'. I then had to resort the initial array to start my linear search and binary search and print all the results. For the hash table, I instantiate the table with the proper size as well as other needed array lists to store the outputs. I then add all the magic items strings in the table. I then take 42 random items from the magic items(your wording in the instructions was a bit unclear so I hope this is ok) and search through the hash table for the items. I then print the results.

See Lines 43-187 of the Main class.(Find under Appendix/Code/Main)

2 Unresolved Issues and Errors

After spending hours repeatedly combing over my code, I ended up with a few errors that I could not fix. The central error being that that my Binary search seems to either not work at all, or work very slowly, I could not figure out what the true problem was or how to fix it. I used ChatGPT and researched online and was still unable to fix the issue. I had trouble with many other parts of my code, especially the hashing, nut I was able to use Chat to Trouble shoot issues or heavily edit skeleton code it generated.

3 Results

results may vary

Algorithm	Number of Comparisons	Time in Nanoseconds
Insertion Sort	115987	7392700
Linear Search	322	18689.48
Binary Search	6	10300.00
Hashing	2	103.15

4 Appendix

4.1 Code

4.1.1 KnuthShuffle

```java
import java.util.Random;
import java.util.Arrays;

public class KnuthShuffle {
    String[] array;
    Random index;

    public KnuthShuffle(String[] array){
        this.array = array;
        index = new Random();
    }

    //used ChatGPT to help me trouble shoot some errors with this section of code
    public void shuffle(){
        for(int i=array.length-1;i>0;i--){
            int j = index.nextInt(i+1);
            String temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    //used for testing if array was actually shuffled
    public String[] getArray(){
        return array;
    }
}
```

4.1.2 InsertionSort

```java
import java.util.Arrays;

public class InsertionSort {
    String[] array;

    public InsertionSort(String[] array){
        this.array = array;
    }

    //used class psudo code as a template, then modified into java with my unique classes and methods
    //used ChatGPT to help fix errors with timer
    public void sort(){
        long startTime = System.nanoTime();
        int comparisons = 0;
        int i = 1;
        int j;

        while(i < array.length){
            j = i;
            /*
            reference code(what the comparison looks like in psudeo code):
            while(j>0 && array[j-1]>array[j]){

            .compareTo gives a positive or negative int depending on whether
            array[j] is less than or greater than array[jMin] respectivly
            gives 0 if they are equal

            found this solution of comparing strings through the link below:
            https://www.w3schools.com/java/ref_string_comparetoignorecase.asp
            */
            while(j>0 && array[j-1].compareToIgnoreCase(array[j]) > 0){
                comparisons++;
                String temp = array[j];
                array[j] = array[j-1];
                array[j-1] = temp;
                j = j - 1;
            }
            i = i + 1;
        }
        //prints the time(in nanoseconds aka  s ) & number of comparisons
        long endTime = System.nanoTime();
        long elapsedTime = endTime - startTime;
        System.out.println("Insertion Sort:");
        System.out.println("Number of comparisons: " + comparisons);
        //The symbol '  ' woldn't show up so I removed it
        System.out.println("Sorting time in nanoseconds: " + elapsedTime);
    }

    //used for testing if array was actually sorted
    public String[] getArray(){
        return array;
    }
}
```

4.1.3 LinearSearch

```java
import java.util.Arrays;

public class linearSearch {
    //the array that is being searched
    String[] array;

    public linearSearch(String[] array){
        this.array = array;
    }

    public void search(String[] array, String[] targetArray){
        int comparisons = 0;
        int totalCompare = 0;
        double averageCompare;
        long startTime;
        long endTime;
        long elapsedTime;
        double totalTime = 0;
        double averageTime;

        for(int i=0;i<targetArray.length;i++){
            startTime = System.nanoTime();
            for(int j=0;j<array.length;j++){
                if(targetArray[i].compareToIgnoreCase(array[j]) == 0){
                    comparisons = j+1;
                    break;
                }
            }
            //may have to clear the timers
            endTime = System.nanoTime();
            System.out.println("Number of comparisons for target " + targetArray[i] + ": " + comparison
            elapsedTime = endTime - startTime;
            totalTime = totalTime + elapsedTime;
            //The symbol '  ' woldn't show up so I removed it
            System.out.println("Search time in nanoseconds: " + elapsedTime);
            totalCompare = totalCompare + comparisons;
            comparisons = 0;
        }
        averageCompare = totalCompare/targetArray.length-1;
        averageTime = totalTime/targetArray.length-1;
        System.out.println("Average comparisons:");
        //found .format from Bob Beechey from https://www.quora.com/How-do-I-cut-off-decimals-in-Java
        //it cuts the double's decimals down to 2 places without replacing or deleting the other places
        System.out.format("%.2f%n", averageCompare);
        System.out.println("Average time in nanoseconds:");
        System.out.format("%.2f%n", averageTime);
    }
}
```

4.1.4 BinarySearch

```java
import java.util.Arrays;

public class binarySearch {
    //the array that is being searched
    String[] array;

    public binarySearch(String[] array){
        this.array = array;
    }

    //use divide and conquer + class notes
    //print where target is found, don't have 2 exits
    //may not need recursion
    public void search(String[] array, String[] targetArray){
        int comparisons = 0;
        int totalCompare = 0;
        double averageCompare;
        //-1 means that the target is not in the array
        int targetLocation = -1;
        int midIndex = array.length/2;
        int endIndex = array.length;
        int temp;
        long startTime;
        long endTime;
        long elapsedTime;
        double totalTime = 0;
        double averageTime;

        for(int i=0;i<targetArray.length;i++){
            startTime = System.nanoTime();
            for(int j=0;j<endIndex;j++){
                if(targetArray[i].compareToIgnoreCase(array[midIndex]) == 0){
                    comparisons++;
                    targetLocation = midIndex;
                    j = endIndex;
                }else if(targetArray[i].compareToIgnoreCase(array[midIndex]) < 0){
                    comparisons++;
                    temp = midIndex;
                    endIndex = midIndex;
                    if(temp/2 > 0){
                        midIndex = temp/2;
                    }
                }else{
                    comparisons++;
                    j = midIndex;
                    //stops out of bounds errors
                    if(midIndex + (midIndex/2) < 666){
                        midIndex = midIndex + (midIndex/2);
                    }
                }
            }
            //may have to reset the timers
            endTime = System.nanoTime();
            System.out.println("Number of comparisons for target " + targetArray[i] + ": " + comparison
            System.out.println("Target location: " + targetLocation);
            elapsedTime = endTime - startTime;
            totalTime = totalTime + elapsedTime;
            //The symbol '  ' woldn't show up so I removed it
            System.out.println("Search time in nanoseconds: " + elapsedTime);
            totalCompare = totalCompare + comparisons;
            comparisons = 0;
            targetLocation = -1;
            midIndex = array.length/2;
            //array.length-1 maybe?
            endIndex = array.length;
        }
        averageCompare = totalCompare/targetArray.length-1;
        averageTime = totalTime/targetArray.length-1;
        System.out.println("Average comparisons:");
        //found .format from Bob Beechey from https://www.quora.com/How-do-I-cut-off-decimals-in-Java
        //it cuts the double's decimals down to 2 places without replacing or deleting the other places
        System.out.format("%.2f%n", averageCompare);
        System.out.println("Average time in nanoseconds:");
        System.out.format("%.2f%n", averageTime);
    }
}
```

4.1.5 HashTable

```
import java.util.LinkedList;

public class hashTable<K, V> {
    private int size;

    //I'm not fully sure if this is allowed but since we
    // are allowed to use arraylists for the read functionality
    //I'm using a premade linked list instead of my own
    //It is also what ChatGPT suggested to use so I did
    private LinkedList<Entry<K, V>>[] table;
    int comparisons = 0;

    public hashTable(int size){
        this.size = size;
        this.table = new LinkedList[size];
    }

    /*
    I guenuinly had no Idea how to do hashing so
    I used ChatGPT a lot for the parts below:
    I used it to help me start the hashisng code(layout and some code itself)
    I used it to help me troubleshoot issues from the many errors I had and have
    And I used it to help me write some parts of code I genuinly had no clue how to even attempt
    */
    private int hashFunction(K key){
        return Math.abs(key.hashCode() % size);
    }

    public void put(K key, V value){
        int index = hashFunction(key);
        Entry<K, V> entry = new Entry<>(key, value);

        if(table[index] == null){
            table[index] = new LinkedList<>();
        }

        //Checks if key already exists
        for(Entry<K, V> e : table[index]){
            if(e.getKey().equals(key)){
                e.setValue(value);
                return;
            }
        }

        //If the key is not found, a new entry is added to the linked list
        table[index].add(entry);
    }

    public V get(K key){
        int index = hashFunction(key);

        if(table[index] == null){
            return null;
        }

        //Searches for an entry with given the key
        for(Entry<K, V> e : table[index]){
            comparisons++;
            if(e.getKey().equals(key)){
                return e.getValue();
            }
        }

        //If key is not found
        return null;
    }

    //returns the number of comparisons then resets the count
    public int getComparisons(){
        int temp = comparisons;
        comparisons = 0;
        return temp;
    }

    public void remove(K key){
        int index = hashFunction(key);

        if(table[index] == null){
            return;
        }

        //Searches for an entry with the given key and removes it
        for(Entry<K, V> e : table[index]){
            if(e.getKey().equals(key)){
                table[index].remove(e);
                return;
            }
        }
    }

    private static class Entry<K, V>{
        private K key;
        private V value;

        public Entry(K key, V value){
            this.key = key;
            this.value = value;
        }

        public K getKey(){
            return key;
        }

        public V getValue(){
            return value;
        }

        public void setValue(V value){
            this.value = value;
        }
    }
}
```

4.1.6 Main

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Arrays;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        //Read and Write text file into an array

        //This wasn't working for me:
        String filename = "magicitems.txt";

        //It only worked when hard coded, so this is what I used for testing:
        //String filename = "C:\\Users\\goldh\\OneDrive\\Documents\\GitHub\\RSchenck-435\\Lab 2\\magici

        ArrayList<String> lines = new ArrayList<String>();

        try {
            BufferedReader reader = new BufferedReader(new FileReader(filename));
            String line = reader.readLine();
            while (line != null) {
                lines.add(line);
                line = reader.readLine();
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        String[] linesArray = lines.toArray(new String[lines.size()]);
        //prints out the read lines, used for testing
        //System.out.println("Lines read from file:");
        //for (String l : linesArray) {
        //    System.out.println(l);
        //}


        //insertion sort
        InsertionSort insert = new InsertionSort(linesArray);
        insert.sort();
        String[] sorted = insert.getArray();
        System.out.println(Arrays.toString(sorted));

        System.out.println(" \n");


        //initializes shuffler and shuffles
        KnuthShuffle shuffler = new KnuthShuffle(sorted);
        shuffler.shuffle();
        String[] shuffledArray = shuffler.getArray();
        //System.out.println(Arrays.toString(shuffledArray));

        /*
        I then use this shuffled array and put
        the first 42 target items into a new array.
        The shuffler essentially randomizes the list so
        the first 42 items can be considered to be random
        */

        //Found copyOfRange from:
        //https://www.geeksforgeeks.org/java-util-arrays-copyofrange-java/

        //copy elements from shuffledArray from index 0(inclusive) to 42(exclusive)
        String[] targetElements = Arrays.copyOfRange(shuffledArray, 0, 42);

        //used for testing
        //System.out.println("Length of targetElements Array: " + targetElements.length);
        //System.out.println("Target Elements: " + Arrays.toString(targetElements));


        //needed to create a new array to hold the sorted items after the knuth shuffle
        //as trying to use the original sorted array led to numerous errors
        //This was the only way I could figure out how to solve them
        insert.sort();
        String[] sorted2 = insert.getArray();
        //used for testing
        //System.out.println(Arrays.toString(sorted2));

        System.out.println(" \n");

        //linear search
        System.out.println("Linear Search:");
        linearSearch linear = new linearSearch(sorted2);
        linear.search(sorted2, targetElements);

        System.out.println(" \n");

        //reasoning is the same as for linear search
        insert.sort();
        String[] sorted3 = insert.getArray();
        //used for testing
        //System.out.println(Arrays.toString(sorted3));

        System.out.println(" \n");

        //binary search
        //if commented out everything else works
        System.out.println("Binary Search:");
        binarySearch binary = new binarySearch(sorted3);
        binary.search(sorted3, targetElements);

        System.out.println(" \n");

        /*
        done print statement used for testing as:
        binary search seems to be running for very long
        amounts of time and thus done is never printed out
        sometimes binary search prints out a few comparisons but
        other times it doesn't manage to print anything out at all
        I don't know why this is or how to fix it
        */
        System.out.println("done");
        System.out.println("\n");

        //hashing
        System.out.println("Hashing:");

        //create hash table with size 250
        hashTable<String, Integer> hash = new hashTable<>(250);

        //lines is the array list of magic items before being converted to a string list
        //I used this because it was easier to implement and troubleshoot with ChatGPT
        for(int i=0;i<lines.size();i++){
            hash.put(lines.get(i), i);
        }

        //holds the 42 items gotten from the hash table
        List<String> retrieved = new ArrayList<>();
        List<Long> times = new ArrayList<>();
        List<Integer> allComparisons = new ArrayList<>();

        Random rand = new Random();

        long startTime;
        long endTime;
        long elapsedTime;
        double totalTime = 0;
        double averageTime;
        int comparisons = 0;
        int totalCompare = 0;
        double averageCompare;

        //gets 42 random items from magicItems
        //and finds them in the hashing table
        //used ChatGPT for some help with to printing results
        for(int i=0;i<42;i++){
            int index = rand.nextInt(lines.size());
            String key = lines.get(index);
            //start search
            startTime = System.nanoTime();
            Integer value = hash.get(key);
            if(value != null){
                endTime = System.nanoTime();
                retrieved.add(key);
                //end search

                elapsedTime = endTime - startTime;
                totalTime = totalTime + elapsedTime;
                times.add(elapsedTime);

                comparisons = hash.getComparisons();
                totalCompare = totalCompare + comparisons;
                allComparisons.add(comparisons);
            }
        }

        //Prints the retrieved strings and times
        //The symbol '  ' woldn't show up so I removed it
        System.out.println("Retrieved strings & search time in nanoseconds:");
        int index = 0;
        for(String str : retrieved){
            System.out.println(str + ": " + times.get(index));
            System.out.println("Number of comparisons: " + allComparisons.get(index));
            index++;
        }
        averageTime = totalTime/lines.size();
        averageCompare = totalCompare/42;
        System.out.println("Average comparisons:");
        //found .format from Bob Beechey from https://www.quora.com/How-do-I-cut-off-decimals-in-Java
        //it cuts the double's decimals down to 2 places without replacing or deleting the other places
        System.out.format("%.2f%n", averageCompare);
        System.out.println("Average time in nanoseconds:");
        System.out.format("%.2f%n", averageTime);
    }
}
```

4.1.7 Text File (magicitems)

I’m not putting all of the words here because it’s 666 lines of text/code