Assignment Two

Shannon Cordoni

Shannon.Cordoni@Marist.edu

October 10, 2021

1 Problem One: Sorting

1.1 The Data Structure

Given a list of strings our job was to create an algorithm to go through this list and sort them. To do this we were assigned to read each element of the list into an array, and using different sorting methods such as selection, insertion, merge, and quick sort we were to sort them in alphabetical order.

1.2 Main Class

1.2.1 Description

For this class we created multiple instances of the word array created from the *magicitems.txt* files. This allowed us to pass each of these arrays to their respective sort. The code below shows this and each of these sorting methods.

```
1
2  /*
3   *
4   * Assignment 2
5   * Due Date and Time: 10/8/21 before 12:00am
6   * Purpose: To develop multiple sorting methods
7   * Input: The user will be inputting a file containing a list of words/statements
8   * Output: The program will use differnt methods to sort them
9   * @author Shannon Cordoni
10  *
11  */
12
13 import java.io.File;
14 import java.io.FileNotFoundException;
15 import java.util.Scanner;
16
17 import java.util.Random;
18
19 public class Cordoni {
20
```

```
21      //Declare keyboard
22      static Scanner keyboard = new Scanner(System.in);
23
24      public static void main(String[] args) {
25
26          //Declare and initialize variables
27          String line;
28
29          String[] wordarray = new String[666];
30          String[] selectionWordArray = new String[666];
31          String[] insertionWordArray = new String[666];
32          String[] mergeWordArray = new String[666];
33          String[] quickWordArray = new String[666];
34
35          //create new file object
36          File myFile = new File("magicitems.txt");
37
38          try
39          {
40              //create scanner
41              Scanner input = new Scanner(myFile);
42              line = null;
43
44              int i = 0;
45
46              //while there are more lines in the file it inputs them into a word array
47              while(input.hasNext())
48               {
49                  //Input into array
50                  wordarray[i] = input.nextLine();
51                  i++;
52               }//while
53
54              input.close();
55
56          }//try
57
58          //error for file not found
59          catch(FileNotFoundException ex)
60          {
61            System.out.println("Failed to find file: " + myFile.getAbsolutePath());
62          }//catch
63
64          //Error in case of a null pointer exception
65          catch(NullPointerException ex)
66          {
67              System.out.println("Null pointer exception.");
68              System.out.println(ex.getMessage());
69          }//catch
70
71          //General error message
72          catch(Exception ex)
73          {
74              System.out.println("Something went wrong");
75              ex.printStackTrace();
76          }//catch
77
78          int p = 1;
79          int r = 665;
80
81          //create selection array to match word array
82          selectionWordArray = wordarray;
83
84          //pass selection array to selection sort
85          selectionSort(selectionWordArray);
```

```
86
87          //read file again to create new array
88          //create new file object
89          File myFile2 = new File("magicitems.txt");
90
91          try
92          {
93              //create scanner
94              Scanner input = new Scanner(myFile2);
95              line = null;
96
97              int i = 0;
98
99              //while there are more lines in the file it inputs them into a word array
100             while(input.hasNext())
101              {
102                 //Input into array
103                 wordarray[i] = input.nextLine();
104                 i++;
105              }//while
106
107             input.close();
108
109         }//try
110
111         //error for file not found
112         catch(FileNotFoundException ex)
113         {
114            System.out.println("Failed to find file: " + myFile.getAbsolutePath());
115         }//catch
116
117         //create insertion array to match word array
118         insertionWordArray = wordarray;
119
120         //pass insertion array to insertion sort method
121         insertionSort(insertionWordArray);
122
123         //read file again to create new array
124         //create new file object
125         File myFile3 = new File("magicitems.txt");
126
127         try
128         {
129             //create scanner
130             Scanner input = new Scanner(myFile3);
131             line = null;
132
133             int i = 0;
134
135             //while there are more lines in the file it inputs them into a word array
136             while(input.hasNext())
137              {
138                 //Input into array
139                 wordarray[i] = input.nextLine();
140                 i++;
141              }//while
142
143             input.close();
144
145         }//try
146
147         //error for file not found
148         catch(FileNotFoundException ex)
149         {
150            System.out.println("Failed to find file: " + myFile.getAbsolutePath());
```

```
151         }//catch
152 
153         //create merge array to match word array
154         mergeWordArray = wordarray;
155 
156         //pass merge array to merge sort method
157         merge(mergeWordArray, p, r);
158 
159         //read file again to create new array
160         //create new file object
161         File myFile4 = new File("magicitems.txt");
162 
163         try
164         {
165             //create scanner
166             Scanner input = new Scanner(myFile4);
167             line = null;
168 
169             int i = 0;
170 
171             //while there are more lines in the file it inputs them into a word array
172             while(input.hasNext())
173              {
174                 //Input into array
175                 wordarray[i] = input.nextLine();
176                 i++;
177              }//while
178 
179             input.close();
180 
181         }//try
182 
183         //error for file not found
184         catch(FileNotFoundException ex)
185         {
186            System.out.println("Failed to find file: " + myFile.getAbsolutePath());
187         }//catch
188 
189         //create quick array to match word array
190         quickWordArray = wordarray;
191 
192         //pass quick array to quick sort method
193         quickSort(quickWordArray, p, r);
194 
195 
196     }//main
197 
198     //This method is the selection sort method that goes through and sorts the array using a
199     //Big Oh of n squared
200     public static void selectionSort(String[] selectionWordArray)
201     {
202 
203         int numberOfSortComparisons = 0;
204 
205        //to loop through the array to determine the next smallest position
206        for(int i = 0; i < selectionWordArray.length - 2; i++){
207 
208             int smallpostion = i;
209 
210             //to loop through the array to to compare small position with the rest of the
211             //array
212             for(int j = i + 1; j < selectionWordArray.length - 1; j++){
213 
214                 //compares to see if the value of j comes before the value of small position
215                 //in the alphabet
```

```
216                 if (selectionWordArray[j].compareToIgnoreCase(selectionWordArray[smallpostion]) < 0){
217                     smallpostion = j;
218                     numberOfSortComparisons++;
219                 }//if
220 
221                 numberOfSortComparisons++;
222             }//for j
223 
224             //swap wordarray[i] with wordarray[smallpostion]
225             if (selectionWordArray[smallpostion]!= selectionWordArray[i]){
226 
227                 String temp = selectionWordArray[i];
228                 selectionWordArray[i] = selectionWordArray[smallpostion];
229                 selectionWordArray[smallpostion] = temp;
230 
231             }//if
232 
233        }//for i
234 
235        System.out.println("Selection Sort Comparisons: " + numberOfSortComparisons);
236 
237     }//selection sort
238 
239     //This method is the insertion sort method that goes through and sorts the array using a
240     //Big Oh of n squared
241     public static void insertionSort(String[] insertionWordArray)
242     {
243         int numberOfInsertComparisons = 0;
244 
245         //to loop through the array to determine the next key
246         for(int i = 1; i < insertionWordArray.length - 2; i++){
247 
248             //sets the key to be an value of the array
249             String key = insertionWordArray[i];
250 
251             int j = i - 1;
252 
253             //while j comes before the key this loop pushes the key to where it
254             //falls in the array
255             while(( j >= 0)&&(insertionWordArray[j].compareToIgnoreCase(key) < 0)){
256 
257                 insertionWordArray[j + 1] = insertionWordArray[j];
258                 j = j - 1;
259                 numberOfInsertComparisons++;
260 
261             }//while
262 
263             //this sets the new key
264             insertionWordArray[j + 1] = key;
265 
266        }//for i
267 
268        System.out.println("Insertion Sort Comparisons: " + numberOfInsertComparisons);
269 
270     }//insertion sort
271 
272     //This method is the merge sort method that goes through and sorts the array using a Big
273     //Oh of n log n
274     public static void merge(String[] wordarray, int p, int r){
275 
276         //if the first value comes before the last value then we can move to the merge sort
277         if (wordarray[p].compareToIgnoreCase(wordarray[r]) < 0){
278             //numberOfMergeComparisons++;
279 
280             int q = p + ((r-1)/2);
```

```
281             merge(wordarray, p, q);
282             merge(wordarray, q + 1, r);
283             mergeSort(wordarray, p, q, r);
284         }//if
285     }//merge
286 
287     //This method merges the subarrays back together
288     public static void mergeSort(String[] wordarray, int p, int q, int r)
289     {
290         int numberOfMergeComparisons = 0;
291 
292         int i = 0;
293         int j = 0;
294 
295         int n1 = q - p + 1;
296         int n2 = r - q;
297 
298         String [] temparray1 = new String[n1];
299         String [] temparray2 = new String[n2];
300 
301         //sets the values of the first temp array
302         for (i = 0; i < n1; i++){
303             temparray1[i] = wordarray[p+i];
304         }//for
305 
306         //sets the values of the second temp array
307         for(j = 0; j < n2; j++){
308             temparray2[j] = wordarray[q + 1 + j];
309         }//for
310 
311         //this  helps put the smallest elements of the temp arrays in sorted order
312         for(int k = p; k < r ; k++){
313             if(temparray1[i].compareToIgnoreCase(temparray2[j]) < 0 ){
314                 wordarray[k] = temparray1[i];
315                 i = i + 1;
316                 numberOfMergeComparisons++;
317             }//if
318             else if (wordarray[k] == temparray2[j]){
319                 j = j+1;
320                 numberOfMergeComparisons++;
321             }//else
322         }//for
323        System.out.println("Merge Sort Comparisons: " + numberOfMergeComparisons);
324     }//merge sort
325 
326     //This method is the quick sort method that goes through and sorts the array using a Big
327     //Oh of n log n
328     public static void quickSort(String[] wordarray, int p, int r)
329     {
330         int numberOfQuickComparisons = 0;
331 
332         //this looks to see if the first value comes before the last value in the alphabet
333         //if so we can move to creat the partition the array
334         if (wordarray[p].compareToIgnoreCase(wordarray[r]) < 0){
335             numberOfQuickComparisons++;
336 
337             //creates the partition
338             int q = partition(wordarray, p, r);
339 
340             //calls quick sort to sort both halfs of the array
341             quickSort(wordarray, p, q - 1);
342             quickSort(wordarray, q + 1, r);
343         }//if
344 
345         System.out.println("Quick Sort Comparisons: " + numberOfQuickComparisons);
```

```
346
347     }//quick sort
348
349     //this method creates the partition of the arary
350     public static int partition(String[] wordarray, int p, int r){
351
352         //int numberOfQuickComparisons = 0;
353
354         String temp = "none";
355         String temp2 = "none";
356         String temp3 = "none";
357
358         int i = 0;
359         int j = 0;
360
361         temp = wordarray[r];
362         i = p-1;
363
364         //this looks to create the partition
365         for (j = p; j < r-1; j++){
366
367             //this looks to see whether the value is smaller than the pivot value
368             if (wordarray[j].compareToIgnoreCase(temp) < 0){
369                 //numberOfQuickComparisons++;
370                 i = i + 1;
371
372                 //these 2 swaps help swap the pivot with the leftmost element greater
373                 //than the temp value, putting the pivot in its correct place.
374                 temp2 = wordarray[i];
375                 wordarray[i] = wordarray[j];
376                 wordarray[j] = temp2;
377             }//if
378
379             temp3 = wordarray[i + 1];
380             wordarray[i + 1] = wordarray[r];
381             wordarray[r] = temp3;
382
383         }//for
384
385         return i + 1;
386     }//partition
387
388 }//MainCordoni
```

1.2.2 Description of Main Code

The code above is the code inside the main class, this includes reading in of the file, and the multiple sorting methods. These methods include selection sort, insertion sort, merge sort, and quick sort. Sadly some of these sorts were unsuccessful in execution, however this does not mean that these sorts cannot be analyzed. To go through each of these sorts we can create a table for better data understanding, this table will show each sort and their asymptotic running time:

Selection Sort	Insertion Sort	Merge Sort	Quick Sort
$O(n^2)$	$O(n^2)$	$O(nlog(n))$	$O(nlog(n))$

The table above shows a quick understanding of the sorting methods used here. To go into more detail the Selection Sort method has an asymptotic running time of $O(n^2)$. This is because if we go through each line of the selection or insertion sort and note the number of times each line is done, we can come up with a mathematical equation using these running times and the formula for an arithmetic sequence $(n(n+1))/2$.

This formula once reduced and no longer including constants boils down to $n^2 + n$, because n is smaller than n^2 we throw it away due to the fact that as n grows closer to infinity n becomes insignificant. This then leaves n^2 as the running time for selection and insertion sort. Another, quicker way to look at this is that both selection and insertion contain 2 loops, each of these loops have their own running time of n. Since we throw away constants, and since these loops are nested we can multiply these values to gain the total running time of the method. After multiplying we get n^2 as the total asymptotic running time of these methods. For quick sort and merge sort, both have an asymptotic running time of $nlog(n)$. This is because the recursion tree for both merge and quick sort has a running time of $T(n/2^k)$ at each level with k being the level of the tree, now the height of the tree is equal to $log(n)$. With the height or merging time of the tree being $log(n)$ we then have to multiply it by the size of our data set producing $nlog(n)$ to be the asymptotic running time of merge and quick sort.