CPSC 319

ASSIGNMENT #2

JOHN EZEKIEL JULIANO

30000523 T05

RASHID MAMUNUR [TA]

FEBRUARY 16, 2018

READ ME FILE

To run the program using **command line**:

Step 1:

javac Anagram.java CustomList.java SortFuncs.java

Step 2:

java Anagram [input file name].txt [output file name].txt

To run the program using **Eclipse**:

Make sure to change the arguments by going to “Run Configurations”, then the “Arguments” tab.

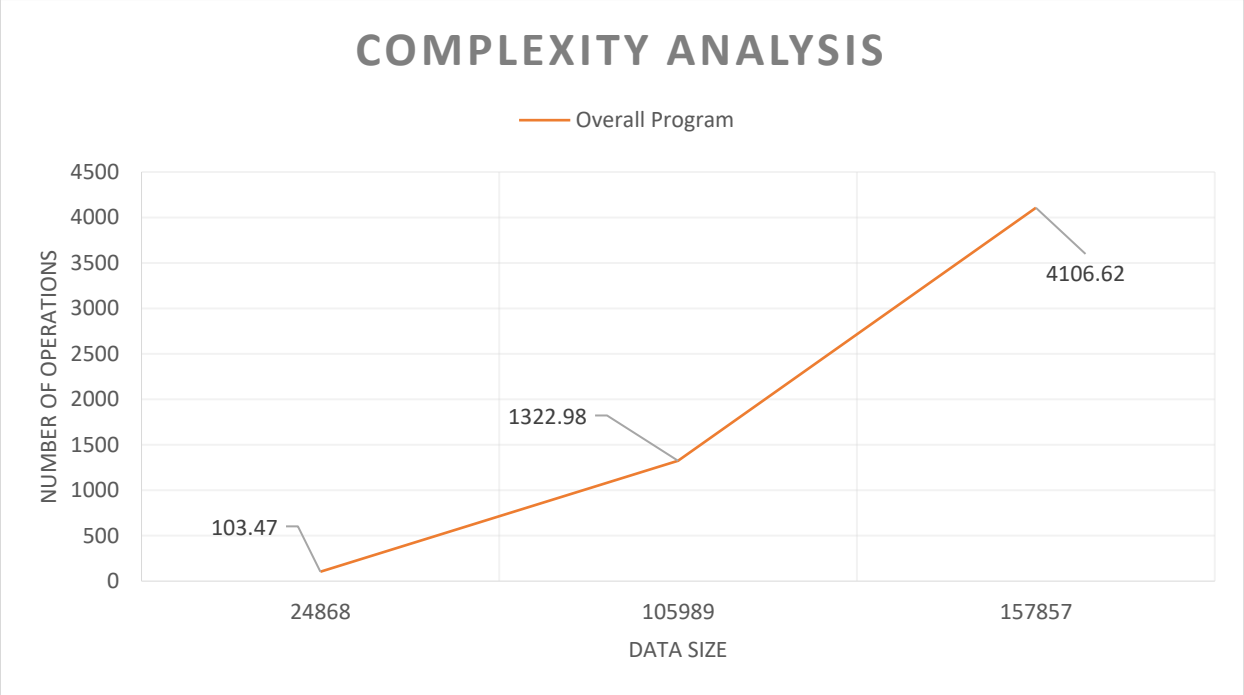
In the first box enter:

[input file name].txt [ouput file name].txt

Apply your changes, then run.

I. Data

Data Size	Overall Program
24868	103.47 seconds
105989	1322.98 seconds
157857	4106.62 seconds



II. Complexity Analysis

1. The worst-case complexity analysis of all the individual methods involved in the program can be found in the pages in the end of the file. These theoretical results will be used to compare with the experimental runs. The method used in determining whether two words are anagrams of the other is isAnagram(). Within in is another method charSort(). The total big-O value for this is $O(k^2n)$. It is assumed that 'k' is the maximum length of any word from the input file.

(Refer to the next few pages for the theoretical analysis of the program)

2. Using the same complexity analysis results however in this case the variables are:

$$n \rightarrow N \quad \text{and} \quad k \rightarrow L$$

The total theoretical big-O running time of the program is:

$$O(k^2n^2)$$

This is mainly generated by the file reader method which also contains the method to determine whether two words are anagrams or not. Even though there is a method that generated a $O(n^4)$ – this is insertionSort() inside a for-loop. It is a known fact that reading from files take longer than most operations. Therefore, I assumed that most of the program's running time will be invested in reading the input file and not on sorting the items and arrived at a conclusion that the total big-O of the program is $O(k^2n^2)$.

Observing the graph of the complexity analysis of the whole program, it can be approximated that the relationship between the data size and the number of operations is quartic (x^4). This yielded similar results as the theoretical ones; $O(k^2n^2)$ and is also true for $O(n^4)$.

```
public static void charSort(char[] in) {
for(int i = 1,j; i<in.length; i++) {
char temp = in[i];
for(j = i; (j>0)&&(temp<in[j-1]); j--)
in[j] = in[j-1];
in[j] = temp;
}
}
```

note: will use 'k^2' for simplification of analysis

(charSort)

```
i=1;
while (i<in.length)
{ temp = in[i];
  j = i;
  while (j>0 && temp<in[j-1])
  { in[j] = in[j-1];
    j--; }
  in[j] = temp;
  i++; }
```

1
$(k-1)$
$2(k-2)$
$(k-2)$
$(k-2)(k-1)$
$3(k-2)^2$
$(k-2)^2$
$2(k-2)$
$k-2$
$5k^2-12k+6$
$\Rightarrow O(k^2)$

```
public static void insertionSort(CustomList in) {
for(int i=1, j; i<in.size(); i++){
String temp = in.getData(i);
for(j=i; (j>0)&&((temp.compareTo(in.getData(j-1)))<0); j--){
String tmp = in.getData(j-1);
in.set(tmp,j);
}
in.set(temp,j);
}
}
```

(insertionSort)

```
i=1;
while (i<in.size())
{ temp = in.getData(i);
  j = i;
  while (j>0 && temp.compareTo(in.getData(j-1))<0)
  { tmp = in.getData(j-1);
    in.set(tmp, j);
    j--; }
  in.set(temp, j);
  i++;
}
```

1
$n-1$
$n(n-2)$
$n-2$
$(n-1)(n-2)$
$n(n-2)(n-2)$
$n(n-2)(n-2)$
$(n-2)(n-2)$
$n(n-2)$
$n-2$

$2(n-2)+1+n-1+2n(n-2)^2+n(n-2)^2+(n-1)(n-2)+2n(n-2)$

by inspection this results to $O(n^2)$

→ The worst case scenario used for insertionSort is when all supplied words are anagrams of each other.

→ The complexity analysis of getData(i) and set(string, i) were used to find the total # of operations for this method

```
public static void quickSort(int lo, int hi, CustomList[]
in) {
int first = lo, last = hi;
CustomList temp;
String pivot = in[(lo+hi)/2].getHead().data;
while(first<=last){
while((in[first].getHead().data).compareTo(pivot) < 0){
first++;
}
while((in[last].getHead().data).compareTo(pivot) > 0){
last--;
}
if(first<=last){
temp = in[first];
in[first] = in[last];
in[last] = temp;
first++;
last--;
}
}
if(lo < last)
quickSort(lo, last, in);
if(first < hi)
quickSort(first, hi, in);
}
```

1 1 4 ops (n+1) (n+1)/2 * (n+1)/2 - 1 * (n+1)/2 * (n+1)/2 - 1 * 1 2 3 2 1 1 1 1 1

* denotes that even subarrays are assumed to be created

Quicksort gains a log(n) proportionality due to its recursive calls.

Using the product rule, this will result into $O(n \log n)$

```
public void addFront(String text) {
Node temp = new Node();
temp.data = text;
temp.next = head;
head = temp;
size++;
}
```

⇒ O(1) because # of operations = 5

note: will use 1 as to simplify analysis for other methods

```
public String getData(int n) {
if((n<0)||(n>=size)) {
System.err.println("Invalid access. Program will now exit");
System.exit(0);
}
Node temp = head;
for(int index = 0; index<n; index++)
temp = temp.next;
return temp.data;
}
```

1op 1op (n+1) (n-1) 1op (n-1) 1op

$\Sigma = 4 + 2(n-1) + n = 3n - 3 \therefore O(n)$

note: will use 'n' as to simplify analysis for other methods

→ worst case for getData is if n is the last element & if block never runs

```
public void set(String text, int n) {
if((n<0)||(n>=size)) {
System.err.println("Invalid index. Program will now exit.");
System.exit(0);
}
Node temp = head;
for(int index = 0; index<n; index++)
temp = temp.next;
temp.data = text;
}
```

1op 1op n n-1 n-1 1op

$\Sigma = 4 + 2(n-1) + n = 3n - 3 \therefore O(n)$

note: will use 'n' as to simplify analysis of other methods

→ worst case is when setting data of the last element & if block never runs

```
public void readInputFile() throws IOException{

BufferedReader buffer = new BufferedReader(new
FileReader(fileIN));  2ops

String data;

arraySize = 0;  1op

while((data = buffer.readLine()) != null) {   n

if(!isAnagram(data)) {   (n-1)(k²n)

wordMat[arraySize] = new CustomList(data);  2(n-1)

arraySize++;  (n-1)

}

}

buffer.close();  1 op

}
```

$$\sum = 4 + n + 3(n-1) + k^2 n(n-1)$$
$$= k^2n^2 + (4 - k^2)n + 1$$
$$\therefore O(k^2n^2)$$

```
public boolean isAnagram(String text) {

char[] inputAsChar = text.toCharArray();  2ops

SortFuncs.charSort(inputAsChar);  k²

for(int i = 0; i<arraySize; i++) {
    1op      n        n-1

char[] currentAsChar =
wordMat[i].getHead().data.toCharArray();  5ops (n-1)

SortFuncs.charSort(currentAsChar);  k² (n-1)

if(Arrays.equals(inputAsChar, currentAsChar)) {  1op (n-1)

wordMat[i].addFront(text);  2ops (n-1)

return true;  1op (n-1)

}

}

return false;  1op

}
```

→ worst case is when all words are anagrams of the other

$$\sum = 4 + n + n-1 + 5(n-1) + k^2(n-1) + 2(n-1) + 2(n-1)$$
$$= 4 + n + 10(n-1) + k^2(n-1)$$
$$= (k^2 + 11)n + (-k^2 - 6)$$

m is can be treated as $O(k^2 n)$

```
public void printToFile() {

try {

cursor.print("This is the sorted list of anagrams.\n");
for(int i = 0; wordMat[i] != null; i++) {
for(int j = 0; j < wordMat[i].size(); j++) {

cursor.print(wordMat[i].getData(j) + " ");

}

cursor.println();

}

}

catch(Exception e) {
e.printStackTrace();
System.out.println("File does not exist.");

}

}
```

1 (1) n (n-1) n-1 1(n-1) 2(n-1) 1(n-1) 3(n-1) (n-1)

This analysis assumed that no anagrams were found from the input file.

$\Sigma = 9(n-1) + n + 5 = 10n + 4$

$\therefore O(n)$

```
public int numberOfWords() throws IOException {

BufferedReader reader = new BufferedReader(new
FileReader(fileIN));

lines = 0;
while (reader.readLine() != null) {

lines++;

}

reader.close();

return lines;

}
```

20ps 1op n n-1 1op 1op

$\Sigma = 5 + n + n-1 = 2n - 4 \therefore O(n)$

note: will use 'n' as simplification for other analysis

<< SortFuncs.java >>

```
public class SortFuncs {
	/*
	 * This is a modified implementation of insertion sort enabled
	 * to sort chars instead of integers
	 * @param in array to be sorted
	 */
	public static void charSort(char[] in) {
		for(int i = 1,j; i<in.length; i++) {
			char temp = in[i];
			for(j = i; (j>0)&&(temp<in[j-1]); j--)
				in[j] = in[j-1];
			in[j] = temp;
		}
	}
	/*
	 * This is an adapted verison of the insertionsort from the lectures
	 * to be able to sort through String objects
	 * @param in object that contains the String to be sorted
	 */
	public static void insertionSort(CustomList in) {
		for(int i=1, j; i<in.size(); i++){
			String temp = in.getData(i);
			for(j=i; (j>0)&&((temp.compareTo(in.getData(j-1)))<0); j--){
				String tmp = in.getData(j-1);
				in.set(tmp,j);
			}
			in.set(temp,j);
		}
	}
	/*
	 * This is an adapted version of quicksort from the lectures
	 * to be able to sort through String objects
	 * @param lo first element
	 * @param hi last element
	 * @param in array to be sorted
	 */
	public static void quickSort(int lo, int hi, CustomList[] in) {
		int first = lo, last = hi;
		CustomList temp;
		// set a pivot element
		String pivot = in[(lo+hi)/2].getHead().data;
```

```
		// divide arrays
		while(first<=last){
			// identify a number greater than pivot value from left subarray
			while((in[first].getHead().data).compareTo(pivot) < 0){
				first++;
			}
			// identify a number less than pivot value from right subarray
			while((in[last].getHead().data).compareTo(pivot) > 0){
				last--;
			}
			// swap
			if(first<=last){
				temp = in[first];
				in[first] = in[last];
				in[last] = temp;
				first++;
				last--;
			}
		}
		// recursive method call
		if(lo < last)
			quickSort(lo, last, in);
		if(first < hi)
			quickSort(first, hi, in);
	}
}
```

<< CustomList.java >>

```
public class CustomList{
	/*
	 * Defines the contents of each element of the array
	 */
	public class Node{
		String data;
		Node next = null;
	}
	/*
	 * Head pointer
	 */
	private Node head;
	/*
	 * Size of the list
```

```
 */
private int size;
/*
 * Constructors
 */
public CustomList() {
	head = null;
	size = 0;
}
public CustomList(String text) {
	addFront(text);
}
/*
 * Returns the size of the list
 */
public int size() {return size;}
/*
 * Returns the head pointer
 */
public Node getHead() {return head;}
/*
 * Adds a node in the beginning of the list and increases
 * size by 1.
 * @param text specifies data inside the node
 */
public void addFront(String text) {
	Node temp = new Node();
	temp.data = text;
	temp.next = head;
	head = temp;
	size++;
}
/*
 * Retrieves the data of the node in the nth location
 * @param n specifies the location of the node
 */
public String getData(int n) {
	if((n<0)||(n>=size)) {
		System.err.println("Invalid access. Program will now exit");
		System.exit(0);
	}
	Node temp = head;
```

```
            for(int index = 0; index<n; index++)
                  temp = temp.next;
            return temp.data;
      }
      /*
       * Changes the data within the nth node to text
       * @param text the new data
       * @param n the position of the node
       */
      public void set(String text, int n) {
            if((n<0)||(n>=size)) {
                  System.err.println("Invalid index. Program will now exit.");
                  System.exit(0);
            }
            Node temp = head;
            for(int index = 0; index<n; index++)
                  temp = temp.next;
            temp.data = text;
      }
}
```

<< Anagram.java >>

```
import java.io.*;
import java.util.Arrays;

public class Anagram {
      /*
       * User defined list to contain anagram matrix
       */
      CustomList[] wordMat;
      /*
       * File name of the input text file
       */
      String fileIN;
      /*
       * File name of the output text file
       */
      String fileOUT;
      /*
       * Time measurement fields
       */
      double start, stop, totalStart, totalStop;
      /*
```

```
 * File printing field
 */
PrintWriter cursor;
/*
 * Input size and storage array size
 */
int arraySize, lines;
/*
 * Read input text file and store into custom list
 */
public void readInputFile() throws IOException{
	BufferedReader buffer = new BufferedReader(new FileReader(fileIN));
	String data;
	arraySize = 0;
	// scans the input file by checking if the next characters is an EOL
	double now, later;
	now = System.nanoTime();
	while((data = buffer.readLine()) != null) {
		if(!isAnagram(data)) {
			wordMat[arraySize] = new CustomList(data);
			arraySize++;
		}
	}
	later = System.nanoTime();
	cursor.println("The method to determine if two words are anagrams took "+(later-now)+" nanoseconds.");
	buffer.close();
}
/*
 * Identifies the number of words to read from the file
 */
public void numberOfWords() throws IOException {
	BufferedReader reader = new BufferedReader(new FileReader(fileIN));
	lines = 0;
	while (reader.readLine() != null) {
	    lines++;
	}
	reader.close();
}
/*
 * Determines if two words are anagrams of each other
 */
public boolean isAnagram(String text) {
```

```
		char[] inputAsChar = text.toCharArray();
		SortFuncs.charSort(inputAsChar);
		// String inputText = inputAsChar.toString();
		for(int i = 0; i<arraySize; i++) {
			char[] currentAsChar = wordMat[i].getHead().data.toCharArray();
			SortFuncs.charSort(currentAsChar);
			// String currentText = currentAsChar.toString();
			if(Arrays.equals(inputAsChar, currentAsChar)) {
				wordMat[i].addFront(text);
				return true;
			}
		}
		return false;
	}
	/*
	 * Prints the output to a file
	 */
	public void printToFile() {
		try {
			cursor.print("This is the sorted list of anagrams.\n");
			// goes through all the pointers
			for(int i = 0; wordMat[i] != null; i++) {
				// goes through all the contents within pointer[i]
				for(int j = 0; j < wordMat[i].size(); j++) {
					// prints to file
					cursor.print(wordMat[i].getData(j) + " ");
				}
				cursor.println();
			}
		}
		catch(Exception e) {
			e.printStackTrace();
			System.out.println("File does not exist.");
		}
	}

	public static void main(String[] args) throws IOException {
		Anagram sample = new Anagram();

		sample.fileIN= args[0];
		sample.fileOUT	= args[1];
		sample.cursor = new PrintWriter(sample.fileOUT);
```

```
            sample.totalStart = System.nanoTime();
            System.out.println("The program has started.");
            sample.cursor.println("The program has started.");

            sample.start = System.nanoTime();
            sample.numberOfWords();
            sample.wordMat = new CustomList[sample.lines];
            sample.readInputFile();
            sample.stop = System.nanoTime();
            sample.cursor.print("Reading the input file took "+(sample.stop-sample.start)/1000000000.0+" seconds.\n");

            sample.start = System.nanoTime();
            for(int i = 0; i<sample.arraySize; i++)
                  SortFuncs.insertionSort(sample.wordMat[i]);
            sample.stop = System.nanoTime();
            sample.cursor.print("Sorting each row took "+(sample.stop-sample.start)/1000000000.0+" seconds.\n");

            sample.start = System.nanoTime();
            SortFuncs.quickSort(0,sample.arraySize-1,sample.wordMat);
            sample.stop = System.nanoTime();
            sample.cursor.print("Sorting the rows took "+(sample.stop-sample.start)/1000000000.0+" seconds.\n");

            sample.start = System.nanoTime();
            sample.printToFile();
            sample.stop = System.nanoTime();
            sample.cursor.print("Printing output to file took "+(sample.stop-sample.start)/1000000000.0+" seconds.\n");

            System.out.println("The program has ended.");
            sample.totalStop = System.nanoTime();
            sample.cursor.print("Processing the input file that contains "+sample.lines+" words took "+(sample.totalStop-
sample.totalStart)/1000000000.0+" seconds.\n");

            sample.cursor.println("The program has ended.");
            sample.cursor.close();

            System.out.print("Processing the input file that contains "+sample.lines+" words took "+(sample.totalStop-
sample.totalStart)/1000000000.0+" seconds.\n");
      }
}
```