```java
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 'K²' for simplification of analysis

(charSort)

i=1;	1
while (i<in.length)	(k-1)
{ temp =in[i];	2(k-2)
j=i;	(k-2)
while(j>0 && temp<in[j-1])	(k-2)(k-1)
{ in[j] = in[j-1];	$3(k-2)^2$
j--; }	$(k-2)^2$
in[j]=temp;	2(k-2)
i++; }	k-2

$5k^2-12k+6$

$\Rightarrow O(k^2)$

```java
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;	1
while (i<in.size())	n-1
{ temp = in.getData(i);	n(n-2)
j=i;	n-2
while (j>0 && temp.compareTo(in.getData(j-1)) <0)	(n-1)(n-2)
{ tmp = in.getData(j-1);	n(n-2)(n-2)
in.set(tmp,j);	n(n-2)(n-2)
j--; }	(n-2)(n-2)
in.set(temp,j);	n(n-2)
i++;	n-2
}	

$2(n-2) + 1 + n-1 + 2n(n-2)^2 + 3(n-2)^2 + (n-1)(n-2) + 2n(n-2)$

by inspection this results to

$O(n^3)$

→ 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;   1   1
CustomList temp;
String pivot = in[(lo+hi)/2].getHead().data;   4 ops
while(first<=last){   (n+1)
while((in[first].getHead().data).compareTo(pivot) < 0){   (n+1)/2 *
first++;   ((n+1)/2) -1 *
}
while((in[last].getHead().data).compareTo(pivot) > 0){   (n+1)/2 *
last--;   ((n+1)/2) -1 *
}
if(first<=last){   1
temp = in[first];   2
in[first] = in[last];   3
in[last] = temp;   2
first++;   1
last--;   1
}
}
if(lo < last)   1
quickSort(lo, last, in);   1
if(first < hi)   1
quickSort(first, hi, in);   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)) {        1op
System.err.println("Invalid access. Program will now
exit");
System.exit(0);
}
Node temp = head;   1op
        1op           (n)        (n-1)
for(int index = 0; index<n; index++)
temp = temp.next;   (n-1)
return temp.data;   1op
}
```

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

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)) {   1op
System.err.println("Invalid index. Program will now
exit.");
System.exit(0);
}
Node temp = head;   1op
        1op          n          n-1
for(int index = 0; index<n; index++)
temp = temp.next;   n-1
temp.data = text;   1op
}
```

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

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

if(!isAnagram(data)) {   (n-1)(k^2 n)

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

arraySize++;   (n-1)

}

}

buffer.close();   1 op

}
```

$$\Sigma = 4 + n + 3(n-1) + k^2 n(n-1)$$
$$= k^2 n^2 + (4 - k^2 \cancel{n})n + 1$$
$$\therefore O(k^2 n^2)$$

```
public boolean isAnagram(String text) {

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

SortFuncs.charSort(inputAsChar);   k^2
         1op            n          n-1
for(int i = 0; i<arraySize; i++) {

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

SortFuncs.charSort(currentAsChar);   k^2 (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

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

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

```
public void printToFile() {

try {

cursor.print("This is the sorted list of anagrams.\n");   1
           1                  n              n-1
for(int i = 0; wordMat[i] != null; i++) {
        1(n-1)          2(n-1)           1(n-1)
for(int j = 0; j < wordMat[i].size(); j++) {

cursor.print(wordMat[i].getData(j) + " ");   3(n-1)

}

cursor.println();   (n-1)

}

}

catch(Exception e) {   1

e.printStackTrace();   1

System.out.println("File does not exist.");   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));   2ops

lines = 0;   1op
                            n
while (reader.readLine() != null) {

lines++;   n-1

}

reader.close();   1op

return lines;   1op

}
```

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

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