## 1 Importing needed assets

**This section simply imports what we need from the Java library. This includes things like files, queues, stacks, scanners, and random.**

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Queue;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.Stack;
import java.util.Random;
```

## 2 Main

**This section is the main function for the class. This is what runs once the program is started so there are calls to all the functions that need to be implemented.**

```
public static void main(String[] args) throws FileNotFoundException {
    File magicItemsFile = new File(”magicitems.txt”);
    String[] magicItemsArray = fileToArray(magicItemsFile);
    String[] palindromes = findPalindrome(magicItemsArray);

    // Loops through the palindromes array and displays them
    for (int i = 0; i ¡ palindromes.length; i++) {
    System.out.println(palindromes[i]);
    }

    // Shuffles and implement selection sort
    shuffle(magicItemsArray);
    int selectionSortComparisons = selectionSort(magicItemsArray);
    System.out.println(”Selection Sort Comparisons: ” + selectionSortComparisons);

    // Shuffles and implement insertion sort
    shuffle(magicItemsArray);
    int insertionSortComparisons = insertionSort(magicItemsArray);
    System.out.println(”Insertion Sort Comparisons: ” + insertionSortComparisons);

    // Shuffles and implement merge sort
    shuffle(magicItemsArray);
    int mergeSortComparisons = mergeSort(magicItemsArray, 0, magicItemsArray.length - 1, 0);
    System.out.println(”Merge Sort Comparisons: ” + mergeSortComparisons);

    // Shuffles and implement quick sort
    shuffle(magicItemsArray);
    int quickSortComparisons = quickSort(magicItemsArray, 0, magicItemsArray.length - 1, 0);
    System.out.println(”Quick Sort Comparisons: ” + quickSortComparisons);

    // Prints out the sorted magicItemsArray
    /*
    * for (int i = 0; i ¡ magicItemsArray.length; i++) {
    * System.out.println(magicItemsArray[i]);
    * }
    */
}
```

## 3 File To Array

**This section is what is used to turn the magicitems.txt file into an array. This function scans the file, creates and array, and inputs each line of the file into the array.**

```
public static String[] fileToArray(File file) throws FileNotFoundException {

    Scanner scan = new Scanner(file);

    // Creates the array to store the magic itmes in
    String[] itemsArray = new String[666];
    int i = 0;

    // Goes through the whole text file and adds each line to the array while
    // converting them to lower case
    while (scan.hasNextLine()) {
    String data = scan.nextLine();
    data = data.toLowerCase();
    itemsArray[i] = data;
    i++;
    }
    scan.close();
    return itemsArray;
    }
```

## 4 Find Palindromes

**This section is used to find the palindromes in the magicItemsArray. Using the array we just created, this function takes each item of the array and puts them into a stack and queue. Then the stack is popped and the queue is dequeued each into a temporary string. Those strings are then compared and if every letter in the stack and queue match, we have a palindrome. Finally, any palindromes found are put into a new array which is then returned.**

```
public static String[] findPalindrome(String[] arr) {
    String[] palindromes = new String[12];
    int palindromesCounter = 0;

    Queue¡String¿ queue = new LinkedList¡¿();
    Stack¡String¿ stack = new Stack¡¿();

    // Loops through the array inputed and returns the 12 palindromes
    for (int i = 0; i ¡ arr.length; i++) {
    boolean isPalindrome = false;

    // Adds each character to a queue and stack
    for (int j = 0; j ¡ arr[i].length(); j++) {
    if (arr[i].charAt(j) != ' ') {
    String temp = Character.toString(arr[i].charAt(j));
    queue.add(temp);
    stack.push(temp);
    }
    }

    // Loops through the queue and stack chcecking if the string is a palindrome
    for (int k = 0; k ¡ queue.size(); k++) {
    String tempForQueue = queue.poll();
    String tempForStack = stack.pop();
    if (tempForQueue.compareTo(tempForStack) == 0) {
    // If all the letters match up, then a palindrome is found
    isPalindrome = true;
    } else {
    // If the letters don't match up, the loop is broken and the queue and stack are
    // cleared for another string to be entered
    isPalindrome = false;
    queue.clear();
    stack.clear();
    break;
    }
    }

    // Adds the palindromes to an array that is returned to the main function
    if (isPalindrome) {
    palindromes[palindromesCounter] = arr[i];
    palindromesCounter++;
    }
    }
    return palindromes;
    }
```

## 5 Shuffle

**This section shuffles the magicItemsArray. The shuffle used here is based off of the Knuth shuffle and takes a random number and swaps the value with another value determined by the length of the array. This happens for each value in the array.**

```
public static final Random gen = new Random();

    // Implements a shuffle based off of Knuth shuffle
    public static void shuffle(String[] arr) {
    int n = arr.length;
    while (n ¿ 1) {
    int r = gen.nextInt(n–);
    String temp = arr[n];
    arr[n] = arr[r];
    arr[r] = temp;
    }

    }
```

## 6 Selection Sort

**This section is used to sort the magicItemsArray using selection sort. This function loops through the array and for every position in the array anther loop is performed and said position's value is compared to every other position's value in the array. The smallest value found is then swapped with the value that you started with and this continues for the whole array.**

```
public static int selectionSort(String[] arr) {
    int comparisons = 0;
    String temp = "";

    // Loops through the array for each position in the array
    for (int i = 0; i ¡ arr.length - 1; i++) {
```

```
    int small = i;
    for (int j = i + 1; j ¡ arr.length; j++) {
    // Compares each value of the array to the previous value and swaps
    // them if a smaller value is found
    if (arr[j].compareTo(arr[small]) ¡ 0) {
    small = j;
    }
    comparisons++;
    }

    // Swaps the positions of the current value and smallest value
    temp = arr[i];
    arr[i] = arr[small];
    arr[small] = temp;
    }
    return comparisons;
    }
```

## 7 Insertion Sort

**This section sorts the magicItemsArray using insertion sort. Similar to selection sort, for every position in the array this function loops through every other position and compares the values. However, instead of swapping positions this function drags the first smallest value it finds until it is sorted.**

```
public static int insertionSort(String[] arr) {
    int comparisons = 0;
    String temp = "";

    // Loops through the array for each position in the array
    for (int i = 0; i ¡ arr.length; i++) {
    for (int j = i + 1; j ¡ arr.length; j++) {
    // Compares the value of the previous position to the next position and swaps
    // them
    // if a smaller value is found
    if (arr[i].compareTo(arr[j]) ¿ 0) {
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    comparisons++;
    }
    }
    }
    return comparisons;
    }
```

## 8 Merge Sort

**This section uses merge sort to sort the magicItemsArray. This function takes in the start and end of the array and splits the array until each item is split into one array. Then these are combined into slightly larger arrays while being compared to one another. Once the arrays are all put back into one, they are sorted because they are slowly being put into place at each step of the merge.**

```
public static int mergeSort(String[] arr, int from, int to, int comparisons) {
    if (from == to) {
```

```
return comparisons;
}

// Gathers the midpoint to break the array into two halves
int mid = (from + to) / 2;

// Reccursively calls the mergeSort function until the array is broken into
// individual pieces
mergeSort(arr, from, mid, comparisons);
mergeSort(arr, mid + 1, to, comparisons);

// Calls the merge function to combine the individual pieces to one array
comparisons = merge(arr, from, mid, to, comparisons);
return comparisons;
}

public static int merge(String[] arr, int from, int mid, int to, int comparisons) {
int n = to - from + 1;
String[] tempArr = new String[n];
int i1 = from;
int i2 = mid + 1;
int j = 0;

// While merging the pieces of the array, each piece is compared to the other
// and sorted
while (i1 ¡= mid && i2 ¡= to) {
if (arr[i1].compareTo(arr[i2]) ¡ 0) {
tempArr[j] = arr[i1];
i1++;
comparisons++;
} else {
tempArr[j] = arr[i2];
i2++;
comparisons++;
}
j++;
}

while (i1 ¡= mid) {
tempArr[j] = arr[i1];
i1++;
j++;
comparisons++;
}

while (i2 ¡= to) {
tempArr[j] = arr[i2];
i2++;
j++;
comparisons++;
}

for (j = 0; j ¡ n; j++) {
arr[from + j] = tempArr[j];
comparisons++;
}
return comparisons;
}
```

## 9 Quick Sort

**This section uses quick sort to sort the magicItemsArray. Similarly to merge sort, this function splits the array into smaller pieces. However, while the array(s) is/are being split, that is when they are compared and sorted. Then once each item is by itself they are already sorted and just have to be combined back into one array.**

```
public static int quickSort(String[] arr, int lower, int higher, int comparisons) {
    int i = lower;
    int j = higher;
    String pivot = arr[i + (j - i) / 2];
    String temp = ””;

    while (i ¡= j) {

    while (arr[i].compareTo(pivot) ¡ 0) {
    i++;
    comparisons++;
    }

    while (arr[j].compareTo(pivot) ¿ 0) {
    j–;
    comparisons++;
    }

    // If the lower index is still less than the higher index, then the values of
    // said
    // indexes are swapped
    if (i ¡= j) {
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j–;
    comparisons++;
    }

    // Calls the quickSort function recursively
    if (lower ¡ j) {
    comparisons++;
    quickSort(arr, lower, j, comparisons);
    }
    if (i ¡ higher) {
    comparisons++;
    quickSort(arr, i, higher, comparisons);
    }
    }
    return comparisons;
    }
```

## 10 Results

### 10.1 Palindromes

1. boccob
2. ebuc cube
3. olah halo
4. radar
5. robot tobor
6. dacad
7. ufo tofu
8. dior droid
9. taco cat
10. golf flog
11. was it a rat i saw
12. aibohphobia

### 10.2 Sorting Comparisons

| Sorting Type | Comparisons | Running Time |
|---|---|---|
| Selection Sort | 221445 | O$n$ |
| Insertion Sort | 109666 | O$n$ |
| Merge Sort | 1332 | O$log(n)$ |
| Quick Sort | 669 | O$log(n)$ |