

Assignment Two – L^AT_EX Sorts

Daniel Bilotto
danielbilotto1@marist.edu

October 9, 2021

1 Main Method

```
.  
  
//this main method prints out the number of comparisons from all the sorts  
//it reads in the array from magic items after every sort as well  
public static void main(String[] args)  
{  
    String[] list = new String[size];  
    file(list);  
    selectSort(list);  
    file(list);  
    insertSort(list);  
    file(list);  
    System.out.println("Merge_sort_comparisons:" + MergeSort.sort(list));  
    file(list);  
    System.out.println("Quick_sort_comparisons:"  
    + QuickSort.sort(list, 0, list.length - 1));  
  
} //main
```

2 File Method

```
.  
  
//this is basically the same file as lab 1  
//all it does is read in the items from magic items.
```

```

public static void file(String[] items)
{
    String fileName = null;
    String line = null;

    try
    {
        fileName = "magicitems.txt";

        File theFile = new File(fileName);

        Scanner input = new Scanner(theFile);

        for (int i = 0; i < size; i++)
        {
            line = input.nextLine().toLowerCase();
            items[i] = line;
        }

        input.close();
        keyboard.close();
    } // try

    catch (Exception ex)
    {
        System.out.println("Oops, something went wrong!");
    } // catch

    // this sends back a message if something goes wrong in
    // importing the text into the array from magic items
} // file

```

3 Selection Sort Method

This method is the selection sort method. It runs through the whole array comparing the smallest item (aka earliest in the alphabet) with the other items and swaps if necessary and increments compare, basically it shifts which item is considered min and then once it runs through the whole array it will send it to the left of the array. The reason why it is has the run time of $O(n^2)$ is because of the two for loops that go to n each that are nested. This is shown in the table below.

```

public static void selectSort(String[] items)
{
    int n = items.length;

```

```

    int compare = 0;
    for (int i = 0; i < n - 1; i++)
    {
        int smallPos = i;
        for (int j = smallPos + 1; j < n; j++)
        {
            compare++;
            if (items[j].compareTo(items[smallPos]) < 0)
            {
                smallPos = j;
            }
        }
        String swap = items[i];
        items[i] = items[smallPos];
        items[smallPos] = swap;
    }
    System.out.println("Selection_sort_comparisons:" + compare);
} //SS

```

4 Insert Sort Method

. What this method does looks to the left of of the index and compares it with that number and swaps if needed The reason why it is has the run time of $O(n^2)$ is because it has a while loop thats nested in a for loop that goes to n , and the while effectively goes to n because it runs through the whole array because of the for loop it's nested in. This is shown in the table below.

```

public static void insertSort(String[] items)
{
    int n = items.length;
    int compare = 0;

    for (int i = 1; i < n ; i++)
    {
        String key = items[i];
        int j = i - 1;
        while ((j > -1) && (items[j].compareTo(key) > 0))
        {
            compare++;
            items[j + 1] = items [j];
            j--;
        }
        items[j+1] = key;
    }
}

```

```

        System.out.println("Insert_sort_comparisons:" + compare);
    } //IS

```

5 Merge Sort Class

This method was inspired from a lot of pseudocode from the text book and online sources and was easily the hardest one. Basically the first half is the divide part hence why you have many lines that say $/2$, and they divide into 2 arrays called left and right. After that it will run through the array checking if the left is smaller than the right and if so make the value in the array that item. And that also means by process of elimination, if it isn't smaller then the right is smaller and hence that item should go in the array first. It's runtime is big o of $n \log n$ because the loops are going to the sub array of left and right as well as its being divided by 2. This is shown in the table below.

```

public class MergeSort
{
    static int compare = 0;
    public static int sort(String[] items)
    {
        if (items.length > 1)
        {
            String[] left = new String[items.length / 2];
            String[] right =
            new String[items.length - items.length / 2];

            for (int i = 0; i < left.length; i++)
            {
                left[i] = items[i];
            }

            for (int i = 0; i < right.length; i++)
            {
                right[i] = items[i + items.length / 2];
                compare++;
            }

            sort(left);
            sort(right);
            merge(items, left, right);
        }
        return compare;
    } //sort

    public static void merge(String[] items, String[] left, String[] right)

```

```

{
    int i = 0;
    int j = 0;

    for (int k = 0; k < items.length; k++)
    {
        if (j >= right.length || (i < left.length &&
        left[i].compareToIgnoreCase(right[j]) < 0))
        {
            items[k] = left[i];
            i++;
            compare++;
        }
        else
        {
            items[k] = right[j];
            j++;
            compare++;
        }
    }
} //merge
}

```

6 Quick Sort Method

This method was mostly from the textbook, basically you pass it a array, the starting num, and the end number so it knows how many times to go through the array. Which looking back I could of used a for loop for... whooops. Anyways the pivot number is going to start at whatever the end number is when you pass it in the sort method. Once the pivot is set, it will start to run through and compare the j (number in the loop) to the pivot and swap when needed. It's runtime is big o of n log n because it's being split apart by each pivot and having to work with multiple arrays on the left and right. This is shown in the table below.

```

public class QuickSort
{
    static int compare = 0;

    public static int sort(String[] items, int begin, int end)
    {
        if (begin < end)
        {
            int partitionInd = partition(items, begin, end);

```

```

        sort(items, begin, partitionInd - 1);
        sort(items, partitionInd + 1, end);
    }

    return compare;
}

public static int partition(String[] array, int begin, int end)
{
    String pivot = array[end];
    int i = (begin - 1);

    for (int j = begin; j < end; j++)
    {
        if (array[j].compareTo(pivot) < 0)
        {
            i++;
            String swap = array[i];
            array[i] = array[j];
            array[j] = swap;
        }
        compare++;
    }

    String swapTwo = array[i + 1];
    array[i + 1] = array[end];
    array[end] = swapTwo;

    return i + 1;
} // partition
}

```

Sort Name	Complexity	number of compares
Selection	$O(n \text{ to the second})$	221445
Insert	$O(n \text{ to the second})$	67906
Merge	$O(n * \log n)$	9615
Quick	$O(n * \log n)$	7238