Lab One

Your Name

Robert.Perrone1@Marist.edu

October 1, 2020

1 Results

Table 1.1: Asymptotic Running Time

Selection Sort	Insertion Sort	Merge Sort	Quick Sort
3459	114309	5413	3643

The asymptotic running time for Selection Sort is O(n$\hat{2}$), for Insertion Sort is O(n), for Mergesort is O(nlogn(n)), and Quicksort is also O(nlog(n)). This is the worst case for each and will never be worse than these functions. Selection Sort is always O(n$\hat{2}$) because we take the cost of each line, which resolves to n*n when all of the constants have been disregarded. Insertion sort is always O(n) because all of its lines of code involve a constant times n and when the constants are removed, we are left with just n. Mergesort and Quicksort are both always O(nlogn(n)) because they both divide and conquer to sort. While Mergesort divides and then conquers, Quicksort divides and conquers at the same time. Both, however, lead to the same runtime when constants are removed.

2 Main Class

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
   public static void main(String[] args) throws FileNotFoundException {
      //Initialize Selection, Insertion, Merge, and Quick Sorts
      SelectionSort selectionSort = new SelectionSort();
      InsertionSort insertionSort = new InsertionSort();
      MergeSort mergeSort = new MergeSort();
      QuickSort quickSort = new QuickSort();

      //Initialize scanner to scan through magicitems.txt file
      Scanner input = new Scanner(new File("src/magicitems.txt"));

      final int arrSize = 666;

      //Initialize array to hold the magicitems
      String[] magicitemsSelection = new String[arrSize];
      String[] magicitemsInsertion = new String[arrSize];
      String[] magicitemsMerge = new String[arrSize];
      String[] magicitemsQuick = new String[arrSize];

      //Add each item in magicitems to the array
      for(int i = 0; input.hasNextLine(); i++) {
         String item = input.nextLine().toUpperCase();
         magicitemsSelection[i] = item; //add next line to array
         magicitemsInsertion[i] = item;
         magicitemsMerge[i] = item;
         magicitemsQuick[i] = item;
         //System.out.println(item);
      }//end for i
      System.out.println(selectionSort.selectionSort(magicitemsSelection));
      System.out.println(insertionSort.insertionSort(magicitemsInsertion));
      System.out.println(mergeSort.mergeSort(magicitemsMerge));
      System.out.println(quickSort.quickSort(magicitemsQuick));
   }//end main
}//end Main Class
```

3 Selection Sort Class

```
public class SelectionSort {
   public int selectionSort(String[] array) {
      int comparisons = 0;
      int n = array.length;

      //Go through each item in the string array to compare
      for (int i = 0; i < n; i++) {
         //set min index
         int smallPos = i;
         //set min string based off of min index
         String smallString = array[i];
         //Compare minString to the next item in the array
         for (int j = i + 1; j < n; j++) {
            if (array[j].compareTo(smallString) < 0) {
               smallPos = j;
               smallString = array[j];
               comparisons++;
            }//end if
         }//end for x
         if (smallPos != i) {
            String temp = array[smallPos];
            array[smallPos] = array[i];
            array[i] = temp;
         }//end if
      }//end for i

      //print out array to check
      /*for(int x = 0; x < n; x++) {
         System.out.println(array[x]);
      }*/ //end for x
      //System.out.println();
      //number of comparisons done during sort
      System.out.println("Selection Sort Comparisons:");
      return comparisons;
   }//end selectionSort
}//end class SelectionSort
```

4 Insertion Sort Class

```java
public class InsertionSort {
   public int insertionSort(String[] array) {
      /*Insertion-Sort(A)
       *    for j = 2 to A.length
       *       key = A[j]
       *       //insert A[j] into the sorted sequence A[1..j-1]
       *       i = j - 1
       *       while i > 0 and A[i] > key
       *          A[i + 1] = A[i]
       *          i = i - 1
       *       A[i + 1] = key */
      int comparisons = 0;
      int n = array.length;

      for(int i = 0; i < n - 1; i++) {
         for(int j = i + 1; j < n; j++) {
            if(array[i].compareTo(array[j]) > 0) {
               String temp = array[i];
               array[i] = array[j];
               array[j] = temp;
               comparisons++;
            }//end if
         }//end for j
      } //end for i

      //print out array to check
      /*for(int x = 0; x < n; x++) {
         System.out.println(array[x]);
      }*/ //end for x
      //System.out.println();
      //number of comparisons done during sort
      System.out.println("Insertion Sort Comparisons:");
      return comparisons;
   }//end insertionSort
}//end class InsertionSort
```

5 Merge Sort Class

```
public class MergeSort {
   public static int comparisons = 0;
   public int mergeSort(String[] array) {
      /*Merge(A,p,q,r)
      *  n1 = q - p + 1
      *  n2 = r - q
      *  let L[1..n1 + 1] R[1..n2 + 1] be new arrays
      *  for i = 1 to n1
      *     L[i] = A[p + i - 1]
      *  for j = 1 to n2
      *     R[j] = A[q + j]
      *  L[n1 + 1] = infinity
      *  R[n2 + 1] = infinity
      *  i = 1
      *  j = 1
      *  for k = p to r
      *     if L[i] <= R[j]
      *        A[k] = L[i]
      *        i = i + 1
      *     else A[k] = R[j]
      *        j = j + 1
      * Merge-Sort(A, p, r)
      *  if p < r
      *     q = [(p + r)/2]
      *     Merge-Sort(A, p, q)
      *     Merge-Sort(A, q + 1, r)
      *     Merge(A, p, q, r)*/
      int n = array.length;
      int  mid = n/2;

      this.divide(array, n);

      //print out array to check
      /*for(int x = 0; x < n; x++) {
         System.out.println(array[x]);
      }*/ //end for x
      //System.out.println();
      //number of comparisons done during sort
      System.out.println("Merge Sort Comparisons:");
      return comparisons;
   }//end mergeSort

   public static void divide(String[] array, int n) {
      if(n == 1) {
         return;
      } else {
         int mid = n / 2;
         String[] left = new String[mid];
         String[] right = new String[n - mid];//n-mid to account for odd numbers

         //Divide array into left and right arrays
         int j = 0;
         for (int i = 0; i < n; i++) {
            if (i < mid) {
```

```java
               left[i] = array[i];
            } else {
               right[j] = array[i];
               j += 1;
            }// end if
         }// end for i

         //Keep dividing the whole until n == 1
         //if(n == 1 ){
         divide(left, mid);
         divide(right, n - mid);
         //}
         //merge left and right arrays into the whole array. mid=leftLength n-mid=rightLength
         //else {
         merge(left, right, array, mid, n - mid);
         //}
      }//end if else
   }//end sort
   public static void merge(String[] left, String[] right, String array[], int left_n, int right_n) {
      int arrayInt = 0;
      int leftInt = 0;
      int rightInt = 0;
      //Only merge if in order, while loops to check
      while(leftInt < left_n && rightInt < right_n) {
         if(left[leftInt].compareTo(right[rightInt]) < 0) {
            array[arrayInt++] = left[leftInt++];
            comparisons++;
         } else {
            array[arrayInt++] = right[rightInt++];
            comparisons++;
         }//end if
      }//end while
      //while left is less than the size of left array
      while(leftInt < left_n) {
         array[arrayInt++] = left[leftInt++];
      }//end while
      //While right is less than the size of the right array
      while(rightInt < right_n) {
         array[arrayInt++] = right[rightInt++];
      }//end while
      /*if(left_n != 0 && right_n != 0) {
         merge(left, right, array, left_n, right_n);
      }*/
   }//end merge
}//end class MergeSort
```

6 Quick Sort Class

```
import java.util.ArrayList;
import java.util.Random;
public class QuickSort {
   public static int comparisons = 0;
   public int quickSort(String[] array) {
      /*QuickSort(A,p,r)
      *  if p < r
      *     q = Partition(A, p , r)
      *     QuickSort(A, p, q - 1)
      *     QuickSort(A, q + 1, r)
      * Partition(A, p, r)
      *  x = A[r]
      *  i = p - 1
      *  for j = p to r - 1
      *     if A[j] <= x
      *        i = i + 1
      *        exchange A[i] with A[j]
      *  exchange A[i + 1] with A[r]
      *  return i + 1 */
      int n = array.length;

      this.sort(array, 0, n - 1);

      //print out array to check
      /*for(int x = 0; x < n; x++) {
         System.out.println(array[x]);
      } //end for x
      System.out.println();*/
      //number of comparisons done during sort
      System.out.println("Quick Sort Comparisons:");
      return comparisons;
   }//end quickSort
   public static void sort(String array[], int begin, int n) {
      if(begin < n) {
         int q = partition(array, begin, n);

         sort(array, begin, q - 1);
         sort(array, q + 1, n);
      }//end if
   } //end divide
   public static int partition(String array[], int begin, int n) {
      //Select Pivot Randomly
      Random random = new Random();
      //generate three random values
      int a = random.nextInt(n);
      int b = random.nextInt(n);
      int c = random.nextInt(n);

      //Select three random pivot points
      String pivot1 = array[a];
      String pivot2 = array[b];
      String pivot3 = array[c];
      String medPivot = "";

      //Select a median pivot to actually pivot around
      /*if((b < a && a < c)
            || (c < a && a < b)) { //Check pivot1
         medPivot = pivot1;
      } else if((a < b && b < c)
            || (c < b && b < a)) { //Check pivot2
         medPivot = pivot2;
      } else {
         medPivot = pivot3;
      }*/

      /*if((pivot2.compareTo(pivot1) < 0 && pivot1.compareTo(pivot3) < 0)
            || (pivot3.compareTo(pivot1) < 0 && pivot1.compareTo(pivot2) < 0)) { //Check pivot1
         medPivot = pivot1;
      } else if((pivot1.compareTo(pivot2) < 0 && pivot2.compareTo(pivot3) < 0)
            || (pivot3.compareTo(pivot2) < 0 && pivot2.compareTo(pivot1) < 0)) { //Check pivot2
         medPivot = pivot2;
      } else {
         medPivot = pivot3;
      }*/

      /*String temp1 = medPivot;
      medPivot = array[n];
      array[n] = temp1;*/

      medPivot = array[n];

      int i = (begin - 1);
      for(int j = begin; j < n; j++) {
         if(array[j].compareTo(medPivot) < 0) {
            i++;
            String temp2 = array[i];
            array[i] = array[j];
            array[j] = temp2;
            comparisons++;
         }//end if
      }//end for j
      String temp3 = array[i+1];
      array[i+1] = array[n];
      array[n] = temp3;

      return i + 1;
   }//end partition
}//end class QuickSort
```