CSCI 3412 – Algorithms

Jeremy Diamond

Problem Set 1 – Sorting Algorithms

Part 1 – Select Sorting Algorithms

For my naive sort O(n^2) I selected bubble sort for familiarity. For my O(n ln(n)) sort I chose Merge sort for ease of implementation and consistent behavior. For my O(n) sort I chose counting sort for simplicity.

Part 2 – Write Pseudo-Code for each of the algorithms

- Bubble Sort

```
BUBBLESORT(A)
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

Taken from page 40 of the textbook. This algorithm swaps elements one at a time to cause the smallest values to ”bubble” up to the top.

- Merge Sort
  Taken from pages 30-34 of the textbook. This algorithm is made up of 2 distinct methods. MergeSort and Merge.

  - MergeSort

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

This method receives an array and two integers to serve a top and bottom indexes for a subarray. If that sub array has 2 or more elements in it the sub array is divided in half and both halves reclusively call the method. This recursive call is the main loop that will be discussed in part 3.

- Merge

```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1 . . n_1 + 1] and R[1 . . n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

This method is a subloop of the MergeSort method. It takes two sorted subarrays and combines them into a signal array. It does this by looking at the first element of each moving the smaller one into the new array and looping this behavior until both are empty. This Pseudo-code contains a feature I never implanted where you make the last element of each subarray an arbitrarily large number to speed up merging. For more complete explications see pages 30-34 of the text.

- Counting Sort

  There was no available version of this to be references so I wrote it myself. I likely didn't look hard enough.

  CountingSort(A)

  1. B = array with indexes 1 to x where x is the max value of an element in A
  2. for i = 1 to A.length
  3. n = A[i]
  4. B[n-1]++

  This algorithm simply increments the value of an index n-1 in B for every n that is an element of A. It only works if the elements are integers, with both an upper, and lower bound.

Part 3 – Static Analysis

- Bubble Sort
    - The loop invariant for the main loop (lines 1-4) of Bubble sort is:
      The smallest value form index i to index A.length-1 in not yet known to be in index i.
    - Initialization: Initially no values have been evaluated so the lowest value can't be know to be in index 1.
    - Maintenance: Assume that i != A.length. Between these two indexes there must be a lowest value to be moves to the lowest position. After that is done i is incremented and the loop restarts.
    - Termination: The loop terminates when i = A.lenght. At this point all of the previous sub arrays of i to A.lenght have had their smallest value moved to the first index to the array must be sorted.
- Merge Sort
    - The loop invariant for the main loop (lines 1-5) of Merge sort is:
      The sub array A[p] to A[r] contains more than 1 element.
    - Initialization: Initially no sub arrays have been split. Thus any unsorted set A must have more then 1 element.
    - Maintenance: Assume that the subarray used to call MergeSort has more then 1 element in it. The function then calls itself recursively with 2 new smaller sub arrays
    - Termination: The loop terminates when the recursive tree hits the level of sub arrays with 1 or 0 elements an which point all the respective Merge methods are called.
- Counting Sort
    - The loop invariant for the main loop (lines 2-4) of Counting sort is:
      There is another value in set A to be sorted.
    - Initialization: Initially no values have been sorted so if set A exists there must be a next value.
    - Maintenance: Assume that set i <= A.length. The value is sorted and the loop restarts.
    - Termination: The loop terminates when i = A.length. At this point all of the previous elements of A must have been sorted, thus the array is sorted.

Part 4 – Implementation

All three algorithms were coded in Java in 3 separate files. One for running bubble sort on the million value file. Another for running all files on the three sorts. The third for running various sizes on all three sorts. As a result of my lack of Java experience there are a few oddities with this code

- The sorts are implemented inside of try blocks because I don't yet know the proper way to get around the required try catch ion Java file io.
- In order to count the operations and swaps I needed to pass an int by reference to MergeSort and Merge. To do this I used 1 element arrays.
- Merge doesn't have the lower bound optimization of the above Pseudo-Code

Bubble_alg_HW1.java

```java

package bubble_alg_hw1;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author Jeremy G Diamond
 */
public class Bubble_alg_HW1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int numOfComps = 0;
        String file = "one-million-randoms.txt";


        bubble(file);
    }

    public static void bubble(String fileName) {
        int[] toBeSorted;
        String currentLine = null;
        int convert = 0;
        int counter = 0;
        int size = 0;
        long compCounter = 0;
        long swapCounter = 0;

       try{

        FileReader reader = new FileReader(fileName);// reads text file

         BufferedReader bufferedReader =
                 new BufferedReader(reader);

         currentLine = bufferedReader.readLine(); //skips first line

         currentLine = bufferedReader.readLine();// reads in size as a string
         convert = Integer.parseInt(currentLine);// converts string to int
         toBeSorted = new int[convert];// allowcates array
         size = convert; // sets the size of the array

```

```
48
49
50          while((currentLine = bufferedReader.readLine()) != null) {
51                  convert = Integer.parseInt(currentLine);
52                  toBeSorted[counter] = convert;
53
54                  counter++;
55                  //System.out.println(currentLine);
56              }
57
58              for (int i = 0; i < size-1; i++)// bubble sort algor
59               for (int j = 0; j < size-i-1; j++){
60                  compCounter++;
61                  if (toBeSorted[j] > toBeSorted[j+1])
62                  {
63                      // swap temp and arr[i]
64                      int temp = toBeSorted[j];
65                      toBeSorted[j] = toBeSorted[j+1];
66                      toBeSorted[j+1] = temp;
67                      swapCounter++;
68                  }
69              }
70              //for (int i = 0; i < size; i++) //print results used for testing
71                //  System.out.println(toBeSorted[i]);
72              System.out.println("num of comps is");
73              System.out.println(compCounter);
74              System.out.println("num of swaps is");
75              System.out.println(swapCounter);
76
77          }
78          catch(FileNotFoundException ex) { // default catch for FileNotFoundException
79              System.out.println(
80                  "Unable to open file '" +
81                  fileName + "'");
82          }
83
84          catch(IOException ex) { // default catch for IOException
85              System.out.println(
86                  "Error reading file '"
87                  + fileName + "'");
88              // Or we could just do this:
89              // ex.printStackTrace();
90          }
91
92      }
93
94  }
95
```

MultiSize.java

```java

package multisize;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;


public class MultiSize {


    public static void main(String[] args) {

        for (int i = 1; i < 1000000; i = i*10 ){

            System.out.print(i);
            System.out.println(" elements");
            meargeHead(i);
            countSort(i);
            bubble(i);//commented out to save time
            System.out.println("");
        }
    }


    public static void bubble(int sizeOf) {

        String fileName = "one-million-randoms.txt";
        int[] toBeSorted;
        toBeSorted = new int[sizeOf];// allowcates array
        String currentLine = null;
        int convert = 0;
        int counter = 0;
        int size = 0;
        long compCounter = 0;
        long swapCounter = 0;

       try{

        FileReader reader = new FileReader(fileName);// reads text file

         BufferedReader bufferedReader =
                 new BufferedReader(reader);

         currentLine = bufferedReader.readLine(); //skips first line

         currentLine = bufferedReader.readLine();// reads in size as a string
         convert = Integer.parseInt(currentLine);// converts string to int

```

```
50
51              size = convert; // sets the size of the array
52
53              for (int i = 0; i < sizeOf; i++)
54              {
55                  currentLine = bufferedReader.readLine();
56                  convert = Integer.parseInt(currentLine);
57                  toBeSorted[i] = convert;
58
59              }
60
61               for (int i = 0; i < sizeOf-1; i++)// bubble sort algor
62                for (int j = 0; j < sizeOf-i-1; j++){
63                  compCounter++;
64                  if (toBeSorted[j] > toBeSorted[j+1])
65                  {
66                      // swap temp and arr[i]
67                      swapCounter++;
68                      int temp = toBeSorted[j];
69                      toBeSorted[j] = toBeSorted[j+1];
70                      toBeSorted[j+1] = temp;
71                  }
72                }
73               //for (int i = 0; i < size; i++) //print results used for testing
74                //  System.out.println(toBeSorted[i]);
75               System.out.println("num of comps for bubble is");
76               System.out.println(compCounter);
77               System.out.println("num of swaps for bubble is");
78               System.out.println(swapCounter);
79
80          }
81              catch(FileNotFoundException ex) { // default catch for FileNotFoundException
82                 System.out.println(
83                      "Unable to open file '" +
84                      fileName + "'");
85              }
86
87             catch(IOException ex) { // default catch for IOException
88                  System.out.println(
89                      "Error reading file '"
90                      + fileName + "'");
91                  // Or we could just do this:
92                  // ex.printStackTrace();
93              }
94
95
96
97
98      }
99
```

```
99
100     public static void meargeHead(int sizeOf) {
101
102         String fileName = "one-million-randoms.txt";
103         int[] toBeSorted;
104         toBeSorted = new int[sizeOf];// allowcates array
105         String currentLine = null;
106         int convert = 0;
107         int counter = 0;
108         int size = 0;
109         int[] compCounter = new int[1]; //this is hacked up way to pass a pointer to an int
110         compCounter[0] = 0;
111         int[] swapCounter = new int[1]; //this is hacked up way to pass a pointer to an int
112         swapCounter[0] = 0;
113
114        try{
115
116         FileReader reader = new FileReader(fileName);// reads text file
117
118         BufferedReader bufferedReader =
119                 new BufferedReader(reader);
120
121         currentLine = bufferedReader.readLine(); //skips first line
122
123         currentLine = bufferedReader.readLine();// reads in size as a string
124         convert = Integer.parseInt(currentLine);// converts string to int
125         size = convert; // sets the size of the array
126
127
128         for (int i = 0; i < sizeOf; i++)
129         {
130                 currentLine = bufferedReader.readLine();
131                 convert = Integer.parseInt(currentLine);
132                 toBeSorted[i] = convert;
133
134         }
135
136           mergeSort (toBeSorted, 0, sizeOf-1, compCounter, swapCounter);
137
138          //for (int i = 0; i < size; i++) //print results used for testing
139            // System.out.println(toBeSorted[i]);
140          System.out.println("num of comps for merge is");
141          System.out.println(compCounter[0]);
142          System.out.println("num of swaps for merge is");
143          System.out.println(swapCounter[0]);
144
145        }
146         catch(FileNotFoundException ex) { // default catch for FileNotFoundException
147           System.out.println(
148               "Unable to open file '" +
```

```
149                     fileName + "'");
150             }
151
152             catch(IOException ex) { // default catch for IOException
153                 System.out.println(
154                     "Error reading file '"
155                     + fileName + "'");
156                 // Or we could just do this:
157                 // ex.printStackTrace();
158             }
159
160
161     }
162
163     public static void countSort(int sizeOf) {
164
165         String fileName = "one-million-randoms.txt";
166         String currentLine = null;
167         int convert = 0;
168         int counter = 0;
169         int size = 0;
170         int swapCounter = 0;
171         int[] countArr = new int[100];
172
173         for (int i = 0; i <100; i++)// set default value of array
174             countArr[i] = 0;
175
176         try{
177
178         FileReader reader = new FileReader(fileName);// reads text file
179
180          BufferedReader bufferedReader =
181                  new BufferedReader(reader);
182
183          currentLine = bufferedReader.readLine(); //skips first line
184
185          currentLine = bufferedReader.readLine();// reads in size as a string
186          convert = Integer.parseInt(currentLine);// converts string to int
187          size = convert; // sets the size of the array
188
189
190          for (int i = 0; i < sizeOf; i++)
191          {
192                  currentLine = bufferedReader.readLine();
193                  convert = Integer.parseInt(currentLine);
194                  countArr[convert]++;
195                  swapCounter++;
196             }
197
198
```

```
194                     countArr[convert]++;
195                     swapCounter++;
196                 }
197
198
199
200
201             //for (int i = 0; i < size; i++) //print results used for testing
202               // System.out.println(toBeSorted[i]);
203             System.out.println("num of comps for count is");
204             System.out.println("0");//hard coded to save time and space
205             System.out.println("num of swaps for count is");
206             System.out.println(swapCounter);
207
208         }
209           catch(FileNotFoundException ex) { // default catch for FileNotFoundException
210             System.out.println(
211                 "Unable to open file '" +
212                 fileName + "'");
213           }
214
215           catch(IOException ex) { // default catch for IOException
216               System.out.println(
217                   "Error reading file '"
218                   + fileName + "'");
219               // Or we could just do this:
220               // ex.printStackTrace();
221 }
222
223
224
225     }
226
227     public static void mergeSort(int arrToSort[], int x, int y, int daCount[], int swaCount[]){
228         daCount[0]++;
229         if (x < y) {
230
231             int m = (x + y) / 2;// Find the mid
232
233             mergeSort(arrToSort, x, m, daCount, swaCount);// Sort both halves
234             mergeSort(arrToSort, m + 1, y, daCount, swaCount);
235
236             merge(arrToSort, x, m, y, daCount, swaCount); // Merge the sorted halves
237         }
238
239     }
240
241     public static void merge(int arrToSort[], int x, int m, int y, int daCount[], int swaCount[]){
242
243         int s1 = m - x + 1; // set sizes of two subarrays
```

```
243             int s1 = m - x + 1; // set sizes of two subarrays
244             int s2 = y - m;
245 
246             int left[] = new int [s1]; // temp arrays
247             int right[] = new int [s2];
248 
249 
250             for (int i=0; i<s1; ++i) //Copy data to temp arrays
251                 left[i] = arrToSort[ x + i];
252             for (int j=0; j<s2; ++j)
253                 right[j] = arrToSort[m + 1+ j];
254 
255 
256             int i = 0, j = 0; //indexes of first and second subarrays
257 
258             int k = x ;// index of merged array
259 
260             while (i < s1 && j < s2) //merge the 2 arrays
261             {
262                 swaCount[0]++;
263                 daCount[0]++;
264                 if (left[i] <= right[j])
265                 {
266                     arrToSort[k] = left[i];
267                     i++;
268                 }
269                 else
270                 {
271                     arrToSort[k] = right[j];
272                     j++;
273                 }
274                 k++;
275             }
276 
277             while (i < s1)
278             {
279                 daCount[0]++;
280                 arrToSort[k] = left[i];
281                 i++;
282                 k++;
283             }
284 
285             while (j < s2)//copy extra elements
286             {
287                 daCount[0]++;
288                 arrToSort[k] = right[j];
289                 j++;
290                 k++;
291             }
292         }
293 
294 
295     }
296 
```

Jeremy_Diamond_algor_HW1.java

```java
package jeremy_diamond_algor_hw1;

import java.io.*;
public class Jeremy_Diamond_algor_HW1 {

    public static void main(String args[]) {

        String file = "shuffled.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "duplicate.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "nearly-sorted.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "nearly-unsorted.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "sorted.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "unsorted.txt";
        System.out.println(file);
        meargeHead(file);
        countSort(file);
        bubble(file);
        System.out.println("");

        file = "one-million-randoms.txt";
```

```
51          System.out.println(file);
52          meargeHead(file);
53          countSort(file);
54          //bubble(file);//commented out to save time
55          System.out.println("");
56      }
57
58      public static void bubble(String fileName) {
59          int[] toBeSorted;
60          String currentLine = null;
61          int convert = 0;
62          int counter = 0;
63          int size = 0;
64          int compCounter = 0;
65          int swapCounter = 0;
66
67          try{
68
69          FileReader reader = new FileReader(fileName);// reads text file
70
71           BufferedReader bufferedReader =
72                   new BufferedReader(reader);
73
74           currentLine = bufferedReader.readLine(); //skips first line
75
76           currentLine = bufferedReader.readLine();// reads in size as a string
77           convert = Integer.parseInt(currentLine);// converts string to int
78           toBeSorted = new int[convert];// allowcates array
79           size = convert; // sets the size of the array
80
81
82           while((currentLine = bufferedReader.readLine()) != null) {
83                   convert = Integer.parseInt(currentLine);
84                   toBeSorted[counter] = convert;
85
86                   counter++;
87                   //System.out.println(currentLine);
88              }
89
90             for (int i = 0; i < size-1; i++)// bubble sort algor
91              for (int j = 0; j < size-i-1; j++){
92                   compCounter++;
93                   if (toBeSorted[j] > toBeSorted[j+1])
94                   {
95                       // swap temp and arr[i]
96                       swapCounter++;
97                       int temp = toBeSorted[j];
98                       toBeSorted[j] = toBeSorted[j+1];
99                       toBeSorted[j+1] = temp;
100                  }
```

```
101                 }
102                 //for (int i = 0; i < size; i++) //print results used for testing
103                 //  System.out.println(toBeSorted[i]);
104                 System.out.println("num of comps for bubble is");
105                 System.out.println(compCounter);
106                 System.out.println("num of swaps for bubble is");
107                 System.out.println(swapCounter);
108 
109             }
110             catch(FileNotFoundException ex) { // default catch for FileNotFoundException
111                 System.out.println(
112                     "Unable to open file '" +
113                     fileName + "'");
114             }
115 
116             catch(IOException ex) { // default catch for IOException
117                 System.out.println(
118                     "Error reading file '"
119                     + fileName + "'");
120                 // Or we could just do this:
121                 // ex.printStackTrace();
122             }
123 
124 
125 
126 
127     }
128 
129     public static void meargeHead(String fileName) {
130 
131         int[] toBeSorted;
132         String currentLine = null;
133         int convert = 0;
134         int counter = 0;
135         int size = 0;
136         int[] compCounter = new int[1]; //this is hacked up way to pass a pointer to an int
137         compCounter[0] = 0;
138         int[] swapCounter = new int[1]; //this is hacked up way to pass a pointer to an int
139         swapCounter[0] = 0;
140 
141         try{
142 
143         FileReader reader = new FileReader(fileName);// reads text file
144 
145          BufferedReader bufferedReader =
146                 new BufferedReader(reader);
147 
148          currentLine = bufferedReader.readLine(); //skips first line
149 
150          currentLine = bufferedReader.readLine();// reads in size as a string
```

```
201             int swapCounter = 0;
202             int[] countArr = new int[100];
203
204             for (int i = 0; i <100; i++)// set default value of array
205                 countArr[i] = 0;
206
207            try{
208
209             FileReader reader = new FileReader(fileName);// reads text file
210
211              BufferedReader bufferedReader =
212                      new BufferedReader(reader);
213
214              currentLine = bufferedReader.readLine(); //skips first line
215
216              currentLine = bufferedReader.readLine();// reads in size as a string
217              convert = Integer.parseInt(currentLine);// converts string to int
218              size = convert; // sets the size of the array
219
220
221              while((currentLine = bufferedReader.readLine()) != null) {
222                      convert = Integer.parseInt(currentLine);
223                      countArr[convert]++;
224                      swapCounter++;
225
226                      //System.out.println(currentLine);
227                 }
228
229
230
231
232               //for (int i = 0; i < size; i++) //print results used for testing
233                 // System.out.println(toBeSorted[i]);
234              System.out.println("num of comps for count is");
235              System.out.println("0");//hard coded to save time and space
236              System.out.println("num of swaps for count is");
237              System.out.println(swapCounter);
238
239            }
240              catch(FileNotFoundException ex) { // default catch for FileNotFoundException
241                 System.out.println(
242                      "Unable to open file '" +
243                      fileName + "'");
244              }
245
246              catch(IOException ex) { // default catch for IOException
247                   System.out.println(
248                       "Error reading file '"
249                       + fileName + "'");
250                  // Or we could just do this:
```

```
                // ex.printStackTrace();
            }



    }

    public static void mergeSort(int arrToSort[], int x, int y, int daCount[], int swaCount[]){
        daCount[0]++;
        if (x < y) {

            int m = (x + y) / 2;// Find the mid

            mergeSort(arrToSort, x, m, daCount, swaCount);// Sort both halves
            mergeSort(arrToSort, m + 1, y, daCount, swaCount);

            merge(arrToSort, x, m, y, daCount, swaCount); // Merge the sorted halves
        }

    }

    public static void merge(int arrToSort[], int x, int m, int y, int daCount[], int swaCount[]){

        int s1 = m - x + 1; // set sizes of two subarrays
        int s2 = y - m;

        int left[] = new int [s1]; // temp arrays
        int right[] = new int [s2];


        for (int i=0; i<s1; ++i) //Copy data to temp arrays
            left[i] = arrToSort[ x + i];
        for (int j=0; j<s2; ++j)
            right[j] = arrToSort[m + 1+ j];


        int i = 0, j = 0; //indexes of first and second subarrays

        int k = x ;// index of merged array

        while (i < s1 && j < s2) //merge the 2 arrays
        {
            swaCount[0]++;
            daCount[0]++;
            if (left[i] <= right[j])
            {
                arrToSort[k] = left[i];
                i++;
            }
            else
            {
                arrToSort[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < s1)
        {
            daCount[0]++;
            arrToSort[k] = left[i];
            i++;
            k++;
        }

        while (j < s2)//copy extra elements
        {
            daCount[0]++;
            arrToSort[k] = right[j];
            j++;
            k++;
        }
    }

}
```

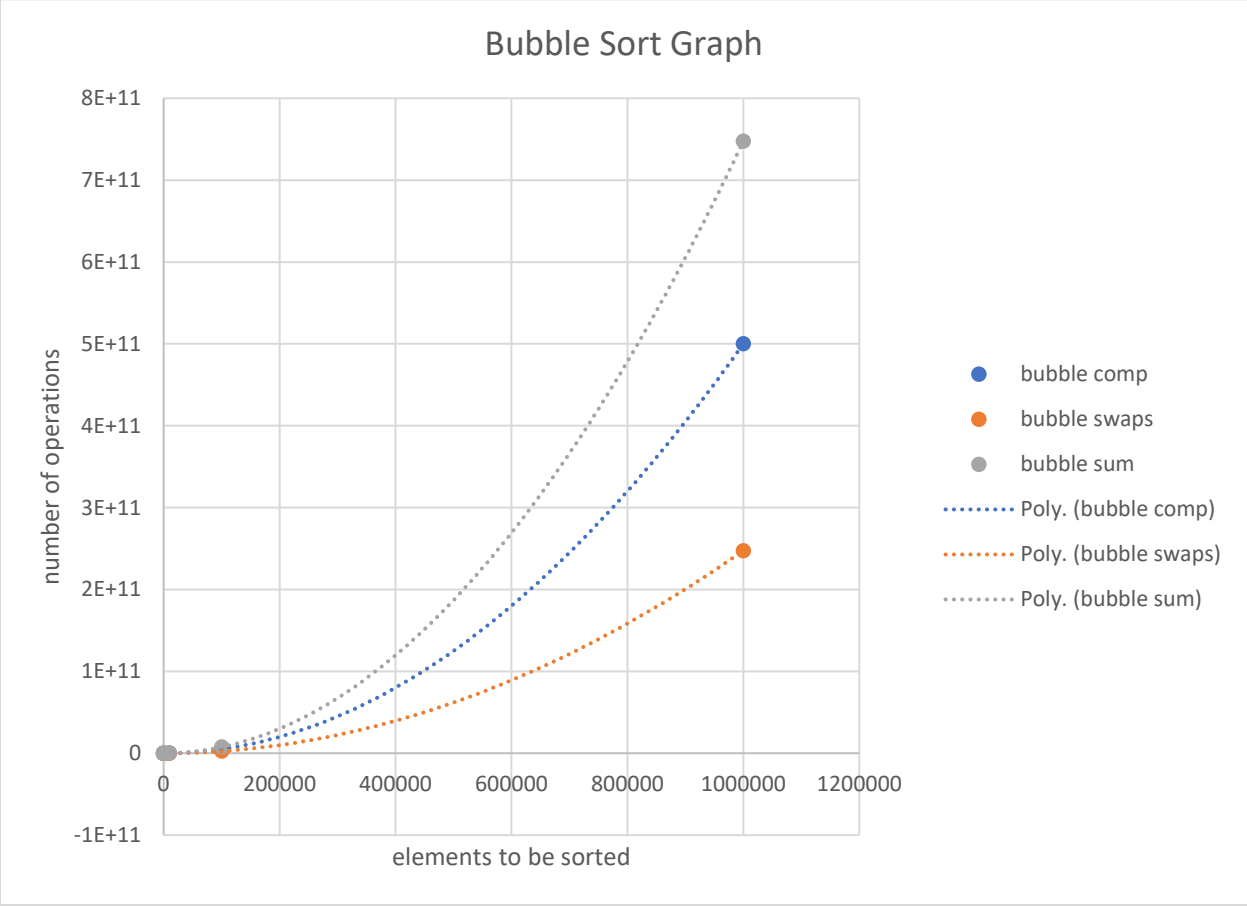
Part 5 – Analysis

- Analysis of Growth

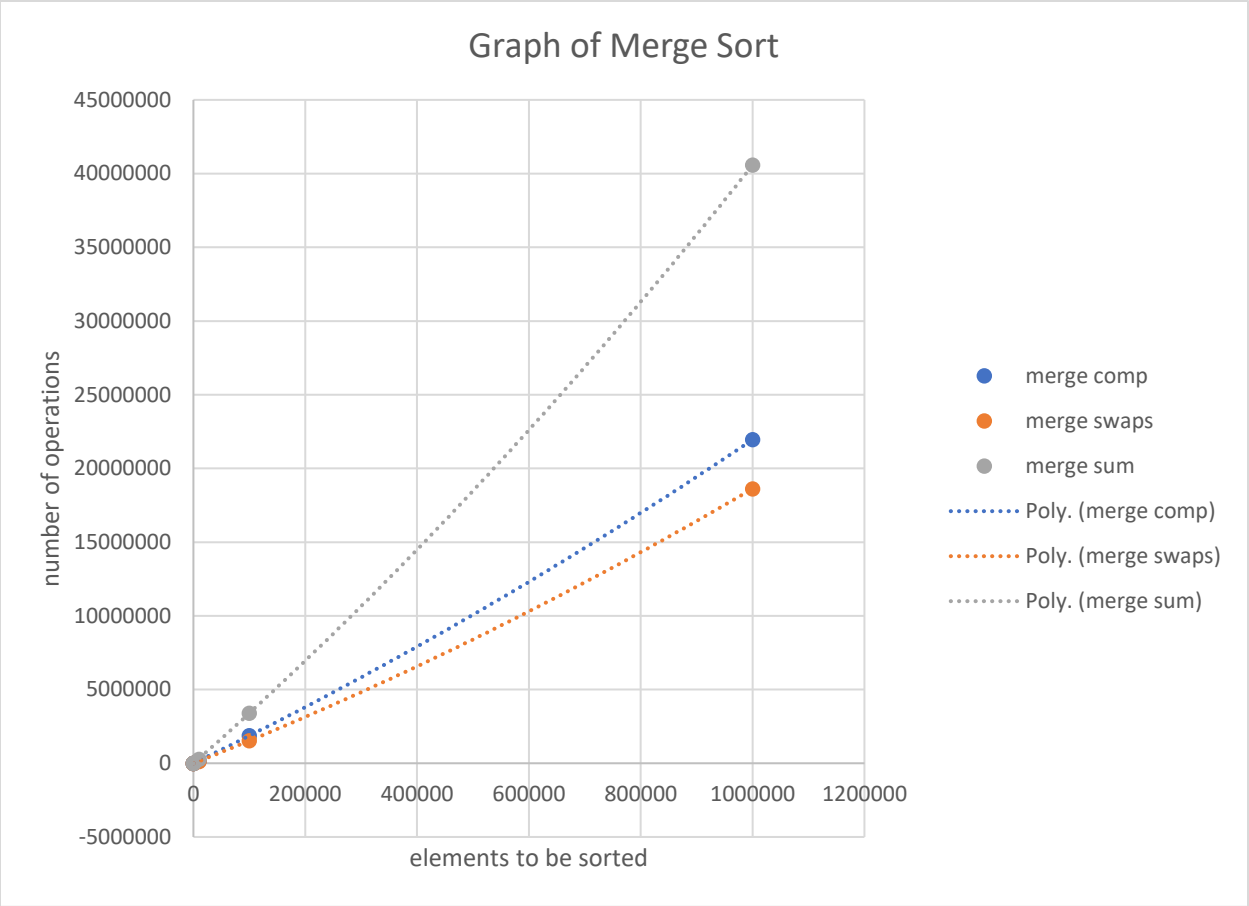
  Growth was paprbolic on bubble sort was nln(n) on Merge Sort and constant on Counting Sort see the graphs and table below.

n	bubble comp	merge comp	count comp	bubble swaps	merge swaps	count swaps	bubble sum	merge sum	count sum
1	1	1	0	0	0	1	1	1	1
10	45	53	0	17	24	10	62	77	10
100	4950	871	0	2319	549	100	7269	1420	100
1000	499500	11975	0	245606	8710	1000	745106	20685	1000
10000	49995000	153615	0	24694055	120315	10000	74689055	273930	10000
100000	4999950000	1868927	0	2485926493	1532834	100000	7485876493	3401761	100000
1000000	5E+11	21951423	0	2.47471E+11	18620724	1000000	7.4747E+11	40572147	1000000

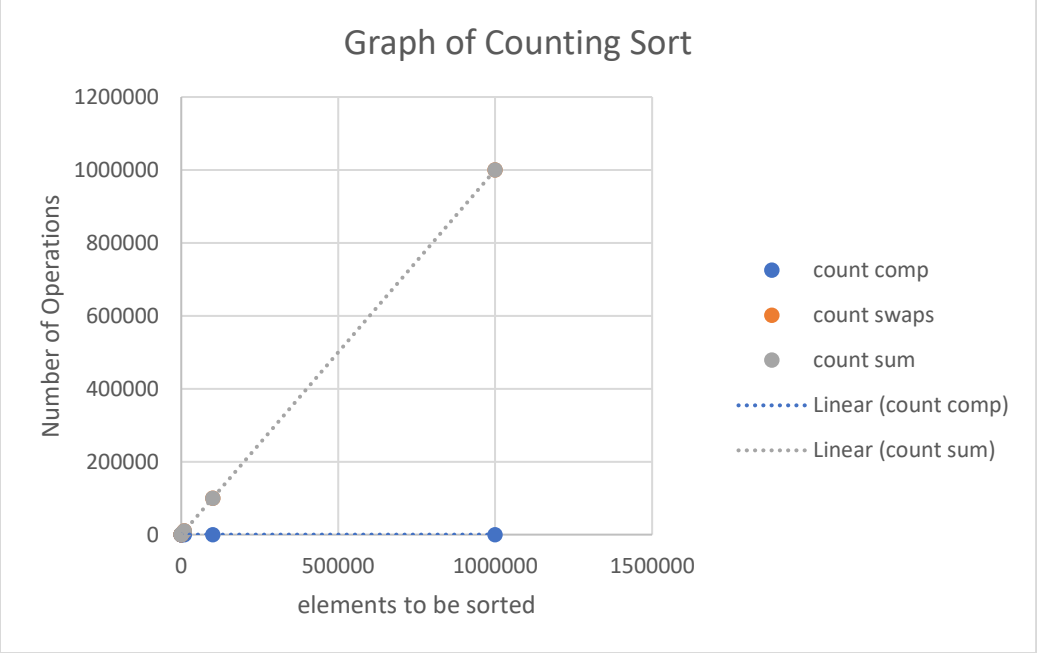
Graph of bubble sort



Graph of Merge Sort



Graph of comparison sort



As can be plainly seen from the data and graphs the sorts all perform in practice as is predicted by the above analysis.

- Comparison Across Data Sets

  All three Sorts preformed identically for every dataset of size 100. By this I mean bubble sort did not vary, in number of comparisons or swaps, with different files of the same length. The same holds true for counting sort and merge sort. I suspect there are lower bound optimizations that I didn't consider before implantation.

- Conclusions

  If ever possible use information about the set to reduce the sorting time. If ever in a situation where the previous just isn't possible use a O(n ln(n)) sort so other don't use a bad sort by mistake.