Design & Analysis of Algorithms Programming Assignment Report

1. **Random Array Generator Pseudocode**

```
Print "What size array do you want?"

Size = input

Int[] arr = new int[size]

For I = 0 to size

        Arr[i] = random number

        Write arr[i] to file

Close file
```

2. **Sorting Algorithms Pseudocode**

```
Print "Enter a name of a file for input"
filename = input
open file named filename
If file doesn't exit
        Print "dosent exist"
        Break
While (file has lines of text)
        Size++
Close file
Int[] arr = new int[size]
Reopen file
Int I = 0
While (file has more lines of text)
        Arr[i] = line of text
Close file
Print "How do you want to sort the array"
Print "b for bubble sort"
Print "I for insertion sort"
Print "m for merge sort"
Choice = input

If choice == b bubbleSort(arr)
If choice == I insertionSort(arr)
If choice == m mergeSort(arr)

toPrint (int[] arr, String inputFile)
        create and open a file named SORTED + filename
        for I = 0 to arr.length
                print arr[i] to file
        close file
        print "Finished! Check for the sorted version of the file!"
```

3. **Time Evaluation**

```
Print “What size would you like”
Size  = input
Int arr = new int [arr]
For I = 0 to size
        arr[i] = random number
print “How do you want to sort the array”
Print “b for bubble sort”
Print “I for insertion sort”
Print “m for merge sort”
Choice = input

If choice == b bubbleSort(arr)
        startTime
        int[] result = bubbleSort(arr)
        endTime
        duration = endTime – startTime
        toString(result, duration)
If choice == I insertionSort(arr)
        startTime
        int[] result = insertionSort(arr)
        endTime
        duration = endTime – startTime
        toString(result, duration)
If choice == m mergeSort(arr)
        startTime
        int[] result = mergeSort(arr)
        endTime
        duration = endTime – startTime
        toString(result, duration)

toString(int[] arr, duration)
        for I = 0 in arr.length
                result + “ “
        System.out.println(result)
        System.out.print("Finished! duration: " + duration + " nanoseconds")
```

4. **Efficiency Measure**

NOTE: My Java Virtual Machine would not accept arrays larger than 10,000,000 integers, so the sorting algorithms run until they take 300,000,000 nanoseconds, or 0.005 minutes to complete

```
Class TimeEvalutaion
        N = 1
        testSize = 10
        PrintWriter writer = new PrintWriter("EfficiencyMeasure.txt", "UTF-8")
        writer.println("Array Size,Bubble Sort,Insertion Sort,Merge Sort,QuickSort")
        while (durationB < 300000000 || durationI < 300000000 || durationM < 300000000 ||
durationQ < 300000000)
        for(int i = 0; i < testSize; i++)
              {myarray[i] = rn.nextInt(10);}
```

```
te.array = myarray;
writer.print(myarray.length + ",")
if(durationB < 300000000)
        durationB = te.sort(myarray, 'b')
        writer.print(durationB + ",")
      else
        writer.print(",");//prints comm
      if(durationI < 300000000)
        durationI = te.sort(myarray, 'i')
        writer.print(durationI + ",")
      else
        writer.print(",")
      if(durationM < 300000000)
        durationM = te.sort(myarray, 'm')
        writer.print(durationM + ",")
      else
        writer.print(",")
      if(durationQ < 300000000)
        durationQ = te.sort(myarray, 'q')
        writer.print(durationQ + ",")
      else
        writer.print(",")
      writer.println()
      testSize *= 10
    writer.close()
```

5. **Bubble Sort Pseudocode**

```
bubbleSort(int[] arr)

for I = 0 to arr.length

        for a = 0  to arr.length-1

                if arr[a] > arr[a+1]

                        swap arr[a] and arr[a+1]

return arr
```

6. **Insertion Sort Pseudocode**

```
insertionSort(int[] arr)
for a = 0 in arr
        for b = a-1; b >= 0; b—
                if arr[e+1] < arr[e]
                        swap arr[e+1] and arr[e]
return arr
```

7. **Merge Sort Pseudocode**

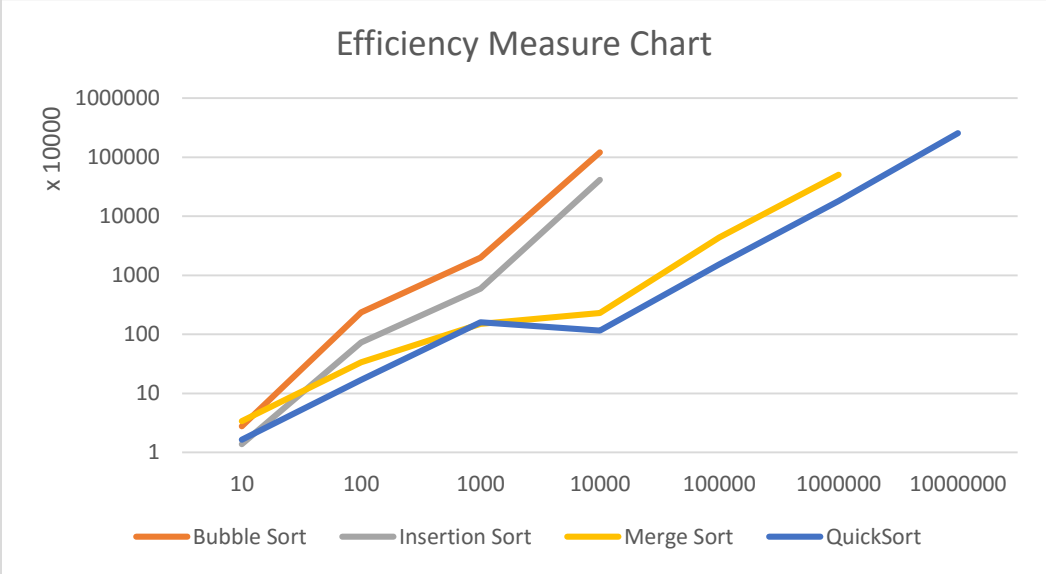
```
mergeSort(int[] arr)
buffer = new int[arr.length]
split(arr, buffer, 0, arr.length)
return arr

split(int[] arr, int[] buffer, int start, int end)
if start < end
        int middle = start + end /2
        split(arr, buffer, start, middle)
        split(arr, buffer, middle+1, end)
        merge(arr, buffer, start, middle, end)


merge(int[] arr, int[] buffer, int start, int middle, int end)

buffer = array

bufferStart = start

bufferMiddle = middle +1

current = start

while(bufferMiddle <= middle && bufferStart <= end)

        if buffer[bufferMiddle] <= buffer[bufferStart]

                array[bufferMiddle] = buffer[bufferMiddle]

                bufferStart++

        else

                array[current] = buffer[bufferMiddle]

                bufferStart++

        current++

int lastElems = middle – bufferMiddle

for g = 0 g <= lastElems g++

        array[current + g] = buffer[bufferMiddle + g]
```

8. **QuickSort**

```
quicksort(int lowerIndex, int higherIndex)
        if higherIndex ==  -1 // array is empty
                return
        int partlow = lowerIndex
        int parthigh = higherIndex
        int pivot = array[lowerIndex+(higherIndex-lowerIndex)/2];
```

```
while partlow <= parthigh
        while array[partlow] < pivot
                partlow++
        while array[parthigh] > pivot
                parthigh—
        if(partlow <= parthigh)
                int temp = array[partlow]
                array[partlow] = array[parthigh]
                array[parthigh] = temp
                partlow++
                parthigh—
if lowerIndex < parthigh
        quicksort(lowerIndex, parthigh)
if (partlow < higherIndex)
        quicksort(partlow, higherIndex)
```

9. **Efficiency Measure Graph**



Although the test had to stop short of 5 minutes, you can see that both bubble and insertion sort are much more inefficient than both merge and quick sort, because the time it takes for them to sort increases drastically.

Merge sort outperforms quick sort for a while, then quick sort proves to be the most efficient and grows slower than all of the other algorithms