DESIGN AND ANALYSIS OF ALGORITHMS

CSCI 323 - 32

PROGRAMMING ASSIGNMENT # 1

Due Date: April 14, 2021

Humaira Qadeer
Humaira.qadeer29@qmail.cuny.edu

Source Code

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

public class Main {
    public static void main(String[] args) throws IOException {

        BufferedReader readText=new BufferedReader(new FileReader("Num8.txt"));
        int[] num8=sortNumArray(readText, 8);
        readText=new BufferedReader(new FileReader("Num16.txt"));
        int[] num16=sortNumArray(readText, 16);
        readText=new BufferedReader(new FileReader("Num32.txt"));
        int[] num32=sortNumArray(readText, 32);
        readText=new BufferedReader(new FileReader("Num64.txt"));
        int[] num64=sortNumArray(readText, 64);
        readText=new BufferedReader(new FileReader("Num128.txt"));
        int[] num128=sortNumArray(readText, 128);
        readText=new BufferedReader(new FileReader("Num256.txt"));
        int[] num256=sortNumArray(readText, 256);
        readText=new BufferedReader(new FileReader("Num512.txt"));
        int[] num512=sortNumArray(readText, 512);
        readText=new BufferedReader(new FileReader("Num1024.txt"));
        int[] num1024=sortNumArray(readText, 1024);
        readText=new BufferedReader(new FileReader("Num2048.txt"));
        int[] num2048=sortNumArray(readText, 2048);
        readText=new BufferedReader(new FileReader("Num4096.txt"));
        int[] num4096=sortNumArray(readText, 4096);
        readText=new BufferedReader(new FileReader("Num8192.txt"));
        int[] num8192=sortNumArray(readText, 8192);
        readText=new BufferedReader(new FileReader("Num16384.txt"));
        int[] num16384=sortNumArray(readText, 16384);

        System.out.println("\n" + "INSERTION SORT" + "\n");
        InsertionSort InsertionSort=new InsertionSort();
        InsertionSort.InsertionSort(num8);
        InsertionSort.InsertionSort(num16);
        InsertionSort.InsertionSort(num32);
        InsertionSort.InsertionSort(num64);
        InsertionSort.InsertionSort(num128);
        InsertionSort.InsertionSort(num256);
        InsertionSort.InsertionSort(num512);
        InsertionSort.InsertionSort(num1024);
        InsertionSort.InsertionSort(num2048);
        InsertionSort.InsertionSort(num4096);
        InsertionSort.InsertionSort(num8192);
        InsertionSort.InsertionSort(num16384);
        readText.close();
        MergeSort MergeSort=new MergeSort();
        System.out.println("MERGE SORT" + "\n");
        MergeSort.MergeSort(num8);
        MergeSort.MergeSort(num16);
        MergeSort.MergeSort(num32);
        MergeSort.MergeSort(num64);
        MergeSort.MergeSort(num128);
        MergeSort.MergeSort(num256);
        MergeSort.MergeSort(num512);
        MergeSort.MergeSort(num1024);
        MergeSort.MergeSort(num2048);
        MergeSort.MergeSort(num4096);
        MergeSort.MergeSort(num8192);
        MergeSort.MergeSort(num16384);

        HeapSort HeapSort=new HeapSort();
        System.out.println("HEAP SORT" + "\n");
        HeapSort.HeapSort(num8);
        HeapSort.HeapSort(num16);
        HeapSort.HeapSort(num32);
        HeapSort.HeapSort(num64);
        HeapSort.HeapSort(num128);
        HeapSort.HeapSort(num256);
        HeapSort.HeapSort(num512);
        HeapSort.HeapSort(num1024);
        HeapSort.HeapSort(num2048);
        HeapSort.HeapSort(num4096);
        HeapSort.HeapSort(num8192);
```

```
        HeapSort.HeapSort(num16384);

        QuickSort QuickSort=new QuickSort();
        System.out.println("QUICK SORT" + "\n");
        QuickSort.QuickSort(num8);
        QuickSort.QuickSort(num16);
        QuickSort.QuickSort(num32);
        QuickSort.QuickSort(num64);
        QuickSort.QuickSort(num128);
        QuickSort.QuickSort(num256);
        QuickSort.QuickSort(num512);
        QuickSort.QuickSort(num1024);
        QuickSort.QuickSort(num2048);
        QuickSort.QuickSort(num4096);
        QuickSort.QuickSort(num8192);
        QuickSort.QuickSort(num16384);

    }

    public static int[] sortNumArray(BufferedReader readText, int size) throws NumberFormatException, IOExcepti
        int[] numArray=new int[size];
        for (int a=0; a < size; a++) {
            numArray[a]=Integer.parseInt(readText.readLine());
        }
        return numArray;
    }


}
```

```java
import java.util.Arrays;

public class HeapSort {

    public static int heapsize;
    static int count=0;

    public static void HeapSort(int[] A) {
        A=Arrays.copyOf(A, A.length);
        heapsize=A.length - 1;
        Build_Max_Heap(A);

        for (int i=A.length - 1; i > 0; i--) {
            int temp=A[0];
            A[0]=A[i];
            A[i]=temp;
            heapsize--;
            Max_Heapify(A, 0);

        }
        System.out.println("HeapSort: " + "Num" + A.length + "\n" + "Cost to sort " + "Num" + A.length + " = "
        if (A.length <= 64) {
            System.out.println(Arrays.toString(A) + "\n");
        } else {
            System.out.println(Arrays.toString((Arrays.copyOfRange(A, 50, 100))) + "\n");

        }
        count=0;
    }


    private static int left(int i) {
        return 2 * i + 1;
    }

    private static int right(int i) {
        return 2 * i + 2;
    }

    public static void Max_Heapify(int[] A, int i) {
        int l=left(i);
        int r=right(i);
        int largest;
        if (l <= heapsize && A[l] > A[i]) {
            largest=l;
        } else {
            largest=i;
        }

        if (r <= heapsize && A[r] > A[largest]) {
            largest=r;
        }

        if (largest != i) {
            int temp=A[i];
            A[i]=A[largest];
            A[largest]=temp;
            count++;
            Max_Heapify(A, largest);
        }
    }

    public static void Build_Max_Heap(int[] A) {

        for (int i=(A.length / 2) - 1; i >= 0; i--) {
            Max_Heapify(A, i);
        }

    }

}
```

```java
import java.util.Arrays;


public class MergeSort {

    public static int count=0;

    public static void MergeSort(int[] A) {
        A=Arrays.copyOf(A, A.length);
        mergeSort(A, 0, A.length - 1);
        System.out.println("MergeSort: " + "Num" + A.length + "\n" + "Cost to sort " + "Num" + A.length + " = '
        if (A.length <= 64) {
            System.out.println(Arrays.toString(A) + "\n");
        } else {
            System.out.println(Arrays.toString((Arrays.copyOfRange(A, 50, 100))) + "\n");

        }
        count=0;
    }

    public static void mergeSort(int[] A, int p, int r) {
        if (p < r) {
            int q=(p + r) / 2;
            mergeSort(A, p, q);
            mergeSort(A, q + 1, r);
            merge(A, p, q, r);
        }
    }

    public static void merge(int[] A, int p, int q, int r) {
        int n1=(q - p) + 2;
        int n2=(r - q) + 1;
        int[] L=new int[n1];
        int[] R=new int[n2];
        for (int i=0; i < n1 - 1; i++) {
            L[i]=A[p + i];
        }
        for (int i=0; i < n2 - 1; i++) {
            R[i]=A[q + i + 1];
        }
        L[L.length - 1]=Integer.MAX_VALUE;
        R[R.length - 1]=Integer.MAX_VALUE;
        int i=0;
        int j=0;
        for (int k=p; k <= r; k++) {
            count++;
            if (L[i] <= R[j]) {
                A[k]=L[i];
                i++;
            } else {
                A[k]=R[j];
                j++;
            }
        }
    }
}
```

```java
import java.util.Arrays;

public class QuickSort {
    public static int count=0;

    public static void QuickSort(int[] A) {
        A=Arrays.copyOf(A, A.length);
        QuickSort(A, 0, A.length - 1);
        System.out.println("Quicksort on " + "Num" + A.length + "\n" + "Cost to sort " + "Num" + A.length + " :
        if (A.length <= 64) {
            System.out.println(Arrays.toString(A) + "\n");
        } else {
            System.out.println(Arrays.toString((Arrays.copyOfRange(A, 50, 100))) + "\n");

        }
        count=0;
    }


    public static void QuickSort(int[] A, int p, int r) {
        if (p < r) {
            int q=Partition(A, p, r);
            QuickSort(A, p, q - 1);
            QuickSort(A, q + 1, r);
        }


    }

    public static int Partition(int[] A, int p, int r) {
        int x=A[r];
        int i=(p - 1);

        for (int j=p; j <= (r - 1); j++) {
            count++;
            if (A[j] <= x) {
                i=i + 1;

                int temp=A[i];
                A[i]=A[j];
                A[j]=temp;
            }
        }
        int tempTwo=A[i + 1];
        A[i + 1]=A[r];
        A[r]=tempTwo;
        return (i + 1);
    }

}
```

```java
import java.util.Arrays;

public class InsertionSort {
    public static int count=0;

    public static void InsertionSort(int[] A) {

        int i, key;
        A=Arrays.copyOf(A, A.length);
        ;

        for (int j=1; j < A.length; j++) {
            key=A[j];
            i=j - 1;

            while (i >= 0 && A[i] >= key) {
                count=count + 1;
                A[i + 1]=A[i];
                i=i - 1;
            }
            A[i + 1]=key;
        }
        System.out.println("InsertionSort: " + "Num" + A.length + "\n" + "Cost to sort " + "Num" + A.length + '
        if (A.length <= 64) {
            System.out.println(Arrays.toString(A) + "\n");
        } else {
            System.out.println(Arrays.toString((Arrays.copyOfRange(A, 50, 100))) + "\n");

        }
        count=0;
    }
}
```

Test Case Output

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\
lib\idea_rt.jar=56789:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\bin" -Dfile.encoding=UTF-8 -classpath C
:\Users\humai\IdeaProjects\Project323\out\production\Project323 com.company.Main

INSERTION SORT

InsertionSort: Num8
Cost to sort Num8 = 14
[1, 2, 3, 4, 5, 6, 7, 8]

InsertionSort: Num16
Cost to sort Num16 = 84
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

InsertionSort: Num32
Cost to sort Num32 = 249
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32]

InsertionSort: Num64
Cost to sort Num64 = 966
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64]

InsertionSort: Num128
Cost to sort Num128 = 4196
[51, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num256
Cost to sort Num256 = 15373
[51, 52, 53, 54, 55, 56, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num512
Cost to sort Num512 = 67409
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num1024
Cost to sort Num1024 = 248967
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num2048
Cost to sort Num2048 = 1009705
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 71, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num4096
Cost to sort Num4096 = 4024740
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 76,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num8192
Cost to sort Num8192 = 16014589
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 81, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

InsertionSort: Num16384
Cost to sort Num16384 = 64403842
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
MERGE SORT

MergeSort: Num8
Cost to sort Num8 = 24
[1, 2, 3, 4, 5, 6, 7, 8]

MergeSort: Num16
Cost to sort Num16 = 64
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

MergeSort: Num32
Cost to sort Num32 = 160
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32]

MergeSort: Num64
Cost to sort Num64 = 384
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64]

MergeSort: Num128
Cost to sort Num128 = 896
[51, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num256
Cost to sort Num256 = 2048
[51, 52, 53, 54, 55, 56, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num512
Cost to sort Num512 = 4608
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num1024
Cost to sort Num1024 = 10240
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num2048
Cost to sort Num2048 = 22528
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 71, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num4096
Cost to sort Num4096 = 49152
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 76,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num8192
Cost to sort Num8192 = 106496
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 81, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

MergeSort: Num16384
Cost to sort Num16384 = 229376
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
HEAP SORT

HeapSort: Num8
Cost to sort Num8 = 11
[1, 2, 3, 4, 5, 6, 7, 8]

HeapSort: Num16
Cost to sort Num16 = 33
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

HeapSort: Num32
Cost to sort Num32 = 108
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32]

HeapSort: Num64
Cost to sort Num64 = 277
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64]

HeapSort: Num128
Cost to sort Num128 = 650
[51, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num256
Cost to sort Num256 = 1580
[51, 52, 53, 54, 55, 56, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num512
Cost to sort Num512 = 3646
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num1024
Cost to sort Num1024 = 8376
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num2048
Cost to sort Num2048 = 18716
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 71, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num4096
Cost to sort Num4096 = 41533
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 76,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num8192
Cost to sort Num8192 = 91358
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 81, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

HeapSort: Num16384
Cost to sort Num16384 = 199027
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
QUICK SORT

Quicksort on Num8
Cost to sort Num8 = 16
[1, 2, 3, 4, 5, 6, 7, 8]

Quicksort on Num16
Cost to sort Num16 = 55
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

Quicksort on Num32
Cost to sort Num32 = 123
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32]

Quicksort on Num64
Cost to sort Num64 = 394
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64]

Quicksort on Num128
Cost to sort Num128 = 901
[51, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num256
Cost to sort Num256 = 2123
[51, 52, 53, 54, 55, 56, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num512
Cost to sort Num512 = 4873
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num1024
Cost to sort Num1024 = 11049
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num2048
Cost to sort Num2048 = 24573
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 71, 73, 74, 75, 76, 77
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num4096
Cost to sort Num4096 = 54225
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 76,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num8192
Cost to sort Num8192 = 128095
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 81, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Quicksort on Num16384
Cost to sort Num16384 = 285822
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]


Process finished with exit code 0
```

Test Case Summary

	Insertion sort	*Merge sort*	*Heapsort*	*Quicksort*
Num8.txt	14	24	11	16
Num16.txt	84	64	33	55
Num32.txt	249	160	108	123
Num64.txt	966	384	277	394
Num128.txt	4196	896	650	901
Num256.txt	15373	2048	1580	2123
Num512.txt	67409	4608	3646	4873
Num1024.txt	248967	10240	8376	11049
Num2048.txt	1009705	22528	18716	24573
Num4096.txt	4024740	49152	41533	54225
Num8192.txt	16014589	106496	91358	128095
Num16284.txt	64403842	229376	199027	285822

Minimum and maximum run time for each size

	8	16	32	64	128	256	512	1024	2048	4196	8192	16384
MIN	Heapsort	HeapSort										
MAX	Merge Sort	Insertion Sort										

Calculation

	(n2)	(n lg n)
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896
256	65536	2048
512	262144	4608
1024	1048576	10240
2048	4194304	22528
4196	17606416	50498.02
8192	67108864	106496
16384	268435456	229376

Comparison of theoretical run time and actual run

QuickSort	(n2)
14	64
84	256
249	1024
966	4096
4196	16384
15373	65536
67409	262144
248967	1048576
1009705	4194304
4024740	17606416
16014589	67108864
64403842	268435456

Merge sort	(n lg n)
24	24
64	64
160	160
384	384
896	896
2048	2048
4608	4608
10240	10240
22528	22528
49152	50498.02
106496	106496
229376	229376

Heapsort	(n lg n)
11	24
33	64
108	160
277	384
650	896
1580	2048
3646	4608
8376	10240
18716	22528
41533	50498.02
91358	106496
199027	229376

Quicksort	(n lg n)
16	24
55	64
123	160
394	384
901	896
2123	2048
4873	4608
11049	10240
24573	22528
54225	50498.02
128095	106496
285822	229376

4) Analysis:

My results conclude the theoretical run time of each algorithm. It is evident that insertion sort performed at a run time of $O(n^2)$, Merge sort performed at a run time of O(n lg n), Heap sort performed at a run time of O(n lg n), and Quick sort performed at a run time of O(n lg n) with a slight deviation in constant factors. Upon examination of the "Minimum and maximum run time for each size" chart and "test case output" chart, the statement of insertion sort running faster than merge sort at relatively smaller problem sizes is confirmed by the output values. At n = 8, insertion sort had a run time of 14 whereas merge sort had a run time of 24. However, for the rest of the problem sizes, insertion sort ran for a longer duration as compared to merge sort. Heap sort presented to be the fastest in comparison to all the other algorithms, confirming its run time of O(n lg n). Merge sort presented to be the second fastest in comparison with heap sort and quick sort came in third. It is apparent that the sorts did take $O(n^2)$ and O(n lg n) steps to run.

Upon analyzing the comparison of theoretical run time and actual run time chart, I came to the conclusion that insertion sort took $O(n^2)$ run time, being the slowest when compared to mergesort, heap sort and quicksort. Merge sort ran at the theoretical run time of O(n log n), heap sort being the fastest ran O(n log n) and Quick sort being the third fast ran O(n log n). Insertion sort presented to take the most steps as compared to the other algorithms.

Out of the O(n log n) sorts, quick sort takes the most steps. Quick sort presented to be slower than merge sort, presumably because of an unbalanced partitioning. At very small inputs, I would prefer to use insertion sort but as the data sizes grow, the matter of choice is conditional based on the scenario of the data. When the size of the data is large and presumably in reverse order, meaning the data would need to swapped in worst case, I would choose heapsort. However if the data is very large and it is known that there is not much to sort, quick sort may be the better option.