Jeffrey Pulinat

CS-323, SECTION: 11 (7912)

Lab#1

3/26/17

Due date: 3/27/17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion Sort | Merge Sort | Heap Sort | Quick Sort |
| Num8.txt | 14 | 24 | 11 | 16 |
| Num16.txt | 84 | 64 | 33 | 55 |
| Num32.txt | 249 | 160 | 108 | 123 |
| Num128.txt | 966 | 384 | 277 | 394 |
| Num256.txt | 4195 | 896 | 650 | 901 |
| Num256.txt | 15372 | 2048 | 1580 | 2123 |
| Num512.txt | 67408 | 4608 | 3646 | 4873 |
| Num1024.txt | 248966 | 10240 | 8376 | 11049 |
| Num2048.txt | 1009704 | 22528 | 18716 | 24573 |
| Num4096.txt | 4024739 | 49152 | 41533 | 54225 |
| Num8192.txt | 16014588 | 106496 | 91358 | 128095 |
| Num16284.txt | 64403841 | 229376 | 199027 | 285822 |

**Discuss what your results mean regarding the theoretical run-time of the different algorithms.**

The graph shows that HeapSort is the ideal sorting algorithm out of insertion, merge and quick sorts. Merge Sort and Heap Sort are very similar in runtime however Heap Sort separates itself with more input to sort. Quick Sort is highly dependable on data randomness and pivot.

**b) Do the sorts really take O(n2 ) and O(n lg n) steps to run? c) Explain how you got your answer to this question. d) Which of the sorts takes the most steps?**

The sorts do take 0(n2) and O(n lg2n). Insertions Sort is very similar too O(n2). We can see this by dividing the count times for all the text files with n2. In Quick Sort, Merge Sort and Heap Sort the runtime we get runtimes very similar to O(n log2n). We do this by diving n log2n with its repsective count run time. There is some discrepancy when comparing the counts to the O run times because the O run times represent worst case running times.

**e) Which of the two O(n lg n) sorts takes the most steps?**

Insertion Sort took the most steps. Algorithms from longest to smallest is insertion, quick, merge, heap.

**f) Why?**

The running time of quick sort usually takes in consideration how the unsorted data is placed and our pivot. I suspect our pivot was not the best placed pivot however it still performed better than the other sorts.

**g) Under what circumstances might you prefer to use one of the sorts versus others?**

Quick Sort is best for random data and pivot is picked randomly. Merge Sort is used when most of the data is already sorted. Insertion Sort should almost never be used unless your data size is very small. Heap sort would be the ideal sort in all cases.

**h) In general, which sort seems preferable? i) Why?**

According to my test and research Merge Sort and Heap Sort is the best. In a real case scenario data will likely be randomized, in which case quick sort could be preferred, however the worst case runtime for quick sort is O(n2) and the worst case for merge and heap would be O(nlogn). I believe it would be safer and quicker on average to use Heap Sort because it does not need data to be randomized or to be mostly sorted.

**Source Code**

//Jeffrey Pulinat

//CS 323 -11 Changhe Yuan

//Homework #1

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Arrays;

public class hw1JeffreyPulinat {

//global variable count

public static int count;

public static String Output = "";

public static boolean flagprint= false;

public static void main(String[] args) {

// Array of textfile names

String [] FileNames = { "Num8.txt", "Num16.txt", "Num32.txt", "Num64.txt", "Num128.txt","Num256.txt",

"Num512.txt","Num1024.txt", "Num2048.txt", "Num4096.txt","Num8192.txt", "Num16384.txt" };

String DataFile = "";

try{

//loops on all textfiles

for(int FileNamesIndex = 0; FileNamesIndex < FileNames.length; FileNamesIndex++){

DataFile = FileNames[FileNamesIndex];

FileReader fileReader = new FileReader(new File(DataFile));

BufferedReader bufferedReader = new BufferedReader(fileReader);

int ArraySize = Integer.parseInt(DataFile.replaceAll("[^0-9]", "")); //gets int from String

int []SetArray1 = new int[ArraySize];

int []SetArray2 = new int[ArraySize];

int []SetArray3 = new int[ArraySize];

int []SetArray4 = new int[ArraySize];

for(int i=0; i<SetArray1.length; ++i){

String aLine = bufferedReader.readLine();

SetArray1[i] = Integer.parseInt(aLine);

SetArray2[i] = Integer.parseInt(aLine);

SetArray3[i] = Integer.parseInt(aLine);

SetArray4[i] = Integer.parseInt(aLine);

}

fileReader.close();

if(DataFile.equals("Num128.txt"))flagprint=true;

OutputMold(SetArray1, SetArray2, SetArray3, SetArray4, DataFile);

}

}catch (IOException e) {

e.printStackTrace();

}

}

//OutputMold is the way String should be printed

public static void OutputMold(int array1[], int array2[], int array3[], int array4[],String DataFile){

//Start Sort

Output = Output + "\n-------" + DataFile + "------------\n";

//Set count o zero before every sort

//INSERT

count = 0;

InsertionSort(array1);

Output = Output + "InsertionSort: " + count + "\n";

PrintArray(array1);

//MERGE

count = 0;

MergeSort(array2, 0, array2.length-1);

Output = Output + "\nMergeSort: " + count + "\n";

PrintArray(array2);

//HEAP

count = 0;

HeapSort(array3);

Output = Output + "\nHeapSort: " + count + "\n";

PrintArray(array3);

//QUICK

count = 0;

QuickSort(array4);

//System.out.println(count);

Output = Output + "\nQuickSort: " + count + "\n";

PrintArray(array4);

System.out.println(Output);

try{

PrintWriter out = new PrintWriter( "Output.txt" );

out.println(Output );

out.close();

} catch (IOException e) {

e.printStackTrace();

}

}

//Prints out the array

public static void PrintArray(int array[]){

if(flagprint==true){

int copy[] = new int [50]; //wanted to use Array to String because its cleaner. Need to create copy array to print specific indexes.

int copyindex=0;

for(int i = 51 ;i <=100; i++){

copy[copyindex] = array[i];

copyindex++;

}

Output = Output + (Arrays.toString(copy));

}else

Output = Output + (Arrays.toString(array));

}

public static void InsertionSort (int myArray[]){

for(int index= 1; index<myArray.length; index++){ //start from index 1 compare to index 0

int key = myArray[index];

int i = index - 1;

while(i >= 0 && myArray[i] > key){

count = count + 1;

myArray[i + 1] = myArray[i];

i = i - 1;

}

myArray[i + 1] = key;

}

}

public static void MergeSort(int [] anArray, int index, int Alength){

if( index<Alength ){

int OtherIndex = ( index+Alength )/2; //int will give floor value automatically

MergeSort(anArray, index, OtherIndex); //left subarray

MergeSort(anArray, OtherIndex+1, Alength); //right subarray

Merge(anArray, index, OtherIndex, Alength); //make more subarrays

}

}

public static void Merge(int [] anArray, int index, int OtherIndex, int Alength){

int n1 = ( OtherIndex-index )+1;

int n2 = ( Alength-OtherIndex ); //split into left and right subarray

int [] leftArray = new int[ n1+1 ];

int [] rightArray= new int[ n2+1 ];

for(int i=0; i<n1; ++i) // fill left subarray with half of original array

leftArray[i] = anArray[ index+i ]; //psuedo says index + i -1. but might give me indexOutofBound

for(int j=0; j<n2; ++j) // fill right subarray with half of original array

rightArray[j] = anArray[ OtherIndex+j+1 ];

leftArray[ n1 ] = 999999999;

rightArray[n2 ] = 999999999;

int indexL = 0; // index left

int indexR = 0; // index right

for(int x = index; x<=Alength; ++x){

count = count + 1;

if( leftArray[indexL]<=rightArray[indexR] ){

anArray[x] = leftArray[indexL];

indexL = indexL+1;

}

else{

anArray[x] = rightArray[indexR];

indexR = indexR+1;

}

}

}

public static void HeapSort(int [] myArray){

Build\_Max\_Heap(myArray);

for(int RootIndex = myArray.length-1; RootIndex>=0; --RootIndex){

exchange(myArray, 0, RootIndex); // swap array[0](first) and array[RootIndex](last)

Max\_Heapify(myArray, 0, RootIndex); //go down the heap (like a tree)

}

}

//Creates Heap

public static void Build\_Max\_Heap(int [] myArray){

int HeapSize = myArray.length;

for(int RootIndex = (myArray.length/2)-1; RootIndex>=0; --RootIndex){

Max\_Heapify(myArray, RootIndex, HeapSize);

}

}

//Going down the HeapArray

public static void Max\_Heapify(int [] myArray, int RootIndex, int HeapSize){

int largest=RootIndex;

int left = 2\*RootIndex+1;

int right = 2\*RootIndex+2;

//find larger left child or Root

if( left<HeapSize && myArray[left] > myArray[largest] )

largest = left;

// find larger right or Root

if( right<HeapSize && myArray[right] > myArray[largest] )

largest = right;

// Root isnt the largest element

if(largest != RootIndex){

exchange(myArray, RootIndex, largest); //swap rootindex with largest element

count = count + 1;

Max\_Heapify(myArray, largest, HeapSize); //largest is the rootIndex now

}

}

// swap A[i] and A[j]

public static void exchange(int A[], int i, int j){

int temporary = A[i];

A[i] = A[j];

A[j] = temporary;

}

//Swap on Pivot. highly dependable on pivot

public static void QuickSort(int [] myArray){

QUICKSORT(myArray, 0, myArray.length-1 );

}

//index1 is start index index 2 is end index originally

public static void QUICKSORT(int [] myArray, int index1, int index2){

if(index1 < index2){

int q = PARTITION(myArray, index1, index2); //Pivot

QUICKSORT(myArray, index1, q-1); // before the pivot index

QUICKSORT(myArray, q+1, index2); // after the pivot index

}

}

//Makes Pivot

public static int PARTITION(int [] myArray, int index1, int index2){

int x = myArray[index2]; // end element of subarray

int i = index1 - 1; //smallest element

for( int currentIndex = index1; currentIndex<index2; ++currentIndex ){

count = count + 1;

if( myArray[currentIndex] <= x) { // Checks if current element <= pivot

i = i + 1; // increment small

exchange(myArray, i, currentIndex); //swap small with current

}

}

exchange(myArray, i+1, index2);

return i+1;

}

}

**OUTPUT:**

-------Num8.txt------------

InsertionSort: 14

[1, 2, 3, 4, 5, 6, 7, 8]

MergeSort: 24

[1, 2, 3, 4, 5, 6, 7, 8]

HeapSort: 11

[1, 2, 3, 4, 5, 6, 7, 8]

QuickSort: 16

[1, 2, 3, 4, 5, 6, 7, 8]

-------Num16.txt------------

InsertionSort: 84

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

MergeSort: 64

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

HeapSort: 33

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

QuickSort: 55

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

-------Num32.txt------------

InsertionSort: 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]

MergeSort: 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]

HeapSort: 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]

QuickSort: 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]

-------Num64.txt------------

InsertionSort: 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]

MergeSort: 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]

HeapSort: 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]

QuickSort: 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]

-------Num128.txt------------

InsertionSort: 4195

[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, 101]

MergeSort: 896

[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, 101]

HeapSort: 650

[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, 101]

QuickSort: 901

[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, 101]

-------Num256.txt------------

InsertionSort: 15372

[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, 101]

MergeSort: 2048

[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, 101]

HeapSort: 1580

[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, 101]

QuickSort: 2123

[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, 101]

-------Num512.txt------------

InsertionSort: 67408

[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, 101]

MergeSort: 4608

[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, 101]

HeapSort: 3646

[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, 101]

QuickSort: 4873

[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, 101]

-------Num1024.txt------------

InsertionSort: 248966

[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, 101]

MergeSort: 10240

[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, 101]

HeapSort: 8376

[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, 101]

QuickSort: 11049

[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, 101]

-------Num2048.txt------------

InsertionSort: 1009704

[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, 101]

MergeSort: 22528

[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, 101]

HeapSort: 18716

[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, 101]

QuickSort: 24573

[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, 101]

-------Num4096.txt------------

InsertionSort: 4024739

[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, 101]

MergeSort: 49152

[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, 101]

HeapSort: 41533

[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, 101]

QuickSort: 54225

[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, 101]

-------Num8192.txt------------

InsertionSort: 16014588

[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, 101]

MergeSort: 106496

[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, 101]

HeapSort: 91358

[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, 101]

QuickSort: 128095

[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, 101]

-------Num16384.txt------------

InsertionSort: 64403841

[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, 101]

MergeSort: 229376

[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, 101]

HeapSort: 199027

[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, 101]

QuickSort: 285822

[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, 101]