**Lab 3 Report**

**CS303L Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

* Implement merge sort algorithm
* Compare performance of insertion sort with merge sort
* Improve the performance of merge sort by using insertion for sorting short arrays

**In-class Assignment:**

1. Implement a method that will sort a given array using the merge sort algorithm.
2. Write a driver program to test the merge sort algorithm.

|  |
| --- |
| Source code: package Lab3;  import java.io.File; import java.io.FileNotFoundException; import java.util.Arrays; import java.util.Scanner;  public class lab3 {   public static void main(String[] args) throws FileNotFoundException {   File file = new File("input\_100.txt");  Scanner s = new Scanner(file);   int[] array = new int[100];   int i = 0;  while(s.hasNextInt()) {  array[i] = s.nextInt();  i++;  }   mergeSort(array);  System.out.println(Arrays.toString(array)); }   public static void mergeSort(int[] a){  int p = 0;  int r = a.length - 1;  int[] temp = Arrays.copyOf(a, a.length);  mergeSort(a, temp, p, r);  }   public static void mergeSort(int[] a, int[] temp, int p, int r){  if(p < r) {  int q = (int)(Math.floor((p + r)/2));  mergeSort(a, temp, p , q);  mergeSort(a, temp, q + 1, r);  merge(a, temp, p, q , r);  }  }   private static void merge(int[] a, int[] temp, int p, int q, int r) {  int i = p;  int j = q + 1;  for (int k = p;k <= r; k++) {  temp[k] = a[k];  }     for (int k = p;k <= r; k++) {  if(i > q) {  a[k] = temp[j];  j = j + 1;  }  else if(j > r) {  a[k] = temp[i];  i = i + 1;  }  else if(temp[j] < temp[i]) {  a[k] = temp[j];  j = j + 1;  }  else {  a[k] = temp[i];  i = i + 1;  }  }  } } |

Output:

[0, 0, 0, 1, 1, 3, 4, 5, 7, 8, 10, 12, 12, 13, 14, 15, 15, 16, 18, 22, 22, 22, 23, 24, 24, 25, 25, 27, 29, 29, 30, 34, 36, 36, 37, 38, 38, 39, 40, 40, 41, 43, 45, 47, 47, 49, 50, 50, 52, 52, 53, 54, 54, 54, 54, 57, 60, 61, 63, 64, 64, 67, 67, 68, 69, 69, 70, 72, 73, 73, 74, 74, 75, 75, 75, 76, 77, 77, 78, 80, 81, 81, 81, 83, 83, 84, 85, 85, 86, 87, 88, 89, 92, 93, 94, 94, 94, 95, 96, 98]

**Homework Assignment:**

1. Compare the execution time of merge sort with insertion sort.
2. Modify the merge sort algorithm to improve the performance of merge sort by using insertion for sorting short arrays
3. Compare original merge sort implementation with the improved merge sort implementation

Source code:

|  |
| --- |
| package Lab3;  import java.io.File; import java.io.FileNotFoundException; import java.io.PrintWriter; import java.util.Arrays; import java.util.Scanner;  public class lab3 {   public static void main(String[] args) throws FileNotFoundException {   File[] files = new File[7];  files[0]= new File("input\_100.txt");  files[1]= new File("input\_1000.txt");  files[2]= new File("input\_5000.txt");  files[3]= new File("input\_10000.txt");  files[4]= new File("input\_50000.txt");  files[5]= new File("input\_100000.txt");  files[6]= new File("input\_500000.txt");   int[] size = {100, 1000, 5000, 10000, 50000, 100000, 500000};  for(int i = 0; i <= 6; i++){  int length = size[i];  int[] array = new int[length];  Scanner s = new Scanner(files[i]);  int j = 0;  while(s.hasNextInt()) {  array[j] = s.nextInt();  j++;  }   int[] in = Arrays.copyOf(array, array.length);  long InserstionSortTime = System.nanoTime();  insertionSort(in);  InserstionSortTime = System.nanoTime() - InserstionSortTime;   int[] me = Arrays.copyOf(array, array.length);  long MergeSortTime = System.nanoTime();  mergeSort(me);  MergeSortTime = System.nanoTime() - MergeSortTime;   int[] inme = Arrays.copyOf(array, array.length);  long InsertionMergeSortTime = System.nanoTime();  mergeInsertionSort(inme, size[i]/10);  InsertionMergeSortTime = System.nanoTime() - InsertionMergeSortTime;   System.out.println("Array Size: " + size[i] + "\n" + "Insertion Sort Time: " + InserstionSortTime  + "\n" + "Merge Sort Time: " + MergeSortTime + "\n" +  "Merge Sort with Insertion Time: " + InsertionMergeSortTime + "\n");  }   }   public static void insertionSort(int[] a){  int p = a.length;  int r = 0;  insertionSort(a, p, r);  }  public static int[] insertionSort(int[] a, int p, int r) {  for (int j = r; j < p; j++) {  int key = a[j];  int i = j - 1;  while(i >= 0 && a[i] > key) {  a[i+1]= a[i];  i = i - 1;  }  a[i+1] = key;  }  return a;  }   public static void mergeInsertionSort(int[] a, int size){  int p = 0;  int r = a.length - 1;  int[] temp = Arrays.copyOf(a, a.length);  mergeInsertionSort(a, temp, p, r, size);  }    public static void mergeInsertionSort(int[] a, int[] temp, int p, int r, int size){  int q;  if (r - p <= size) {  insertionSort(a, p, r);  }  else {  q = (int)(Math.floor((p + r)/2));  mergeInsertionSort(a, temp, p , q, size);  mergeInsertionSort(a, temp, q + 1, r, size);  merge(a, temp, p, q , r);   }   }   public static void mergeSort(int[] a){  int p = 0;  int r = a.length - 1;  int[] temp = Arrays.copyOf(a, a.length);  mergeSort(a, temp, p, r);  }   public static void mergeSort(int[] a, int[] temp, int p, int r){  if(p < r) {  int q = (int)(Math.floor((p + r)/2));  mergeSort(a, temp, p , q);  mergeSort(a, temp, q + 1, r);  merge(a, temp, p, q , r);  }  }   private static void merge(int[] a, int[] temp, int p, int q, int r) {  int i = p;  int j = q + 1;   for (int k = p;k <= r; k++) {  temp[k] = a[k];  }   for (int k = p;k <= r; k++) {  if(i > q) {  a[k] = temp[j];  j = j + 1;  }  else if(j > r) {  a[k] = temp[i];  i = i + 1;  }  else if(temp[j] < temp[i]) {  a[k] = temp[j];  j = j + 1;  }  else {  a[k] = temp[i];  i = i + 1;  }  }  } } |

The output was:

Array Size: 100

Insertion Sort Time: 83397

Merge Sort Time: 447961

Merge Sort with Insertion Time: 44469

Array Size: 1000

Insertion Sort Time: 3396746

Merge Sort Time: 1090439

Merge Sort with Insertion Time: 73830

Array Size: 5000

Insertion Sort Time: 18538928

Merge Sort Time: 725608

Merge Sort with Insertion Time: 85225

Array Size: 10000

Insertion Sort Time: 13493616

Merge Sort Time: 1425287

Merge Sort with Insertion Time: 119632

Array Size: 50000

Insertion Sort Time: 326473961

Merge Sort Time: 6748986

Merge Sort with Insertion Time: 765497

Array Size: 100000

Insertion Sort Time: 1050239333

Merge Sort Time: 9625979

Merge Sort with Insertion Time: 661180

Array Size: 500000

Insertion Sort Time: 23626165384

Merge Sort Time: 59259786

Merge Sort with Insertion Time: 6233770

Analysis:

Insertion is faster than merge sort for shorter arrays, however merge sort is faster for large arrays. Using a conditional to choose the best sorting algorithm for the array size will get the best run time for arrays of all sizes.