CSCI 421 Design and Analysis of Algorithms Spring 2019

Lecture 2 Activity 2

1. Apply the tree practical improvements on the merge sort algorithm. Compare the algorithms with various cutoff to insertion sort in terms of their running time. Say cutoff is set to 7, 12, 20, 25. You need to run the algorithms against large-size random arrays, say N=1000, 2000, 4000, 8000.

import java.util.\*;

@SuppressWarnings("unchecked")

public class Lec2Act2\_1 {

private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {

for (int k = lo; k <= hi; k++)

aux[k] = a[k];

int i = lo, j = mid + 1;

for (int k = lo; k <= hi; k++) {

if (i > mid)

aux[k] = a[j++];

else if (j > hi)

aux[k] = a[i++];

else if (less(aux[j], aux[i]))

aux[k] = a[j++];

else

aux[k] = a[i++];

}

}

private static void mergeSort(Comparable[] a, Comparable[] aux, int lo, int hi) {

if (hi <= lo + 7 - 1) {

insSort(a);

return;

}

int mid = lo + (hi - lo) / 2;

mergeSort(aux, a, lo, mid);

mergeSort(aux, a, mid + 1, hi);

if (!less(a[mid + 1], a[mid]))

return;

merge(a, aux, lo, mid, hi);

}

public static void insSort(Comparable[] a) {

int N = a.length;

for (int i = 0; i < N; i++)

for (int j = i; j > 0; j--)

if (less(a[j], a[j - 1]))

exch(a, j, j - 1);

else

break;

}

public static void mergeSort(Comparable[] a) {

Comparable[] aux = new Comparable[a.length];

mergeSort(a, aux, 0, a.length - 1);

}

private static void exch(Comparable[] a, int i, int j) {

Comparable swap = a[i];

a[i] = a[j];

a[j] = swap;

}

private static boolean less(Comparable v, Comparable w) {

return v.compareTo(w) < 0;

}

public static void main(String[] args) {

Random rand = new Random();

Comparable arr[] = new Comparable[1000];

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

arr[i] = rand.nextInt();

long startTime, endTime, elapsed;

System.out.println("Merge sort with array size " + 1000);

startTime = System.currentTimeMillis();

mergeSort(arr);

endTime = System.currentTimeMillis();

elapsed = endTime - startTime;

System.out.println("Merge Sort Time :" + elapsed);

}

}

\*\*Please note I kept getting null pointer exceptions….I tried for many hours to get this working with no success…

1. For the quicksort algorithm, its running time is different in best-case and worst- case scenarios depending on the pivot item in the partitioning stage.
2. Show the derivation to obtain the theoretical running time in best-case and worst-case scenarios.
   1. Worst-Case:
   2. Best-Case:
3. Construct examples on best-case and worst-case, respectively. Compare the running time in both situations. You need to run the algorithm against large-size arrays, say N=1000, 2000, 4000, 8000.