CSCI 421 Design and Analysis of Algorithms Spring 2019

Lecture 2 Activity 1

**Insertion sort with practical improvements**

1. As discussed in class, we can have some practical improvement on insertion sort
2. Develop a modified version of insertion sort based on the first approach in slide 34 that moves larger elements to the right one position with one array access per entry, rather than using exch().

class Lec2Act1A{

void sort(int arrList[]){

int n = arrList.length;

for (int i=1; i<n; ++i){

int key = arrList[i];

int j = i-1;

while (j>=0 && arrList[j] > key){

arrList[j+1] = arrList[j];

j = j-1;

}

arrList[j+1] = key;

}

}

static void printArray(int arrList[]){

int n = arrList.length;

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

System.out.print(arrList[i] + " ");

System.out.println();

}

public static void main(String args[]){

int arrList[] = {36, 26, 10, 15, 19, 76, 69, 42, 105,27, 57};

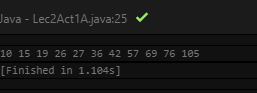
Lec2Act1A ob = new Lec2Act1A();

ob.sort(arrList);

printArray(arrList);

}

}



1. Develop a modified version of insertion sort based on the binary search approach in slide 34.

import java.util.Arrays;

class Lec2Act1B {

public static void main(String[] args) {

final int[] arrList = {36, 26, 10, 15, 19, 76, 69, 42, 105,27, 57};

new Lec2Act1B().sort(arrList);

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

System.out.print(arrList[i]+" ");

}

public void sort(int array[]) {

for (int i = 1; i < array.length; i++){

int x = array[i];

int j = Math.abs(Arrays.binarySearch(array, 0, i, x) + 1);

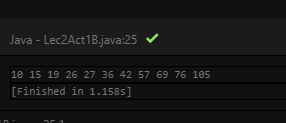
System.arraycopy(array, j, array, j+1, i-j);

array[j] = x;

}

}

}



1. Compare the modified versions of insertion sort with its original version shown in slide 26 in terms of running time. You need to run the algorithms against large-size random arrays, say N=1000, 2000, 4000, 8000.