Vaishak Menon

CS303 Lab 6 – Novel Sort

Problem Specification:

There were two problems to solve for this homework assignment.

The first was to sort an array of numbers by finding the smallest and largest elements and swapping them with the respective positions within the array. This is done repeatedly till the array is completely sorted.

The second was to sort a text file that had a city and time respectively in two columns. The time was already in sorted order and the goal was to sort the cities in alphabetical order, while keeping the time in sorted order.

Program Design:

The first program is designed to find the smallest and largest numbers and sort the array by swapping them. My code has three functions to carry out this purpose. The first is findSmallLarge(), which takes in an array, a starting point, and an ending point as parameters/inputs. Within this function, the code goes through the array starting at the inputted starting point and ending at the inputted ending point. After finding the smallest and largest, the indices are put into an array. The second function is just a swap function that will swap to digits within the array given the indices. The final function is the sort function which takes in input of an integer array. This function has two variables, i and j, which holds the lowest index and highest index respectively. It has a while loop that repeats till the difference between j and i is greater than 0. The main purpose of this loop is to repeatedly find the smallest and largest numbers using the findSmallLarge function and swap them with the indices they were found at. There is an if statement within the sort function that checks if the largest element is at the smallest index, which if true, the order of swapping changes due to a bug that I found while testing. After doing all the swaps and making it to the middle, I also added an insertion sort implementation from lab2, to make sure that the middle element is where it is supposed to be and if it isn't put it in the right place and to make sure that the rest of the array was sorted completely. This is more of just a check in place to make sure everything went well. The driver for this class reads in the input file, splits it up into an array and then calls the sort and produces the time it took to sort it.

The second program is designed to take in a log that has been provided and sort the city data alphabetically while also keeping the time data sorted if there are multiple logs of one city. My code has a sort function that essentially just uses insertion sort on both arraylists, cities and times. The cities get sorted in alphabetical order using insertion sort and comparing the strings and at the same time whenever an insert/swap is made that same thing is done to the time ArrayList since they both have the same exact order as the txt file before any sorting begins. I also have a convert function that takes the times array and turns it back into strings because originally, I was going to work with Map and HashMap and this required the times to become

integers. The driver class for this problem reads in each line, splits it by whitespace, and gets the city and puts it into an ArrayList and gets the time and puts it in a different ArrayList. Then calls the sort and displays the sorted cities and corresponding times.

Testing Plan:

The testing plan for the first problem is to use the sort with 100 and 1000 random numbers provided from previous labs. I print out the numbers showing that the numbers are sorted after sorting for the 100 random numbers test case, but anything after that will be too much as the numbers get exponentially larger.

The testing plan for the second problem is to sort the given sample input file. Due to the specific test cases, I decided to just show that this one test case does work, but the code has been implemented so that any other test cases should work.

Test Cases:

Problem 1:

```
100 Elements:
Time taken to novel sort + insertion sort array of 100 numbers: 185700
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
1000 Elements:
Time taken to novel sort + insertion sort array of 1000 numbers: 4262200
```

```
100 Elements:
Time taken to novel sort + insertion sort array of 100 numbers: 113600
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
1000 Elements:
Time taken to novel sort + insertion sort array of 1000 numbers: 4311100
```

Problem 2:

```
Unsorted Cities:
Chicago Phoenix Houston Chicago Houston Chicago Seattle Seattle Phoenix Chicago Chicago Chicago Seattle Seattle Chicago Chicago Phoenix
09:00:00 09:00:03 09:00:13 09:00:59 09:01:10 09:03:13 09:10:11 09:10:25 09:14:25 09:19:32 09:19:46 09:21:05 09:22:43 09:22:54 09:25:52 09:36:14 09:37:44
Sorted Cities:
Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago Houston Houston Phoenix Phoenix Phoenix Seattle Seattle Seattle Seattle
09:00:00 09:00:59 09:03:13 09:19:32 09:19:46 09:21:05 09:25:52 09:36:14 09:00:13 09:01:10 09:00:03 09:14:25 09:37:44 09:10:11 09:10:25 09:22:43 09:22:54
```

Analysis/Conclusion:

Given the data and test cases, for problem 1 the time it took to sort is expected due to my addition of the insertion sort because that will add more time but it also ensures a sorted array. Although the time might be longer especially after the number of elements gets much larger, the sort is completely sorted within a reasonable time considering that most of the time is only from swapping and not from the insertion sort.

For the second problem, I used insertion sort because it was easiest to implement. Looking back, a counting sort might work better, if it completely reads and indexes strings correctly, since it is a bit more geared to this kind of situation but with a different data type. Quick sort also might work better since it would apply the same logic I applied with insertion sort, but this time with an algorithm that is faster.

References:

I did not need any outside references for the code I used, other than myself. I used previous insertion sort code from lab 2 in both problems. Input files have been provided to us previously, this is just reuse for problem 1. For problem 2, the input file was provided with the instructions. Pictures of code below.

```
public void sort(ArrayList<String> A, ArrayList<Integer> B)
{
    int i = 0;
    String key = "";
    int k = 0;
    int length = A.size();
    for (int j = 1; j < length; j++)
    {
        key = A.get(j);
        k = B.get(j);
        i = j - 1;
        while(i >= 0 && A.get(i).compareTo(key) > 0)
        {
            A.set(i+1,A.get(i));
            B.set(i+1,B.get(i));
            i = i - 1;
        }
        A.set(i+1, key);
        B.set(i+1, k);
    }
}

public ArrayList<String> convert(ArrayList<Integer> A)
{
    ArrayList<String> B = new ArrayList<String>();

    for(int i : A)
    {
        String s = "0" + String.valueOf(i);
        StringBuffer sB = new StringBuffer(s);
        for(int j = 2; j < s.length(); j+=3)
        {
            sB.insert(j, ':');
        }
        String ns = sB.toString();
        B.add(ns);
    }
    return B;
}
```

```
LogSort ls = new LogSort();

File f = new File("NovelSortInput.txt");
ArrayList<String> cities = new ArrayList<String>();
ArrayList<Integer> times = new ArrayList<Integer>();

try
{
    Scanner sc = new Scanner(f);
    while (sc.hasNextLine())
    {
        String[] log = sc.nextLine().trim().split(" ");
        //Index for last element
        int last = log.length-1;

        if (log.length == 2)
        {
            String city = log[0];
            String time = "";
            if (!(log[0].isEmpty()))
            {
                time = log[last];
                String[] timeSplit = time.split(":");
                String s = "";
                for (int i = 0; i < timeSplit.length; i++)
                {
                    s += timeSplit[i];
                }
                int newTime = Integer.parseInt(s);
                cities.add(city);
                times.add(newTime);
            }
        }
        else
        {
            String city = "";
            for(int i = 0; i < last;i++)
            {
                city = city + log[i];
            }

            String time = "";
            if (!(log[0].isEmpty()))
            {
                time = log[last];
                String[] timeSplit = time.split(":");
                String s = "";
                for (int i = 0; i < timeSplit.length; i++)
                {
                    s += timeSplit[i];
                }
                int newTime = Integer.parseInt(s);
                cities.add(city);
                times.add(newTime);
            }
        }

    }
    sc.close();
}
catch (FileNotFoundException e)
{
    e.printStackTrace();
}
```

```
ArrayList<String> newTimes = ls.convert(times);

System.out.println("Unsorted Cities: ");
for (String s : cities) {System.out.print(s + " ");}
System.out.println();
for (String x : newTimes) {System.out.print(x + " ");}

ls.sort(cities, times);
newTimes = ls.convert(times);

System.out.println();
System.out.println("Sorted Cities: ");
for (String s : cities) {System.out.print(s + " ");}
System.out.println();
for (String x : newTimes) {System.out.print(x + " ");}
```

```java
private static int[] nums = new int[2];

private int[] findSmallLarge(int[] A, int a, int b)
{
    int smallest = A[a], largest = A[b], sIndex = a, lIndex = b;

    for (int i = a; i < b; i++)
    {
        if (A[i] <= smallest)
        {
            smallest = A[i];
            sIndex = i;
        }
        if (A[i] >= largest)
        {
            largest = A[i];
            lIndex = i;
        }
    }

    nums[0] = sIndex;
    nums[1] = lIndex;

    return nums;
}

private void swap(int[] A, int x, int y) {
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
}
```

```java
public void sort(int[] A)
{
    int i = 0, j = A.length;

    while (j - i > 0)
    {
        findSmallLarge(A, i, j-1);

        int sIndex = nums[0];
        int lIndex = nums[1];

        if(i == lIndex)
        {
            swap(A, j-1, lIndex);
            swap(A, i, sIndex);
        }
        else
        {
            swap(A, i, sIndex);
            swap(A, j-1, lIndex);
        }

        i++;
        j--;
    }

    if (A.length % 2 == 0 & i == A.length / 2)
    {
        int a = 0;
        int key = A[A.length / 2];
        int length = A.length;

        for (int k = 1; k < length; k++)
        {
            key = A[k];
            a = k - 1;
            while (a >= 0 && A[a] > key)
            {
                A[a + 1] = A[a];
                a--;
            }
            A[a + 1] = key;
        }
    }
}
```

```java
private static int[] array = new int[]{};

public static void updateArray(File f)
{
    try
    {
        Scanner sc = new Scanner(f);
        String[] sArray = sc.nextLine().trim().split(" ");
        int sLength = sArray.length;

        array = new int[sLength];

        for(int i = 0; i < sLength; i++)
        {
            array[i] = Integer.parseInt(sArray[i]);
        }
        sc.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}
Run | Debug
public static void main(String[] args)
{
    NovelSwap ns = new NovelSwap();
    File f = new File("input_100.txt");
    updateArray(f);

    System.out.println(array.length + " Elements: ");
    long iTime = System.nanoTime();
    ns.sort(array);
    long eiTime = System.nanoTime() - iTime;
    System.out.println("Time taken to novel sort + insertion sort array of " + array.length + " numbers: " + eiTime);

    for(int i : array)
    {
        System.out.print(i + " ");
    }

    File f2 = new File("input_1000.txt");
    updateArray(f2);

    System.out.println("\n" + array.length + " Elements: ");
    long iTime2 = System.nanoTime();
    ns.sort(array);
    long eiTime2 = System.nanoTime() - iTime2;
    System.out.println("Time taken to novel sort + insertion sort array of " + array.length + " numbers: " + eiTime2);

}
```