Vaishak Menon

CS303 Lab 7 – Selection Sort and Bubble Sort

Problem Specification:

The problem addressed for this lab is to create and test selection sort and bubble sort. I created my own random number files and ran the sorts against those to create new test cases. I used the previously given 1000 number input file and then created my own 2000, 4000, 8000, and 16000. I also ran specifically selection sort and bubble sort against a bonus_random.txt. This file was just a random txt file that has 1024 random numbers. Afterwards I sorted and reverse sorted the file and ran those numbers in the selection sort and bubble sort. This file along with its sorted and reverse sorted counterparts provide possible worst-case scenarios for both selection and bubble sort.

Program Design:

The Lab7Sorts java file only contains three functions. One being the selection sort, another being bubble sort, and the last one is a swap function. The selection sort function and the bubble sort function both solely have the code required to sort, while the swap function is a self-created function whose purpose is just to swap the values at two indices. I also imported all the previous sorts from the previous labs to implement and change them to sort in reverse order. I changed all the sorts to sort in reverse order so that the largest number comes first while the smallest is last. I also

Testing Plan:

To test this, the process was the same as all the other labs. I read in all the files and ran both sorts on them and printed out the time that it took to sort each of the functions. I ran 1000, 2000, 4000, 8000, and 16000 for all sorts from insertion to bubble sort. I only ran the Random, Reverse Sorted, and Sorted for selection sort and bubble sort.

Test Cases:

Array Size	Insertion Sort (ns)		Merge Sort (ns)		Heap Sort (ns)		Quick Sort (ns)		Selection Sort (ns)		Bubble Sort (ns)
1000	8111900		3040700		1579700		1100200		3087900		13663900
2000	587900		690500		709900		448400		1566700		7205100
4000	3637900		2729100		1331700		575400		11549600		29675800
8000	14690900		4983400		3333600		1932400		42418900		99594500
16000	36364500		2745600		4126300		2029500		87797900		413678300

Array Type	Selection Sort Time	Bubble Sort Time
Random	6239900	7302000
Reverse Sorted	4224200	5333500
Sorted	3744900	2209400

Analysis/Conclusion:

With the results gained from running the sorts, the great amount of time it takes to run each sort fits with the time complexity of both sorts. Based on the algorithm and pseudocode, the time complexity of selection sort and bubble sort should be $O(n^2)$. With the amount of time that it takes even as the size grows, it is expected that the time increases at such a large rate because of this time complexity. Compared to all the other sorts, selection sort and bubble sort will and always does take longer. Due to the n^2 time complexity, it will always be slower than the n or (n*log(n)) time complexities.

References:

The only references I must make are to the pseudocode and my previous lab in which I used a similar format for writing this report. Below I have added images of all my code for this lab.

```
private static Lab7Sorts ls = new Lab7Sorts();
private static InsertionSort is = new InsertionSort();
private static Merge ms = new Merge();
private static HeapSort hp = new HeapSort();
private static QuickSort qs = new QuickSort();

private static int[] iArray = new int[]{};
private static int[] mArray = new int[]{};
private static int[] hArray = new int[]{};
private static int[] qsArray = new int[]{};
private static int[] tempArray = new int[]{};
private static int[] ssArray = new int[]{};
private static int[] bsArray = new int[]{};

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

        iArray = new int[sLength];
        mArray = new int[sLength];
        hArray = new int[sLength];
        qsArray = new int[sLength];
        ssArray = new int[sLength];
        bsArray = new int[sLength];

        for(int i = 0; i < sLength; i++)
        {
            iArray[i] = Integer.parseInt(sArray[i]);
            mArray[i] = Integer.parseInt(sArray[i]);
            hArray[i] = Integer.parseInt(sArray[i]);
            qsArray[i] = Integer.parseInt(sArray[i]);
            ssArray[i] = Integer.parseInt(sArray[i]);
            bsArray[i] = Integer.parseInt(sArray[i]);
        }
        sc.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }

    tempArray = new int[mArray.length];
    for(int x = 0; x < mArray.length; x++)
    {
        tempArray[x] = mArray[x];
    }

    return tempArray.length;

}
```

```
private static void getTimes(int size)
{
    long iTime = System.nanoTime();
    is.iSortR(iArray);
    long eiTime = System.nanoTime() - iTime;
    System.out.println("Time taken to insertion sort array of " + size + " numbers: " + eiTime);

    long mTime = System.nanoTime();
    ms.mergeSort(mArray, tempArray, 0, mArray.length-1);
    long emTime = System.nanoTime() - mTime;
    System.out.println("Time taken to merge sort array of " + size + " numbers: " + emTime);

    long hTime = System.nanoTime();
    hp.sort(hArray);
    long ehTime = System.nanoTime() - hTime;
    System.out.println("Time taken to heapsort array of " + size + " numbers: " + ehTime);

    long sTime = System.nanoTime();
    qs.sort(qsArray, 0, qsArray.length-1);
    long eTime = System.nanoTime() - sTime;
    System.out.println("Time taken to quick sort array of " + size + " numbers: " + eTime);

    long ssTime = System.nanoTime();
    ls.selectionSortR(ssArray);
    long esTime = System.nanoTime() - ssTime;
    System.out.println("Time taken to selection sort array of " + size + " numbers: " + esTime);

    long bTime = System.nanoTime();
    ls.bubbleSortR(bsArray);
    long ebTime = System.nanoTime() - bTime;
    System.out.println("Time taken to bubble sort array of " + size + " numbers: " + ebTime + "\n");
}
```

```java
public static void main(String[] args)
{
    NumberGenerator ng = new NumberGenerator();
    Random r = new Random();
    int randomSize = 1024;

    File f = new File("bonus_random.txt");

    int[] temp = new int[]{};
    int[] random = new int[]{};
    int[] reversed = new int[]{};
    int[] sorted = new int[]{};

    try
    {
        random = new int[randomSize];
        FileWriter fw = new FileWriter(f);
        for(int i = 0; i < randomSize; i++)
        {
            int randNum = r.nextInt(randomSize);
            fw.write(randNum + " ");
            random[i] = randNum;
        }
        fw.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    temp = random;

    is.iSortR(random);
    reversed = random;

    is.iSort(random);
    sorted = random;

    long rdTime = System.nanoTime();
    ls.selectionSort(random);
    long erdTime = System.nanoTime() - rdTime;
    System.out.println("Time taken to selection sort array of random numbers: " + erdTime);

    long rsTime = System.nanoTime();
    ls.selectionSort(reversed);
    long ersTime = System.nanoTime() - rsTime;
    System.out.println("Time taken to selection sort array of reversed sorted numbers: " + ersTime);

    long sTime = System.nanoTime();
    ls.selectionSort(sorted);
    long esTime = System.nanoTime() - sTime;
    System.out.println("Time taken to selection sort array of sorted numbers: " + esTime + "\n");
```

```java
int size;

/*
File f4 = new File("input_100.txt");
size = updateArray(f4);
getTimes(size);
*/

File f4 = new File("input_1000.txt");
size = updateArray(f4);
getTimes(size);

size *= 2;

String newFile = ng.getNewFile(size);
File f5 = new File(newFile);
size = updateArray(f5);
getTimes(size);

size *= 2;

String newFile2 = ng.getNewFile(size);
File f6 = new File(newFile2);
size = updateArray(f6);
getTimes(size);

size *= 2;

String newFile3 = ng.getNewFile(size);
File f7 = new File(newFile3);
size = updateArray(f7);
getTimes(size);

size *= 2;

String newFile4 = ng.getNewFile(size);
File f8 = new File(newFile4);
size = updateArray(f8);
getTimes(size);
```

```java
random = temp;

long rdTime2 = System.nanoTime();
ls.bubbleSort(random);
long erdTime2 = System.nanoTime() - rdTime2;
System.out.println("Time taken to bubble sort array of random numbers: " + erdTime2);

long rsTime2 = System.nanoTime();
ls.bubbleSort(reversed);
long ersTime2 = System.nanoTime() - rsTime2;
System.out.println("Time taken to bubble sort array of reversed sorted numbers: " + ersTime2);

long sTime2 = System.nanoTime();
ls.bubbleSort(sorted);
long esTime2 = System.nanoTime() - sTime2;
System.out.println("Time taken to bubble sort array of sorted numbers: " + esTime2 + "\n");
```

```java
public void selectionSort(int[] A)
{
    int length = A.length;
    for(int i = 0; i < length - 1; i++)
    {
        int min = i;
        for(int j = i + 1; j < length; j++)
        {
            if(A[j] < A[min])
            {
                min = j;
            }
        }
        if(i != min)
        {
            swap(A, min, i);
        }
    }
}

public void bubbleSort(int[] A)
{
    int length = A.length;
    for(int i = 0; i < length -1; i++)
    {
        for(int j = length; j > i + 1; j--)
        {
            if(A[j-1] < A[j-2])
            {
                swap(A, j-1, j-2);
            }
        }
    }
}
```

```java
package CS303.Lab7;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class NumberGenerator
{
    public String getNewFile(int size)
    {
        Random r = new Random();
        String file = "input_" + size + ".txt";
        File f = new File(file);
        try
        {
            FileWriter fw = new FileWriter(f);
            for(int i = 0; i < size; i++)
            {
                int randNum = r.nextInt(size);
                fw.write(randNum + " ");
            }
            fw.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

        return file;
    }
}
```

```java
public void selectionSortR(int[] A)
{
    int length = A.length;
    for(int i = 0; i < length - 1; i++)
    {
        int min = i;
        for(int j = i + 1; j < length; j++)
        {
            if(A[j] > A[min])
            {
                min = j;
            }
        }
        if(i != min)
        {
            swap(A, min, i);
        }
    }
}

public void bubbleSortR(int[] A)
{
    int length = A.length;
    for(int i = 0; i < length -1; i++)
    {
        for(int j = length; j > i + 1; j--)
        {
            if(A[j-1] > A[j-2])
            {
                swap(A, j-1, j-2);
            }
        }
    }
}

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