Assignment #8

Rikiya Takehi
1W21CS04-6
Algorithms and Data Structures
07/06/2022

Problem Statement

Write a bubble sort algorithm in java programming. There are three ways of bubble sort algorithms shown in class: the basic bubble sort program, bubble sort program using exchg, and bubble sort program using last. Test the programs use “testdata-sort-1.txt”, “testdata-sort-2.txt”, and “testdata-sort-3.txt” as test data. For each test data, measure the time taken to sort the program in milliseconds.

Programs

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.Scanner;

public class BubbleSort {
    static void swap(int[] a, int idx1, int idx2) {
        int d=a[idx1];
        a[idx1]=a[idx2];
        a[idx2]=d;
    }
    static void bubbleSort(int[] a, int n) {
        for(int i=0; i<n-2; i++){
            for(int j=n-1; j>=i+1; j--){
                if(a[j-1] > a[j]){
                    swap(a, j-1, j);
                }
            }
        }
    }
    static void bubbleSort1(int[] a, int n) {
        for(int i=n-1; i<=n-2; i++){
            int exchg=0;
            for(int j=n-1; j>=i+1; j--){
                if(a[j-1]>a[j]){
                    swap(a, j-1, j);
                    exchg = exchg+1;
                }
            }
            if(exchg==0){
                break;
            }
        }
    }
    static void bubbleSort2(int[] a, int n) {
        int k=0;
        while(k<n-1){
            int last=n-1;
            for(int j=n-1; j>=k+1; j--){
                if(a[j-1]>a[j]){
                    swap(a, j-1, j);
                    last=j;
                }
            }
            k=last;
        }
    }
    public static void main(String[] args) {
        try{
            Path file=Paths.get("./testdata-sort-1.txt");
            List<String>stringData=Files.readAllLines(file);
            int [] x = new int[stringData.size()];
            int nx = x.length;
            for (int i = 0; i < x.length; i++) {
                x[i]=Integer.parseInt(stringData.get(i));
            }

            System.out.println("Bubble sort");
            long start = System.currentTimeMillis();
            bubbleSort(x, nx);

            System.out.println("Sorted array");
            for (int i = 0; i < nx; i++){
                System.out.println("x[" + i + "]=" + x[i]);
            }
            long end = System.currentTimeMillis();
            System.out.println("Execution time: "+(end - start));
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Program explanation

Bubble sort program is a sorting program where it compares each of the adjacent element pairs, and swaps them if they are not placed in order.

Lines 9-11 are codes where two numbers (idx1 and idx2) can be swapped. By using an integer d, it replaces the numbers. In the later codes, it only requires to recall "swap" in order to swap two specific numbers.

```
int d=a[idx1];
a[idx1]=a[idx2];
a[idx2]=d;
```

In lines 15-21, the bubble sorting program is written. It searches the sequence from the right side, and if the number on the left is bigger than the number on the right, it recalls the "swap" method and swaps the two numbers. This is repeated until the end of the whole sequence.

```
for(int i=0; i<n-2; i++){
    for(int j=n-1; j>=i+1; j--){
        if(a[j-1] > a[j]){
            swap(a, j-1, j);
        }
    }
}
```

In lines 24-35, the improvement code for bubble sort is written ("BubbleSort1"). This basically follows the same sequences as the normal bubble sort. However, using the integer "exchg", minimizes the number of steps needed to scan the data. If there are no swaps done after a pass, no more passes are needed because they are all sorted. When the needed exchange is 0, it is programmed to stop in lines 32-34.

```
static void bubbleSort1(int[] a, int n) {
    for(int i=n-1; i<=n-2; i++){
        int exchg=0;
        for(int j=n-1; j>=i+1; j--){
            if(a[j-1]>a[j]){
                swap(a, j-1, j);
                exchg = exchg+1;
            }
        }
        if(exchg==0){
            break;
        }
    }
```

In lines 37-49, there is another bubble sort method using “last”. In the program, the leftmost integers of the swapped variables. When this variable is named “last”, all the sequences on the left-hand side of “last” do not need further sorting. The variable “last” is updated after every pass, and the variable “last” is substituted with the variable “k” after the current pass (line 47). The program minimizes the size of the sequence after every pass, making the sorting easier.

```
static void bubbleSort2(int[] a, int n) {
    int k=0;
    while(k<n-1){
        int last=n-1;
        for(int j=n-1; j>=k+1; j--){
            if(a[j-1]>a[j]){
                swap(a, j-1, j);
                last=j;
            }
        }
        k=last;
    }
}
```

In line 61, it records the exact time using the following code and states “start”.

```
long start = System.currentTimeMillis();
```

In line 68, it records the time again using the following code and states “end”. Then in line 69, it shows the time taken to complete the sorting by subtracting the time recorded as “end” by time recorded at “start”.

```
long end = System.currentTimeMillis();
System.out.println("Execution time: " + (end - start));
```

When switching the data and the sorting method, change the names in lines 52 and 62.

```
Path file=Paths.get("./testdata-sort-1.txt");
```

```
bubbleSort(x, nx);
```

Experimental settings

Input files:

“testdata-sort-1.txt”, “testdata-sort-2.txt”, “testdata-sort-3.txt”

Tested programs:

“bubbleSort”, “bubbleSort1”, “bubbleSort2”

Results

Table showing the time taken for each bubble sort program to sort each test data in milliseconds

	testdata-sort-1.txt	testdata-sort-2.txt	testdata-sort-3.txt
bubbleSort	28 ms	16096 ms	6350 ms
bubbleSort1	21 ms	17052 ms	496 ms
bubbleSort2	23 ms	16682 ms	384 ms

Discussion of the Results

The time taken to sort testdata-sort-1.txt was almost the same between the normal bubble sort, bubble sort using variable "exchg", and bubble sort using the variable "last". One is faster or slower than the other, but the difference is very small so it is dismissible.

For the time taken for sorting "testdata-sort-2.txt", the normal bubble sort was faster by 966 milliseconds than bubbleSort1 and 586 seconds faster than bubbleSort2. If most or all passes are performed by bubbleSort1 and bubbleSort2, these programs would take more time. Since the programs using variable "exchg" needs a step more each time in order to set the "exchg", it does take longer time, but only by a little bit. The same goes for the bubble sort program using "last"; it takes a few more steps and time because it needs to set the variable as "last" each time. Therefore, it can be guessed that the program was completely random, and most/all steps were needed to be performed by bubbleSort1 and bubbleSort2.

The time taken to sort "testdata-sort-3.txt", the normal bubble sort was drastically slower than the other two improved bubble sorts. It can be guessed that the sequences were already almost sorted. If the sequences are sorted to some extent, the two improved bubble sorts need drastically less time. For the improved versions of bubble sort, if the arrays are completely sorted, the time complexity is $O(n)$, whereas if all passes were done, it requires $O(n^2)$ as time complexity. The normal bubble sort always requires $O(n^2)$ for all cases.