Algorithm and Data Structures

Assignment 8: Report

Name: Maeda Taishin

Student ID: 1W21CF15

Date: 06/06/2023

Bubble Sort: Basic bubble sort algorithm

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
public class bubbleS {
    Run | Debug
    public static void main(String[] args){
        try{
            Path file = Paths.get (first:"testdata-sort-1.txt");
            List<String> stringData = Files.readAllLines(file);

            int[] arry = new int[stringData.size()];
            for (int i = 0; i<arry.length; i++){
                arry[i] = Integer.parseInt(stringData.get(i));
            }
            int n = arry.length;
            long start = System.nanoTime();
            bubS(arry, n);
            long end = System.nanoTime();
            //System.out.println("Sorted Array: ");
            //printArry(arry);
            System.out.println("Execution time: " + (end - start));

        }catch (IOException e){
            e.printStackTrace();
    }   }
    private static void bubS(int[] arry, int n){
        for(int i=0; i<=n-2; i++){
            for(int j=n-1; j>=i+1; j--){
                if(arry[j-1] > arry[j]){
                    swap(arry, j);
                }
            }
        }
    }
    private static void swap(int[] arry, int j){
        int temp = arry[j];
        arry[j] = arry[j-1];
        arry[j-1] = temp;
    }
    /*private static void printArry(int[] arry){
        for(int i=0; i<arry.length; i++){
            System.out.println("x[" + i + "]=" + arry[i]);
        }
        System.out.println();
    }*/
}
```

Code Explanation:

Main function (Lines: 8-26)

Line: 9

Initially, the main function will be described. Talking about reading a text file, the idea is that we first assumed the text file, “testdata-sort-1.txt” in this case, is the same directory.

Line: 10

Next, we needed to read all lines in the files provided and store data in a list of string.

Lines: 12-15

Then, we needed to convert the list that is recognized as a string in this case to an integer array. In order to do that, several Java libraries are needed to be stored at the beginning of the code (Lines: 1-5). Lastly, the code inside the main function will be placed inside the try-catch exception, allowing us to define blocks of codes to be tested or executed if an error occurs (Lines: 8 and 24-25).

Line: 16

After that, a variable called “n” is created to represent the size of the array, as the program starts counting the first element as the zeroth position. The function in this program is the sorting function using the Bubble sort algorithm which will be discussed later.

Lines: 17 and 19

Furthermore, several syntaxes are included to measure the execution time of the program after calling the select sorting function. For instance, the data type “long” is used since it contains the minimum value of -2 to the power of 63 and a maximum value of, 2 to the power of 63, minus 1, so the program can return the time in nanoseconds when calculating the time interval. The program is designed to record the current time right before entering the function in line 17 and record again after the function is called and executed. Next, the amount of time spent will be calculated by subtracting the “end” and “start” which are the time after and before respectively, and printing it out in line 22 of this program.

Line: 18

The function “bubS” is set and called by passing the array “arry” an integer “n” to the function to perform the bubble sorting algorithm.

Comment codes (lines: 20-21 and 41-46)

I just used them to make sure that the code actually executes the sorted array by using the testdata-sort-1.txt file since it contains the smallest amount of data (100) I was able to make the program print it out to see and confirm the outcomes.

Sorting Function (Lines: 27-35)

Moving on to the explanation about the function, in this program, the array “arry” and the array’s length “n” are passed to the Bubblel sort function called “bubS”. Generally speaking, the Bubble sort is the algorithm of comparing adjacent elements and swapping them if they are in the wrong order.

Line: 28

A for-loop is defined with variable “i” set to zero and the loop will keep iterating as the value of “i” is incremented by one until the value is equal to n-2 which is the length of the array “arry” subtracted by two. This identifies the second leftmost element inside the array which will later be discussed later.

Line: 29

A nested for-loop is created with variable “j” initialized to n-1, array’s length minus 1, meaning that it is represented as the rightmost element inside the array “arry”. This loop will keep running until “j” is equal to “i”+1 as the value of “j” is decremented by one. For instance, i+1 refers to the position i+1th which is one of the adjacent elements (right one since the position is plus one) that will be compared.

Lines: 30 and 31

Inside the nested for loop, an if-else statement is created aiming to compare the two adjacent elements in the array. The condition is satisfied when the value of the element in the position “j-1” (this will be the second rightmost element when the for-loop is entered for the first time) is greater than the value in the position “j” which is the value right next to it. In other words, the condition is satisfied when the elements are placed in the wrong order (non-accending order). Then, both elements will be swapped by passing them (“arry” and “j”) into the swap function.

Lines: 36-40 (Swapping function)

After “arry” and “j” are passed into this function, a temporary variable “temp” is created and is assigned to arry[j], the arry[j] is assigned to arry[j-1], and arry[j-1] is assigned to “temp”. This will swap those two elements to a sorted position.

Note:

The program will be comparing two adjacent elements from the two rightmost elements to the two leftmost elements in the array considering that the position of the leftmost element will be increased by one position every time the loop is entered. Note that, When the first pass, i = 0, which means that element at the 0th position was already sorted with the final value of “j” as 1. All in all, the value of elements will be checked and sorted until every condition are satisfied.

Bubble Sort1: Improve bubble sort algorithm with exchg

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
public class bubbleS1 {
    Run | Debug
    public static void main(String[] args){
        try{
            Path file = Paths.get (first:"testdata-sort-2.txt");
            List<String> stringData = Files.readAllLines(file);

            int[] arry = new int[stringData.size()];
            for (int i = 0; i<arry.length; i++){
                arry[i] = Integer.parseInt(stringData.get(i));
            }
            int n = arry.length;
            long start = System.currentTimeMillis();
            bubS1(arry, n);
            long end = System.currentTimeMillis();
            //System.out.println("Sorted Array: ");
            //printArry(arry);
            System.out.println("Execution time: " + (end - start));

        }catch (IOException e){
            e.printStackTrace();
    }   }
    private static void bubS1(int[] arry, int n){
        for(int i=0; i<=n-2; i++){
            int exchg = 0;
            for(int j=n-1; j>=i+1; j--){
                if(arry[j-1] > arry[j]){
                    swap(arry, j);
                    exchg += 1;
                }
            }
            if(exchg == 0){
                break;
            }
        }
    }
    private static void swap(int[] arry, int j){
        int temp = arry[j];
        arry[j] = arry[j-1];
        arry[j-1] = temp;
    }
    /*private static void printArry(int[] arry){
        for(int i=0; i<arry.length; i++){
            System.out.println("x[" + i + "]=" + arry[i]);
        }
        System.out.println();
    }*/

}
```

Code Explanation:

Main Function(Lines: 8-26)

Similar to the previous bubble sort code. The only different thing is the function name which is "bubS1" in this case, and the time calculation which has been done in milliseconds by using "currentTimeMillis()".

Sorting Function (Lines: 27-40)

The idea of this bubble sort algorithm is the upgrade version of the previous one which can be done by creating an extra variable, "exchg" which will be discussed later, to check how many exchanges or swaps occurred in each pass. The program will exit the loop when there is no exchange or swap occurred meaning that the array has been already sorted and no more pass is needed. This will improve the algorithm to avoid unnecessary checking and reduce the time to execute.

Line: 28

For-loop (Same as the previous bubble sort algorithm)

Line: 29

A variable called "exchg" is created and set to zero and will be used to check whether there are any exchanges or swaps that existed in a nested for-loop below. Note that, when "exchg" is equal to zero, it represents that the number of exchanges or swaps is equal to zero.

Line: 30

Nested for-loop (Same as the previous bubble sort algorithm)

Lines: 31-32

If-else statement with a condition that leads to swapping (Same as the previous bubble sort algorithm)

Line: 33

After the swapping occurred, the variable "exchg" will be incremented by one indicating that there is an additional swapping, and the value will keep increasing as the number of swapping increases within the loop.

Line: 36-38

Another if-else statement is created. It is located outside the nested for-loop but still inside the initial for-loop. This is the place where the program can exit the loop if the value of "exchg" is equal to zero since there was no exchange or swap occurring in the array. In other words, when the number of swaps is zero after a pass, it means that all of the elements in the array are already sorted and no more pass is needed.

Lines: 41-44: (Swapping function) (Same as the previous bubble sort algorithm)

Bubble Sort2: Improve bubble sort algorithm with last

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
public class bubbleS2 {
    Run | Debug
    public static void main(String[] args){
        try{
            Path file = Paths.get (first:"testdata-sort-1.txt");
            List<String> stringData = Files.readAllLines(file);

            int[] arry = new int[stringData.size()];
            for (int i = 0; i<arry.length; i++){
                arry[i] = Integer.parseInt(stringData.get(i));
            }
            int n = arry.length;
            long start = System.currentTimeMillis();
            bubS2(arry, n);
            long end = System.currentTimeMillis();
            //System.out.println("Sorted Array: ");
            //printArry(arry);
            System.out.println("Execution time: " + (end - start));

        }catch (IOException e){
            e.printStackTrace();
    }   }
    private static void bubS2(int[] arry, int n){
        int k = 0;
        while(k < n-1){
            int last = n-1;
            for(int j=n-1; j>=k+1; j--){
                if(arry[j-1] > arry[j]){
                    swap(arry, j);
                    last = j;
                }
            }
            k = last;
        }
    }
    private static void swap(int[] arry, int j){
        int temp = arry[j];
        arry[j] = arry[j-1];
        arry[j-1] = temp;
    }
    /*private static void printArry(int[] arry){
        for(int i=0; i<arry.length; i++){
            System.out.println("x[" + i + "]=" + arry[i]);
        }
        System.out.println();
    }*/

}
```

Code Explanation:

Main Function(Lines: 8-26)

Similar to the previous bubble sort code. The only thing that is different is the function name which is “bubS2” in this case.

Sorting Function (Lines: 27-39)

This is another upgrade version of the bubble sort algorithm. The idea is to create a variable called “last” that represents the position of the swapped right element when swapped. For instance, the“last” will indicate the latest position that swapping had occurred which will always be the right side element when comparing two elements. Thus, when the program entered the next pass/loop, the program will avoid comparing elements in the sorted part and start checking the condition until at one position to the right of the sorted element resulting in reduced execution time. The program will be discussed more later.

Line: 28

A variable “k” is initialized to zero which will later be used to represent the position of the latest swapped right element. It is also used in the while-loop condition later.

Line: 29

A while-loop is created, and the condition keeps running as long as the value of “k” is less than the array length minus one (the rightmost position of the array).

Line: 30

A variable “last” is created and set to n-1 which is the length of the array minus one which is the rightmost position of the array in this case.

Lines: 31-35

Firstly, a for-loop is defined with variable “j” set to n-1 and the loop will keep iterating as the value of “j” is decremented by one until the value is equal to k+1 which is one position next to “k”. “k” will be indicated as the latest position that the swapping had occurred which be mentioned later. Next, an if-else statement is created with the condition the same as the two previous bubble sort algorithms as well as for the swapping function. Then, the value of “last” is set equal to “j” when there is swapping in order to record the latest right element swapping position of the array since “j” is the position at that moment. Note that, “last” will always be updated every time when swapped.

Line: 37

The value of “last” that had been finalized in the previous condition above will be assigned to “k” when the completion of pass. Now, “k” will have the latest swapping position, and the program will decide whether the loop should be entered again depending on the value of “k”. If “k” is equal to n-1 then the program will terminate indicating that the array is completely sorted.

Lines: 40-44: (Swapping function) (Same as the two previous bubble sort algorithms)

Execution Results and Discussions:

testdata-sort-1.txt

	basic	exchg	last
Time in milliseconds	0.147875	0.158542	0.163333

Table 1. Execution time in milliseconds of three different bubble sort types for the testdata-sort-1.txt

testdata-sort-2.txt

	basic	exchg	last
Time in milliseconds	12600	12716	12855

Table 2. Execution time in milliseconds of three different bubble sort types for the testdata-sort-2.txt

testdata-sort-3.txt

	basic	exchg	last
Time in milliseconds	3557	19	18

Table 3. Execution time in milliseconds of three different bubble sort types for the testdata-sort-3.txt

Time complexity	basic	exchg	last
Best-case	$O(n^2)$	$O(n)$	$O(n)$
Average-case	$O(n^2)$	$O(n^2)$	$O(n^2)$
Worst-case	$O(n^2)$	$O(n^2)$	$O(n^2)$

Table 4. The time complexity of three different bubble sort types

As can be seen in Table 1-2, the execution times of three different bubble sort algorithms in the first two test data files do not show significant differences as the data in those files are mostly randomized. The two improved versions even take slightly more time to execute with the execution time of 0.147875, 0.158542, and 0.163333 milliseconds for the first data files by basic, exchg, and last version respectively. For the second data file, the basic, exchg, and last versions took 12600, 12716, and 12855 milliseconds respectively. The idea of exchg and last have shown significant improvements in execution performances when it comes to the third test data file where the data is partially sorted. From the third data file, there will be relatively less amount of swaps to be done compared to the first two files which can be clearly seen in Table 3 where it took approximately 18-19 milliseconds for exchg and last version to sort while it took approximately 3557 milliseconds for the basic version to execute. This indicates how powerful the improved versions could be in order to make the algorithm be more efficient and less time-consuming.

From Table 4, we can observe that the basic version of the bubble sort algorithm has a time complexity $O(n^2)$ in every case since the outer loop always iterates n-1 times regardless of whether the data is already sorted or not. The exchg and last version share the same time complexity. In the best-case time complexity of when the data is already sorted, the outer loops of the two versions will iterate only once, so $O(n)$ will be their time complexities. For the worst-case time complexity, all of the three different bubble sort algorithms have shown the same outcome. The worst-case is achieved when the data is sorted in reverse meaning that the outer loop will always iterate n-1 times with a swap that occurred in every inner/nested loop. For the average-case and worst-case of exchg and last version, the time complexity will be $O(n^2)$. As can be seen, the average-case and worst-case of all three versions have the same time complexity.

In addition, the bubble sort algorithm will perform the swapping of unequal adjacent elements. Hence, the order of equal elements in data files stays the same after sorting which is why the bubble sort is a stable algorithm. However, with the time complexity of $O(n^2)$ when it comes to average-case and worst-case, it is not suitable for sorting a large unsorted size of data. Meanwhile, it is one of the useful algorithms for sorting a small data size or almost sorted data.