Algorithm and Data Structures

Assignment 4: Report

Name: Maeda Taishin

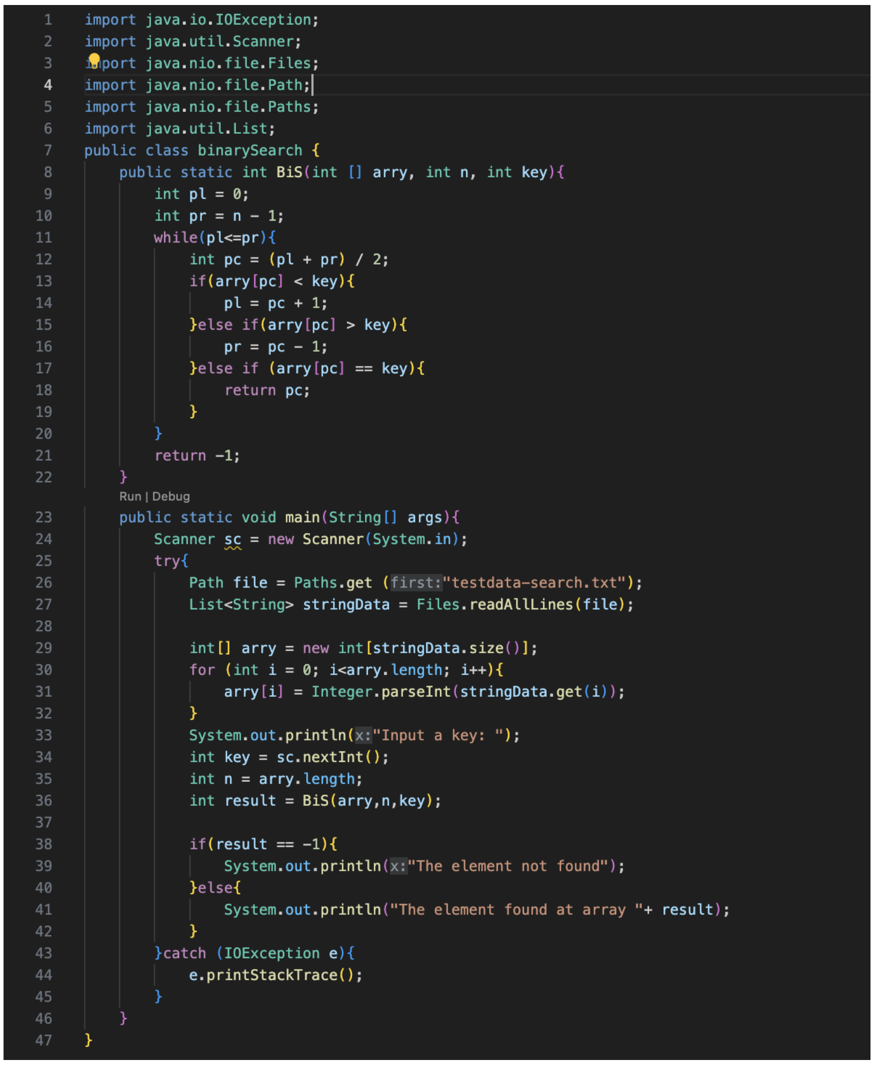
Student ID: 1W21CF15

Date: 09/05/2023

Ex. 1: Binary search in Java

- Write binary search algorithm in Java
  - A pseudo code has been shown in the class #3

  - Assumptions
    - A **sorted** integer array is given
      - e.g., {1, 2, 3, 6, 9, 10}
    - A key will be obtained from the standard input
      - use java.util.Scanner



```
1   import java.io.IOException;
2   import java.util.Scanner;
3   import java.nio.file.Files;
4   import java.nio.file.Path;
5   import java.nio.file.Paths;
6   import java.util.List;
7   public class binarySearch {
8       public static int BiS(int [] arry, int n, int key){
9           int pl = 0;
10          int pr = n - 1;
11          while(pl<=pr){
12              int pc = (pl + pr) / 2;
13              if(arry[pc] < key){
14                  pl = pc + 1;
15              }else if(arry[pc] > key){
16                  pr = pc - 1;
17              }else if (arry[pc] == key){
18                  return pc;
19              }
20          }
21          return -1;
22      }
        Run | Debug
23      public static void main(String[] args){
24          Scanner sc = new Scanner(System.in);
25          try{
26              Path file = Paths.get (first:"testdata-search.txt");
27              List<String> stringData = Files.readAllLines(file);
28
29              int[] arry = new int[stringData.size()];
30              for (int i = 0; i<arry.length; i++){
31                  arry[i] = Integer.parseInt(stringData.get(i));
32              }
33              System.out.println(x:"Input a key: ");
34              int key = sc.nextInt();
35              int n = arry.length;
36              int result = BiS(arry,n,key);
37
38              if(result == -1){
39                  System.out.println(x:"The element not found");
40              }else{
41                  System.out.println("The element found at array "+ result);
42              }
43          }catch (IOException e){
44              e.printStackTrace();
45          }
46      }
47  }
```

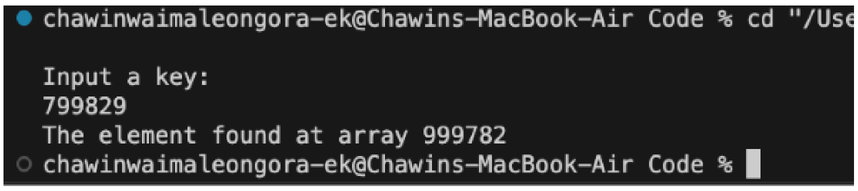
Code Explanation:

Initially, I would like to explain the code in the main function. As seen in the picture (line: 24), I start by setting up a scanner object called “sc” to get user input by importing the Scanner library. The explanation of reading a file from a text file will be discussed later. Next, I prompt users to input the element they want to find. I create a variable called “key” to store the integer input from the users, which will be used later when calling a searching function. I also create a variable called “n” to represent the size of the array, as the program starts counting the first element as the zeroth position. In line 37 of this program, I create a variable called “result” to call the searching function named “BiS” representing the binary search, passing the array “arry”, the array's size “n”, and the input “key” (The function will be discussed later). Finally, I use an if-else statement (Line: 38-42) to check whether the condition for finding the element holds, based on the value of “result” returned from the searching function. For instance, if the value of “result” is -1, indicating that the element is not found in the array, the program will execute “The element not found”. On the other hand, if the value of “result” is not -1, then the element is found in the array and will be printed out as “The element found at array” followed by the position of the element located inside the array.

For an explanation about reading a text file, the idea is that we first assumed the text file, “testdata-search.txt” in this case, is the same directory (Line: 26). Next, we needed to read all lines in the files provided and store data in a list if string(Line: 27). Then, we needed to convert the list that is recognized as a string in this case to an integer array (Lines: 29-32). In order to do that, several Java libraries are needed to be stored at the beginning of the code (Lines: 1, 3-6). Lastly, the code inside the main function will be placed inside the try-catch exception which allows us to define blocks of codes to be tested or executed if an error occurs (Lines: 25 and 43-44).

After passing the array “arry”, the array’s length “n”, and the prompted number “key” to the function, now, I would like to discuss the binary search function part which is named “Bis” in this case. The idea behind this algorithm is that we repeatedly divide the searching span of the sorted array into half until we find or do not find the prompted number. In this function, firstly, I initialized the leftmost and the rightmost position (pointer) of the searching span which are the variable called “pl” and “pr” respectively (Line: 1-2). For instance, I set “pl” to zero as the first position of the array and set “pr” to n - 1 which represents the last position of the array initially. Then, the while-loop (Line: 11-20) is used in this function and it keeps running whenever the value of “pl” is less than or equal to the value of “pr”. Entering the loop, every time that the loop is repeated, the variable “pc”, current position or pointer, is set to be the value in the middle of “pl” and “pr” in order to be ready to be checked whether it suits in which condition (Line: 12). This is consistent with the idea of dividing the search span in half. Lines 13-19, it is about checking the condition by using an if-else statement where considering whether the element in the “pc”th position of the “arry” which is in the middle is less than, greater than, or equal to the prompted value “key”. If the value of the “key” is greater than, the value of “pl” will be equal to “pc” plus one since there is a possibility that the key is located in the next searching span which is the list of elements starting from the new “pl” that is greater than the value in the “pc”th position while “pr” remains the same. On the other hand, if the value of the “key is less than, the value of “pr” will be equal to “pc” minus one which is one position to the left from the middle. This is because there is a possibility that the key is located in the next searching span which is the list of elements starting from the original “pc” to the new “pr”th position. Consequently, the program keeps running until the value of an element in the “pc”th position of the “arry” is equal to the value of the “key” meaning that the program found the prompted value in the array, and the program will return the value “pc” which is the position of the array to the main function to print the result out. Another case is when the program could not find the prompted number in the array which is the while-loop keeps repeating the loop until the value of “pl” is greater than “pr”. The program will break the loop and return -1 (Line: 21) to the main function which will be printed out as “The element not found”.

testdata-search.txt:



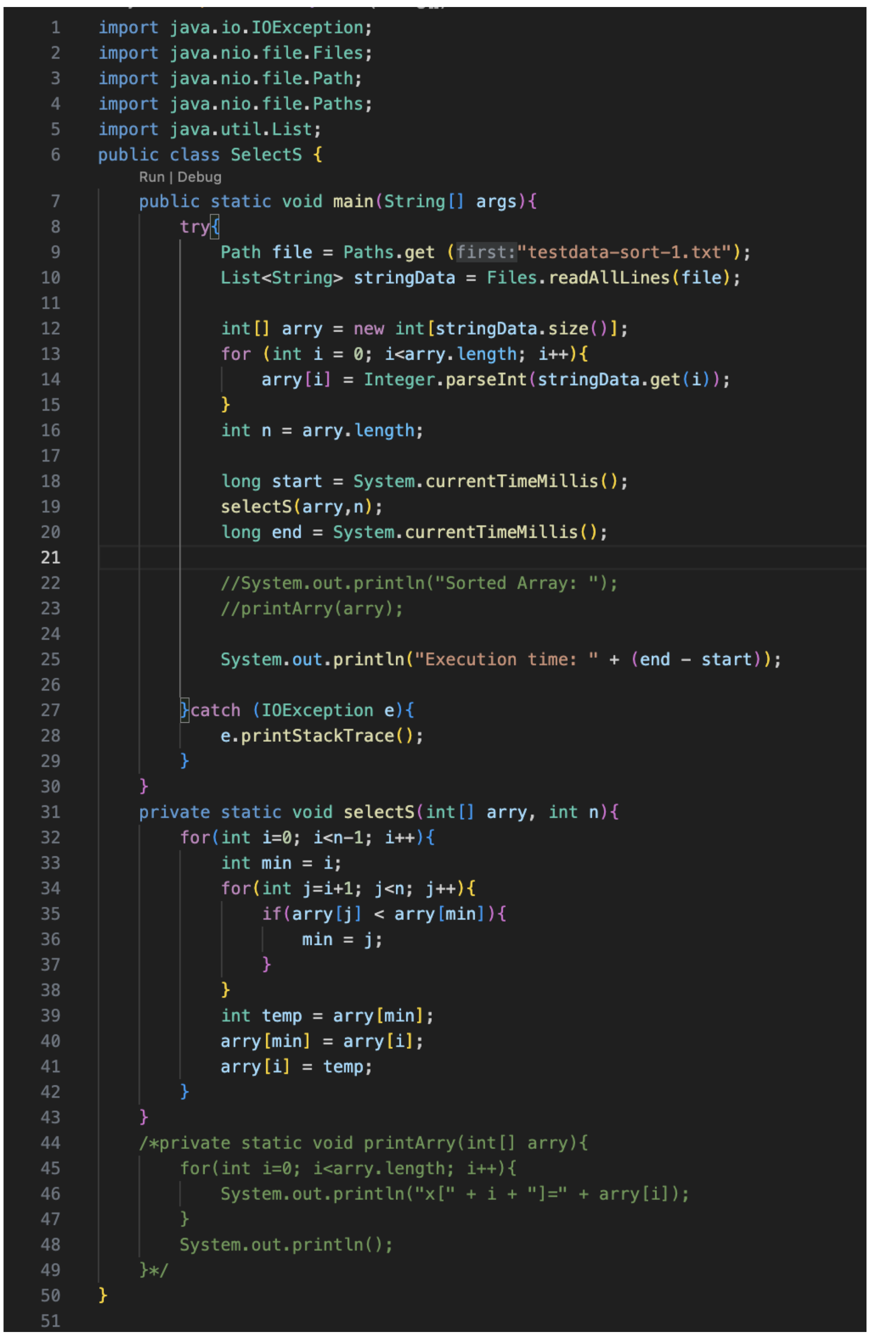
```
● chawinwaimaleongora-ek@Chawins-MacBook-Air Code % cd "/Use

  Input a key:
  799829
  The element found at array 999782
○ chawinwaimaleongora-ek@Chawins-MacBook-Air Code %
```

This is the result showing that the prompted value 799829 is found in the array at the 999782nd position.

Ex. 2: Selection sort in Java

- Write selection sort algorithm in Java
  - A pseudo code has been shown in the class #3
  - Assumptions
    - An integer array is initialized by using data from the standard input
      - use java.util.Scanner



```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 SelectS {
    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();
            selectS(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 selectS(int[] arry, int n){
        for(int i=0; i<n-1; i++){
            int min = i;
            for(int j=i+1; j<n; j++){
                if(arry[j] < arry[min]){
                    min = j;
                }
            }
            int temp = arry[min];
            arry[min] = arry[i];
            arry[i] = 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();
    }*/
}
```

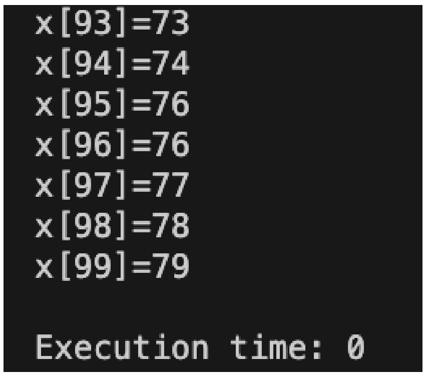
Code Explanation:

The main function of this program is similar to the previous program. However, the function in this program is the sorting function by using the select sorting algorithm which will be discussed later. Furthermore, several syntaxes are included in lines 18 and 20 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 milliseconds when calculating the time interval. The program is designed to record the current time right before entering the function in line 18 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 25 of this program.

In this program, the array “arry” and the array’s length “n” are passed to the select sorting function. The idea of the select sorting algorithm is to repeatedly find the minimum element in the unsorted array and put (swap) it at the sorted position. As can be seen in line 32, the for-loop is created with the initialized “i” value that equals zero and increments by one representing the position of the array. The loop will keep going until to value of “i” is less than the array’s length “n” minus one. Inside the loop (Line: 33), the variable “min” is created and set to be equal to “i” meaning that “min” will also be incremented by one. The variable “min” will be recognized as the minimum number. Next, the nested for-loop is created (Line: 34) with the variable “j” that is set to be “i” plus one representing the next position to “i” in the array in order to be used in comparing each other in the latter condition. Note that, “j” will be incremented only until it is less than “n” since it is started at one greater value. And then inside the nested loop, the if-else statement is created (Line: 35) to compare elements in the position “i” and “j” which can be written as arry[i] and arry[j] respectively. The condition is designed to be that when the arry[j] is less than arry[min], the element that is considered to be the smallest in the current situation, the value of “min” will be set equal to “j”. This process will keep repeating itself until the minimum element is found. After that, the smallest element (array[min]) will be swapped with the first element (arry[i]) in the nested loop. As can be seen in lines 39-41, the variable “temp” is created as a temporary variable which is assigned to arry[min], the arry[min] is assigned to arry[i], and arry[i] is assigned to “temp”. This will swap those two elements to a sorted position. Note that, the swapping is happening outside the nested loop but still inside the first loop. For instance, this program wants to find the minimum element every time when the value of “i” changed and then swaps with the first element, and the number of repeated times to find the minimum element will become less as “i” and “j” are increased and get closer to “n”. Thus, the program will eventually give the sorted array.

For the comment codes (lines: 22-23 and 44-49), 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.

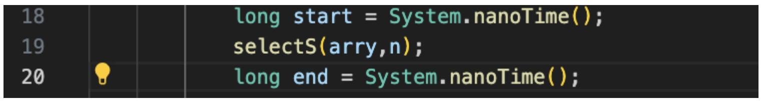
testdata-sort-1.txt:



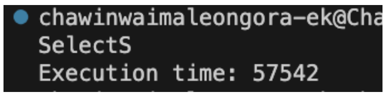
```
x[93]=73
x[94]=74
x[95]=76
x[96]=76
x[97]=77
x[98]=78
x[99]=79

Execution time: 0
```

For the first data file, it says that it took 0 milliseconds to sort. However, it is not exactly 0, but the program can compute extremely fast for the small array. Therefore, I changed the time measurement to nanoseconds, and the time taken is 57542 nanoseconds which is equal to 0.057542 milliseconds.

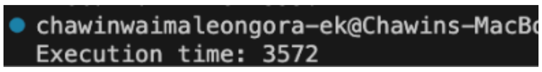


```
        long start = System.nanoTime();
        selectS(arry,n);
        long end = System.nanoTime();
```



```
● chawinwaimaleongora-ek@Cha
  SelectS
  Execution time: 57542
```

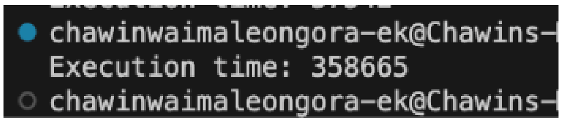
testdata-sort-2.txt:



```
● chawinwaimaleongora-ek@Chawins-MacB
  Execution time: 3572
```

For the second data file, it took 3572 milliseconds (3.572 seconds) to sort.

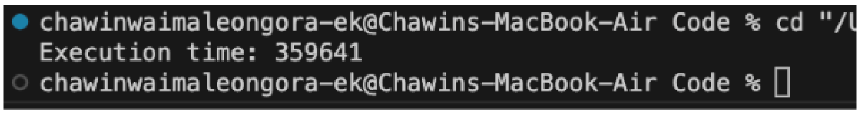
testdata-sort-3.txt:



```
● chawinwaimaleongora-ek@Chawins-
  Execution time: 358665
○ chawinwaimaleongora-ek@Chawins-
```

For the third data file, it took 358665 milliseconds (5.97775 minutes) to sort.

testdata-sort-4.txt:

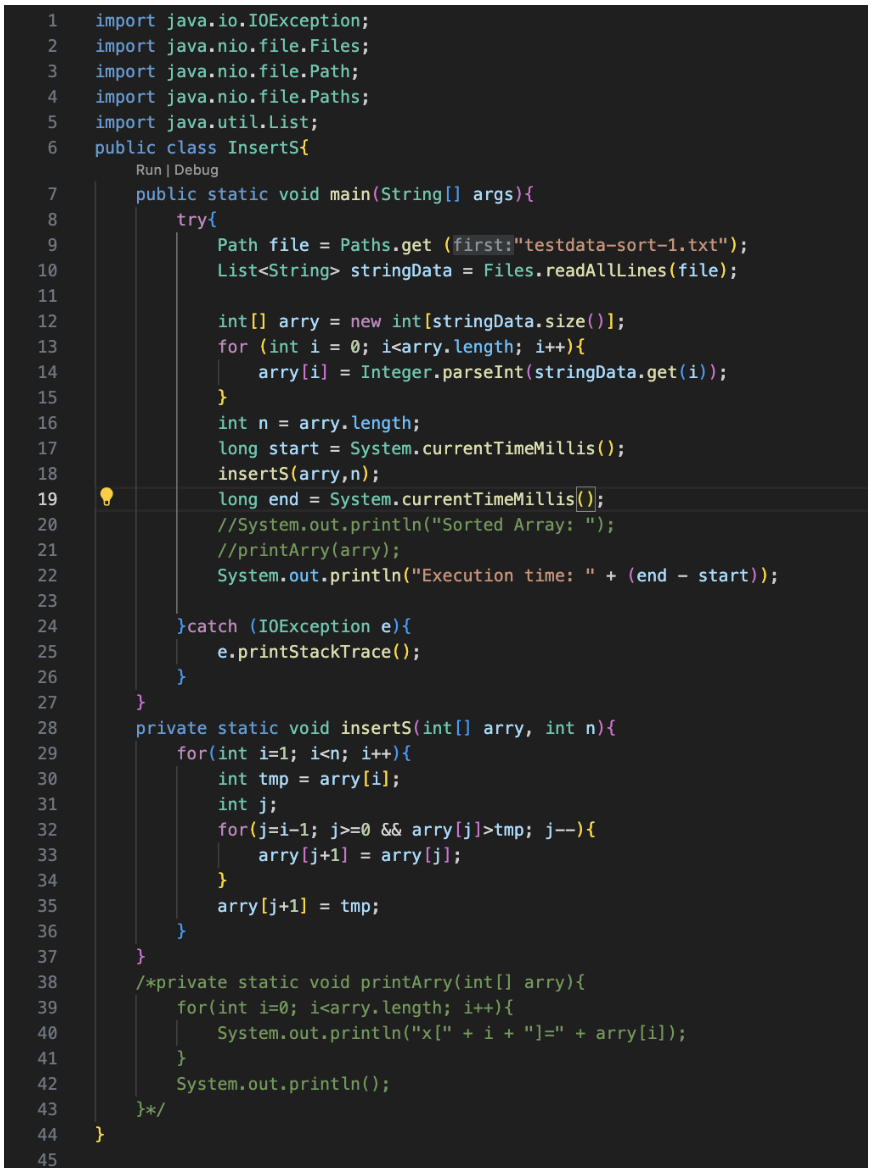


```
● chawinwaimaleongora-ek@Chawins-MacBook-Air Code % cd "/U
  Execution time: 359641
○ chawinwaimaleongora-ek@Chawins-MacBook-Air Code % ▯
```

For the fourth data file, it took 359641 milliseconds (about 5.994 minutes) to sort.

Ex. 3: Insertion sort in Java

- Write insertion sort algorithm in Java
  - A pseudo code has been shown in the class #3
  - Assumptions
    - An integer array is inputted from the standard input
      - use java.util.Scanner



```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 InsertS{
    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();
            insertS(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 insertS(int[] arry, int n){
        for(int i=1; i<n; i++){
            int tmp = arry[i];
            int j;
            for(j=i-1; j>=0 && arry[j]>tmp; j--){
                arry[j+1] = arry[j];
            }
            arry[j+1] = tmp;
        }
    }
    /*private static void printArry(int[] arry){
        for(int i=0; i<arry.length; i++){
            System.out.println("x[" + i + "]=" + arry[i]);
        }
        System.out.println();
    }*/
}

```

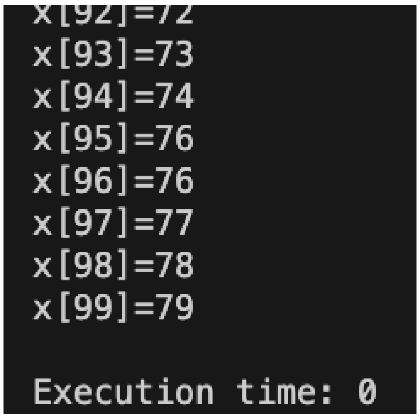
Code Explanation:

The main function of this program is similar to the Selection Sorting program. However, this is a program with an Insertion Sorting algorithm. The idea of insert sorting is to separate the array into the sorted part and the unsorted part and create a variable that temporarily stores the selected element.

Then, finds the appropriate place (hole position) in the unsorted part and then insert the element from the hole into that position.

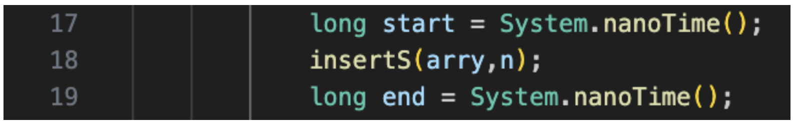
To be more specific, after passing the array “arry” and the array’s length “n” to the function, the for-loop is created (Line: 29). However, this time, “i” will be initialized and started from 1 and will be incremented until it is less than “n” because it will be considered as the position in the array and compared with the element that has a lower position in the upcoming nested-loop. Inside the first for-loop, the variable “tmp” is created and set to the element of the array at the position “i” in which “tmp” will work as storage that temporarily stores the element (Line: 30). In addition, the variable “j” is created to be used inside the nested loop below and outside it later (Line: 31). From line 32-34, the condition of the nested for-loop is set as “j” is equal to “i-1” representing the element in one position before “i”, and the value of “j” will be decremented by one whenever “j” is greater than equal to zero and arry[j] is greater than “tmp”. When the condition is satisfied arry[j+1], the element at one position after “j”, will be equal to “tmp”. Then, outside the nested loop but still inside the first loop (Line: 35) shows that the element at one position after “j” which is array[j+1] will be set equal to “tmp”. These processes will make elements that have a greater value than the selected element “tmp” shift to the unsorted part to be considered again and again until the array is completely sorted.

testdata-sort-1.txt

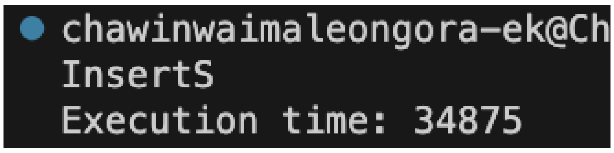

```
x[92]=72
x[93]=73
x[94]=74
x[95]=76
x[96]=76
x[97]=77
x[98]=78
x[99]=79

Execution time: 0
```

For the first data file, it says that it took 0 milliseconds to sort. However, it is not exactly 0, but the program can compute extremely fast for the small array. Therefore, I changed the time measurement to nanoseconds, and the time taken is 34875 nanoseconds which is equal to 0.034875 milliseconds.

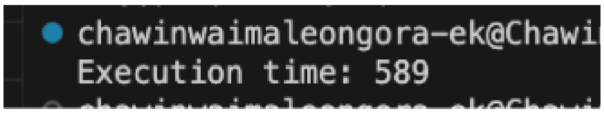


```
long start = System.nanoTime();
insertS(arry,n);
long end = System.nanoTime();
```



```
● chawinwaimaleongora-ek@Ch
  InsertS
  Execution time: 34875
```

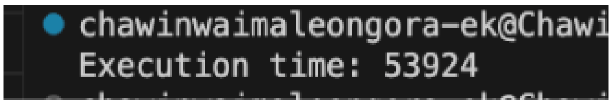
testdata-sort-2.txt:



```
● chawinwaimaleongora-ek@Chawi
  Execution time: 589
```

For the second data file, it took 589 milliseconds (0.589 seconds) to sort.

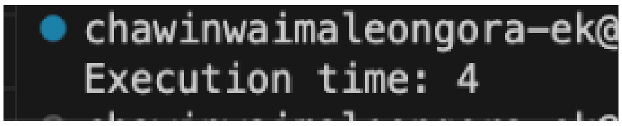
testdata-sort-3.txt:



```
● chawinwaimaleongora-ek@Chawi
  Execution time: 53924
```

For the third data file, it took 53924 milliseconds (about 0.8987 minutes) to sort.

testdata-sort-4.txt:



```
● chawinwaimaleongora-ek@
  Execution time: 4
```

For the fourth data file, it took 4 milliseconds to sort.

Performance discussion mong sorting algorithms:

For the selection sort algorithm, the execution time increases as the amount of data increases. As can be seen from the results, the testdata-sort-3.txt and testdata-sort-4.txt consumed way more time than the first two data. Since the algorithm is constructed with nested loops with the number of data "n" to be sorted, the time complexity, in this case, will be $O(n^2)$.

For the insertion sort algorithm, the execution time increases as the amount of data increases. However, there is one case that happened to have a unique outcome which will be discussed later. Despite that, it can be seen that the execution time is improved to be faster than the selection sort algorithm. The algorithm is also constructed with the nest-loop associating with the number of data "n" to be sorted which gives the (average and worst-case) time complexity of $O(n^2)$ as well. However, the case when the input data as an array is partially sorted before will make the inner loop to find the hole position to insert more easily as there is no need to compare every single data in the sorted subarray. This will lead to the best-case time complexity of $O(n)$ which occurs when the inner loop does not need to shift anything. As a result, the program only took approximately 0.89 minutes to sort the testdata-sort-3.txt while it took approximately 5.97 minutes for the selecting sort program to sort which is a huge difference in terms of efficiency. Now, considering the case where the testdata-sort-4.txt gives the strange outcome, even though the file contains 10^6 data, the execution time is significantly lower than that of testdata-sort-3.txt. The reason behind this is that most of the data are sorted, and only a few elements in the array are swapped. Thus, the time complexity will be $O(n)$ in the best case. Note that, the insertion sort algorithm is not suitable for sorting a large array since it has the average time complexity of $O(n^2)$ as mentioned before.