Assignment #4

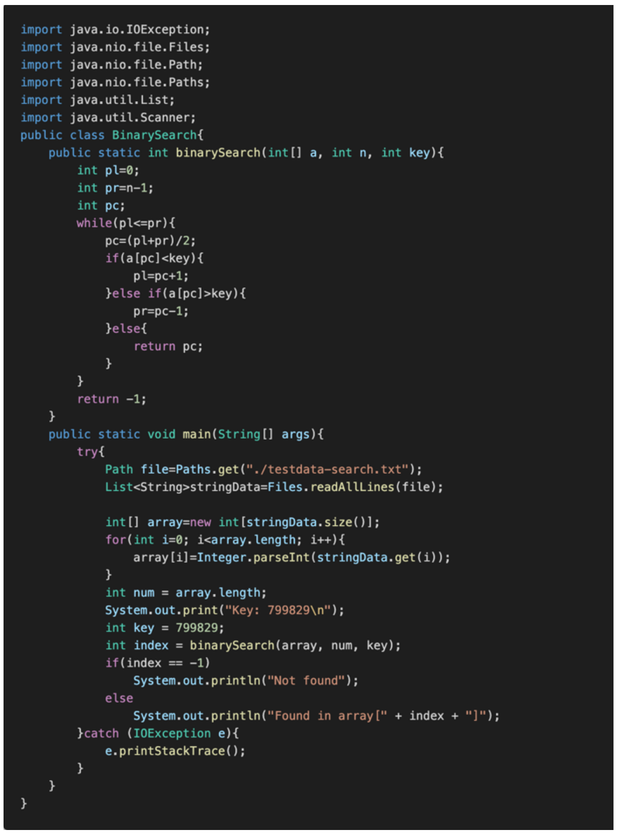
Rikiya Takehi
1W21CS04-6
Algorithms and Data Structures
02/05/2022

Problem Statement

Create binary search, selection sort, insertion sort programs using Java. Then, test the programs using the given files: testdata-search.txt, testdata-sort-1.txt, testdata-sort-2.txt, testdata-sort-3.txt, and testdata-sort-4.txt. For selection sort and insertion sort, measure the execution time as well.

Programs

Binary Search program

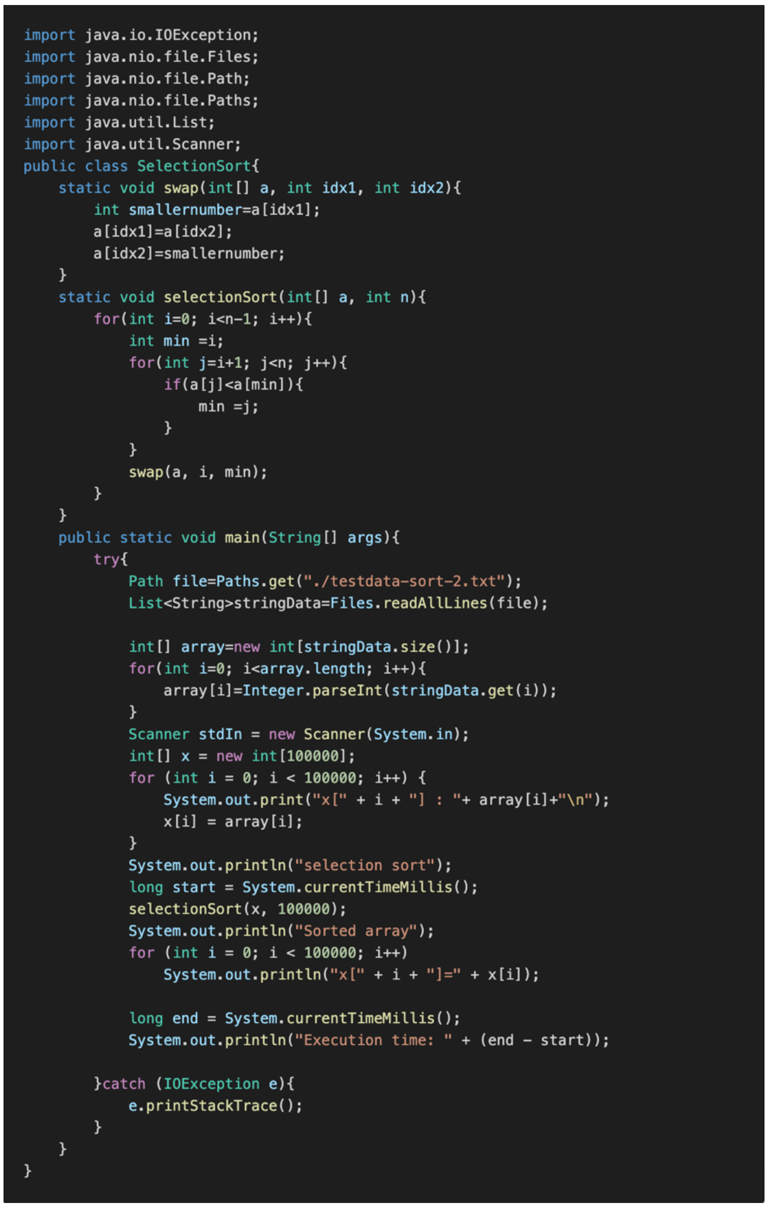


```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.Scanner;
public class BinarySearch{
    public static int binarySearch(int[] a, int n, int key){
        int pl=0;
        int pr=n-1;
        int pc;
        while(pl<=pr){
            pc=(pl+pr)/2;
            if(a[pc]<key){
                pl=pc+1;
            }else if(a[pc]>key){
                pr=pc-1;
            }else{
                return pc;
            }
        }
        return -1;
    }
    public static void main(String[] args){
        try{
            Path file=Paths.get("./testdata-search.txt");
            List<String>stringData=Files.readAllLines(file);

            int[] array=new int[stringData.size()];
            for(int i=0; i<array.length; i++){
                array[i]=Integer.parseInt(stringData.get(i));
            }
            int num = array.length;
            System.out.print("Key: 799829\n");
            int key = 799829;
            int index = binarySearch(array, num, key);
            if(index == -1)
                System.out.println("Not found");
            else
                System.out.println("Found in array[" + index + "]");
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Program Explanation:

Line 13 (pc=(pl+pr)/2;) means that they have searched for the center of the number sequence. If the number in the center is bigger than the key number, the program looks for the center of the first half, and repeats the procedure until the key number is found (lines 12~20). If the number in the center is smaller than the key number, the program looks for the center of the second half, and repeats the procedure until the key number is found. This way, the key number can often be found faster than the normal way, where the program searched the key from the start to end. In this code, "pl" is the lowest number to search. If the program finds out that the key number is in the latter hald, "pl" becomes "1+pc", where "pc" means the center(lines 14, 15). This way, "pc=(pl+pr)/2" will give the next center of the sequence (line 13).

Selection sort program

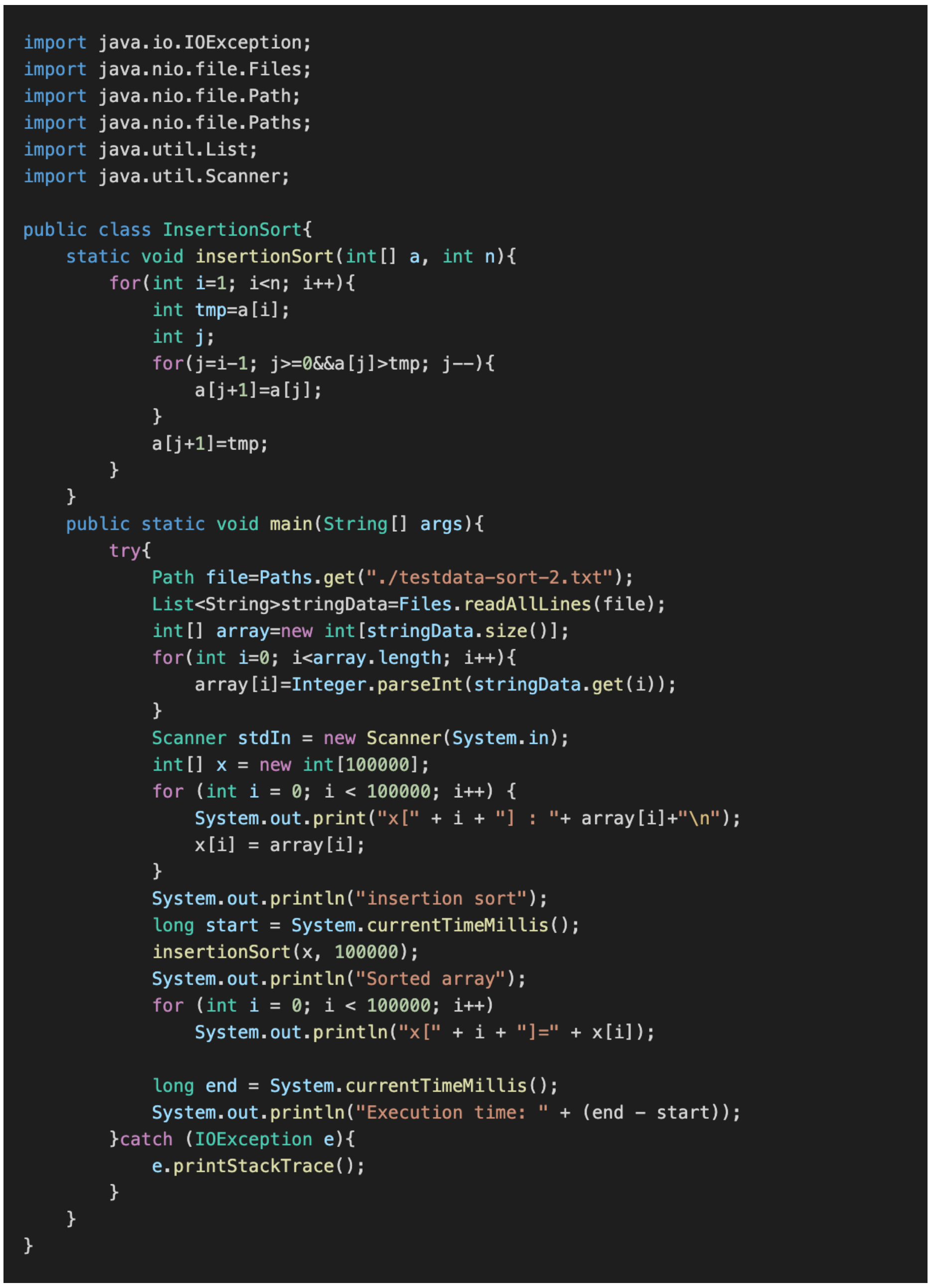


```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.Scanner;
public class SelectionSort{
    static void swap(int[] a, int idx1, int idx2){
        int smallernumber=a[idx1];
        a[idx1]=a[idx2];
        a[idx2]=smallernumber;
    }
    static void selectionSort(int[] a, int n){
        for(int i=0; i<n-1; i++){
            int min =i;
            for(int j=i+1; j<n; j++){
                if(a[j]<a[min]){
                    min =j;
                }
            }
            swap(a, i, min);
        }
    }
    public static void main(String[] args){
        try{
            Path file=Paths.get("./testdata-sort-2.txt");
            List<String>stringData=Files.readAllLines(file);

            int[] array=new int[stringData.size()];
            for(int i=0; i<array.length; i++){
                array[i]=Integer.parseInt(stringData.get(i));
            }
            Scanner stdIn = new Scanner(System.in);
            int[] x = new int[100000];
            for (int i = 0; i < 100000; i++) {
                System.out.print("x[" + i + "] : "+ array[i]+"\n");
                x[i] = array[i];
            }
            System.out.println("selection sort");
            long start = System.currentTimeMillis();
            selectionSort(x, 100000);
            System.out.println("Sorted array");
            for (int i = 0; i < 100000; 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:

This code makes the sequence in an increasing order. Selection sort is a program where the smallest element in the unsorted sequence is selected, and it gets swapped with the leftmost element in the unsorted sequence. In lines 8~12, a code that swaps index 1 with index 2 is written. In lines 13-23, the code searches the smallest number and the leftmost number in the unsorted sequence, and uses the swapping code previously mentioned (line 21). In line 40 and 46, the program counts the current time, and in line 47, it gives the time difference that took to sort the sequence in order.

Insertion sort program



```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 InsertionSort{
    static void insertionSort(int[] a, int n){
        for(int i=1; i<n; i++){
            int tmp=a[i];
            int j;
            for(j=i-1; j>=0&&a[j]>tmp; j--){
                a[j+1]=a[j];
            }
            a[j+1]=tmp;
        }
    }
    public static void main(String[] args){
        try{
            Path file=Paths.get("./testdata-sort-2.txt");
            List<String>stringData=Files.readAllLines(file);
            int[] array=new int[stringData.size()];
            for(int i=0; i<array.length; i++){
                array[i]=Integer.parseInt(stringData.get(i));
            }
            Scanner stdIn = new Scanner(System.in);
            int[] x = new int[100000];
            for (int i = 0; i < 100000; i++) {
                System.out.print("x[" + i + "] : "+ array[i]+"\n");
                x[i] = array[i];
            }
            System.out.println("insertion sort");
            long start = System.currentTimeMillis();
            insertionSort(x, 100000);
            System.out.println("Sorted array");
            for (int i = 0; i < 100000; 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

The insertion sort searches the numbers from the left side, and whenever the number on the right side is smaller than the number on the left side, it swaps. After the swap, if the left-hand side of the number is bigger than the number, it swaps again. This sequence is repeated until the numbers are in numerical order. In line 11, “tmp” is the leftmost number of the unsorted sequence. Lines 13~15 compares all the elements in the sorted part and swaps all numbers that are greater than “tmp”. In line 40 and 46, the program counts the current time, and in line 47, it gives the time difference that took to sort the sequence in order.

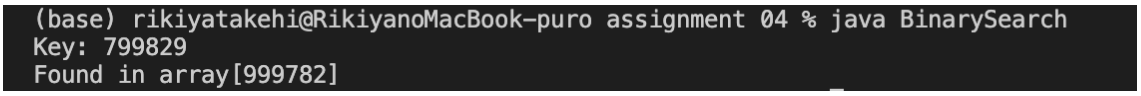
Results

Binary Search

Experimental Settings:

The program aims to search a number in an array, and show the number of the array. The program searches the number from the data "testdata-search.txt". From this array, the task was to look for the number 799,829.

Results:



```
(base) rikiyatakehi@RikiyanoMacBook-puro assignment 04 % java BinarySearch
Key: 799829
Found in array[999782]
```

As shown in the results, the number 799829 was found in the array 999782.

Discussion of the results:

Since 999782>799829, it would show that until the number 799829, some numbers should have repeated itself. Therefore, there could be more than one 799829 in the array, but the first one that appeared was on the 999782th array.

Selection Sort & Insertion Sort

Experimental settings:

The program aims to order the given sequence from smallest to greatest using the selection/insertion sorting method. The program reorders the sequence in the data files: testdata-sort-1.txt (100 data), testdata-sort-2.txt (100000 data), testdata-sort-3.txt (1000000 data), and testdata-sort-4.txt (1000000 data). The task was also to get the execution time taken to order the sequence in numerical order using "public static long System.currentTimeMillis()".

Results:

Selection Sort

testdata-sort-1.txt:

The sorted array was from 1~79. The execution time was 2 milliseconds.

testdata-sort-2.txt:

The sorted array was from 1~79999. The execution time was 7437 milliseconds.

testdata-sort-3.txt:

The sorted array was from 1~799999. The execution time was 524848 milliseconds.

Testdata-sort-4.txt:
The sorted array was from 1~799999. The execution time was 558898 milliseconds.

Insertion Sort
Testdata-sort-1.txt:
The sorted array was from 1~79. The execution time was 5 milliseconds.

testdata-sort-2.txt:
The sorted array was from 1~79999. The execution time was 2897 milliseconds.

testdata-sort-3.txt:
The sorted array was from 1~799999. The execution time was 121389 milliseconds.

Testdata-sort-4.txt:
The sorted array was from 1~799999. The execution time was 23716 milliseconds.

Discussion of the Results:

The test results of the selection sort and the insertion sort were probably identical. However, for the testdata-sort-1.txt, the selection sort was faster by three milliseconds. For the testdata-sort-2.txt, the selection sort was slower than the insertion sort by 4540 seconds. For the testdata-sort-3.txt, the selection sort was slower than the insertion sort by 403,459 milliseconds. For the testdata-sort-4.txt, the selection sort was slower than the insertion sort by 535,182 milliseconds. This implies that for three of the test data, insertion sorting is the better option.

This is probably because the number of steps needed to order the sequence is very different. For the insertion sort, it only needs to scan "one" element that would fit the next element (k+1st element). On the other hand, in order to search for the next element, the program needs to scan all remaining elements in the unsorted sequence.

It could become very fast to combine binary search with insertion sort. The smallest element in the unsorted sequence can be put into the sorted sequence by using binary search. This could reduce the number of steps needed to complete insertion sort.