

Algorithm and Data Structures

Assignment 12: Report

Name: Maeda Taishin

Student ID: 1W21CF15

Date: 04/07/2023

Ex. 1: Quick sort

- Write quick sort algorithm in Java

- Assumptions

- Integer values are initialized by using data from the standard input
 - use java.util.Scanner

Ex1: Quick Sort Algorithm

```
1  import java.io.IOException;
2  import java.nio.file.Files;
3  import java.nio.file.Path;
4  import java.nio.file.Paths;
5  import java.util.List;
6  public class Qsort {
7      Run | Debug
8      public static void main(String[] args){
9          try{
10             Path file = Paths.get (first:"testdata-sort-4.txt");
11             List<String> stringData = Files.readAllLines(file);
12
13             int[] array = new int[stringData.size()];
14             for (int i = 0; i<array.length; i++){
15                 array[i] = Integer.parseInt(stringData.get(i));
16             }
17             int n = array.length;
18             long start = System.nanoTime();
19             quick(array, left:0, n - 1);
20             long end = System.nanoTime();
21             System.out.println(x:"Sorted Array: ");
22             printArray(array);
23             System.out.println("Execution time: " + (end - start)/1e6 + " milliseconds.");
24         }catch (IOException e){
25             e.printStackTrace();
26         }
27     }
```

```

28     private static void quick(int[] array, int left, int right){
29         int pl = left;
30         int pr = right;
31         int pivot = array[(pl+pr)/2];
32         while(pl <= pr){
33             while (array[pl] < pivot){
34                 pl++;
35             }
36             while(array[pr] > pivot){
37                 pr--;
38             }
39             if(pl <= pr){
40                 swap(array, pl, pr);
41                 pl++;
42                 pr--;
43             }
44         }
45         if(left < pr){
46             quick(array, left, pr);
47         }
48         if(right > pl){
49             quick(array, pl, right);
50         }
51     }
52     private static void swap(int[] array, int x, int y){
53         int temp = array[x];
54         array[x] = array[y];
55         array[y] = temp;
56     }
57     private static void printArray(int[] array){
58         for(int i=0; i<array.length; i++){
59             System.out.println("x[" + i + "]= " + array[i]);
60         }
61         System.out.println();
62     }
63 }

```

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-4.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 strings.

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 quick 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 “quick” is set and called by passing the array “array”, an integer “0” indicating the leftmost element, and the array’s length “n-1” to the function to perform the quick sorting algorithm.

lines: 20-21 and 57-62

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: 28-44)

Line 29-31:

Three variables are initialized namely “pl”, “pr”, and “pivot” which represent left (the first element to be sorted), right (the last element to be sorted), and pivot as $\text{array}[(\text{pl}+\text{pr})/2]$ (the element at the middle between “pl” and “pr”) respectively.

Line 32:

A while-loop is defined to keep track of “pl” and “pr” as long as the value of “pl” is less than or equal to “pr”.

Lines 33-35:

A while-loop is again defined inside in which while the value of an element in the position pl in array “array” is less than the value of pivot, the value of “pl” will be incremented by 1.

Lines 36-38:

Another while-loop is again defined inside the one created in line 33 in which while the value of an element in the position “pr”th in array “array” is greater than the pivot, the value of “pr” is decremented by 1.

Lines 39-43:

At this point, the value of an element in the position “pl”th will be greater than or equal to “pivot” and that of in the position “pr”th which can be described as: $\text{array}[\text{pl}] \geq \text{pivot} \geq \text{array}[\text{pr}]$. Thus, an if-else statement is created to check the condition of “pl” and “pr” once again. If “pl” is less than or equal to “pr”, the condition is satisfied. The contents at position “pl”th and “pr”th will be swapped. This will give $\text{array}[\text{pl}] \leq \text{pivot} \leq \text{array}[\text{pr}]$ indicating the sorting. Then, the value of “pl” will be incremented by 1, and “pr” will be decremented by 1.

Lines 45-46:

Now, “pl” is greater than “pr” with either of two conditions. First, all of the elements in the position before “pl”th are not greater than “pivot”. Second, all of the elements in the position after “pr”th are not less than “pivot”. These will indicate that the array can be classified into 2 sub-arrays. One is the array that contains elements from position “left”th to “pr”th inclusively. Another is the array that contains elements from position “pl”th to “right”th. Both sub-arrays are again sorted recursively in the “quick” function as long as the size of the array is larger than 1. In other words, $\text{left} < \text{pr}$, $\text{right} > \text{pl}$).

Lines 52-56 (Swapping function):

After “array”, “pl” and “pr” are passed into this function, 3 parameters are created including array “array”, “x” representing pl, and “y” representing pr. Later, a temporary variable “temp” is created and is assigned to array[x], the array[x] is assigned to array[y], and array[y] is assigned to “temp”. This will swap those two elements to a sorted position.

Execution Results and Discussion:

Test-data-sort1.txt	Test-data-sort2.txt	Test-data-sort3.txt	Test-data-sort4.txt
0.022083	26.26575	81.934291	31.7045

Table1. Execution time in milliseconds of the quick sort algorithm from 4 different datasets.

As can be seen from Table 1, the time for the program to execute the results increases as the number of data increases. However, the execution time spent in the test-data-sort4 is less than that of the test-data-sort3. This is due to the fact that the data is already partially sorted in the test-data-sort4 which is not the case for test-data-sort3.

Discussing the time complexity, the quick sort algorithm will have the best-case time complexity when the partition reduces the size of the sub-array by half. Meaning that the middle element or close to the middle element is selected as the pivot. n will be the total size of the sub-array from when it is partitioned in the same level. Since there are a total of $\log n$ levels, the best-case time complexity will be $O(n \log n)$. For the average-case time complexity, if the array is arranged in disorder sequence without having increasing or decreasing properties, the complexity will be the same as the best-case which is $O(n \log n)$. For the worst-case time complexity, it will happen when the partitioning algorithm selects the largest or smallest element as the pivot every single time. Since the nested while-loops will perform only the constant time tasks in which each of the tasks is linearly associated with the range of the array's position, every time that the recursion is called, the time complexity will be increased by the size of the sub-arrays. Hence, the worst-case time complexity will be $O(n^2)$.

Algorithm	Average Time Complexity	Test-data-sort 1.txt	Test-data-sort 2.txt	Test-data-sort 3.txt	Test-data-sort 4.txt
Selection Sort	$O(n^2)$	0.05765	3553.645	361052	359641
Insertion sort	$O(n^2)$	0.03601	560.2367	55623.195	3.911
Shell sort	$O(n^2)$	0.0223	16.504	155.055	18.075
Bubble sort	$O(n^2)$	0.1525	12868.74	1347988.5	32.48
Quick sort	$O(n \log n)$	0.022083	26.26575	81.934291	31.7045

Table 2. The comparison of execution time in milliseconds among different sorting algorithms.

As can be seen from Table 2, the execution time will be extremely close to each other in every sorting algorithm when it comes to a small size of data (test-data-sort1.txt). Every sorting algorithm executes under 1 milliseconds. The execution time starts to be significantly different when the amount of data is getting larger and larger. It can be clearly observed that the algorithms having the time complexity of $O(n^2)$ including Selection, Insertion, and Bubble sort took more time to sort a large size of array than that of $O(n \log n)$ including Shell, and Quick sort. Note that, the execution results for test-data-sort4.txt gives a unique outcome. Due to the fact that the dataset is partially sorted initially, algorithms that have the best-case time complexity of $O(n)$ including Insertion and Bubble sort can execute slightly faster than the others. Therefore, the different types of sorting algorithms can vary depending on the dataset's conditions to get an optimum outcome.

Ex. 2: Binary search by recursion in Java

- Write binary search algorithm **by recursion** in Java
 - avoid to use loops in the algorithm
 - instead use recursion
- 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

Ex2: Binary Search by Recursion

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


Code Explanation:Binary Search by recursion function (Lines 8-20:)Line 8:

The function is created including 4 parameters which are the array “array”, the left pointer integer “pl”, the right pointer integer “pr”, and a key integer “key”.

Line 9:

A if-else statement is defined to check the condition such that if “pl” is less than or equal to “pr”, the condition is satisfied.

Line 10:

Inside the statement, the variable “pc” is defined to be the middle pointer between “pl” and “pr” $((pl+pr)/2)$.

Line 11-17:

Another if-else statement is defined inside to check if the element in the “pc”th position of the array is less than “key”, the program will recursively apply a Binary search to the right partition by returning “BiSRe(array, pc+1, pr, key)” in which each variable represents an array, the next position to the middle pointer, “pr”, and “key” respectively. Else if the element in the “pc”th position of the array is greater than “key”, the program will recursively apply a Binary search to the left partition by returning “BiSRe(array, pl, pc-1, key)” in which each variable represents an array, “pl”, the position before the middle pointer, and “key” respectively. Otherwise, the program will return “pc”.

Line 18:

With respect to the first if-else statement defined in line 9, if the condition does not satisfy, the program will return -1 indicating that the prompted element is not founded in this array.

The main function (Lines 21-45):Line 22:

Scanner “sc” is initialized.

Line 24:

Initially, the main function will be described. Talking 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 25:

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

Lines 27-30

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: 1-6). 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: 23 and 42-43).

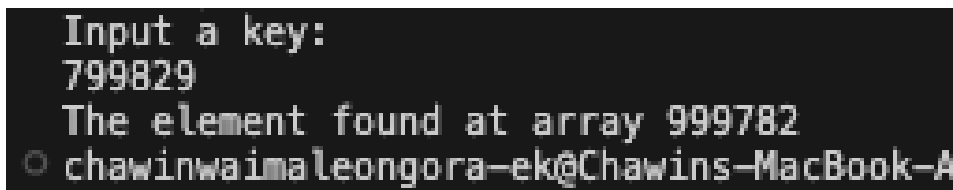
Lines 31-35:

After that, the program is set to ask the user to input a key. A variable called “key” is defined 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 35 of this program, I create a variable called “result” to call the searching function named “BiSRe” representing the Binary search by recursion, passing the array “arry”, 0, the array's size “n” minus 1, and the input “key” (The function will be discussed later).

Lines 37-41:

An if-else statement is defined 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.

Execution Result and Discussion:

A terminal window with a black background and white text. The text shows the program's execution: it prompts for a key, receives the input '799829', and then outputs 'The element found at array 999782'. The prompt character is a circle.

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

When the user prompts the key as 799829, the program will execute and says the element is found in the array at the 999782nd position in the testdata-search.txt. This gives the same result as what I did in assignment 4 as the searching can only be used when the array is sorted, and the time complexity is $O(\log n)$.

Ex. 3: Brute Force Algorithm for String Search

- Write brute force algorithm for string search in Java
 - Assumptions
 - target text and pattern are obtained from the standard input

Ex3: Brute Force Algorithm for String Search

```
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 bfSearch {
8      public static int bfS(String text, String pattern){
9          int pt = 0;
10         int pp = 0;
11
12         while(pt != text.length() && pp != pattern.length()){
13             if(text.charAt(pt) == pattern.charAt(pp)){
14                 pt++;
15                 pp++;
16             }else{
17                 pt -= pp - 1;
18                 pp = 0;
19             }
20         }
21         if(pp == pattern.length()){
22             return pt - pp;
23         }else{
24             return -1;
25         }
26     }
```

```

27 public static void main(String[] args){
28     Scanner sc = new Scanner(System.in);
29     try{
30         Path file = Paths.get (first:"testdata-stringsearch.txt");
31         List<String> stringData = Files.readAllLines(file);
32
33         System.out.println(x:"Pattern: ");
34         String pattern = sc.nextLine();
35         sc.close();
36         int line = -1;
37         int result = -1;
38         for(int i=0; i<stringData.size(); i++){
39             int temp = bfS(stringData.get(i), pattern);
40             if(temp != -1){
41                 line = i;
42                 result = temp;
43                 break;
44             }
45         }
46
47         if(result == -1){
48             System.out.println(x:"There is no pattern found!");
49         }else{
50             int len = result + pattern.length();
51             System.out.println("Matched at "+ (result + 1) + " in line " + (line + 1));
52             System.out.println(stringData.get(line));
53             System.out.printf(String.format(format:"%ds\n", len), pattern);
54         }
55     }catch (IOException e){
56         e.printStackTrace();
57     }
58 }
59 }

```

Code Explanation:

Brutal force string search function (Lines 8-25:)

Line 8:

Two input parameters are defined as string type including “text” and “pattern”.

Lines 9-10:

Variable “pt” and “pp” are set equal to zero which represents the text pointer and pattern pointer respectively.

Line 12:

A while-loop is defined, and the program will keep checking every single character of “text” and “pattern” as long as “pt” and “pp” stay within the range.

Lines 13-19:

An if-else statement is defined to check if the character at the position “pt” in “text” is equal to that of “pp” in “pattern”. If so, then “pt” and “pp” will be incremented by one meaning that both of them will be moved to the next position. Otherwise, the value of “pt” will be set to the initial position prepared to be used in the next iteration, and “pp” will be reset to 0.

Lines 21-25:

Outside the while-loop, another if-else statement is defined to check if “pp” is the length of “pattern” which is the reason to stop the iteration, the program will return “pt” minus “pp” which is the first position of “text” where “pattern” is found. Otherwise, the program will return -1 indicating that the program could not find the pattern.

The main function (Lines 27-58):

Line 28:

Scanner “sc” is initialized.

Line 30:

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

Line 31:

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

Line 33:

The program will ask the user to prompt a pattern.

Lines 34-35:

A variable called “pattern” is defined to store the string input from the users, which will be used later when calling a searching function. Then, the scanner “sc” is closed.

Lines 36-37:

Variables “line” and “result” are set equal to -1.

Line 38:

A for-loop is defined to search for the pattern iterating from “i” = 0 to “i” = stringData.size() - 1.

Line 39:

Inside the for-loop, a variable temp is set to call a function “bfS” which is the brutal force string search method. The aim is to find the “pattern” in the content of the data at position “i”th of “stringData” and store the return value int integer “temp”.

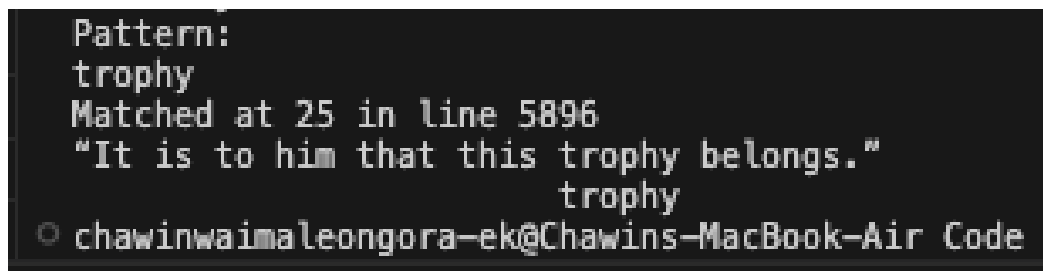
Lines 40-45:

An if-else statement is defined inside the for-loop to check the condition whether “temp” is not equal to -1. If it is not equal to -1, then the program will assign “i” equal to “line” and assign “temp” equal to “result”. Otherwise, the program will exit the loop.

Lines 47-54:

An if-else statement is defined to check whether the condition for finding the pattern holds, based on the value of “result” returned from the searching function. For instance, if the value of “result” is -1, indicating that the pattern is not found in the array, the program will execute “There is no pattern found”. On the other hand, if the value of “result” is not -1, then a variable int “len” is defined to be the total length of characters from the initial of “line” to the last character of “pattern”. Then, the program is set to print “result” plus one indicating the position and “line” plus one indicating the line number. Next, the program will print the line (statement) that contains the “pattern”. And then, in the following line, the program will print out the “pattern” with the string format of length “len”.

Execution Result:

A terminal window with a dark background and light-colored text. The output shows the pattern 'trophy' being found at position 25 in line 5896. The corresponding line of text is displayed, with the pattern highlighted. The terminal prompt shows the user is 'chawinwaimaleongora-ek' on a 'Chawins-MacBook-Air' using 'Code' as the editor.

```
Pattern:
trophy
Matched at 25 in line 5896
"It is to him that this trophy belongs."
      trophy
chawinwaimaleongora-ek@Chawins-MacBook-Air Code $
```

As can be seen, the pattern “trophy” is found at position 25th in line 5896. Although the testdata-stringsearch.txt contains a huge amount of text and characters, the program will search for the pattern’s position and its line number that is found first. This is due to the code that the “pattern” is searched from left to right and from top to bottom of the text data.