Assignment #10

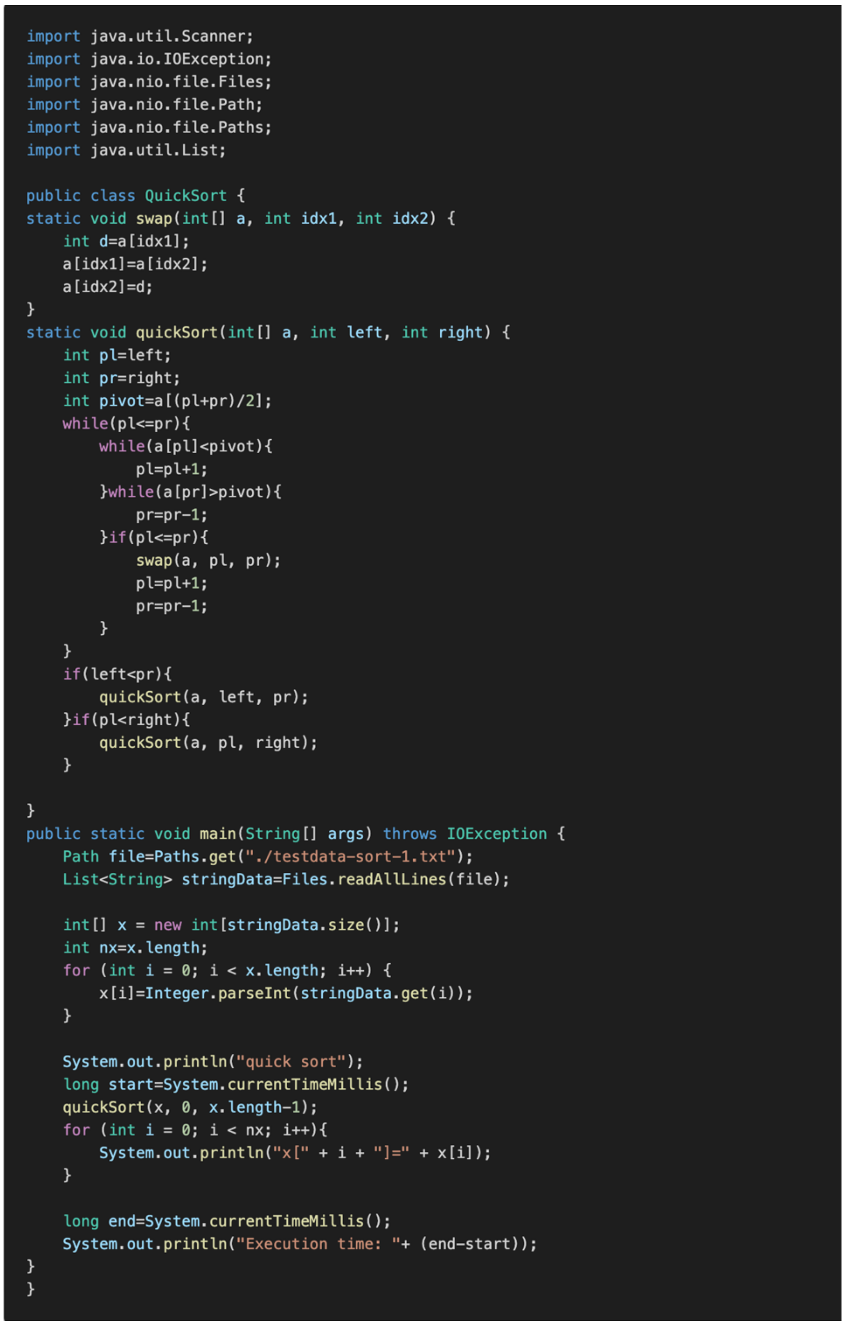
Rikiya Takehi
1W21CS04-6
Algorithms and Data Structures
04/07/2022

Problem Statement

Write a quick sort program, binary search program, and brute force string search program. For the quick sort program, use testdata-sort-{1~4}.txt and measure its execution time for each of the test data. For the binary search program, testdata-search.txt is given and key 799,829 is searched. For brute force string search program, testdata-tringsearch.txt is used, and the pattern "trophy" is searched.

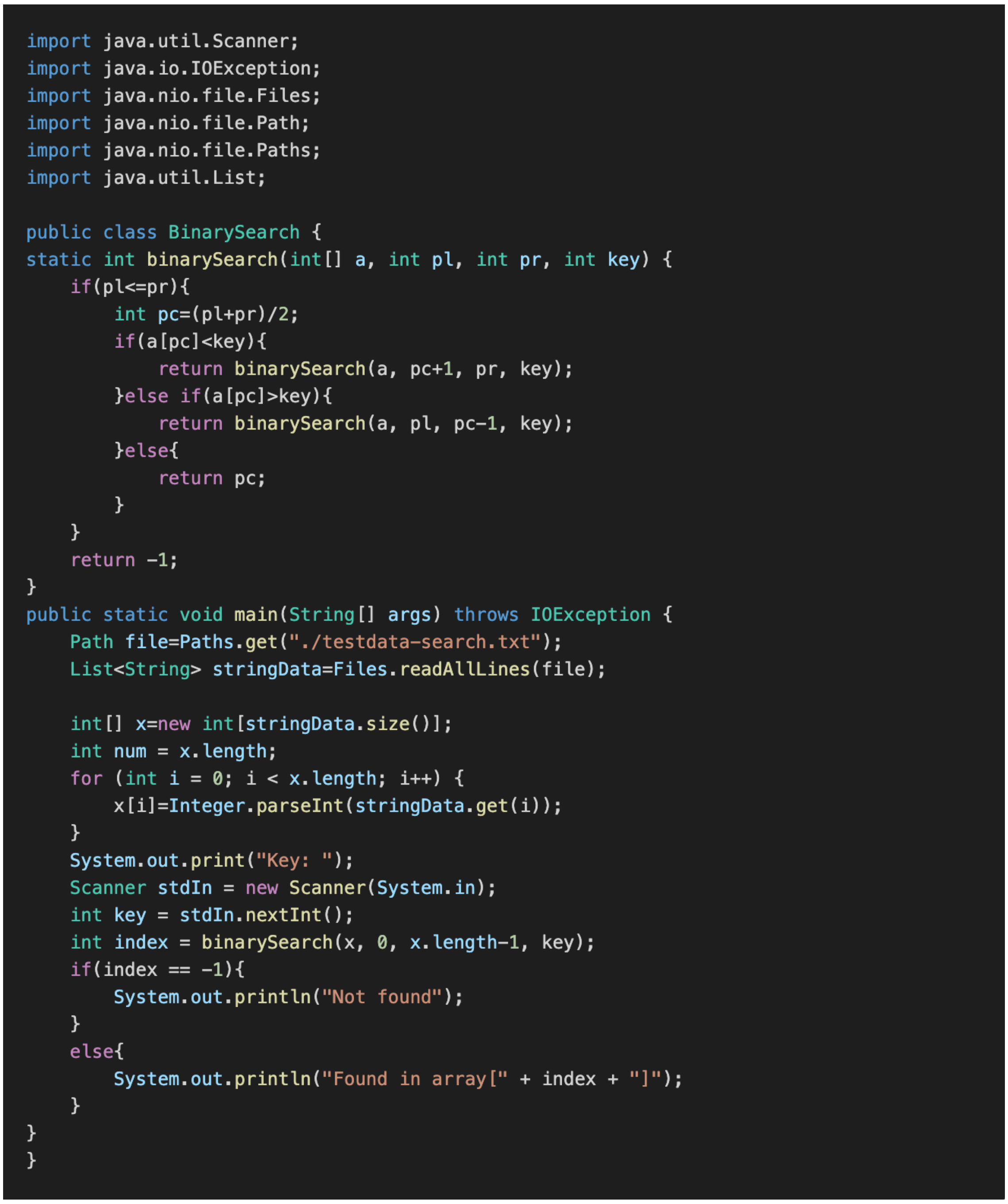
Programs

quick sort program:



```java
import java.util.Scanner;
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 QuickSort {
static void swap(int[] a, int idx1, int idx2) {
    int d=a[idx1];
    a[idx1]=a[idx2];
    a[idx2]=d;
}
static void quickSort(int[] a, int left, int right) {
    int pl=left;
    int pr=right;
    int pivot=a[(pl+pr)/2];
    while(pl<=pr){
        while(a[pl]<pivot){
            pl=pl+1;
        }while(a[pr]>pivot){
            pr=pr-1;
        }if(pl<=pr){
            swap(a, pl, pr);
            pl=pl+1;
            pr=pr-1;
        }
    }
    if(left<pr){
        quickSort(a, left, pr);
    }if(pl<right){
        quickSort(a, pl, right);
    }

}
public static void main(String[] args) throws IOException {
    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("quick sort");
    long start=System.currentTimeMillis();
    quickSort(x, 0, x.length-1);
    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));
}
}
```

binary search program:



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

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

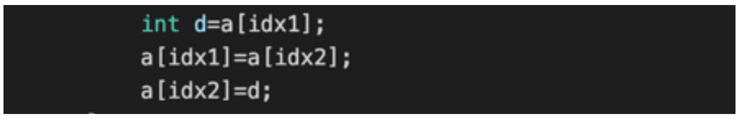
brute force string search program:

```java
import java.util.Scanner;
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 BruteForceStringSearch {
static int bfSearch (String txt, String pattern) {
    int pp=0;
    int pt=0;
    while(pt!=txt.length()&&pp!=pattern.length()){
        if(txt.charAt(pt)==pattern.charAt(pp)){
            pt++;
            pp++;
        }else{
            pt=pt-pp+1;
            pp=0;
        }
    }
    if(pp==pattern.length()){
        return pt-pp;
    }
    return -1;
}
public static void main(String[] args) throws IOException {
    Path file=Paths.get("./testdata-stringsearch.txt");
    List<String> stringData=Files.readAllLines(file);
    String[] array=new String[stringData.size()];
    for (int i = 0; i < array.length; i++){
        array[i]=stringData.get(i);
    }

    int num=array.length;
    System.out.print("pattern:");
    String str = stdIn.nextLine();
    int index=-1;

    for(int i=0; i < array.length; i++){
        index=bfSearch(array[i], str);
        if(index!=-1){
            System.out.println("Found at index"+(index+1)+"in line "+(i+1));
            break;
        }
    }

    if(index==-1){
        System.out.println("Not found");
    }

    /*
    if (idx == -1){
        System.out.println("There is no pattern in text.");
    }
    else {
        int len = 0;
        for (int i = 0; i < idx; i++)
            len += s1.substring(i, i + 1).getBytes().length;
        len += s2.length();
        System.out.println("match at " + (idx + 1));
        System.out.println(s1);
        System.out.printf(String.format("%%%ds¥n", len), s2);
    }
    */
}
}
```

Program explanation

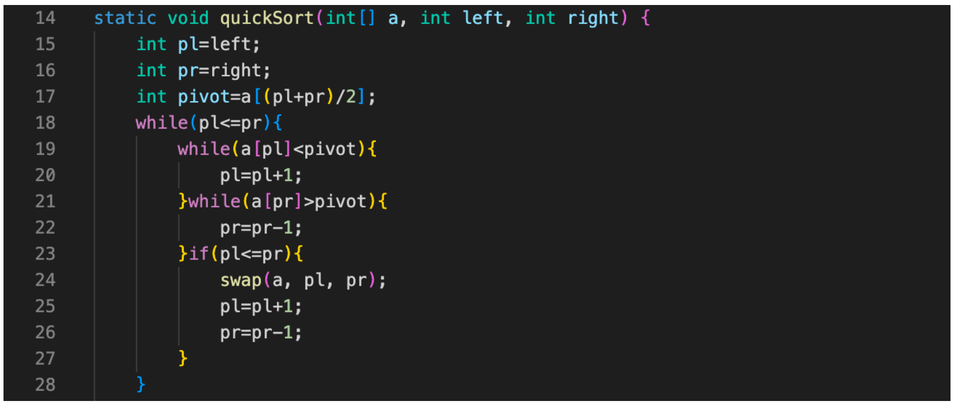
Quick Sort Program

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.



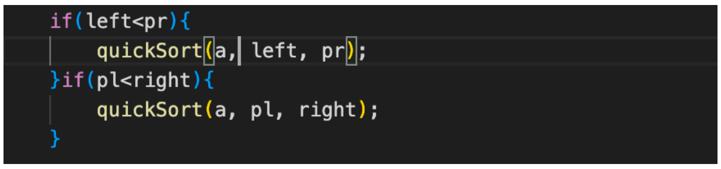
```
        int d=a[idx1];
        a[idx1]=a[idx2];
        a[idx2]=d;
```

A quick sort array makes two arrays and calls itself in a recursive way to sort the two subarrays. In lines 14-28, it identifies the pivot in line 17, and divides it into the left section (pl) and the right section (pr). While the left section is smaller than the right section, the following procedures are done. If the number on the array is smaller than the pivot value, it adds 1 to pl, and if the number is bigger than the pivot value it subtracts 1 from pr. This way, it makes two subarrays.



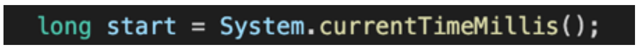
```
14    static void quickSort(int[] a, int left, int right) {
15        int pl=left;
16        int pr=right;
17        int pivot=a[(pl+pr)/2];
18        while(pl<=pr){
19            while(a[pl]<pivot){
20                pl=pl+1;
21            }while(a[pr]>pivot){
22                pr=pr-1;
23            }if(pl<=pr){
24                swap(a, pl, pr);
25                pl=pl+1;
26                pr=pr-1;
27            }
28        }
```

Then, in lines 29-33, it says to quick sort the two subarrays recursively until it is completely sorted.



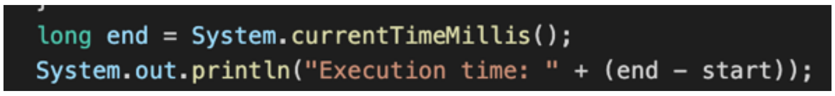
```
    if(left<pr){
        quickSort(a, left, pr);
    }if(pl<right){
        quickSort(a, pl, right);
    }
```

In line 47, it records the exact time using the following code and states "start".



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

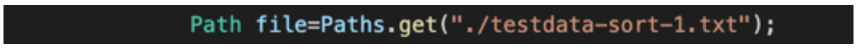
In line 53, it records the time again using the following code and states "end". Then in line 54, it shows the time taken to complete the sorting by subtracting the time recorded as "end"



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

by time recorded at "start".

When switching the data and the sorting method, change the names in line 37.

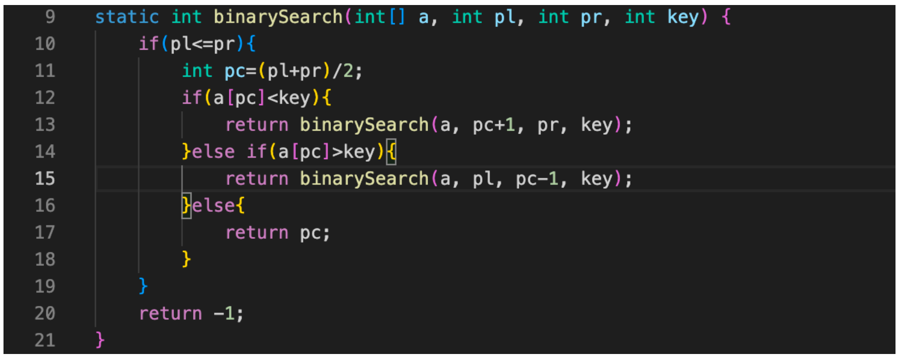


```
Path file=Paths.get("./testdata-sort-1.txt");
```

<u>Binary Search Program:</u>

In this program, the binary search was done using recursion.

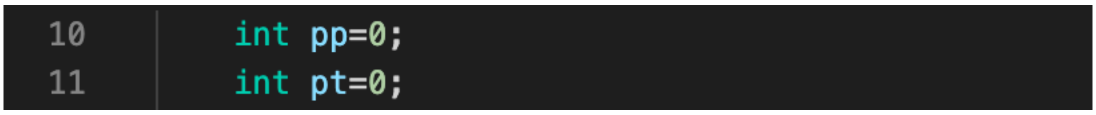
In lines 9 to 21, it shows the step in which the next actions are decided. In line 11, it identifies the center of the array. In line 12 and 13, if the key is bigger than the center number, it returns to do the binary search again but with the latter half of the array. In lines 14 and 15, if the key is smaller than the center number, it returns to do the binary search again but with the first half of the array. If the number is the center, it returns the center number as the key (line 16 and 17). If the procedure is repeated until the end and number is not found, it returns "-1" which shows that it was unsuccessful.



```
 9  static int binarySearch(int[] a, int pl, int pr, int key) {
10      if(pl<=pr){
11          int pc=(pl+pr)/2;
12          if(a[pc]<key){
13              return binarySearch(a, pc+1, pr, key);
14          }else if(a[pc]>key){
15              return binarySearch(a, pl, pc-1, key);
16          }else{
17              return pc;
18          }
19      }
20      return -1;
21  }
```

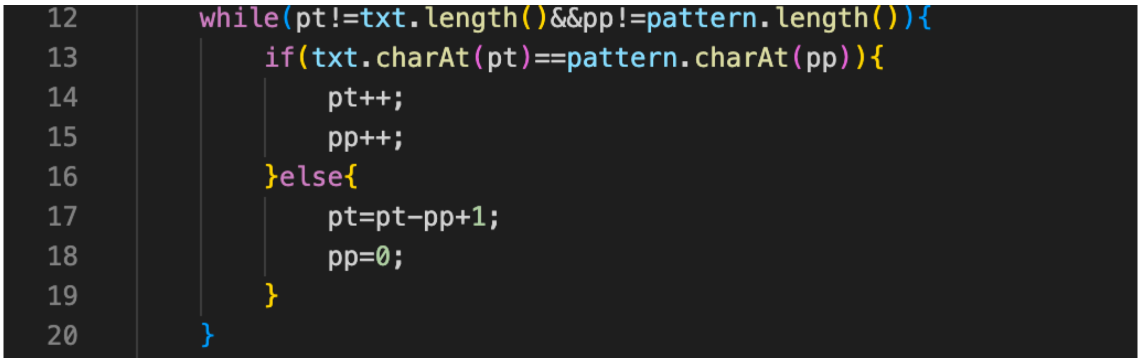
<u>brute force string search program:</u>

Brute force string search program is a string program that searches each of the characters one by one until it has a mismatch. In lines 10, 11, it names pp and pt each for the cursor of txt (string array) and the cursor of pp (pattern).



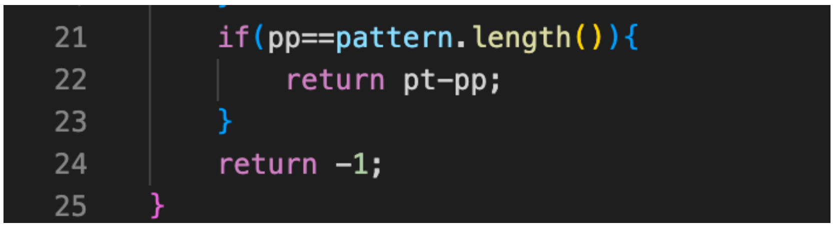
```
10      int pp=0;
11      int pt=0;
```

If the current character of txt is the current character of the pattern, if compares the next character of the pattern and the text is compared (lines 13-15). If the characters are not the same, it shifts the pattern down the text by one character (lines 16-18).



```
12      while(pt!=txt.length()&&pp!=pattern.length()){
13          if(txt.charAt(pt)==pattern.charAt(pp)){
14              pt++;
15              pp++;
16          }else{
17              pt=pt-pp+1;
18              pp=0;
19          }
20      }
```

If pp is the length of the pattern, it shows that the entire length is found, so it returns pt-pp (the place it was found) and ends the procedure (lines 21-25).



```
21      if(pp==pattern.length()){
22          return pt-pp;
23      }
24      return -1;
25  }
```

Experimental settings

Input files:

“testdata-sort-1.txt”, “testdata-sort-2.txt”, “testdata-sort-3.txt”, “testdata-sort-4.txt” for quick sort program.

“testdata-search.txt” for binary search program.

“testdata-stringsearch.txt” for brute force string search program.

Tested programs:

“QuickSort”, “BinarySearch”, “BruteForceStringSearchProgram”

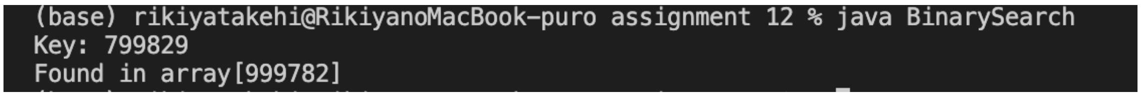
Results

Quick Sort Program:

Table showing the time taken for quick sort program to sort each test data in milliseconds compared with other sorting methods

	testdata-sort-1.txt	testdata-sort-2.txt	testdata-sort-3.txt	testdata-sort-4.txt
Quick sort	21	543	2089	2451
Bubble sort	28	16096	6350	
Bubble sort 1	21	17052	496	
Bubble sort 2	23	16682	384	
Shell sort	2	239	1764	1943
Insertion sort	2	1365	98083	1420
Selection Sort	2	7437	524848	558898

Binary Search Program:



```
(base) rikiyatakehi@RikiyanoMacBook-puro assignment 12 % java BinarySearch
Key: 799829
Found in array[999782]
```

Brute Force String Search Program:

The pattern trophy was found at index 25 in line 5896

Discussion of the Results

Quick Sort Program:

When comparing the quick sort program with others, it can be seen that it is fairly fast overall. The execution time is fairly slow for the test data sort 1, but are fairly fast for the rest of the test data. When compared with shell sort, it can be seen that the time taken for quick sort is slower by a little for every test data. This would show that the it takes time for quick sort to deal with the test data 1 because no matter the size of the test data, it is necessary to find the pivot multiple times. On the other hand, it is faster than many others in long arrays because it does not have to compare every single value in the array.