Name: Georges Hatem

CS 590 Assignment Homework 1: Algorithm Analysis Application Exercises

Due Date: January 30, 2022

Problem 1.6.70

The code below is a code that I have written in Java (I also attached the code as a 2nd document aside from the PDF File that I attached for the assignment). Below is a description of the code and runtime of the Algorithm:

```java
/* Name: Georges Hatem
 *
 * Assignment: CS590 Hw1 Analysis Application Exercises
 *
 * Description: An efficient algorith for reversing an Array A. The running time algorith is O(n).
 *
 * Due Date: January 30, 2022
 *
 */

public class hw1_G_Hatem {

    public static void main(String[] args) {

        int [] A = {3, 4, 1, 5};
        int temp;

        for(int i = 0; i < (A.length/2); i++) {

            temp = A[i];
            A[i] = A[(A.length - 1) - i];
            A[(A.length - 1) - i] = temp;

        }


        for(int i =0; i < A.length; i++) {

            System.out.print("" + A[i]);

        }

        System.out.println("");

    }

}

```

Pseudocode:

Although we are not required to write a Pseudocode, I thought that the best way to explain the runtime of the Algorithm is to write the Pseudocode.

The Pseudocode of the Algorithm in the code above is as follows:

Algorithm invertArray(A, n):
Input: An n-element array A
Output: An n-element array A (which becomes inverted)

$\boldsymbol{for\ i \leftarrow 0\ to\ (\frac{n}{2} - 1)}$ **do**

$\quad \boldsymbol{temp \leftarrow A[i]}$

$\quad \boldsymbol{A[i] \leftarrow A[n-1-i]}$

$\quad \boldsymbol{A[n-1-i] \leftarrow temp}$

$\boldsymbol{for\ i \leftarrow 0\ to\ (n-1) do}$

return A[i]

First off, we are assigning the value 0 to i. So, this is considered 1 primitive operation. Then, we are looping until i < n/2. Therefore, the condition part of the loop has (n/2) primitive operations. The for loop will loop (n/2 -1) times. Temp is assigned to A[i] inside of the for loop. This is a 2 primitive operations because we are indexing into an array and then assigning the value of A[i] to temp. After that, we are doing 2 other primitive operations ($A[i] \leftarrow A[n-1-i]$) as well because we are indexing into an array and assigning a value to A[i]. The last statement ($A[n-1-i] \leftarrow temp$) has 1 primitive operations because we are assigning the value temp to A[n-1-i].

For the 2nd loop, we are assigning to value of 0 to i as well, which adds up to 1 primitive operations. Similarly, the condition (n-1) has 1 primitive operation but that 1 primitive operation is done n times. So, the condition (n-1) has n primitive operations. The loop runs (n-

1) times and return statement has 2 primitive operations. Therefore, return statement has (2)*(n-1)

Let’s compute for the whole thing to get the running time of my algorithm, we have the following:

In the first for loop:

i = 0 (1 primitive operation)
(n/2 – 1) ((1)*(n/2) = n/2 primitive operations)
Temp = A[i] ((2)*((n/2)-1) primitive operations)
A[i] = A[n-1-i] ((2)*((n/2)-1) primitive operations)
A[n-1-i] = temp ((1)*((n/2)-1) primitive operations)

In the second for loop:

I = 0 (1 pimitive operation)
(n-1)((1)*(n) primitive operations)

return A[i] ((2)*(n-1))

Combining all of the following above, we get:

$$(1)+\left(\frac{n}{2}\right)+\left(\left(\frac{n}{2}\right)-1\right)*(2+2+1)+(1)+(n)+(n-1)*(2)$$

Calculating the above, we get:

$$=(1)+\left(\frac{n}{2}\right)+5*\left(\frac{n}{2}\right)-5+(1)+(n)+(2*n)-2$$

$$= \boldsymbol{6n-5} = \textbf{O(n)}$$

Therefore, the running time of my Algorithm is O(n)