ASSIGNMENT 2

WRITTEN QUESTIONS PART

Data Structures and Algorithms

COMP 352

Laila Alhalabi # 40106558

Date: March 2, 2021

Question 1

Consider the following algorithm:

Algorithm 1: Mystery(A, n)

Input : Array A of n numbers $a_0, \cdots a_{n-1}$ with $n \geq 1$
Output: See question (a) below

```
if n = 1 then
    Print the message "Mystery solved"
else
    j ← 1;
    while j ≤ n − 1 do
        if a_{j−1} > a_j then
            swap a_{j−1} and a_j
        j ← j + 1;
    Mystery(A, n-1);
```

(a) Using A = (9,5,11,3,2) as the input array, hand-trace Mystery, showing the values of i, j, and the array elements just before line 7 is executed.

For A = (9, 5, 11, 3, 2), n = 5, j = 1:
$a_0 > a_1$ is true (9 > 5), so A = (5, 9, 11, 3, 2)
Values before line 7 is executed: n = 5, j = 1, A = (5, 9, 11, 3, 2)

However, the algoritm is a sorting algorithm using recursion called bubble sort, solving:
First call: Mystery((9,5,11,3,2), 5), A will be: A = (5,9,3,2,11)
Second call: Mystery((5,9,3,2,11), 4), A will be: A = (5,3,2,9,11)
Third call: Mystery((5,3,2,9,11), 3), A will be: A = (3,2,5,9,11)
Fourth call: Mystery((3,2,5,9,11), 2), A will be: A = (2,3,5,9,11)
Fifth call: Mystery((2,3,5,9,11), 1), will print: "Mystery solved"

(b) Determine the running time $T(n)$ as a function of the number of comparisons made on line 5, and then indicate its time and space efficiency (i.e., $\mathcal{O}$ and Ω bounds).

Space complexity is O(1) since it uses constant space.
Running time best case on line 5 is O(n) and Ω(n) if array was already sorted in ascending order, while the worst case is $O(n^2)$ and $\Omega(n^2)$ if the array is in reverse order and we need to move all elements.

(c) Determine the running time $T(n)$ as a function of the number of swaps made on line 6, and indicate its time and space efficiency.

Space complexity is O(1) since it uses constant space, while time complexity is O(n)

(d) Suggest an improvement, or a better algorithm altogether, and indicate its time and space efficiency. If you cannot do it, explain why it cannot be done?

As [1] illustrated, page 561, a better algorithm can be done using Merge Sort that works by dividing the array into two arrays and dividing these each one of these two arrays into two arrays, sorting all elements and merging the arrays together. The best and worst time complexity will be O(nlogn). However, this algorithm will require larger space and space complexity will be O(1) since we use arrays to store the sorted elements.

(e) Can Mystery be converted into an iterative algorithm? If it cannot be done, explain why; otherwise, do it and determine its running time complexity in terms of the number of comparisons of the array elements.

Since it is tail recursion, then it can be done as an iterative algorithm given an array as follow:

```
for i ← 0 to i < n – 1
    for j ← 0 to i < n – 1 – i
        if A_{i-1} > A_i then
            swap a_{i-1} and a_i
```

Question 2

For each of the following pairs of functions, either $f(n)$ is $\mathcal{O}(g(n))$, $f(n)$ is $\Omega(g(n))$, or $f(n)$ is $\Theta(g(n))$. For each pair, determine which relationship is correct. Justify your answer.

#	$f(n)$	$g(n)$
(a)	$4n \log n + n^2$	$\log n$
(b)	$8 \log n^2$	$(\log n)^2$
(c)	$\log n^2 + n^3$	$\log n + 3$
(d)	$n\sqrt{n} + \log n$	$\log n^2$
(e)	$2^n + 10^n$	$10n^2$
(g)	$\log^2 n$	$\log n$
(h)	n	$\log^2 n$
(i)	$\sqrt{n}$	$\log n$
(j)	4^n	5^n
(k)	3^n	n^n

If L = 0 then O(g(n)), if L = ∞ then Ω(g(n)), if L ≠ 0 then Θ (g(n)). To find L we apply $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$

a) Let f(n) = n^2 & g(n) = log(n) since they are the biggest terms.
Therefore, $\lim_{n\to\infty} \frac{n^2}{log(n)} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

b) Let f(n) = $\log(n^2)$ & g(n) = $\log(n)^2$ since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{log(n^2)}{log(n)^2} = 0$. The limit goes to 0 because f(n) < g(n), so complexity is **O(g(n))**.

c) Let f(n) = n^3 & g(n) = log(n) since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{n^3}{log(n)} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

d) Let f(n) = $n\sqrt{n}$ & g(n) = $\log(n)^2$ since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{n\sqrt{n}}{log(n)^2} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

e) Let f(n) = 10^n & g(n) = n^2 since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{10^n}{n^2} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

g) Let f(n) = $\log(n)^2$ & g(n) = log(n) since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{log(n)^2}{log(n)} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

h) Let f(n) = n & g(n) = $\log(n)^2$ since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{n}{log(n)^2} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

i) Let f(n) = $\sqrt{n}$ & g(n) = log(n) since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{\sqrt{n}}{log(n)} = \infty$. The limit goes to ∞ because f(n) > g(n), so complexity is **Ω(g(n))**.

j) Let f(n) = 4^n & g(n) = 5^n since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{4^n}{5^n} = 0$. The limit goes to 0 because f(n) < g(n), so complexity is **O(g(n))**.

k) Let f(n) = 3^n & g(n) = n^n since they are the biggest terms.
Therefore $\lim_{n\to\infty} \frac{3^n}{n^n} = 0$. The limit goes to 0 because f(n) < g(n), so complexity is **O(g(n))**.

REFERENCES

[1] M. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java, 6th Edition*. John Wiley & Sons, 2014.