DATA STRUCTURES & ALGORITHMS
Tutorial 1 Solution

COMPUTATIONAL COMPLEXITY

Question 1.

Reorder the following efficiencies from the smallest to the largest:

a. \( n + n^2 + n^5 \)
b. 20,000
c. \( 4^n \)
d. \( n^4 \)
e. \( n! \)
f. \( n\log_2(n) \)

**Solution:**

*Efficiency:* a measure of amount of time for an algorithm to execute (Time Efficiency) or a measure of amount of memory needed for an algorithm to execute (Space Efficiency).

\[ 20,000 < n\log_2(n) < n^4 < n + n^2 + n^5 < 4^n < n! \]

Question 2.

Decide whether these statements are True or False.

a. If \( O(f(n)) = O(g(n)) \) then \( f(n) = g(n) \)
b. If \( f(n) = O(g(n)) \) and \( g(n) = O(f(n)) \) then \( f(n) = g(n) \)

**Solution:**

a. False.
Ex: \( f(n) = 2n, g(n) = 3n \)
b. False
Ex: \( f(n) = n, g(n) = n + 1 \).

**Explanation:**

First, we review the definition of Big-O:

\( f(n) = O(g(n)) \iff \text{there are constants } C \in \mathbb{R}^+ \text{ and } k \in \mathbb{N} \text{ such that } 0 \leq f(n) \leq Cg(n) \text{ where } n > k \)

Here, we have: \( f(n) \leq 1 \times g(n) \) where \( n > 1 \). Hence, \( f(n) = O(g(n)) \).
And, we also have: \( g(n) \leq 2 \times f(n) \) where \( n > 1 \). Hence, \( g(n) = O(f(n)) \).
But \( f(n) \neq g(n) \).

**Question 3.**

Determine the big-O notation for the following:

- a. \( n^2 + n^5 \)
- b. \( 100n + 20n^{2/3} + 15n^{5/7} \)
- c. \( 9\log_2(n) + 6n \)
- d. \( n^{100} + 2^n \)
- e. \( n! + 2^n \)
- f. \( n\log_2(n) + 5n \)

**Solution:**

- a. \( O(n^5) \)
- b. \( O(n) \)
- c. \( O(n) \)
- d. \( O(2^n) \)
- e. \( O(n!) \)
- f. \( O(n\log_2(n)) \)

**Question 4.**

Calculate the run-time efficiency of the following program segment:

```
1   i = n
2   loop (i >= n/2)
   1   j = n - i
   2   loop (j < i)
      1   print(i, j)
      2   j = j + 1
3   end loop
4   i = i - 1
3   end loop
```

**Solution:**

Assume that \( n/2 = \lfloor n/2 \rfloor \) in the given program segment.

- If \( n \) is **even**, the run-time efficiency is
  \[ n + (n - 2) + (n - 4) + \ldots + 2 = n(n + 2)/4 = O(n^2) \]
- If \( n \) is **odd**, the run-time efficiency is
  \[ n + (n - 2) + (n - 4) + \ldots + 1 = (n + 1)^2/4 = O(n^2) \]

Or generally, the run-time efficiency of the given program segment is \( O(n^2) \).

**Question 5.**
Estimating the time complexity of the following program segment:

```plaintext
i = 0
loop (i < N)
    j = i
    loop (j < N)
        k = 0
        loop (k < M)
            x = 0
            loop (x < N)
                print(i, j, k)
                x = x + 3
            end loop
        end loop
    k = k + 1
end loop
j = j + 1
end loop
i = i + 1
```

Solution:
The iteration of variable $i$ is executed $N$ times. For each loop of variable $i$, the iteration of variable $j$ is executed $N$, $N-1$, $N-2$, ..., 1 times. For each loop of variable $j$, the first iteration of variable $k$ is executed $M$ times. For each loop of variable $k$, the iteration of variable $x$ is executed $[(N+2)/3]$ times; the second iteration of variable $k$ is executed $2M$ times. Therefore, the run-time efficiency is

\[
N \times (M \cdot [(N+2)/3] + 2M) + (N-1) \times (M \cdot [(N+2)/3] + 2M) + \ldots + 1 \times (M \cdot [(N+2)/3] + 2M)
\]

\[
= N \cdot (N+1) \cdot (M \cdot [(N+2)/3] + 2M) / 2
\]

\[
= O(N^3 M)
\]

Question 6.

If the algorithm `doIt` has an efficiency factor of $7n$, calculate the run time efficiency of the following program segment:

```plaintext
i = 1
loop (i <= n)
    j = 1
    loop (j < n)
        k = 1
```
Solution:
There are 3 nested loops, the iteration of variable $i$ is executed $n$ times, $j$ is executed $n-1$ times, $k$ is executed $n$ times. Therefore, the run-time efficiency is $n(n-1)n(7n) = O(n^4)$.

Question 7.

Write a recurrence equation for the running time $T(n)$ of $f(n)$, and solve that recurrence.

Algorithm $f$ (val $n$ <integer>)

Pre n must be greater than 0
Return integer value of $f$ corresponding to $n$

1 if ($n = 1$)
   1 return 1
2 else
   1 return $f(n - 1) + f(n - 1)$

End $f$

Solution:
$T(1) = 1$
$T(n) = 1 + 2*T(n-1) = 1 + 2*(1 + 2*T(n-2)) = …$
$= 1 + 2 + 2^2 + … + 2^{n-1} = 2^n - 1 = O(2^n)$

Question 8.

Write a recurrence equation for the running time $T(n)$ of $g(n)$, and solve that recurrence.

Algorithm $g$ (val $n$ <integer>)

Pre n must be greater than 0
Return integer value of $g$ corresponding to $n$

1 if ($n = 1$)
   1 return 1
2 else
   1 return $2*g(n - 1)$

End $g$
Solution:
T(1) = 1
T(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = 1 + 1 + … + 1 + 1 + T(1) = n = O(n)

Question 9.

Given that the efficiency of an algorithm is $5n\log_2(n)$, if a step in this algorithm takes 1 nanosecond ($10^{-9}$), how long does it take the algorithm to process an input of size 7000?

Solution:
$5 \times 7000 \times \log_2(7000) \times 10^{-9} = 4.4706 \times 10^4s$

-- End --