COMPLEJIDAD ALGORÍTMICA
PRÁCTICA CALIFICADA 01 –
Reto 01

Nombre Completo: Jose Alberto Quispe Zavaleta

Instrucciones: Lea las preguntas y plantee las respuestas en la misma hoja:

Pregunta 1 (1.5 punto)

Realizar un análisis del siguiente algoritmo y calcule el tiempo asintótico en notación Big-O de la siguiente expresión. Asignar el peso a cada línea de código.

```
list_lengths
```

```
[10, 100, 1000, 10000, 100000, 1000000, 10000000]
```

```
for l in list_lengths:
  lst = [42]*l

  tic = time.clock()
  x = find_max(lst)
  toc = time.clock()

  linear_times.append(toc-tic)
```

Análisis:

```
1    list_lengths
2
3
4    for l in list_lengths:  # -> O(n)
5        lst = [42] * l      # -> O(1)
6
7
8        tic = time.clock()  # ->  O(1)
9        x = find_max(lst)   # -> O(1)
10       toc = time.clock()  # -> O(1)
11
12
13       linear_times.append(toc-tic)  # O(1)
```

El mayor de todos es O(n).

Respuesta: Big O = O(n)

Pregunta 2 (1.5 punto)

Realizar un análisis del siguiente algoritmo y calcule el tiempo asintótico en notación Big-O de la siguiente expresión. Asignar el peso a cada línea de código.

```
def element_multiplier(my_list):
  for i in range(len(my_list)):
    for j in range(len(my_list)):
      x = my_list[i] * my_list[j]
```

```
granular_list_lengths
```

```
[10, 50, 100, 500, 1000, 5000, 10000]
```

```
poly_times = []

for l in granular_list_lengths:
  lst = [42]*l

  tic = time.clock()
  x = element_multiplier(lst)
  toc = time.clock()

  poly_times.append(toc-tic)
```

Análisis:

```
def element_multiplier(my_list):
  for i in range(len(my_list)):   # O(n)
    for j in range(len(my_list)): # O(n)
      x = my_list[i] * my_list[j] # O(1)

# -> al tener un O(n) anidado eso convierte a toda la funcion element_multiplier en O(n^2)

granular_list_lengths

poly_times = []  # O(1)

for l in granular_list_lengths:  # O(k) -> Este bucle se ejecuta k veces, donde k es la longitud de granular_list_lengths
  lst = [42] * l                 # O(l) -> Crea una lista de longitud l

  tic = time.clock()             # O(1)
  x = element_multiplier(lst)    # O(l^2) -> Pq llama a la función con complejidad O(n^2)
  toc = time.clock()             # O(1)

  poly_times.append(toc-tic)     # O(1)

```

- El bucle va a repetirse una cantidad de veces a la que asignamos como k y que es la longitud del arreglo *granular_list_lengths*.
- Dentro del bucle, la operación más costosa de todas es la llamada a la función *element_multiplier,* que tiene una complejidad de O(l^2).

Entonces tenemos como resultado O(k*l^2), siendo k la longitud de *granular_list_lengths* y l es cada valor en *granular_list_lengths.*

Respuesta: Big O = O(k*l^2)