Lautaro Colque

Ejercicio 1:

Calcular el número de OE (operaciones elementales) del siguiente algoritmo para el valor 12 y para el valor 5.

```
1:  def suma_inutil(acumulador,valor):                1:3OE
2:        acumulador=acumulador+valor                 2:2OE
3:        return(acumulador)                          3:2OE
4:
5:  acumulador=0                                      5:1OE
6:  valor=input_int("ingrese un numero")              6:1OE
7:  if valor >10:                                     7:1OE
8:        suma_inutil(acumulador,valor)               8:3OE
9:  else:
10:       print("ingrese un número mayor de 10")      10:1OE
```

para valor=12 tenemos 15 operaciones elementales
para valor=5 tenemos 8 operaciones elementales

Ejercicio 2:

Calcular número estimado de OE del siguiente algoritmo para los siguientes valores:

a) v1=255,v2=12,v3=1
- 10 operaciones elementales

b) v1=1,v2=2,v3=3
- 9 operaciones elementales

c) v1=5,v2=8,v3=2
- 10 operaciones elementales

```
1:  # ordena 3 numeros de mayor a menor
2:  if v1 > v2:                  2:1OE
3:      if v1 >v3:               3:1OE
4:          r1=v1                4:1OE
5:          if v2 >v3:           5:1OE
6:              r2=v2            6:1OE
7:              r3=v3            7:1OE
8:          else:
9:              r2=v3            9:1OE
10:             r3=v2            10:1OE
11:     else:
12:         r3=v2                12:1OE
13:         r2=v1                13:1OE
14:         r1=v3                14:1OE
15: else:
16:     if v2 > v3:              16:1OE
17:         r1=v2                17:1OE
18:         if v1 >v3:           18:1OE
19:             r2=v1            19:1OE
20:             r3=v3            20:1OE
21:         else:
22:             r2=v3            22:1OE
23:             r3=v1            23:1OE
24:     else:
25:        r1=v3                 25:1OE
26:        r2=v2                 26:1OE
27:        r3=v1                 27:1OE
28: print (r1,r2,r3)             28:1OE
```

Ejercicio 3
Calcular la complejidad temporal del algoritmo de forma experimental para el Algoritmo 3. El cálculo se debe efectuar para los siguiente intervalos de la variable monto
Calcular la complejidad para el intervalo [0,100] con paso 10
Calcular la complejidad para el intervalo [100,1000] con paso 100
Calcular la complejidad para el intervalo [1000,10000] con paso 1000
Calcular la complejidad para el intervalo [10000,100000] con paso 10000
Calcular la complejidad para el intervalo [100000,1000000] con paso 100000
Adaptar el siguiente código de ejemplo para calcular el tiempo de ejecución :

```
import time
start = time.time()
print("hello")
end = time.time()
print(end - start)
```

Algoritmo 3

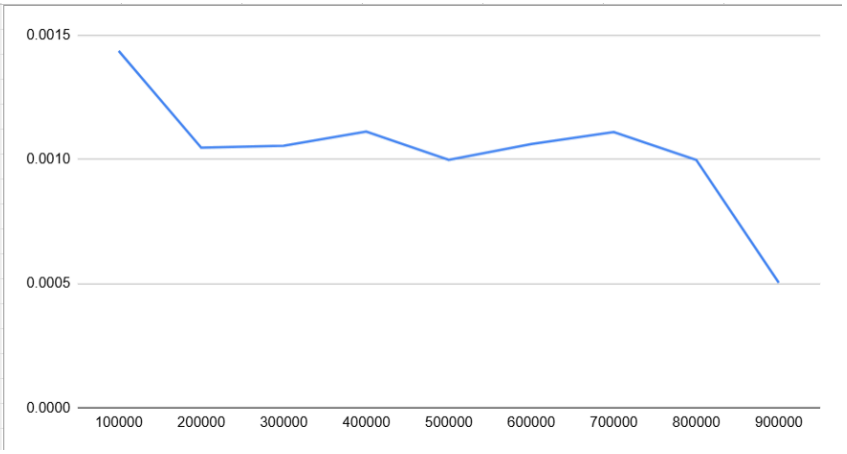
```
1:  # Algoritmo inútil que resta billetes de 100,10 y 1 a un monto dado.
2:  def entrega_billetes_2(monto):
3:          billete=100
4:          inc=0
5:          billete_actual=billete/(10**inc)
6:          while (monto>0):
7:                  if monto >= billete_actual:
8:                          monto=monto-billete_actual
9:
10:                 else:
11:                         inc=inc+1
12:                         billete_actual=billete/(10**inc)
```

Graficar los resultados obtenidos sobre un eje de coordenadas cartesianos donde el eje X representa el tamaño de la entrada (n) y el eje Y el tiempo de ejecución para cada uno de los intervalos.

Use los intervalos:
1-[100000,1000000] con paso 100000
2-[1000000,10000000] con paso 1000000
3-[10000000,100000000] con paso 10000000
4-[100000000,100000000] con paso 100000000
5-[1000000000,1000000000] con paso 1000000000
Use estos datos porque los anteriores planteados por el ejercicio siempre me daban tiempo igual a cero.

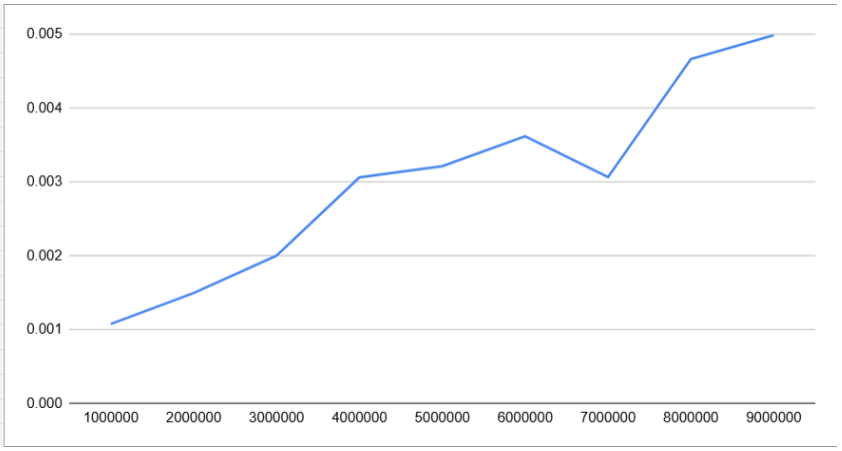
1-

100000	0.001437425613
200000	0.001047611237
300000	0.001055240631
400000	0.001112222672
500000	0.0009984970093
600000	0.001062393188
700000	0.00111079216
800000	0.0009977817535
900000	0.0005035400391



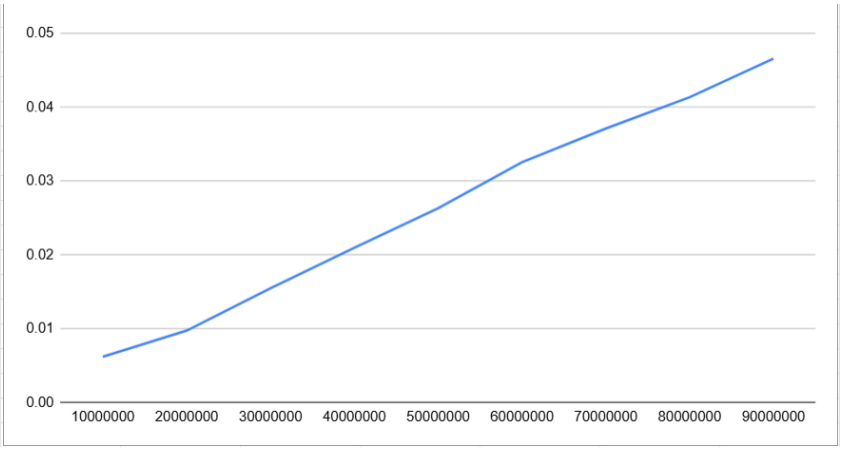
2-

1000000	0.001073122025
2000000	0.001492261887
3000000	0.001998901367
4000000	0.003059864044
5000000	0.003212451935
6000000	0.003617048264
7000000	0.003063201904
8000000	0.004662752151
9000000	0.004986524582



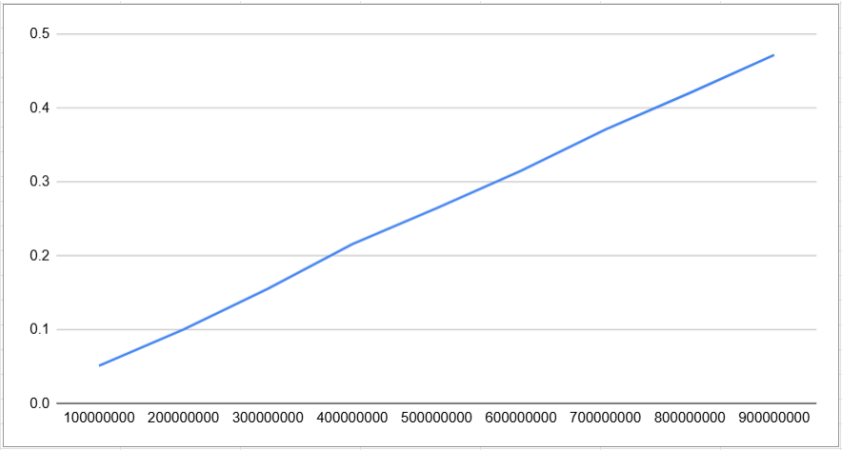
3-

10000000	0.006171226501
20000000	0.009712219238
30000000	0.01546287537
40000000	0.02093672752
50000000	0.02630114555
60000000	0.03251338005
70000000	0.03708434105
80000000	0.04131889343
90000000	0.04654526711



4-

100000000	0.05099534988
200000000	0.1000056267
300000000	0.1552259922
400000000	0.2155082226
500000000	0.264220953
600000000	0.3149657249
700000000	0.3707318306
800000000	0.4202446938
900000000	0.4716618061



5-

1000000000	0.5399389267
2000000000	1.04112339
3000000000	1.564352274
4000000000	2.087882996
5000000000	2.603989124
6000000000	3.122595549
7000000000	3.656460285
8000000000	4.168594837
9000000000	4.694106102