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

valor = 12 -> 1OE + 3OE + 8OE = 12OE

valor = 5 -> 1OE + 3OE + 1OE + 2OE = 7OE

Ejercicio 2

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

a) v1=255,v2=12,v3=1 T(a)= 6OE + 4OE = 10OE
b) v1=1,v2=2,v3=3 T(b) = 9OE
c) v1=5,v2=8,v3=2 T(c) = 10OE

```
1:  # ordena 3 numeros de mayor a menor
2:  if v1 > v2:                                   2: 1OE + max{
3:      if v1 >v3:                                    3: 1OE + max{
4:          r1=v1                                         4: 1OE
5:          if v2 >v3:                                    5: 1OE + max{2OE,2OE}
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 + max{
17:         r1=v2                                         17: 1OE
18:         if v1 >v3:                                    18:
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 + 3OE
```

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

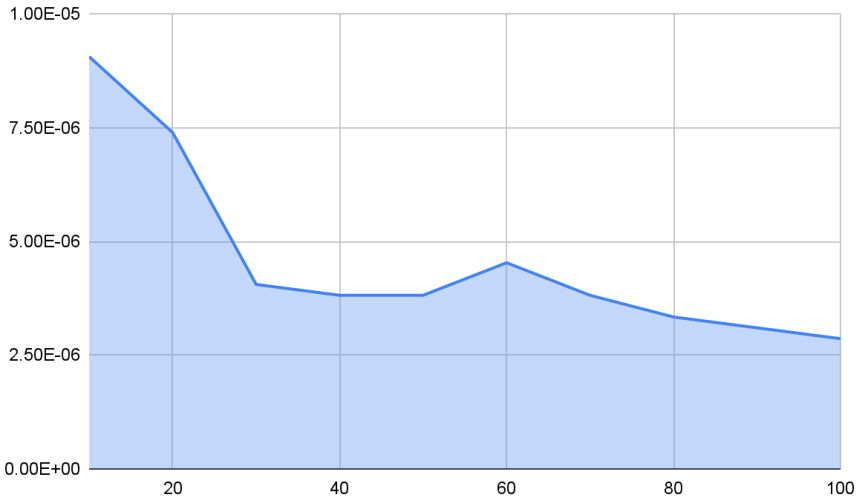
Adaptar el siguiente código de ejemplo para calcular el tiempo de ejecución `:`

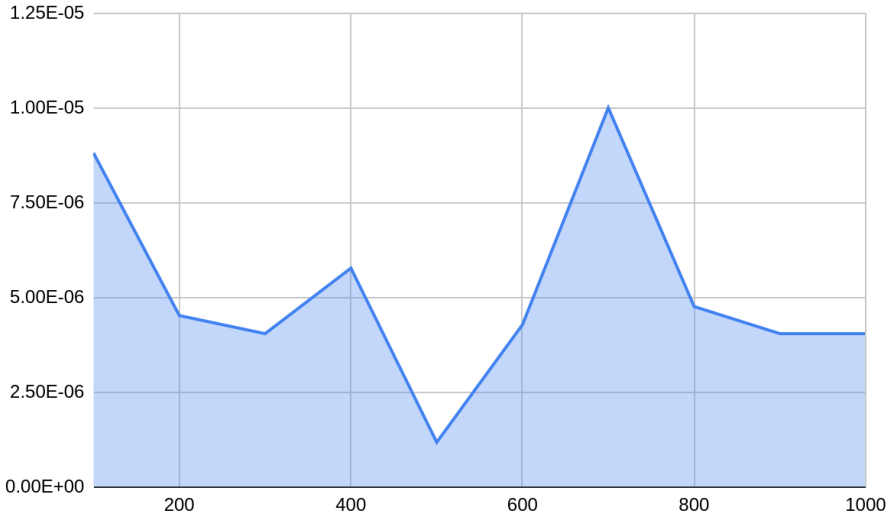
```
import time
start = time.time()
print("hello")
end = time.time()
print(end - start)
```

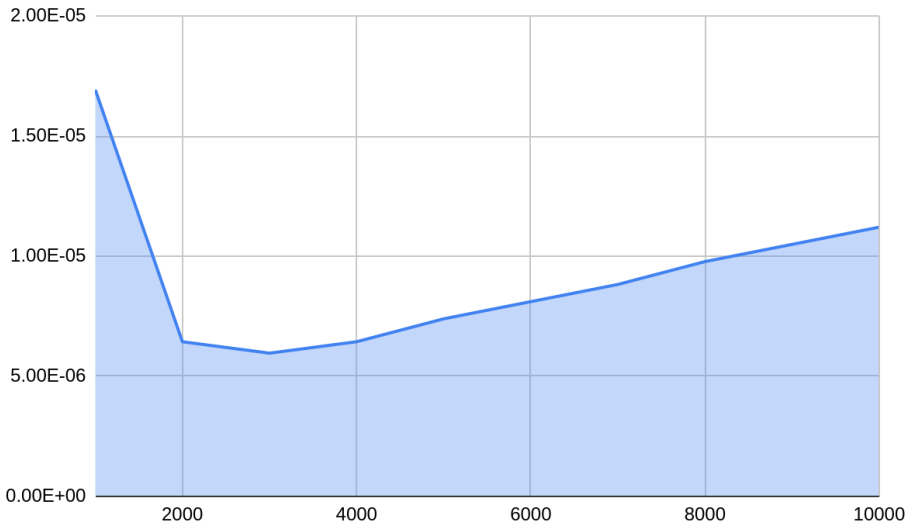
Algoritmo 3.

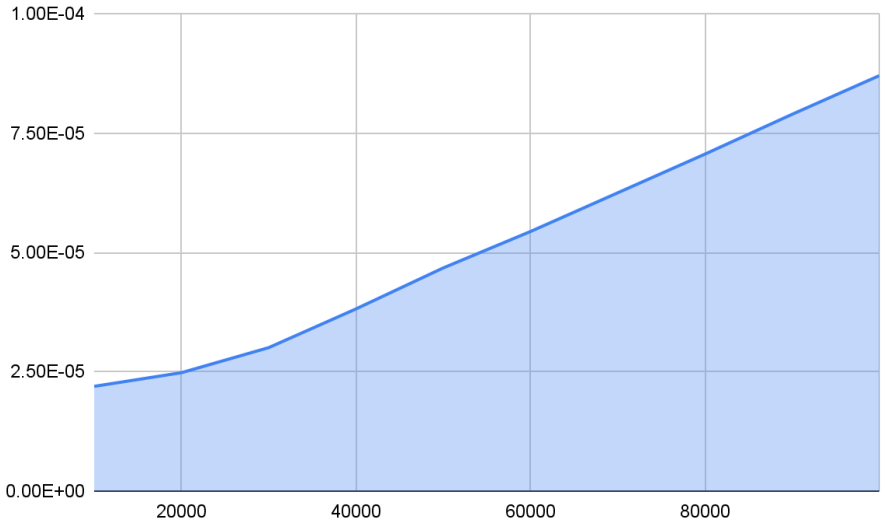
```
1:  # Algoritmo inútil que resta billetes de 100,10 y 1 a un monto dado.
2:  def entrega_billetes_2(monto):
3:          billete=100
4:          inc=0
5:          billete_actual=billete/(10**inc)
6:          while (monto>0):
7:                  if monto >= billete_actual:
8:                      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

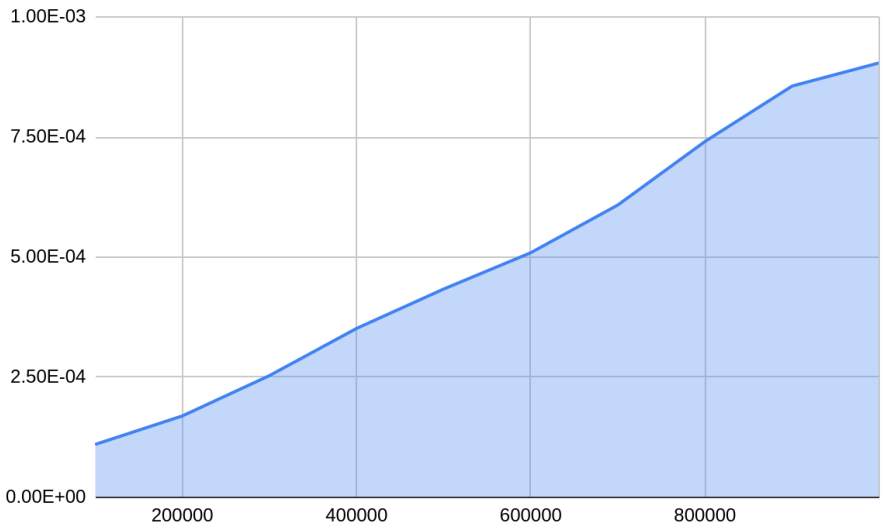
Graficar los resultados obtenidos sobre un eje de coordenadas cartesianos donde el eje X representa el tamaño de la entrada (n) y el eje Y el tiempo de ejecución para cada uno de los intervalos,


1.00E-05
7.50E-06
5.00E-06
2.50E-06
0.00E+00
20
40
60
80
100


1.25E-05
1.00E-05
7.50E-06
5.00E-06
2.50E-06
0.00E+00
200
400
600
800
1000


2.00E-05
1.50E-05
1.00E-05
5.00E-06
0.00E+00
2000
4000
6000
8000
10000


1.00E-04
7.50E-05
5.00E-05
2.50E-05
0.00E+00
20000
40000
60000
80000


1.00E-03
7.50E-04
5.00E-04
2.50E-04
0.00E+00
200000
400000
600000
800000

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Análisis de Complejidad Teórica

Todos los ejercicios son obligatorios

1) Calcular la cantidad de OE (operaciones elementales) para cada una de las operaciones del TAD secuencia implementado sobre arreglos:

   Access(Array,posicion)
   Search(Array,Element)
   Insert(Array,element,posicion)
   Delete(Array,element)

2) Calcular el orden de complejidad *O(f)* para cada una de las operaciones del ejercicio 1.

Access -> O(1)
Search -> O(n)
Insert -> O(n)
Delete -> O(n)

3) Calcular el orden de complejidad *O(f)* para los siguientes códigos:

Codigo I O(1)

```
1. if a>b:
2.   c=a+b
3. else:
4.   for d in range(1,10):
5.     c=a+b*d
```

Codigo 2 O(n)

```
1. a=1
2. while a<n:
3.   a=a+1
```

Codigo 3 $O(n^2)$

```
1. for i in range(1,n):
2.         j=0
3.          while j<i:
4.                  a=a*(1+j)
5.                  j=j+1
```

Integrantes: Dr. Carlos A. Catania, Ing. Lucia Cortés, Lic. Javier Rosenstein

Codigo 4 $O(n^3)$

```
1. for a in range(1,n):
2.     for b in range(a,n):
3.         if L[a]==L[b]:
4.             delete(L,L[b])
```

1)

Access(Array,posición) -> 6OE

```
def access(array, position): # 3OE

    return array[position] # 3OE
```

Search(Array,Element) -> 10 OE + 7n OE

```
def search(array, element): # 3OE

    encontrado = False # 1 OE
    for i in range(0,len(array)): # 1 OE + 5n OE + n(S1)

        if array[i] == element:  # S1 = 1OE + 1OE

            return i # 2OE



    if encontrado == False: # 1OE
        return None # 2OE

    #T(search) = 3OE + 1OE + 1OE + 5nOE + n(2OE) + 2OE + 3OE = 10 + 7n OE
```

Insert(Array,element,posición) -> 22 OE + 7n OE

```
def insert(array, element, index): #4OE

    if out_of_range(index, 0, len(array)): #7OE + 5OE
        return None #2OE

    else:

        for i in range(len(array)-1,index,-1): #4OE + 3n

            array[i] = array[i-1] # 4nOE

        array[index] = element # 2OE

    #T(insert) = 4OE + 12OE + max{2OE, 6OE + 7n} = 16OE + 6OE + 7n = 22OE + 7nOE
```

Delete(Array,element) -> 15 OE + 17n OE

```python
def delete(array,element): # 3

    encontrado = False # 1

    for i in range(0,len(array)): # 1 + 5n
        if array[i] == element: # 2n
            encontrado = True # 1n
            index = i # 1n

    if encontrado: # 1
        for i in range(index,len(array)-2): # 1 + 1 + 4n
            array[i] = array[i+1] # 4n

        array[len(array)-1] = None # 5
        return index # 2
    else:
        return None

    #T(delete) = 15 + 17n
```