Introducción al análisis de la complejidad de los algoritmos

El problema

Supongamos que queremos resolver el siguiente problema
Dada una secuencia de números enteros con al menos 3 elementos, encontrar el mayor número que se puede expresar como como la suma de tres elementos de la lista:

Es decir tendríamos que hacer un programa que se comporte de la siguiente manera:
$ python programa.py 1 ,2 ,1 ,2, 1, 3, 3, 4, 4, 5, 1
13
Ya que 13 se puede expresar como la suma de 5 + 4 + 4

$ python programa.py 1, 2, -1
2
Ya que 2 se puede expresar como 2 + 1 + (-1) (y no hay otra posible suma).

Las soluciones

Vamos a ver dos soluciones diferentes al problema. Ambas soluciones están en el archivo suma_maxima.py, definidas como las funciones sol1 y sol2.

Ejercicios:
1.1) Ejecutá interactivamente el módulo (python3 -i suma_maxima.py) y probá llamar a ambas funciones. ¿Siempre devuelven lo mismo? (deberían...)
1.2) Si tenés dudas de cómo funcionan los algoritmos, podés debuggear con
% python3 -i suma_maxima.py
>>> import pdb
>>> pdb.runcall(sol1, [1, 2, 3])
1.3) ¿Qué sucede si invocás a sol1 y a sol2 con una lista de 10 números? Hint: ¿qué devuelve python si escribís [1, 2, 3] * 10 ?
1.4) ¿Qué sucede si invocás a sol1 y a sol2 con una lista de 30 números?
1.5) ¿Qué sucede si invocás a sol1 y a sol2 con una lista de 100 números?

Comparando las soluciones

Evidentemente ambos algoritmos son distintos. Computan el mismo valor, pero lo hacen de un modo diferente. Y es evidente que cuando la lista es suficientemente grande, uno de ellos comienza a ser muy lento.

timeit

Para comparar ambas soluciones vamos a comenzar tomando un enfoque bien práctico. Utilizaremos el módulo timeit de la librería estándar de python. Por ejemplo, para medir el tiempo que demora sol1 para responder cuando le pasamos la lista [1, 2, 3], ejecutamos
% python3 -m timeit -n 1 -s 'from suma_maxima import sol1' ‘sol1([1, 2, 3])'
1 loops, best of 5: 2.01 usec per loop
2.1) Compará el tiempo de terminación de sol1 vs el tiempo de terminación de sol2 con listas de 10, 25, 50, 75, 100, 125, 150 elementos

n	sol1(n)	sol2(n)
3		
30	0,001480	0,000007
50		
75		
100		

2.2) Con los tiempos obtenidos en el punto 2.1 elabora un gráfico.
Graficar estas tablas:

2.3) ¿Qué podemos decir respecto a las dos soluciones? ¿Cuál te parece mejor? ¿Por qué?

3) Escriba dos funciones en Python para encontrar el número mínimo en una lista. La
+ primera función debe comparar cada número de una lista con todos los demás de la lista. O(n2). /se las doy resuelta)

```
def min1(l:[])->int:
def min2(l:[])->int:

def min1(l):
      min = l[0]
      for j in l:
      for k in l:
            if j <= min:
                  min = j
      return min

print(min1([3,-5,1])) -> n**2
```

+ La segunda función debe ser lineal O(n).

- Opcion bis: graficar estas dos funciones las repeticiones para distintos valores.