
Curso de Nivelación de Algoritmos - Examen

Formularios de Google <forms-receipts-noreply@google.com>
Para: mmastelli@fi.uba.ar

12 de octubre de 2022, 20:23

Gracias por rellenar **Curso de Nivelación de Algoritmos - Examen**

Esto es lo que se recibió.

Curso de Nivelación de Algoritmos - Examen

Tema 1

A la hora de escribir pseudocódigo, por favor no olvides aclarar la indentación usando puntos "." o guiones "_"

Por ejemplo:

if (a>b):

....c=0

else:

....d=0

Correo *

mmastelli@fi.uba.ar

Nombre

*

Mateo

Apellido *

Mastelli

Correo electrónico *

mmastelli@fi.uba.ar

Ejercicio 1 (20 pts)

a) Escribir una función, usando el método iterativo, que dada una lista de números, devuelva la misma lista pero donde a los números impares se les sume +1. Por ejemplo, dado

[2,5,4,3] -> [2,6,4,4]

[3,3,3] -> [4,4,4]

[] -> []

[4] -> [4]

b) ¿Se puede hacer en forma recursiva esta función? En caso afirmativo, ¿qué cambiarías al algoritmo anterior para hacerla recursiva?

*

a)

def Ejercicio1(L):

__ i = 0

__ while i < len(L):

__ if L[i]%2 != 0:

__ L[i] = L[i] + 1

__ i += 1

__ return L

b)

Para hacer la función de forma recursiva necesito un parámetro mas. Por ejemplo, le agrego como parámetro la cantidad de elementos de dicha lista:

(siendo $n = \text{len}(L) - 1$)

def Ejercicio1_r(L, n):

__ if n < 0:

__ return L

__ if L[n]%2 != 0:

__ L[n] = L[n] + 1

__ return Ejercicio1_r(L, n-1)

Ejercicio 2 (15 pts)

Sea la siguiente función (ver imagen), cuya entrada es una lista de números.

Describir en español cuál es el resultado de aplicar esta función sobre una lista (es decir, explicar qué características tienen las entradas para las cuales la función devuelve True y qué características tienen las entradas para las cuales la función devuelve False).

```
def func(lista):  
    a, e = 0, 0  
    i=0  
    while i < len(lista)-1:  
        if lista[i] > lista[i+1]:  
            a = a+1  
            i=i+1  
        elif lista[i] == lista[i+1]:  
            e = e+1  
            i = i+1  
        else:  
            i = i+1  
    return a + e + 1 == len(lista)
```

Esta función recorre la lista dada y cuenta la cantidad de veces que:

_Un elemento es mayor al siguiente elemento en la lista.

_Un elemento es igual al siguiente elemento en la lista.

(Dichos valores los almacena en dos variables por separado, "a" y "e" respectivamente).

Luego la función devuelve True solo si la suma entre estos dos valores + 1 es igual a la longitud de la lista. En otro caso devuelve False.

En otras palabras, la función devuelve True solo a las listas que contienen valores ordenados de manera creciente, con la posibilidad de que los valores se repitan. En cualquier otro caso la función devuelve False.

Ejercicio 3 (5 pts)

Seleccione las afirmaciones que son verdaderas acerca del algoritmo de Merge sort.

- ☐ Es una algoritmo de búsqueda
- ☒ Es un algoritmo de ordenamiento
- ☒ Utiliza recursión

- ☒ Divide el problema en muchos problemas más fáciles de resolver
- ☐ Siempre tiene mayor complejidad que Insertion Sort

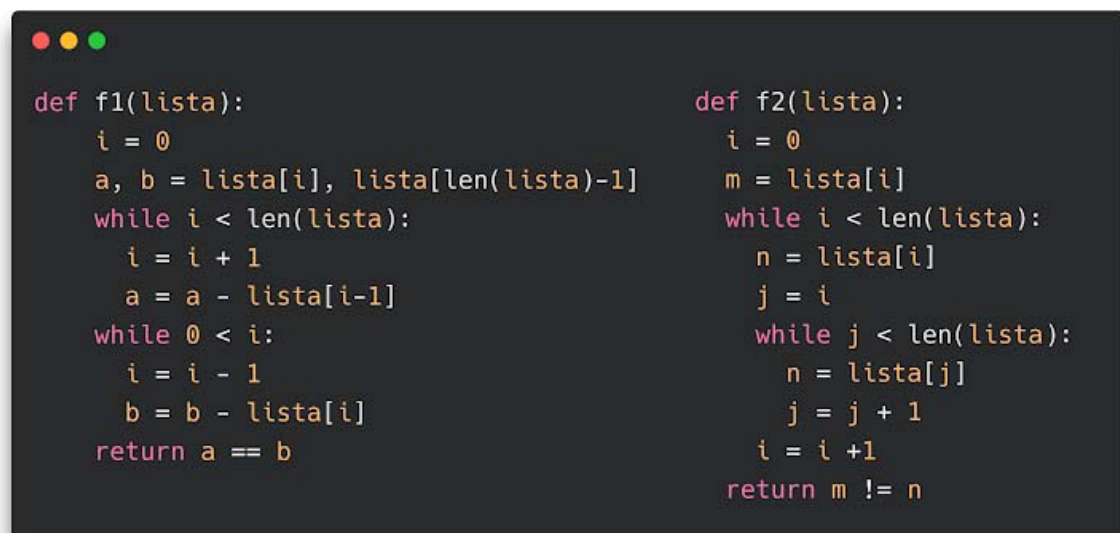
Ejercicio 4 (15 pts)

Dados los siguientes programas (ver imagen).

a) Determinar qué computa cada función. Justificar brevemente.

b) Determinar para cada función cuál es el orden de complejidad algorítmica en el peor caso en función del tamaño de la lista L. Justificar brevemente. Ayuda: cada función se corresponde con alguno de los siguientes órdenes de complejidad, donde $n = \text{len}(\text{lista})$

$O(1)$, $O(n)$, $O(\log(n))$, $O(n)$, $O(n * \log(n))$, $O(n^2)$, $O(n^3)$, $O(n!)$.



```
def f1(lista):
    i = 0
    a, b = lista[i], lista[len(lista)-1]
    while i < len(lista):
        i = i + 1
        a = a - lista[i-1]
    while 0 < i:
        i = i - 1
        b = b - lista[i]
    return a == b

def f2(lista):
    i = 0
    m = lista[i]
    while i < len(lista):
        n = lista[i]
        j = i
        while j < len(lista):
            n = lista[j]
            j = j + 1
        i = i + 1
    return m != n
```

a) La primer función (f1) devuelve True si el primer elemento de la lista es igual al ultimo elemento de la lista, mientras que la segunda función (f2) devuelve True cuando el primer elemento de la lista es distinto del ultimo elemento de la lista.

La primer función lo que hace es inicializar 2 variables:

_a = primer elemento de la lista

_b = ultimo elemento de la lista

luego recorre la lista y le resta cada valor de ella a la variable "a" (mientras va guardando estos resultados en la propia variable "a")

Después recorre la lista nuevamente, esta vez empezando desde el final de la lista, y le resta cada valor de ella a la variable "b" (mientras va guardando estos resultados en la propia variable "b")

Finalmente pide (o pregunta por) la igualdad entre "a" y "b". Lo cual solo se cumple si el primer y el ultimo elemento de la lista son iguales, dado que a las dos variables se le resta el mismo valor (la suma de todos los elementos de la lista).

La segunda función comienza guardando en una variable "m" el primer elemento de la lista. Luego a través de las variables "i" y "j" recorre la lista, por cada movimiento de la variable "i" la variable "j" recorre lo que resta de la lista (esto es debido a como esta estructurado con los whiles anidados). Cada vez que "j" se mueve se almacena el valor lista[j] en la variable "n". dado que los ciclos terminan cuando $i = j = \text{len}(\text{lista}) - 1$, "n" termina tomando el valor del ultimo elemento de la lista.

Finalmente pide (o pregunta por) la NO igualdad entre "m" y "n", siendo estos el primer y el ultimo elemento de la lista respectivamente.

b) La función f1 al tener dos whiles pero NO anidados tiene un orden de complejidad algorítmica $O(n)$. Mientras que la función f2 al tener dos whiles anidados tiene un orden de complejidad $O(n^2)$

Ejercicio 5 (10 pts)

Determinar cuál es el valor de x, y, z, i luego de ejecutar el siguiente programa (ver imagen).

```
x, y, z = 8, 2, 10
y = y + z + 5
i = 4
while i != 0:
    z = y - 3
    y = x - 5
    x = z + 1
    i = i - 2
```

x = 1
y = 10
z = 0
i = 0

Ejercicio 6 (20 pts)

Implemente una función recursiva que reciba dos números positivos y enteros como parámetros (n y p) y devuelva el resultado de ejecutar la siguiente operación:

$$F(n, p) = (1 * p) + (2 * p) + (3 * p) + \dots + (n * p)$$

Por ej, si llamamos F(2, 2), debe devolver 6. Si llamamos F(3,2) debe devolver 12.

```
def Ejercicio6(n,p):  
    __if n == 0:  
        ____ return 0  
    __if n > 0:  
        ____ return Ejercicio6(n-1,p) + n * p
```

Ejercicio 7 (15 pts)

Implementar una clase llamada Pava_electrica. La clase tiene como atributos: cantidad_liquido, temperatura, fabricante y linea. Además, tiene los métodos: __init__, cargar_liquido, vaciar_liquido y calentar_liquido.

```
class Pava_electrica():  
    __def __init__(self, cantidad_liquido = 0, temperatura = 0, fabricante = "-", linea = "-"):  
        ____ self.cantidad_liquido = cantidad_liquido  
        ____ self.temperatura = temperatura  
        ____ self.fabricante = fabricante  
        ____ self.linea = linea  
    __def cargar_liquido(self):  
        ____ self.cantidad_liquido += 1  
        ____ print("la pava electrica tiene ", self.cantidad_liquido, " unidades de liquido")  
    __def vaciar_liquido(self):  
        ____ self.cantidad_liquido = 0  
        ____ self.temperatura = 0  
        ____ print("la pava electrica se vacio, no tiene contenido")  
    __def calentar_liquido(self):  
        ____ self.temperatura += 1  
        ____ if self.cantidad_liquido == 0:  
            ____ self.temperatura = 0  
        ____ print("la pava electrica tiene", self.temperatura, " unidades de temperatura")
```

Crea tu propio formulario de Google

Notificar uso inadecuado