Algoritmia y Complejidad
Entrega ejercicios 3

Grado en Ingeniería Informática

Raúl López Llana
Grupo Laboratorio: Miércoles 12.00 – 14.00 h.

Índice

Problema 6

Enunciado:
Mr. Scrooge ha cobrado una antigua deuda, recibiendo una bolsa con n monedas de oro. Su olfato de usurero le asegura que una de ellas es falsa, pero lo único que la distingue de las demás es su peso, aunque no sabe si este es mayor o menor que el de las otras. Para descubrir cuál es la falsa, Mr. Scrooge solo dispone de una balanza con dos platillos para comparar el peso de dos conjuntos de monedas. En cada pesada lo único que puede observar es si la balanza queda equilibrada, si pesan más los objetos del platillo de la derecha o si pesan más los de la izquierda.
Diseñar un algoritmo Divide y Vencerás para resolver el problema de Mr. Scrooge (encontrar la moneda falsa y decidir si pesa más o menos que las auténticas).

A partir de un ejemplo, se procederá a la explicación.

Se dispone de un array con el peso de las monedas.
Lo que haré es dividir el total de las monedas en grupos de 3 ya que con esta cantidad puedo saber cual es la originar y cual la falsa y si pesa más o menos la moneda.
En cuanto a la posición, sumaré el total de monedas que se encuentren en los grupos previos a la falsa y le sumaré la posición que ocupa dentro den grupo de esta.

Primer grupo

1 1 1

Segundo grupo

2 1 1

Vemos que en el segundo grupo está la falsa y pesa mas

La posición en la que se encuentra será el tamaño del primer grupo +
la posición en la que se encuentra dentro del segundo

Posición 4 y pesa más

Componentes Divide y vencerás en el algoritmo
1.- Conjunto de candidatos: Todos los datos.
2.- Conjunto de decisiones: comparar los elementos de un grupo.
3.- Función que determina la solución del problema: elemento que sea diferente a los del resto de grupo.
4.- Completable: se debe dividir el tamaño del problema hasta encontrar un tamaño con el que podamos trabajar.

5.- Función de selección: se elige el elemento que difiere de los otros elementos con los que se compara.
6.- Objetivo: Devolver el dato que sea distinto al resto.

```
'''Variables que se pueden cambiar'''
Nmonedas=38
PesoReales=2
PesoFalsa=1
PosicionFalsa=25

'''Esta función compara dos numero pasados por referencia
    El resultado será 2 si elemento1>elemento2
        El resultado será 1 si elemento1<elemento2'''
def xmayorquey(elemento1,elemento2):
    if (elemento1 > elemento2):
        return 2
    else:
        return 1

'''Esta función aplicará el algoritmo DyV para decir cual es la moneda falsa
Para poder saber si una moneda es falsa o no, necesitaré compararlo minimo
con otras 2 monedas'''
def funcion(bolsa):
    iguales=False
    resul=0
    posi=-1

    '''Cuando tenemos 3 monedas, podemos compararlas'''
    if (len(bolsa) == 3):
        '''Realizo distintas comparaciones para ver si son iguales.
        En caso de encontrar una que no sea igual, almaceno su posición y si pesa
más o menos'''
        if (bolsa[0] == bolsa[1]):
            if (bolsa[1] != bolsa[2]):
                resul = xmayorquey(bolsa[2], bolsa[0])
                posi =3
            else:
                iguales=True
        else:
            if (bolsa[0] == bolsa[2]):
                # todos igual
                resul = xmayorquey(bolsa[1], bolsa[0])
                posi =2
            else:
                 resul = xmayorquey(bolsa[0], bolsa[1])
                 posi =1

    '''Cuando tenemos 4 monedas, podemos compararlas'''
    if (len(bolsa) == 4):
```

```
        '''Realizo distintas comparaciones para ver si son iguales.
            En caso de encontrar una que no sea igual, almaceno su posición y si
pesa más o menos'''
        if (bolsa[0] == bolsa[1]):
            if (bolsa[1] == bolsa[2]):
                if (bolsa[2] != bolsa[3]):
                    resul = xmayorquey(bolsa[3], bolsa[0])
                    posi=4
                else:
                    iguales = True
            else:
                resul = xmayorquey(bolsa[2], bolsa[0])
                posi =3
        else:
            if (bolsa[0] == bolsa[3]):
                resul = xmayorquey(bolsa[1], bolsa[3])
                posi =2
            else:
                resul = xmayorquey(bolsa[0], bolsa[3])
                posi =1

    '''Cuando tenemos 5 monedas, podemos compararlas'''
    if (len(bolsa) == 5):
        '''Realizo distintas comparaciones para ver si son iguales.
            En caso de encontrar una que no sea igual, almaceno su posición y si
pesa más o menos'''
        if (bolsa[0] == bolsa[1]):
            if (bolsa[1] == bolsa[2]):
                if (bolsa[2] == bolsa[3]):
                    if (bolsa[3] != bolsa[4]):
                        resul = xmayorquey(bolsa[4], bolsa[0])
                        posi=5
                    else:
                        iguales = True
                else:
                    resul = xmayorquey(bolsa[3], bolsa[0])
                    posi=4
            else:
                if (bolsa[1] == bolsa[3]):
                    resul = xmayorquey(bolsa[2], bolsa[3])
                    posi =3
                else:
                    resul = xmayorquey(bolsa[1], bolsa[3])
                    posi =2
        else:
            if (bolsa[0] == bolsa[3]):
                resul = xmayorquey(bolsa[1], bolsa[3])
                posi =2
            else:
                resul = xmayorquey(bolsa[0], bolsa[3])
```

```
            posi =1

    '''En caso de tener un número de monedas demasiado alto como para
empezar a compararlas, entra aquí'''
    if(resul==0 and iguales==False):
        '''Si tenemos 2 o menos monedas es porque se ha introducido mál el
numero de monedas inicial'''
        if(len(bolsa)<=2):
            print("Error en cantidad de bolsa")
            return 0,0
        else:
            '''En caso de tener 6 y 8 monedas en un grupo, tendremos que
dividirnas en 2 grupos'''
            '''Miro cuantos elementos por grupo ha de tener'''
            if(len(bolsa)>=6 and len(bolsa)<=8):
                longitud = len(bolsa) // 2
                '''Divido en grupos las monedas y vuelvo a llamar a la función'''
                primeraDivision = funcion(bolsa[:longitud])
                segundaDivision = funcion(bolsa[longitud:])
                a = max(primeraDivision, segundaDivision)
                '''compruebo si se ha encontrado la moneda falsa'''
                if (primeraDivision[1] == segundaDivision[1]):
                        posi = -1
                else:
                    if (primeraDivision[1] > segundaDivision[1]):
                        posi = primeraDivision[1]
                    else:
                        posi = segundaDivision[1]+ longitud
                '''Devuelvo el resultado obtenido'''
                a = (a[0], posi)
                return a
            else:
                '''En caso de tener más de 8 monedas en un grupo, tendremos que
dividirnas en 2 grupos'''
                '''Miro cuantos elementos por grupo ha de tener'''
                longitud = len(bolsa) // 3
                '''Divido en grupos las monedas y vuelvo a llamar a la función'''
                primeraDivision=funcion(bolsa[:longitud])
                segundaDivision=funcion(bolsa[longitud:longitud*2])
                terceraDivision=funcion(bolsa[longitud*2:])
                a=max(primeraDivision,segundaDivision,terceraDivision)

                '''compruebo si se ha encontrado la moneda falsa'''
                if (primeraDivision[1] == segundaDivision[1]):
                    if (segundaDivision[1] == terceraDivision[1]):
                        posi=-1
                    else:
                        posi=terceraDivision[1]+ 2*longitud
                else:
                    if (primeraDivision[1] ==terceraDivision[1]):
```

```
                posi =segundaDivision[1]+longitud
            else:
                posi=primeraDivision[1]
        '''Devuelvo el resultado obtenido'''
        a=(a[0],posi)
        return a
    else:
        return resul,posi

    return

'''Esta función rellenará la bolsa con modenas segun los parámetros pasados'''
def rellenarBolsa(bolsa,PesoReales,PesoFalsa,PosicionFalsa):
    for i in range(1, Nmonedas + 1):
        '''Posición de la moneda falsa'''
        if(i==PosicionFalsa):
            '''Peso moneda falsa'''
            bolsa.append(PesoFalsa)
        else:
            '''Peso monedas reales'''
            bolsa.append(PesoReales)


    return bolsa



#region main
'''Relleno la bolsa de monedas en función de los parámetros definidos'''
bolsa=[]
bolsa=rellenarBolsa(bolsa,PesoReales,PesoFalsa,PosicionFalsa)

'''Llamo al a funcion que me dirá cual es la moneda falsa usando el algoritmo
DyV'''
masomenos,posi=funcion(bolsa)

'''Muestro por pantalla todas las monedas'''
print(bolsa)

'''Muestro el resultado'''
if(masomenos==2):
    print("pesa mas y se encuentra en la posición: ",posi)
else:
    print("Pesa menos y se encuentra en la posición: ",posi)
#endregion
```

Explicación:

Definimos un array con los pesos de las monedas a las que introduzco una falsa en una posición en concreto y con un peso mayor o menor que el del resto de monedas.

Llamo a la función y comprueba si el tamaño del conjunto es de 3,4 o 5 que son los distintos casos con los que yo puedo trabajar y buscaré dentro de estos grupos si está la moneda falsa comparando unas monedas con otras y guardaré la información asociada a la moneda falsa si la encuentro.

En caso que el numero de monedas sea mayor al indicado, dividiré el numero de monedas en grupos más pequeños hasta encontrar grupos con los que pueda trabajar.

Cálculo de complejidad:

Para calcular la complejidad de este algoritmo, usaremos la siguiente tabla:

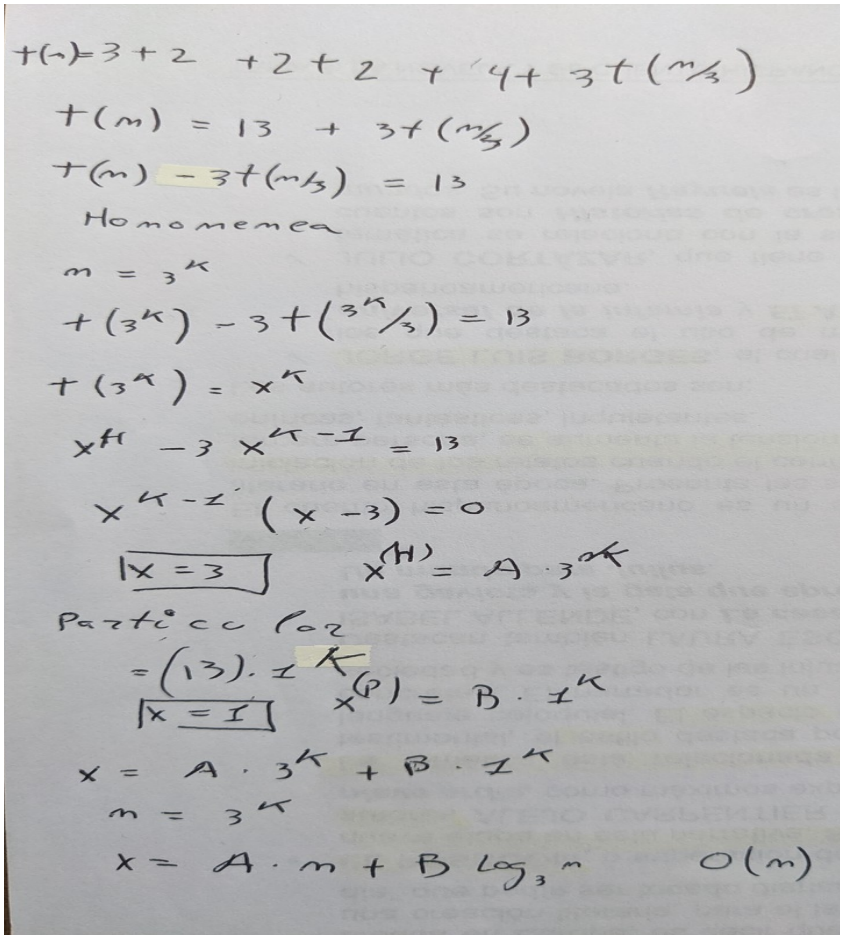
$$T(n) = \begin{cases} O(n^p) & k < b^p \\ O(n^p \log n) & k = b^p \\ O(n^{\log_b k}) & k > b^p \end{cases}$$

K será el número de subcasos
p será el grado máximo del polinomio de la función de DyV
b será el tamaño de los subcasos

En nuestro caso:
k=3 (hago 3 llamas recursivas)
p=1 (es el grado máximo dentro de la función)
b=3 (por cuanto se divide el problema por cada llamada recursiva, se divide en un tercio)

con todo esto podemos afirmar que la complejidad de nuestro programa es $O(n * \log_3(n))$

$$\begin{aligned} T(n) = 3 &+ max(max(max(2,1), max(2,2)), 0) + max(max(max(max(2,1),2), max(2,2)), 0) \\ &+ max(max(max(max(max(2,1),2), max(2,2)), max(2,2)), 0) \\ &+ max(max(1 + 2(n/2) + 1 + max(1, max(1,1) + 2), \\ &\quad 1 + 3t(n/3) + \ max(max(1,1), max(1,1)) + 2), 1) \end{aligned}$$



Ejemplo de ejecución:

Al ejecutar uno de los casos de prueba propuestos, podemos observar que su funcionamiento es correcto:

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
Pesa menos y se encuentra en la posición:  25
```

Problema 4

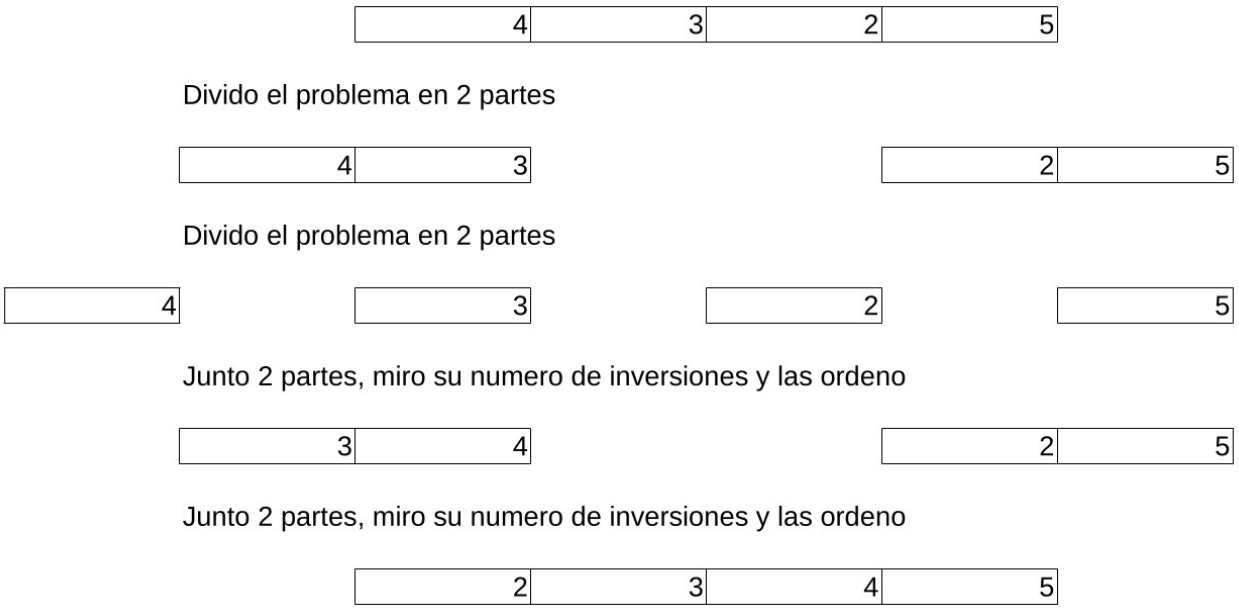
Enunciado:
En una plataforma digital van a analizar los gustos de sus clientes para analizar las similitudes con otros clientes y hacerles recomendaciones personalizadas. Para ello han pedido a los usuarios de dicha plataforma que realicen un ranking de un conjunto de peliculas que ya han visto. Para clientes hay que estudiar cómo de similares son sus gustos

Una de las formas de calcular la similitud entre dos rankings es contar el número de inversiones que existen. Dos películas i y j están invertidas en las preferencias de A y B si el usuario A prefiere la serie i antes que la j (aparece antes en su ranking) mientras que el usuario B prefiere la serie j antes que el i. Cuantas menos inversiones existan entre dos rankings, más similares son las preferencias de esos usuarios. Para facilitar este proceso, podemos suponer que el ranking de uno de los clientes siempre es 1, 2, ..., n (cualquier par de rankings se puede transformar para que se cumpla esto).

De esta forma, podemos trabajar sólo con el otro ranking, y existe una inversión cada vez que un número más pequeño está situado a la derecha de un número más grande. Diseñar un algoritmo, lo más eficiente posible, que permita calcular el número de inversiones entre dos clientes

A partir de un ejemplo, se procederá a la explicación.

Se dispone de un array con 4 posiciones con números no repetidos y ordenados de forma aleatoria. Lo que haré es dividir el array hasta tener elementos individuales y desde ese punto, iré comprobando de elementos para ver si tienen inversión, en dicho caso aumentaré el contador total y los re ordenaré para que queden al final de menor a mayor.



Como se puede ver en el ejemplo de la imagen superior, partimos de un array de números.
Comenzamos a dividirlo en en trozos de dos hasta tener arrays de un solo elemento.
En ese punto empiezo a juntar arrays. Primero junto dos que contienen un elemento y miro se están ordenados, que no tendría que hacer nada, o desordenados, que aumento el contador de inversiones y los ordeno.
Con esta misma metodología, lo replico con el resto y luego paso a juntar arrays de 2 elementos.
Realizo esto hasta llegar a tener un array con la misma longitud que el inicial pero ya ordenado y sabiendo el número de inversiones.

Componentes Voraces en el algoritmo

1.- Conjunto de candidatos: Todos los datos.
2.- Conjunto de decisiones: comparar los elementos de dos grupos.
3.- Función que determina la solución del problema: elemento que sea menor entre dos conjuntos.
4.- Completable: se debe dividir el tamaño del problema hasta encontrar un tamaño con el que podamos trabajar.
5.- Función de selección: se elige primero el elemento de menor y peso y si este se encontraba a la derecha, aumento el numero de inversiones en el tamaño de los elementos de la izquierda.
6.- Objetivo: Devolver el array de datos ordenado y el numero de inversiones total.

```
from random import randint
'''Esta función aplicará el algoritmo DyV para calcular el numero de inversiones de una array'''
def funcion(cliente):
    inversion=0
    '''Si el array contiene más de un elemento, tengo que dividirlo'''
    if(len(cliente)>1):
        '''Divido en 2 grupos'''
        longitud = len(cliente) // 2
        inversion1, cliente1 = funcion(cliente[:longitud])
        inversion2, cliente2 = funcion(cliente[longitud:])
        '''Sumo el numero de inversiones de cada uno'''
        inversion+=inversion1+inversion2
        '''calculo las inversiones y reordeno los 2 arrays'''
        x=y=i=0
        while(x<len(cliente1) and y<len(cliente2)):
            if(cliente1[x]>cliente2[y]):
                cliente[i]=cliente2[y]
                '''Sumo al numero de inveriones todos los elementos que queden por comparar del array 1'''
                inversion+=len(cliente1)-x
                y+=1
                i+=1
            else:
                cliente[i]=cliente1[x]
                x += 1
                i += 1
        '''Miro los arrays y termino de insertar los elementos de mayor peso'''
        while (x < len(cliente1)):
            cliente[i] = cliente1[x]
```

```
            x += 1
            i += 1
        while (y < len(cliente2)):
            cliente[i] = cliente2[y]
            y += 1
            i += 1
    '''Devuelvo el numero de inversiones total y el array ordenado'''
    return inversion,cliente




#region main
'''Declaro el numero de elementos'''
N=10
cliente=[]
'''Relleno las valoraciones del cliente sin repetir notas'''
for i in range(1,5):
    numero = randint(1, 10)
    while(cliente.__contains__(numero)):
        numero = randint(1, 10)
    cliente.append(numero)

'''muestro sus gustos por pantalla'''
print(cliente)

'''Aplico el algorimo de divide y vencerás'''
inversiones, clientes =funcion(cliente)
print("total de inversiones  usando DyV: ",inversiones)
#endregion

'''caso 1'''
cliente=[6, 4, 3, 1, 8, 7, 2, 5]
print("\n\nCaso 1: \n",cliente)
inversiones, clientes =funcion(cliente)
print("total de inversiones  usando DyV: ",inversiones)

'''caso 2'''
cliente=[7, 13, 2, 19, 10, 4, 9, 20, 1, 5, 15, 17, 6, 18, 3, 14, 16, 8, 12, 11]
print("\n\nCaso 2: \n",cliente)
inversiones, clientes =funcion(cliente)
print("total de inversiones  usando DyV: ",inversiones)
```

Explicación:

Creo un array con los gustos del cliente que quiero analizar y los relleno con valores sin repetir.

Llamo a la función que dividirá el conjunto de trabajo que se le ha pasado en caso de que contenga mas de 1 elemento. Una vez divididos los datos empezaré a compara entre dos conjuntos de elementos si existe inversión o no, a su vez iré ordenando los elementos para que queden de menor a mayor ya que me facilitará posteriormente al comparar este grupo resultante con otro.

En caso de que veamos que un elemento que se encuentra a la derecha de otro es mayor, se considera una inversión y se sumará al total el numero de elementos que se encuentre a la izquierda de este.

Cálculo de complejidad:

Para calcular la complejidad de este algoritmo, usaremos la siguiente tabla:

$$T(n) = \begin{cases} O(n^p) & k < b^p \\ O(n^p \log n) & k = b^p \\ O(n^{\log_b k}) & k > b^p \end{cases}$$

K será el número de subcasos
p será el grado máximo del polinomio de la función de DyV
b será el tamaño de los subcasos

En nuestro caso:
k=2 (hago 2 llamas recursivas)
p=1 (es el grado máximo dentro de la función)
b=2 (por cuanto se divide el problema por cada llamada recursiva)

Por lo que podemos afirmar que la complejidad de nuestro problema es $O(n * \log_2(n))$

$$T(n) = 1 + max\left(1 + 2T(n/2) + 2 + \sum(max(3,3)) + \sum(3) + \sum(3) + 1, 0\right)$$
$$T(n) = 4 + 2T(n/2) + 9n$$
$$T(n) - 2T(n/2) = 4 + 9n$$
$$n = 2^k$$
$$T\left(2^k\right) - 2T\left(2^k/2\right) = 4 + 9*2^k$$
$$T\left(2^k\right) = x^k$$
$$Homogenea$$
$$x^k - 2x^{k-1} = 0$$
$$x^{k-1}(x-2) = 0$$
$$Raiz\ x = 2$$
$$x^{(H)} = A*2^k$$
$$Particular$$
$$x^k - 2x^{k-1} = 4 + 9*2^k$$
$$Raices\ 1\ y\ 2$$
$$x^{(P)} = B + c*2^k*k$$

$$X = x^{(H)} + x^{(P)} = A*2^k + B + c*2^k*k$$
$$n = 2^k$$
$$X = x^{(H)} + x^{(P)} = A*n + B + c*n*log\ (n)$$
$$O(n*log(n))$$

Ejemplo de ejecución:

Al ejecutar uno de los casos de prueba propuestos, podemos observar que su funcionamiento es correcto y vemos paso a paso los movimientos que va realizando para poder seguir la traza del ejemplo:

```
Caso 1:
 [6, 4, 3, 1, 8, 7, 2, 5]
Divido  [6, 4, 3, 1, 8, 7, 2, 5]
Divido  [6, 4, 3, 1]
Divido  [6, 4]
ordeno  [6]  y  [4]
Divido  [3, 1]
ordeno  [3]  y  [1]
ordeno  [4, 6]  y  [1, 3]
Divido  [8, 7, 2, 5]
Divido  [8, 7]
ordeno  [8]  y  [7]
Divido  [2, 5]
ordeno  [2]  y  [5]
ordeno  [7, 8]  y  [2, 5]
ordeno  [1, 3, 4, 6]  y  [2, 5, 7, 8]
total de inversiones  usando DyV:  15
```