

Ejercicio (tiempo promedio):

Estudiar $t_p(n)$ para las instrucciones de asignación...

①

```

cont:=0
para i:= 1,...,n hacer
    para j:= 1,...,i-1 hacer
        si a[i] < a[j] entonces
            cont:= cont + 1
        finsi
    finpara
finpara

```

②

```

i:= 1
mientras i ≤ n hacer
    si a[i] ≥ a[n] entonces
        a[n]:=a[i]
    finsi
    i:= i + 1
finmientras

```

$$a[n]:=a[i] \quad \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{1}{4} \cdots \frac{1}{n} = \frac{1}{i+1}$$

①

$$t_p(n) = 1 + \sum_{i=1}^n \left(\sum_{j=1}^{i-1} (0.5 \cdot 1 + 0.5 \cdot 0) \right)$$

$$1 + \sum_{i=1}^n \left(\sum_{j=1}^{i-1} \frac{1}{2} \right) = 1 + \sum_{i=1}^n \left(\frac{1}{2} (i-2)+1 \right) = 1 + \sum_{i=1}^n \frac{1}{2} (i-1) = 1 + \frac{1}{2} \sum_{i=1}^n i - 1 =$$

$$1 + \frac{1}{2} \sum_{i=1}^n i - \frac{n}{2} = 1 + \frac{1}{2} \frac{(n+1) \cdot n}{2} - \frac{n}{2}$$

②

$$t_p(n) = 1 + \sum_{i=1}^n \left(1 + \frac{1}{i+1} \right) = 1 + \sum_{i=1}^n 1 + \sum_{i=1}^n \frac{1}{i+1} = 1 + n + \int_0^n \frac{1}{i+1} di$$

$$1 + n + \int_0^n \ln(i+1) di = 1 + n + \ln(n+1)$$

01 CLASE PRACTICA CONTEO - - - - - ANÁLISIS

2) Dado el algoritmo:

```
for i=1 to n
    c[1,i]=a[i,2]*a[i,i]
    for j=i+1 to n
        if par(a[i,j])
            c[i,j]=
```

Contar el número promedio de asignaciones $c[1,i]=a[i,2]*a[i,i]$: ...**a**..
el número promedio de comparaciones $\text{par}(a[i,j])$: ..**b**...
y el número promedio de actualizaciones $c[i,j]=$...**c**..

a) $\sum_{i=1}^n (1) = n \checkmark$

b) $\sum_{i=1}^n \left(\sum_{j=i+1}^n (1) \right) = \sum_{i=1}^n (n - i - 1 + 1) = \sum_{i=1}^n (n - i) = \sum_{i=1}^n n - \sum_{i=1}^n i = n^2 - \frac{n \cdot (n+1)}{2} =$

$$n^2 - \frac{n^2 + n}{2} \checkmark$$

c) $\sum_{i=1}^n \left(\sum_{j=i+1}^n (0.5 \cdot 1) \right) = \sum_{i=1}^n \left(\sum_{j=i+1}^n \frac{1}{2} \right) = \sum_{i=1}^n \frac{1}{2} \cdot (n - i) = \frac{1}{2} \left(n^2 - \frac{n \cdot (n+1)}{2} \right) =$

$$\frac{n^2}{2} - \frac{(n \cdot (n+1))}{4} \checkmark$$

2) [1,5] Dado el algoritmo:

```
cont=0
for i=1 to n
    for j=1 to n
        if par(i+j)
            cont++
```

Contar el número promedio de asignaciones $\text{cont}++$: ...**a**..
el número promedio de comparaciones $\text{par}(i+j)$: ...**b**..
Obtener el orden exacto del algoritmo: ...**c**..

Y dado el algoritmo:

```
cont=0
for i=1 to n
    for j=1 to n
        if (par(a[i]) OR impar(a[j]))
            cont++
```

Contar el número promedio de asignaciones $\text{cont}++$: ...**d**..
Obtener el orden exacto del algoritmo: ...**e**..

a) $\sum_{i=1}^n \left(\sum_{j=1}^n (0.5 \cdot 1) \right) =$

$$\sum_{i=1}^n \left(\frac{1}{2} n \right) = \frac{1}{2} n^2 \checkmark$$

b) $\sum_{i=1}^n \left(\sum_{j=1}^n (1) \right) = n^2 \checkmark$

c) $\Theta(1) = 1 + \sum_{i=1}^n \left(\sum_{j=1}^n \left(1 + \frac{1}{2} \right) \right) =$

$$1 + \frac{3}{2} n^2 \checkmark$$

d) $\sum_{i=1}^n \left(\sum_{j=1}^n (0.75 \cdot 1) \right) = \frac{3}{4} n^2$

e) $\Theta(1) = 1 + \frac{7}{4} n^2$

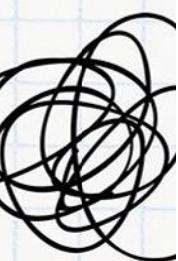
Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

XXXXX
XXXXX

1. a) (2 puntos) Dado el código:

```

for i=1 to n
    suma=0
    if par(i)
        j=1
        while j ≤ i
            suma+= a[i,j]
            j++
        endwhile
    else
        j= i
        while j ≤ n
            suma+= a[i,j]
            j++
        endwhile
    endif
endfor

```

donde aarray[1..n,1..n], estudiar los órdenes O, Omega y exacto de su tiempo de ejecución y el o-pequeño del tiempo promedio.

$$\Theta(n) = \sum_{i=1}^n \left(2 + t_{IF}(i) \right) = \sum_{i=1}^n \left(2 + \sum_{k=1}^{i-1} (6 + 3(n+2)) \right)$$

$$t_{IF}(i) = \begin{cases} 1 + \sum_{j=1}^i 3, & i \text{ PAR} \\ 1 + \sum_{j=i}^n 3, & i \text{ IMPAR} \end{cases}$$

| K | i |
|---|------|
| 1 | 1, 2 |
| 2 | 3, 4 |
| 3 | 5, 6 |

$$\begin{aligned}
 i=1 & \dots 2 + 1 + \sum_{j=1}^n 3 &] 6+3(n-1) \\
 i=2 & \dots 2 + 1 + \sum_{j=1}^2 3 &] 6+3(n-2) \\
 i=3 & \dots 2 + 1 + \sum_{j=1}^3 3 &] 6+3(n-3) \\
 i=4 & \dots 2 + 1 + \sum_{j=1}^4 3 &] 6+3(n-4)
 \end{aligned}$$

2) Tenemos datos enteros en un array a con índices 1 a n, y el algoritmo:

```

if a[1]>a[2]
    max=a[1]
else
    max=a[2]
endif
cont=0
for i=3 to n
    if a[i]>max
        cont++
    endif
endfor

```

Contar el número promedio de ejecuciones de la sentencia cont++:

Solución:

$$a) \sum_{i=3}^n \left(\frac{1}{3} \cdot 1 \right) = \sum_{i=3}^n \frac{1}{3} = \frac{1}{3} (n-3+1) = \frac{1}{3} (n-2) = \frac{n-2}{3}$$

$$b) \Theta(n) = 1 + 0.5 + 0.5 + 1 + \sum_{i=3}^n 1 + \frac{1}{3} = 3 + n + \frac{n-2}{3}$$

```

encontrado:=falso
para i=1..n-1 hacer
    si x[i]=y[1] entonces
        si x[i+1]=y[2] entonces
            encontrado:=verdadero
            lugar:=i
        finsi
    finsi
    i:=i+1
finpara

```

Calcular su tiempo promedio de ejecución suponiendo una probabilidad p de que dos elementos cualesquiera (de x o de y) sean iguales.

operación Una (Imagen[n, n], m)

i:=m+1
repetir
 para j:=m+1 hasta n-m hacer
 Otra (Imagen, i, j, m)
 finpara
 i:=i+1
hasta i > n-m

operación Otra (Imagen[n, n], x, y, tam)
acum:=0
para i:=y-tam hasta y+tam hacer
 para j:=x-tam hasta x+tam hacer
 acum:=acum+Imagen[i, j]
 finpara
finpara

$$t_p(n) = 1 + \sum_{i=1}^{n-1} (1 + p \cdot (1 + 2p) + 1) =$$

$$1 + \sum_{i=1}^{n-1} (1 + p + 2p^2 + 1) =$$

$$1 + \sum_{i=1}^{n-1} (2p^2 + p + 1) =$$

$$1 + \int_0^{n-1} 2p^2 + p + 1 \, dp = 1 + \left[\frac{2p^3}{3} + \frac{p^2}{2} + p \right]_0^{n-1}$$

$$1 + \frac{2(n-1)^3}{3} + \frac{(n-1)^2}{2} + n - 1$$

$$t_p(n) = 1 + \sum_{i=m+1}^{n-m} \left(\sum_{j=m+1}^{n-m} (t_{\text{OTRA}}(n)) + 1 \right)$$

$$t_{\text{OTRA}}(n) = 1 + \sum_{i=y-tam}^{y+tam} \left(\sum_{j=x-tam}^{x+tam} 1 \right) = 1 + \sum_{i=y-tam}^{y+tam} (x+tam - (x-tam) + 1) =$$

$$1 + \sum_{i=y-tam}^{y+tam} (2tam + 1) = 1 + (2tam + 1) \cdot (2tam + 1) = 1 + 4tam^2 + 4tam + 1 =$$

$$4tam^2 + 4tam + 1 \rightarrow 4m^2 + 4m + 1$$

$$t_p(n) = 1 + \sum_{i=m+1}^{n-m} \left(\sum_{j=m+1}^{n-m} (4m^2 + 4m + 1) \right) = 1 + \sum_{i=m+1}^{n-m} ((n-m) - (m+1) + 1)(4m^2 + 4m + 1) = 1 + \sum_{i=m+1}^{n-m} ((n-2m)(4m^2 + 4m + 1)) =$$

$$\left[1 + (n-2m)(n-2m)(4m^2 + 4m + 1) \right]$$

PROFESOR PRÁCTICAS

chb4@um.es

miércoles 15:30 - 17:15

jueves 15:30 - 17:15

Jueves 6 Febrero 2025

SESIÓN 02: notación asintótica

Ejercicio 7.17 Llamamos t al tiempo de ejecución de un algoritmo, t_m al tiempo en el caso más favorable, t_M el tiempo en el caso más desfavorable, y t_p al tiempo promedio. Decir si son ciertas o falsas las siguientes afirmaciones:

- a) $t_p \in O(t_M)$ Verdadero ✓
- b) $t \in \Omega(t_M)$ Falso ✓
- c) $t_m \in O(t_M)$ Verdadero ✓
- d) $t_m \in \Theta(t_M) \Rightarrow t_p \in \Theta(t_m)$ Verdadero ✓
- e) $t_m \in \Omega(t_M) \Rightarrow t \in \Theta(t_m)$ Verdadero ✓
- f) En cualquier algoritmo $t_m \neq t_M$ Falso ✓
- g) $(t_M)^2 \in \Omega(t_M)$ Verdadero ✓
- h) $(t_p)^2 \in \Omega(t_p)$ Verdadero ✓
- i) $t_p + t_m + t_M \in \Theta(t_M)$ Verdadero ✓
- j) $t_p \in \Theta\left(\frac{t_M+t_m}{2}\right)$ Falso ✓

si preguntan verdadero
y falso responder argumentando
con una demostración o
un contrarejemplo !!



3) [1] a) Si para cierto algoritmo tenemos que $t_m(n) \in O(n^2)$ y $t_M(n) \in \Omega(n^3)$. ¿Qué podemos decir de su tiempo de ejecución?

..... no hay orden exacto porque las cotas no coinciden ✓

b) Si para cierto algoritmo tenemos que $t_m(n) \in \Omega(n^2)$ y $t_M(n) \in \theta(n^3)$. ¿Qué podemos decir de su tiempo promedio?

..... $t_p \in \Omega(n^2)$ y $t_p \in O(n^2)$ ✓

..... equivale a $O(\sqrt{n} \cdot \log n)$

..... equivale a $2 \cdot \log n$

4) [1] Ordenar con la relación de inclusión los siguientes órdenes:

$O(\log \log n), O(\sqrt{n} \ln h), O(\log^2 n), O(\log n^2), O(n)$
 $O(n) > O(\sqrt{n} \cdot \ln n) > O(\log^2 n) > O(\log n^2) > O(\log \log n)$ ✓

Ordenar los Ω de las mismas funciones

..... al revés ✓

Ordenar los órdenes exactos de las mismas funciones

..... no se puede ✓ o no ser que hubiera
algunos iguales

5) Si conocemos el o -pequeño del tiempo promedio de un algoritmo, $t_p(n) \in o(f(n))$, de cual de los siguientes órdenes podemos saber algo?

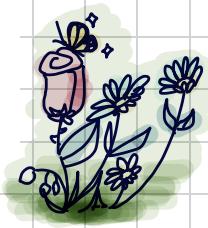
Del tiempo del algoritmo, t, a) O , b) Ω , c) θ , d) o ,

del tiempo en el caso más desfavorable, t_M , e) $O(\Omega, g) \theta, h) o, t_M \in \Omega(f(n))$

del tiempo en el caso más favorable, t_m , f) $O, j) \Omega, k) \theta, l) o, t_m \in O(f(n))$

del tiempo promedio, t_p , m) $O, n) \Omega(o) \theta(p)o t_p(n) \in O \dots$

De los que se puede decir algo, ¿qué se puede decir?



4) Tenemos un array de tamaño n con los datos ordenados

¿Cuál es el Ω de la búsqueda secuencial con centinela? ... 1 ✓

¿Cuál es el O de la búsqueda secuencial con centinela? ... $n+1 \rightarrow n$ ✓

¿Cuál es el Ω de la búsqueda binaria? ... 1 .. ✓

¿Cuál es el O de la búsqueda binaria? $\log(n)$ ✓

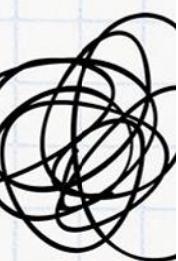
Importante

Puedo eliminar la publi de este documento con 1 coin

→ Plan Turbo: barato
¿Cómo consigo coins?

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

aaaaah

- 4) Si el tiempo de ejecución de un algoritmo sabemos que es $\frac{n^2}{2} + \frac{3n}{3} + n \log n^{n-1} \leq t(n) \leq n^2 + 3^n + n^{\log n^n}$

Dar un buen valor para el Ω del algoritmo $\Omega(n \log n)$ ✓

Dar un buen valor para el O del algoritmo $O(n \log n)$ ✓

¿Cuál es su θ ? ... no hay exacto ✓

... same. ✓

... Y su o-pequeña? ... same. ✓

$(\log n)^2 > 2 \cdot \log n$

no hay orden exacto porque

Ω y O se diferencian en una variable ($\frac{1}{n}$) y
no una constante

- 5) Ordenar con la relación de inclusión los siguientes órdenes, indicando los que son iguales:

$O(\ln n \log n)$, $O(\log^2 n)$, $O(\log \sqrt{n})$, $O(\sqrt{\log \sqrt{n}})$, $O(\ln n \log n + \log^2 n + \log \sqrt{n} + \sqrt{\log \sqrt{n}})$

Ordenar los Ω de las mismas funciones

$1 > 5 > 4 > 3 > 2$

... al revés

i:=1

j:=n

mientras i<j hacer

 si a[i]<a[j] entonces

 i:=i*2

 si no

 j:=j/2

finsi

finmientras

a) Calcular su orden exacto. $\Theta(\log n)$ → va reduciéndose a la mitad la distancia entre i y j

b) Calcular el orden exacto en el caso en que se sustituyan las sentencias $i:=i*2$ y $j:=j/2$ por $i:=i+1$ y $j:=j-1$, respectivamente.

$\Theta(n)$

Ejercicio 7.6 Dado el siguiente algoritmo para obtener el segundo mayor elemento en un array:

max1 := -∞

max2 := -∞

para i:=1...n hacer

 si a[i]>max1 entonces

 max2 := max1

 max1 := a[i]

 si no

 si a[i]>max2 entonces

 max2 := a[i]

 finsi

 finsi

finpara

a) Obtener Θ del algoritmo obteniendo O y Ω . $\Theta(n)$

Ejercicio 7.14 Dado el esquema:

```
para j:=1...n hacer
    para i:=1...j hacer
        si a[i,j]<a[1,j] entonces
            a[1,j]:=a[i,j]
        finsi
    finpara
    para i:=1...n hacer
        si a[1,i]<a[1,1] entonces
            a[1,1]:=a[1,i]
        finsi
    finpara
```

$$\sum_{j=1}^n \sum_{i=1}^j : \frac{n(n+1)}{2} = n^2 = \Omega = O = \Theta$$

Estudiar su O y Ω , cotas para la constante en la notación O , y Θ del número promedio de asignaciones de elementos del array.

```
para i:=1...n hacer
    j:=i+1
    mientras j≤n Y a[j]<a[i] hacer
        a[i]:=a[j]
        j:=j+1
    finmientras
finpara
```

- Obtener O y Ω del programa.
- Obtener Θ del número promedio de instrucciones.

Ejercicios por mi cuenta (10/2/25)

1. **1 - C, A** (Clase) Obtener el tiempo de ejecución $t(n)$, $O(t(n))$, $\Omega(t(n))$ y $\Theta(t(n))$ para el siguiente algoritmo de multiplicación de matrices:

```

for i=1 to n
    for j=1 to n
        suma= 0
        for k=1 to n
            suma= suma + a[i, k] · b[k, j]
        endfor
        c[i, j]= suma
    endfor
endfor

```

$$\begin{aligned}
 t(n) &= \sum_{i=1}^n \left(\sum_{j=1}^n \left(2 + \sum_{k=1}^n (1) \right) \right) = \sum_{i=1}^n \left(\sum_{j=1}^n (2 + n) \right) = \sum_{i=1}^n \left(\sum_{j=1}^n 2 + \sum_{j=1}^n n \right) : \\
 \sum_{i=1}^n (2n + n^2) &= \sum_{i=1}^n 2n + \sum_{i=1}^n n^2 = 2n^2 + n^3
 \end{aligned}$$

$$O(t(n)) = n^3 \quad \Omega(t(n)) = n^3$$

En este caso, $t(n)$ es una función, pues no hay peor ni mejor caso.

2. **2/3 - C, A** (TG 7.1) Obtener O y Ω , y Θ del tiempo promedio para el algoritmo de búsqueda binaria.

```

i=1
j=n
repeat
    m=(i+j) div 2 → media de i y j
    if a[m]>x
        j=m-1
    else
        i=m+1
    endif
until i>j or a[m]=x

```

En este caso $t(n)$ es un conjunto de funciones.

Justifica por qué no es correcto (hablando matemáticamente) definir el tiempo de ejecución $t(n)$ para todos los casos de este algoritmo como una función de N en \mathbb{R}^+ . En consecuencia, ¿cómo se debe definir el tiempo para tener funciones de N en \mathbb{R}^+ ?

$$t(n) = \begin{cases} 2, & n \leq 1 \\ \log(n), & n > 1 \end{cases} \rightarrow O(n) = \log(n) \quad \Theta = \text{no porque } \Omega(n) \neq O(n)$$

$$\Omega(n) = 1$$

4. **2 - C, A** (EX) Decir si son ciertas o falsas las siguientes afirmaciones, razonando la respuesta:

- $\Theta(\log_2 n) = \Theta(\log_{10} n)$ V
- $O(2^n) = O(2^{n+3})$ V
- $O(n^{n+2}) = O(n^{n+3})$ F
- $O(m*n) \subseteq O(2^n)$ F

$$\lim_{n \rightarrow \infty} \frac{2^{n+3}}{2^n} = 2^3 = 8$$

$$\lim_{n \rightarrow \infty} \frac{n^{n+3}}{n^{n+2}} = \infty$$

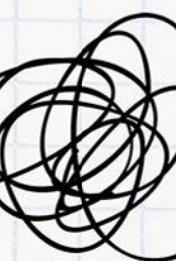
Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

XXXXXX

6. [2/3 - C, A] Calcular el tiempo de ejecución y el orden exacto de este algoritmo

```
p = 0
for i = 1 to n
    p = p + i*i
    for j = 1 to p
        write (a[p, j])
    endfor
endfor
```

$p = 1 \quad p = 1+4 \quad p = 5+9 \quad p = 14+16 \quad p = \dots + n \cdot n$

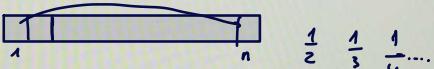
1 5 14 30 ...
+4 +9 +16 ...
2² 3² 4²
 $p + i^2$

$$t(n) = \sum_{i=1}^n \left(\sum_{j=1}^i i^2 \right) = n \cdot \int_0^n i^2 di = n \cdot \left[\frac{i^3}{3} \right] = n \cdot \frac{n^3}{3}$$

la suma de cuadrados crece en orden $O(n^3)$, esto para n veces.

15. [2 - C, A] (TG 6.3) Dado el programa:

```
i=1
while i ≤ n
    if a[i] ≥ a[n]
        a[n]= a[i]
    endif
    i= i * 2
endwhile
```



Calcular:

- Su orden exacto. En caso de aparecer un orden condicionado eliminar la restricción.
- El número promedio de asignaciones en el array.

a) el bucle se ejecuta $\log(n)$ veces porque $i = 1, 2, 4, 8, \dots, n$ $i = 2^k$ hasta llegar a n

b) asignaciones: $\sum_{k=1}^{\log n} \left(\frac{1}{k+1} \right) = \int_0^{\log n} \frac{1}{k+1} dk = \left[\ln(k+1) \right]_0^{\log n} = \ln(\log n + 1) - \ln(1) = \log(\log n)$

9. **1 - C, A, R** (TG 8.2) Calcular el tiempo de ejecución, en función del valor de n , del siguiente programa Pascal. A partir del tiempo, expresar el orden de complejidad.

procedure Algoritmo (x : array [1..MAX, 1..MAX] of real; n : integer)

var

i, j : integer;

a, b, c : array [1..MAX, 1..MAX] of real;

begin

if $n > 1$ then begin

for $i := 1$ to n div 2 do

for $j := 1$ to $n/2$ do begin

$a[i, j] := x[i, j];$

$b[i, j] := x[i, j + n \text{ div } 2];$

$c[i, j] := x[i + n \text{ div } 2, j];$

end;

Algoritmo (a, n div 2);

Algoritmo (b, n div 2);

Algoritmo (c, n div 2);

end;

end;

$$t(n) = \begin{cases} 1, & n = 0; 1 \\ \frac{n^2}{4} + 3t(\frac{n}{2}), & n > 1 \end{cases}$$

$$t(n) = \sum_{i=1}^{\frac{n}{2}} \left(\sum_{j=1}^{\frac{n}{2}} (3) \right) + 3t(\frac{n}{2})$$

$$(\frac{n}{2})^2$$

$$\textcircled{1} \quad t(n) - 3t(\frac{n}{2}) = \frac{n^2}{4}$$

$$\textcircled{1}' \quad t'(k) - 3 \cdot t'(k-1) = \frac{(2^k)^2}{4}$$

$$t'(k) - 3t'(k-1) = \frac{2^{2k}}{4}$$

$$\textcircled{2}' \quad (x-3)(x^2-2) = 0$$

chuleta

$$n = 2^k \quad k = \log n$$

$$t'(k) := t(n) = t(2^k)$$

3. **2 - C, A, R** Demostrar que siendo $a, b, c \in \mathbb{R}^+$ y $d, n_0 \in \mathbb{N}$, con $d > 0$ y:

$$f(n) = \begin{cases} c & \text{Si } n < n_0 \\ a \cdot f(n-d) + b & \text{En otro caso} \end{cases}$$

Se cumple que:

SI $a = 1 \Rightarrow f \in O(n)$

SI NO $\Rightarrow SI a^{1/d} \leq 1 \Rightarrow f \in O(1)$ ✓

SI NO $\Rightarrow f \in O(a^{n/d})$ ✓

para $a = 1$

$$\textcircled{1} \quad f(n) - a \cdot f(n-d) = b$$

$$\textcircled{1}' \quad f(n) - f(n-d) = b \cdot 1^n$$

$$\textcircled{2} \quad f'(k) - a \cdot f'(k-1) = b$$

$$\textcircled{2}' \quad (x-1)(x-1) = 0$$

$$\textcircled{3} \quad (x-a)(x-1) = 0$$

$$\textcircled{3}' \quad x = 1, 1$$

$$\textcircled{4} \quad x = a, 1$$

$$\textcircled{4}' \quad f(n) = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n$$

$$\textcircled{5} \quad f(n) = c_1 \cdot a^{n/d} + c_2 \cdot 1^{n/d}$$

$$\textcircled{5}' \quad O(f(n)) = n$$

chuleta

$$n = d \cdot k \quad k = \frac{n}{d}$$

$$f'(k) := f(n) = f(b \cdot k)$$

8. **[2 - C, A, R]** (Clase) Demostrar que siendo $a, b, c, d \in \mathbb{R}^+$ y $e, n_0 \in \mathbb{N}$, con $e > 0$ y:

$$f(n) = \begin{cases} d & \text{Si } n < n_0 \\ a \cdot f(n-e) + bn + c & \text{En otro caso} \end{cases}$$

Se cumple que:

SI $a = 1 \Rightarrow f \in O(n \cdot \log n)$

SI NO \Rightarrow SI $a^{1/e} \leq 1 \Rightarrow f \in O(n)$
SI NO $\Rightarrow f \in O(n^{\log_e a})$

OJO: está mal, los resultados que pide demostrar saldrían para $f(n/2)$, no para $f(n-e)$; de todos modos, podéis resolver para $f(n-e)$ y ver qué sale ;-

Estudiar que pasaría si alguna de las constantes utilizadas (a, b, c, d ó e) fuera negativa.

$$\textcircled{1} \quad f(n) - a \cdot f(n-e) = bn + c$$

para $a \neq 1$

para $a = 1$

$$f'(n) - a^e f(n-1) : (b \cdot (n-e) + c) \cdot 1^e$$

chuletillo

$$n = k \cdot e \quad k = \frac{n}{e}$$

$$t'(n) := t(n) = t(n/e)$$

$$(x-a)(x-1)^2 = 0$$

$$x = a, 1, 1$$

$$f'(n) = c_1 \cdot a^n + c_2 \cdot 1^n + c_3 \cdot n \cdot 1^n$$

$$f(n) = c_1 \cdot a^{n/e} + c_2 \cdot 1^{n/e} + c_3 \cdot \frac{n}{e} \cdot 1^{n/e}$$

$$(x-1)(x-1)^2 = 0$$

$$x = 1, 1, 1$$

$$f(n) = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n + c_3 \cdot n^2 \cdot 1^n$$

$$f(n) = c_1 + c_2 \cdot \frac{n}{e} + c_3 \cdot \left(\frac{n}{e}\right)^2$$

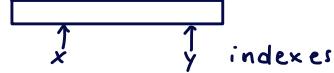
para $a = 1 \rightarrow f \in O(n^2)$

para $a^{1/e} \leq 1 \rightarrow f \in O(n)$

para $a^{1/e} > 1 \rightarrow f \in O(n^{2/e})$

11. **[2/3 - C, A, R]** Calcular una buena cota superior de la complejidad del siguiente algoritmo:

```
procedure dos (a: array [1..n] of integer; x, y: integer)
  if x ≠ y
    if 2 · a[x] < a[y]
      dos (a, x, y div 2) log(n)
    else
      dos (a, 2x, y) log(n)
    endif
  endif
```



$$t(n) \in O(\log n)$$

16. **[1/2 - A]** (EX) Para cada una de las siguientes afirmaciones, decir justificadamente si son ciertas o falsas (se incluye solución parcial y/o pista, faltaría completar):

- $\Omega(f(n)+g(n)) = \Omega(\min(f(n), g(n))) \rightarrow$ falso, ver propiedad 4
- $t(n+1) \in \Theta(t(n))$ para toda función $t: \mathbb{N} \rightarrow \mathbb{R}^+ \rightarrow$ falso, buscar contraejemplo
- $\Theta(2n^2 + 5n - 1) = \Theta(n^2) \rightarrow$ verdadero

$$1) \quad \Omega(f(n) + g(n)) = \Omega(\max(f(n), g(n)))$$

$$2) \text{ contraejemplo } t(n) = \begin{cases} 1, & n < 1 \\ n^2 + 3, & n \geq 1 \end{cases} \quad t(1) = 1 \quad t(1+1) = n^2 + 3$$

$$3) \quad \Theta(2n^2 + 5n - 1) \text{ max } \rightarrow n^2 = \Theta(n^2)$$

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
loquito yo...

X X X X X

19. **1 - C, A** Encontrar O y Ω del algoritmo:

```

max = 0
for i = 1 to n
    cont = 1
    j = i + 1
    while a[i] ≤ a[j] and j ≤ n
        j = j + 1
        cont = cont + 1
    endwhile
    if cont > max
        max = cont
    endif
endfor

```

peor caso : $O(n^2)$, si el while se ejecutara n veces

mejor caso (Ω) : $\Omega(n)$, si el while se ejecuta 1 vez.

20. **1/2 - C, A** (TG 7.3) Calcular la Θ pequeña del número promedio de asignaciones del algoritmo:

```

max = a[1]
for i = 2 to n
    if a[i] > max
        max = a[i]
    endif
endfor

```

$$\text{Asignaciones: } 1 + \sum_{i=2}^n \left(\frac{1}{i+1} \right)$$

$$\downarrow$$

$$\int_1^n \frac{1}{i+1} di = \left[\ln(i+1) \right]_1^n = \ln(n+1) - \ln(2) \approx \ln(n)$$

Sesión 03: recursividad

pág nº 2:

- 5) [1,5] Dadas las ecuaciones de recurrencia siguientes, ¿Cuál será el orden exacto de los tiempos de ejecución correspondientes?

$$t(n) = 2t(n-2) + 2^{\frac{n}{2}} \quad t(n) - 2t(n-2) - (\sqrt{2})^n \quad (x^2 - 2)(x - \sqrt{2}) = 0 \quad x = \sqrt{2}, \sqrt{2}, -\sqrt{2}$$

$$t(n) = 2t(n-2) + 2^{\frac{n}{2}} + 2^n n \quad t(n) - 2t(n-2) = 2^n + 2^n n \quad (x^2 - 2)(x - \sqrt{2})(x - 2)^2 = 0 \quad x = \sqrt{2}, \sqrt{2}, -\sqrt{2}, 2, 2$$

$$t(n) = 2t\left(\frac{n}{2}\right) + 2^{\frac{n}{2}} \quad \text{se hace por expansión de recurrencia * (abajo)}$$

- 6) [1] El tiempo de ejecución de un algoritmo tiene la siguiente ecuación característica:

$$(x - 3)^2(x - 4) = 0 \quad x = 3, 3, 4 \quad t(n) = c_1 \cdot 3^n + c_2 \cdot n \cdot 3^n + c_3 \cdot 4^n$$

y los casos base $t(i) = \log i$ con $1 \leq i \leq 4$

¿Cuál es el orden exacto del algoritmo? $t(n) \in \Theta(4^n)$

¿En qué influyen los casos base?

- 7) [3] Suponemos un array a con índices de 1 a n , y el algoritmo:

alg(int a[],int pos):

 si $\text{pos} > 1$

 si par(a[pos])

 alg(a,pos/2)

 en otro caso

 alg(a,pos/4)

Estudiar los órdenes Ω , O , θ y ω -pequeño del tiempo de ejecución y de la ocupación de memoria en los casos más favorable, más desfavorable y promedio, suponiendo que la primera llamada es $\text{alg}(a,n)$. Justificar en qué casos es posible eliminar la restricción de que n sea potencia de dos.

$$* \quad t(n) = 2 \cdot t\left(\frac{n}{2}\right) + 2^{\frac{n}{2}} = 2 \cdot \left[2 \cdot t\left(\frac{n}{2^2}\right) + 2^{\frac{n}{2^2}} \right] + 2^{\frac{n}{2}} = 2^2 t\left(\frac{n}{2^2}\right) + 2 \cdot 2^{\frac{n}{2^2}} + 2^{\frac{n}{2}} =$$

$$2^2 \left[2 t\left(\frac{n}{2^3}\right) + 2^{\frac{n}{2^3}} \right] + 2 \cdot 2^{\frac{n}{2^2}} + 2^{\frac{n}{2}} = 2^3 t\left(\frac{n}{2^3}\right) + 2^2 \cdot 2^{\frac{n}{2^3}} + 2 \cdot 2^{\frac{n}{2^2}} + 2^{\frac{n}{2}} \rightarrow O(t(n))$$

este n se va haciendo grande, por tanto, el término de mayor orden es $2^{\frac{n}{2}}$

pdf n°3

6) Dadas las ecuaciones de recurrencia siguientes, ¿Cuál será el orden exacto de los tiempos de ejecución correspondientes?:

$$t(n) = 4t(n-2) + n + 1 \dots \Theta(2^n)$$

$$t(n) = 4t(n-2) + 1 \dots \Theta(2^n)$$

$$t(n) = 4t(n-2) + 2^n n \dots \Theta(n^2 2^n)$$

$$t(n) - 4t(n-2) = (n+1) \cdot 1^n \quad (x^2 - 4)(x-1)^2 = 0 \quad t(n) = c_1 \cdot 2^n + c_2 \cdot (-2)^n + c_3 \cdot 1^n + c_4 \cdot n \cdot 1^n \checkmark$$

$$t(n) - 4t(n-2) = 1 \quad (x^2 - 4)(x-1) = 0 \quad t(n) = c_1 \cdot 2^n + c_2 \cdot (-2)^n + c_3 \cdot 1^n \checkmark$$

$$t(n) - 4t(n-2) = 2^n n \quad (x^2 - 4)(x-2)^2 \quad t(n) = c_1 \cdot 2^n + c_2 \cdot (-2)^n + c_3 \cdot n \cdot 2^n + c_4 \cdot n^2 \cdot 2^n \checkmark$$

pdf n°1

Ejercicio 8.5 Resolver la ecuación recurrente: $t(n) - 2t(n-1) = n + 2^n$, con $n > 0$, y $t(0) = 0$.

$$t(n) - 2t(n-1) = n + 2^n \quad (x-2)(x-1)(x+1) = 0 \quad x = 2, 2, 1, -1$$

$$t(n) = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n + c_3 \cdot 1^n + c_4 \cdot n \cdot 1^n$$

vamos a resolver las constantes

$$\begin{cases} t(0) = 0 = c_1 + c_3 \\ t(1) = 3 = 2c_1 + 2c_2 + c_3 + c_4 \\ t(2) = 12 = 4c_1 + 8c_2 + c_3 + 2c_4 \\ t(3) = 35 = 8c_1 + 24c_2 + c_3 + 3c_4 \end{cases}$$

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 2 & 2 & 1 & 1 & 3 \\ 4 & 8 & 1 & 2 & 12 \\ 8 & 24 & 1 & 3 & 35 \end{array} \right] \xrightarrow{\substack{F_2 - 2 \cdot F_1 \\ F_3 - 4 \cdot F_1}} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 4 & 8 & 1 & 2 & 12 \\ 8 & 24 & 1 & 3 & 35 \end{array} \right]$$

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 0 & 0 & 1 & -2 & 0 \\ 0 & 24 & -7 & 3 & 35 \end{array} \right] \xrightarrow{F_4 - 12F_2} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 0 & 0 & 1 & -2 & 0 \\ 8 & 24 & 1 & 3 & 35 \end{array} \right] \xrightarrow{F_3 - 4F_2} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 0 & 0 & 1 & -2 & 0 \\ 8 & 24 & 1 & 3 & 35 \end{array} \right]$$

$$\downarrow F_4 - 12F_2$$

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 0 & 0 & 1 & -2 & 0 \\ 0 & 0 & 5 & -9 & -1 \end{array} \right] \xrightarrow{F_4 - 5F_3} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & -1 & 1 & 3 \\ 0 & 0 & 1 & -2 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{array} \right] \begin{cases} c_1 + c_3 = 0 \\ 2c_2 - c_3 + c_4 = 3 \\ c_3 - 2c_4 = 0 \\ c_4 = -1 \end{cases} \begin{cases} c_4 = -1 \\ c_3 = 2c_4 = -2 \\ c_2 = \frac{3-2+1}{2} = 1 \\ c_1 = -c_3 = 2 \end{cases}$$

$$t(n) = 2 \cdot 2^n + n \cdot 2^n - 2 - n$$

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

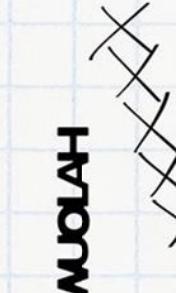
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...



23. **1/2 - C, A** Calcular O, Ω y Θ para el siguiente programa:

```
i = 1           f ∈ O(n)
j = 1
while i ≤ n and j ≤ n
    if a[i, j + 1] < a[i + 1, j]   f ∈ Ω(n)
        j = j + 1
    else
        i = i + 1
    endif
endwhile
```

*El máx de iteraciones
es 2n*

EXAMEN MARZO

para practicar

Dom 16 Febrero 2025

1) (3 puntos) Dado el código:

```
i = 1; aux=0;
while i<=n
    index = M[1,i];
    k=i+1;
    while k<=n and index!=M[k,i]
        if index < M[k,i], index = M[i,k]; else index = M[k,i]; endif
        j=1;
        while j<= n
            if(M[k,j]>aux), aux--; endif
            j=j*2;
        endwhile
        k++;
    endwhile
    i++;
endwhile
```

donde $M:\text{array}[1..n+1, 1..n+1]$ de enteros, estudiar los órdenes O, Ω y Θ de $t(n)$.

no existe $f(n)$ tal que $t(n) \in \Theta(f(n))$

Hay un bucle que lo abarca todo de 1 a n (n veces).

El bucle de dentro tiene n iteraciones como máximo.

El bucle de dentro se itera $\log n$ veces.

$t \in O(n^2 \log n)$, $t \in \Omega(n)$
 \downarrow
 si no entra al segundo bucle

2) (2 puntos) Resolver las ecuaciones de recurrencia:

- a) $t(n) = 3t(n-1) + 4t(n-2) + n3^n + n^3 + 3^n$
 b) $t(n) = 3t(n-1) + 4t(n-2) + n^3 + n^2 4^{\frac{n}{2}} + 1$

a) $\underbrace{t(n) - 3t(n-1) - 4t(n-2)}_{x^2 - 3x - 4} = n3^n + n^3 + 3^n$

$$(x-4)(x+1)(x-3)^2(x-1)^4 = 0$$

$$\frac{3 \pm \sqrt{9-4(1)(-4)}}{2} \rightarrow \frac{3+5}{2} = 4 \quad \frac{3-5}{2} = -1$$

$$t(n) = c_1 \cdot 4^n + c_2 (-1)^n + c_3 \cdot 3^n + c_4 \cdot 1^n + c_5 \cdot n \cdot 1^n + c_6 \cdot n^2 \cdot 1^n +$$

$$c_7 \cdot n^3 \cdot 1^n = 0$$

b) $t(n) - 3t(n-1) - 4t(n-2) = n^3 + n^2 \cdot 4^{\frac{n}{2}} + 1$

$$(x-4)(x+1)(x+2)^3(x-2)^3(x-1)^4 = 0$$

3) (1 punto) Ordenar los siguientes órdenes de ejecución, así como los Ω y Θ correspondientes, con la relación de inclusión:

$$O(2^{0.1n}), O(n^3 \ln^2 n + (\sqrt{n})^5), O(9^{\frac{n}{2}}), O(\log_{10}(n^2) \ln(n) \log_3 n), O(\log_3 n^{\frac{5}{2}})$$

ordenación órdenes:

$$O(2^{0.1n}), O(n^3 \cdot (\log(n))^2), O(9^{\frac{n}{2}}), O(2 \cdot (\log(n))^3), O(\frac{5}{2} \log(n))$$

↓ ↓
close to 0 más de la suma simplified

$$O(9^{\frac{n}{2}}) >> O(n^3 \cdot (\log(n))^2) >> O((\log(n))^3 \cdot 2) >> O(\frac{5}{2} \log(n)) >> O(2^{0.1n})$$

la ordenación de Ω se hace al revés

no se pueden ordenar los Θ

4) (1 punto) Indica si las siguientes afirmaciones son verdaderas o falsas para cualquier algoritmo:

- a) Si $t_M(n) \in \Theta(n^n)$ y $t_m(n) \in \Omega(2^n)$ entonces $t(n) \in \Theta(n!)$
- b) $t_m(n) \in \Omega(t_p(n))$ implica $t_M(n) \in O(t_p(n))$

a) falso, para saber el Θ , el Ω y O deberían coincidir, no tiene porque ser $\Theta(n!)$

b) si $t_m(n) \in \Omega(t_p(n))$ entonces $t_m(n) = t_p(n)$, por tanto, el $t_m(n)$ también debería coincidir con $t_p(n)$. Verdadero.

| | | |
|----------|----------|--------------|
| P | P | V |
| P | \vdots | \checkmark |
| \vdots | P | X |

5) (3 puntos) Dada la función:

```
int g21(int P[], int Q[], int m):int
    cont=0;
    i=1; while i<=m, i=i*2; endwhile
    if(m>8)
        j=0;
        if( par(P[m]) OR impar(Q[m]) )
            for i=1 to 4
                cont+= g21(P,Q,m/2);
            endfor
        else
            for i=1 to 16
                cont+= g21(Q,P,m/4);
            endfor
        endif
    return cont;
```

$$t_p(n) = \begin{cases} \log(n), & n \leq 8 \\ \log(n) + \frac{3}{4} \left(\frac{1}{3} t(n/2) + \frac{1}{4} t(n/4) \right), & n > 8 \end{cases}$$

$$\boxed{n=2^k \quad k=\log_2 n \\ t'(k) := t(n)=t(2^k)}$$

$$t(n) - 3t(n/2) - 4t(n/4) = \log(n)$$

$$t'(k) - 3t'(k-1) - 4t'(k-2) = \underbrace{\log(2^k)}_{k} \cdot 1^k$$

$$x^2 - 3x - 4 \rightarrow 4$$

$\downarrow -1$

$$(x-4)(x+1)(x-1)^2 = 0$$

$$t'(k) = c_1 \cdot 4^k + c_2 \cdot (-1)^k + c_3 \cdot 1^k + c_4 \cdot k \cdot 1^k = 0$$

$$t(n) = c_1 \cdot n^2 + c_2 \cdot (-1)^{\log_2 n} + c_3 \cdot 1^{\log_2 n} + c_4 \cdot \log(n) \cdot 1^{\log_2 n}$$

¿se puede eliminar la condición?

$$t(n) \in O(n^2 \mid n=2^k)$$

\uparrow
 j

END $\xrightarrow{j \vee}$ $\xrightarrow{t(n) \vee}$

f es b-armónica?

$$f(b \cdot n) = b \cdot n^2 \in O(n^2) \vee \text{sí}$$

Repaso examen

Jueves 20 Febrero 2025

Un algoritmo trabaja con listas enlazadas. Lista vacía \rightarrow se insertan n elementos. Para cada elemento, se recorre secuencialmente toda la lista y se inserta en la última posición.

$$t(n) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

| progresión
| aritmética

$\Theta(t(n)) = n^2 \rightarrow$ es un orden exacto porque hemos encontrado una función que lo representa

lista : empty
 $i := 1$
elem = 0

for $i <= n$
while $i < elem$
 $i := i + 1$
insert(i)

Ejercicio 6.9 Dado el programa:

```
max := -∞  
para i:=1,...,n hasta  
  para j:=1,...,m hasta  
    si a[i,j] > max       $\frac{1}{(i-1) \cdot m + j}$   
      max := a[i,j]  
    finsi  
  finpara  
finpara
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | j |
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |

[2,3]

Obtener una cota inferior y superior del número de veces que se ejecuta la asignación interna, y calcular el número promedio de veces que se ejecuta dicha instrucción.

Se va a ejecutar mínimo 1 vez $\rightarrow t(\text{asig}) \in \Omega(1)$

Sólo en el caso en el que la matriz esté ordenada ascendenteamente $\rightarrow t(\text{asig}) \in O(n \cdot m)$

Para el caso promedio, habrá que calcular la probabilidad del IF.

$$\sum_{i=1}^{n \cdot m} 1 \cdot \frac{1}{i} = \int_1^{nm+1} \frac{1}{i} di = \ln(nm+1) - \ln(1) = \ln(nm) \approx \log(n \cdot m) \rightarrow \Theta(t(\text{asig}))$$

para $i := 1, \dots, n$ hacer

la prob será $\frac{1}{d!}$

$j := i + 1$

mientras $j \leq n$ y $a[j] < a[i]$ hacer

$a[i] := a[j]$

$j++$

finmientras estudiar bien

finpara

el algoritmo

antes de escribir

Calcular O , Ω de $t(n)$ y Θ de $t_p(n)$.

$t(n) \in O(n^2) \rightarrow$ array ordenado de mayor a menor

$t(n) \in \Omega(n) \rightarrow$ array ordenado menor a mayor

$$t_p(n) = n \cdot \sum_{j=i+1}^n \left(\frac{1}{j} \right) = n \cdot \log n$$

$t_p(n) \in \Theta(n \cdot \cancel{\log n})$

se anularán el orden
serían

$$\sum_{i=1}^n \left[2 + \sum_{j=i+1}^n 3 \right] = 2n + 3n^2 - 3 \sum_{i=1}^n i \rightarrow 2n + 3n^2 - 3 \left(\frac{n(n+1)}{2} \right) = n^2$$

$$t_p(n) = \sum_{i=1}^n \left[2 + \sum_{j=i+1}^n \frac{1}{j} \cdot 3 \right] =$$

8) Suponemos un array bidimensional a con n filas y n columnas (numeradas de 1 a n), y el algoritmo:

```

h(int t,int f,int c):
    si t=1
        return a[f,c]
    en otro caso si pot2(a[f,c])
        return h(t/4,f+t/2,c+t/2)+h(t/4,f+t/2,c+t/2)
    en otro caso
        return h(t/2,f+t/2,c)
    
```

Estudiar el tiempo de ejecución y la ocupación de memoria para los casos más favorable, más desfavorable y promedio si n es potencia de dos. Habrá que indicar qué órdenes de complejidad se pueden obtener en cada caso. Justificar si es posible eliminar la restricción de que n sea potencia de dos para el tiempo de ejecución.

$$t(n) = \begin{cases} 1, & t=1 \\ \text{Si } \text{pot2}(n): 2 \cdot t(\frac{n}{4}) + c \\ \text{else } t(n) = t(\frac{n}{2}) + c \end{cases}$$

$$n = 2^k$$

$$t(2^k) = 2 \cdot t(2^{k-2}) + c$$

$$(x^2 - 2)(x-1) =$$

$$(x - \sqrt{2})(x + \sqrt{2})(x-1)$$

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

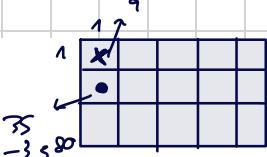
MUERAH

EXAMEN MARZO 2.1

21/12/125

1) (3 puntos) Dado el código:

```
i = 1; aux=0;
while i<=n
    index = M[1,i];
    k=i+1; x
    while k<=n and index!=M[k,i]
        if index < M[k,j], index = M[i,k]; else index = M[k,i]; endif
        j=1;
        while j<= n
            if(M[k,j]>aux), aux--;
            j=j+2;
        endwhile
        k++;
    endwhile
    i++;
endwhile
```



$$t(n) \in \Omega(n)$$

$$t(n) \in O(n^2 \cdot \log n)$$

no existe $f(n)$ tal que $t(n) \in \Theta(f(n))$

donde $M:\text{array}[1..n+1, 1..n+1]$ de enteros, estudiar los órdenes O , Ω y Θ de $t(n)$.

2) (2 puntos) Resolver las ecuaciones de recurrencia:

a) $t(n) = 3t(n-1) + 4t(n-2) + n3^n + n^3 + 3^n$

b) $t(n) = 3t(n-1) + 4t(n-2) + n^3 + n^2 4^{\frac{n}{2}} + 1$

a) $t(n) - 3t(n-1) - 4t(n-2) = n3^n + n^3 + 3^n$

$$(x^2 - 3x - 4)(x-3)^2(x-1)^4 : (x-4)(x+1)(x-3)^2(x-1)^4$$

↓

$$\frac{3 \pm \sqrt{9-4(1)(-4)}}{2} \quad \frac{3+5}{2}=4 \quad \frac{3-5}{2}=-1$$

$$x = 4, -1, 3, 3, 1, 1, 1, 1$$

$$t(n) = c_1 \cdot 4^n + c_2 \cdot (-1)^n + c_3 \cdot 3^n + c_4 \cdot n \cdot 3^n + c_5 \cdot 1^n + c_6 \cdot n \cdot 1^n + c_7 \cdot n^2 \cdot 1^n + c_8 \cdot n^3 \cdot 1^n$$

b) $t(n) - 3t(n-1) - 4t(n-2) = n^3 + n^2 4^{\frac{n}{2}} + 1$

$$(x-4)(x+1)(x-\sqrt{4})(x-1)^4$$

$$(x-4)(x+1)(x-2)^3(x+\cancel{2})^3(x-1)^4$$

3) (1 punto) Ordenar los siguientes órdenes de ejecución, así como los Ω y Θ correspondientes, con la relación de inclusión:

$$O(2^{0.1^n}), O(n^3 \ln^2 n + (\sqrt{n})^5), O(9^{\frac{n}{2}}), O(\log_{10}(n^2) \ln(n) \log_3 n), O(\log_3 n^{\frac{5}{2}})$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$O(\sqrt{2^n}) \quad O(n^3 \cdot (\log n)^2) \quad O((\sqrt{9})^n) \quad O(2 \cdot (\log n)^3) \quad O(25 \cdot \log n)$$

$$O(2^{0.1^n}) \subset O(2.5 \cdot \log n) \subset O(2 \cdot (\log n)^3) \subset O(n^3 \cdot (\log n)^2) \subset O(\sqrt{2^n}) \subset O(3^n)$$

4) (1 punto) Indica si las siguientes afirmaciones son verdaderas o falsas para cualquier algoritmo:

- a) Si $t_M(n) \in \Theta(n^n)$ y $t_m(n) \in \Omega(2^n)$ entonces $t(n) \in \Theta(n!)$
- b) $t_m(n) \in \Omega(t_p(n))$ implica $t_M(n) \in O(t_p(n))$

$$t_M \geq t_p \geq t_m$$

a) $t_M(n) \in \Theta(n^n)$ y $t_m(n) \in \Omega(2^n)$

FALSO Si un algoritmo tiene $\Omega(f(n))$ y $O(f(n))$, entonces $\in \Theta(f(n))$

$$\Omega(n!) \in \Omega(2^n) \text{ y } O(n!) \in O(n^n)$$

b) $t_m(n) \in \Omega(t_p(n))$ implica $t_M(n) \in O(t_p(n))$. FALSO

MARZO 2023 1.4

1) (3 puntos) Dado el código:

```

 $n^3$  for k = 1..n^3
    diag = A[k,n^3-k+1];
    i=n^2;
    while i >= 1 AND impar(A[k,i])
        if diag >= A[i,k], diag = A[i,k]; else diag = A[k,i]; endif
        j = 1;
        while j <= n AND not par(A[j,k])
            if(A[k,j^3]>diag), diag -- ; endif
            j = j*2;
        endwhile
        i--;
    endwhile
endfor
  
```

$\nearrow A \text{ es una matriz}$
 $\nearrow \log n \max$

donde $A:\text{array}[1..n^3, 1..n^3]$ de enteros, estudiar los órdenes O, Ω y Θ de $t(n)$.

Como en los while hay condiciones dobles, existe un mejor y un peor caso.

$$t_{\text{M}}(n) : n^3 \cdot n^2 \cdot \log n : n^5 \cdot \log n \quad t(n) \in O(n^5 \cdot \log n)$$

$$t_{\text{m}}(n) = n^3 \quad t(n) \in \Omega(n^3) \rightarrow \text{no entra en el segundo y tercer bucle.}$$

Como $\Omega(t(n))$ y $O(t(n))$ no coinciden, podemos decir que $t(n)$ no pertenece a un orden exacto.

2) (2 puntos) Resolver las ecuaciones de recurrencia:

- a) $t(n) = 7t(n-1) + 8t(n-2) + 3^n n + n^2 + n + 3^n n^2 + 1$
- b) $t(n) = 7t(n-1) + 8t(n-2) + 2 + 2^n n + n^2 2^{\frac{n}{2}} + n$

$$\text{a)} \quad t(n) - 7t(n-1) - 8t(n-2) = 3^n n + n^2 + n + 3^n n^2 + 1$$

$$(x^2 - 7x - 8) = 1^n (n^2 + n + 1) + 3^n (n + n^2)$$

$$\frac{7 \pm \sqrt{49 - 4(4)(-8)}}{2} \quad \begin{matrix} \nearrow \frac{7+9}{2} = 8 \\ \nearrow \frac{7-9}{2} = -1 \end{matrix}$$

$$(x+1)(x-8)(x-1)^3(x-3)^3 = 0$$

$$x = -1, 8, 1, 1, 1, 3, 3, 3$$

$$t(n) = c_1 \cdot (-1)^n + c_2 \cdot 8^n + c_3 \cdot (1)^n + c_4 \cdot n \cdot 1^n + c_5 \cdot n^2 \cdot 1^n + c_6 \cdot 3^n + c_7 \cdot n \cdot 3^n + c_8 \cdot n^2 \cdot 3^n$$

\downarrow
orden

$$\text{b)} \quad t(n) - 7t(n-1) - 8t(n-2) = 1^n (n+2) + 2^n \cdot n + (\sqrt{2})^n \cdot n^2$$

$$(x+1)(x-8)(x-1)^2(x-2)^2(x-\sqrt{2})^3$$

$$x = -1, 8, 1, 1, 2, 2, \sqrt{2}, \sqrt{2}, \sqrt{2} \quad t(n) = c_1 \cdot (-1)^n + c_2 \cdot 8^n + c_3 \cdot 1^n + c_4 \cdot n \cdot 1^n + c_5 \cdot 2^n + c_6 \cdot n \cdot 2^n + c_7 \cdot (\sqrt{2})^n +$$

$$c_8 \cdot n \cdot (\sqrt{2})^n + c_9 \cdot n^2 \cdot (\sqrt{2})^n$$

MARZO 2023 3.2

1) (3 puntos) Dado el código:

```

for j = 1..n/2
    ff = 1;
    i=n;
    while i >= 1 AND impar(M[k,i])
        if ff >= M[i,k], ff = 0; else ff += M[k,i]; endif
        k = 1;
        while k <=  $\frac{n}{2}$  AND not par(M[j,k])
            if(M[k,j]=ff), ff -- ; endif
            k++;
        endwhile
        i = i/2;
    endwhile
endfor

```

donde M :array[1.. $n+1$, 1.. $n+1$] de enteros, estudiar los órdenes O , Ω y Θ de $t(n)$.

nos damos cuenta de que va a haber peor y mejor caso por las condiciones de los whiles, cuyo resultado depende del valor de la entrada y no solo del tamaño.

$$t_M(n) = \sum_{j=1}^{\frac{n}{2}} \left(2 + \sum_{i=n}^1 (s + 2 \cdot n) \right) = \sum_{j=1}^{\frac{n}{2}} (2 + 3\log n + 2 \cdot \log n \cdot n) = \frac{n}{2} \cdot 2 + 3 \cdot \log n \cdot \frac{n}{2} + 2 \cdot \log n \cdot n \cdot \frac{n}{2}$$

$$t(n) \in O(n^2 \cdot \log n) \quad t(n) \in \Omega(n^2)$$

$$t_M(n) = \sum_{j=1}^{\frac{n}{2}} 2 = 2 \cdot \frac{n}{2} = n$$

No existe $f(n)$ tal que $t(n) \in \Theta(f(n))$

2) (2 puntos) Resolver las ecuaciones de recurrencia:

- a) $t(n) = t(n-1) + 6t(n-2) + 2^n + n^3 + n2^n + 1$
b) $t(n) = t(n-1) + 6t(n-2) + 2 + 2^n n^2 + n + n^2 9^{\frac{n}{2}}$

$$a) \quad t(n) - t(n-1) - 6t(n-2) = 2^n + n^3 + n2^n + 1$$

$$(x^2 - x - 6) : 2^n (1+n) + 1^n (n^3 + 1)$$

$$(x-3)(x+2)(x-2)^2(x-1)^4$$

$$\frac{1 \pm \sqrt{1-4(1)(-6)}}{2} \begin{cases} \frac{1+5}{2} = 3 \\ \frac{1-5}{2} = -2 \end{cases}$$

$$t(n) = c_1 \cdot 3^n + c_2 \cdot 2^n + c_3 \cdot (-2)^n + c_4 \cdot n \cdot 2^n + c_5 \cdot 1^n + c_6 \cdot n \cdot 1^n + c_7 \cdot n^2 \cdot 1^n + c_8 \cdot n^3 \cdot 1^n$$

$$b) \quad t(n) - t(n-1) - 6t(n-2) = 1^n (2+n) + 2^n (n^2) + 3^n (n^2)$$

$$(x-3)(x+2)(x-1)^2(x-2)^3(x-3)^5$$

$$t(n) = c_1 \cdot 3^n + c_2 \cdot (-2)^n + c_3 \cdot 1^n + c_4 \cdot n \cdot 1^n + c_5 \cdot 2^n + c_6 \cdot n \cdot 2^n + c_7 \cdot n^2 \cdot 2^n + c_8 \cdot 3^n + c_9 \cdot n \cdot 3^n + c_{10} \cdot n^2 \cdot 3^n$$

3) (1 punto) Ordenar los siguientes órdenes de ejecución, así como los Ω y Θ correspondientes, con la relación de inclusión:

$$O\left(\log_3 n^{\frac{5}{3}}\right), O(n^3 + n \ln^5 n + 3^n), O(1/n), O\left(n^3 8^{\frac{n}{3}}\right), O(\log_3(n^2) \ln(n) + \log n)$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$O\left(\frac{5}{3} \log n\right) \quad O(3^n) \quad O(n^2 \cdot 2^n) \quad O(2 \cdot (\log n)^2)$$

$$O(1/n) \subset O(\log_3 n^{\frac{5}{3}}) \subset O(\log_3(n^2) \ln(n) + \log n) \subset O(n^3 \cdot 2^n) \subset O(3^n)$$

Ω sería al revés

Θ no se puede

4) (1 punto) Indica si las siguientes afirmaciones son verdaderas o falsas para cualquier algoritmo:

- a) Si $t_p(n) \in \Theta(n^2)$ entonces $t(n) \in O(n^3)$.
- b) $t_M(n) \in O(t_p(n))$ y $t_p(n) \in O(t_m(n))$ implica $t(n) \in \Omega(t_M(n))$.

a) FALSO.

Que $t_p(n) \in \Theta(n^2)$ no implica que no exista un caso mejor y peor, es decir perfectamente puede pasar que $t_M \in \Theta(n^4)$, y entonces la afirmación $t(n) \in O(n^3)$ sería incorrecta, ya que $t(n)$ es tiempo de ejecución en general, engloba a todos los casos.

b) VERDADERO

Si $t_M(n)$ está acotado superiormente por $t_p(n)$, sabiendo que $t_M \geq t_p \geq t_m$, podemos deducir que $t_p = t_M = t_m$. Así que, $t(n) \in \Omega(t_m(n))$ es correcta, ya que $t(n)$, aunque es el tiempo general, ahora mismo no hay diferencia entre peor y mejor caso, el $t(n)$ está acotado tanto superiormente como inferiormente por $t_m(n)$ ($\Theta(t_m(n))$) o $t_p(n)$.

5) (3 puntos) Dada la función:

```
int g32(int M[], int q):int
    cnt=0;
    i=1; while i<=q^2, cnt++; i++; endwhile
    if(q>4)
        if( mod(M[q]), 6 ) = 0 )
            for i=1 to 24
                cnt+= g32(M, q/2);
            endfor
        else
            for i=1 to 6
                cnt+= g32(M, q/4);
            endfor
        endif
    endif
    return cnt;
```

$$t_p(n) = \begin{cases} n^2 & n \leq 4 \\ 3 + 2n + \frac{1}{6} \cdot 24 \cdot t(n/2) + \frac{5}{8} \cdot 8 \cdot t(n/4), & n > 4 \end{cases}$$

$$t_p(n) - 4t(n/2) - 5t(n/4) = 3 + 2n$$

$$t'(k) - 4t'(k-1) - 5t'(k-2) = 3 + 2^{k+1} - 4^k$$

$$(x^2 - 4x - 5)(x-2)(x-1) = 0$$

$$t'(k) = C_1 \cdot 5^k + C_2 \cdot 1^k + C_3 \cdot 2^k + C_4 \cdot (-1)^k$$

$$t(n) = C_1 \cdot 5^{\log_2 n} + C_2 + C_3 \cdot 2^{\log_2 n} - C_4$$

$$\downarrow$$

$$n^{\log_2 5}$$

donde $M: \text{array}[1..n+1]$ de enteros, y siendo la primera llamada con $q = n$, con n potencia de 2, estudiar los órdenes O , Ω , Θ de su tiempo promedio ($t_p(n)$). Indica las condiciones iniciales necesarias para calcular la o-pequeña de $t_p(n)$ (no es necesario calcularla). ¿Es posible quitar la condición de que n sea potencia de 2?

cond. ini (deben ser 4) 2, 4, 8, 16



$t_p(n) \in \Theta(n^{\log_5})$ $t_p(n) \in O(n^{\log_5})$ $t_p(n) \in \Omega(n^{\log_5})$

¿se puede quitar la condición? SÍ!!

$$\begin{cases} t(n) \text{ EMIV} \\ f(n) \text{ EMIV} \\ f(n) \text{ es b-ármónica? } f(b \cdot n) \in O(n^{\log_b}) \end{cases}$$

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

MUERAH

MARZO 2023 2.3

1) (3 puntos) Dado el código:

```

if par(T[1,1]);
    for i=1..n, for j=1..i, tmp++; endfor, endfor
else
    for i=1..n
        k=1; ref = T[1,i];
        while k<=n and ref<=T[k,i]
            for j=1..k
                if(T[k,j]>ref), ref++; endif
            endfor
            k=k*2;
        endwhile
    endfor
endif

```

donde T :array[1..n + 1, 1..n + 1] de enteros, estudiar los órdenes O , Ω y Θ de $t(n)$.

$$t(n)_{par} : \sum_{i=1}^n \left(\sum_{j=1}^i \right) = \sum_{i=1}^n ((i-1)+1) = \sum_{i=1}^n i = \frac{n \cdot (n+1)}{2} = \frac{n^2+n}{2}$$

$$t(n)_{impar} : \begin{matrix} \sum_{i=1}^n & (2 + \log n \cdot n) \\ (\text{peor}) & \downarrow \\ 1+2+4+\dots+n \end{matrix} = 2n + n^2 \log n$$

$$t(n)_{impar} = n \quad (\text{mejor})$$

$t(n) \in \Omega(n)$, $t(n) \in O(n^2 \log n)$, no existe una $f(n)$ tal que $t(n) \in \Theta(f(n))$

2) (2 puntos) Resolver las ecuaciones de recurrencia:

a) $t(n) = t(n-1) + 2t(n-2) + 2^n + n^2 + n2^n$

b) $t(n) = t(n-1) + 2t(n-2) + n + n^2 2^{\frac{n}{2}} + 1$

a) $t(n) - t(n-1) - 2t(n-2) = 2^n + n^2 + n \cdot 2^n$

$(x^2 - x - 2) = 2^n (1 + n) + 1^n (n^2)$

$$\frac{1 \pm \sqrt{1-4(1)(-2)}}{2} \quad \frac{1+3=2}{2} \quad x = 2, 2, 2, 1, 1, 1, -1$$

$$\frac{1-3}{2} = -1 \quad t(n) = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n + c_3 \cdot n^2 \cdot 2^n + c_4 + c_5 \cdot n + c_6 \cdot n^2 - c_7$$

b) $(x^2 - x - 2) = 1^n (1 + n) + (\sqrt{2})^n (n^2)$

$x = 2, -1, 1, 1, \sqrt{2}, \sqrt{2}, \sqrt{2}$

3) (1 punto) Ordenar los siguientes órdenes de ejecución, así como los Ω y Θ correspondientes, con la relación de inclusión:

$O(n^3 \ln^7 n + n^{\frac{5}{3}})$, $O(\log_7 n (\ln n)^3)$, $O(1.5^n)$, $O(8^{\frac{n}{3}})$, $O(\log_3 n^{\frac{4}{3}})$

$O(n^3 (\log n)^7)$, $O((\log n)^4)$, $O(1.5^n)$, $O(2^n)$, $O(\log n)$

$O(\log n) \subset O(\log^4 n) \subset O(n^3 \cdot \log^3 n) \subset O(1.5^n) \subset O(2^n)$

$\Omega(2^n) \subset \Omega(1.5^n) \subset O(n^3 \log^3 n) \subset O(\log^4 n) \subset O(\log n)$

4) (1 punto) Indica si las siguientes afirmaciones son verdaderas o falsas para cualquier algoritmo:

- a) Si $t_m(n) \in \Theta(t_p(n))$ y $t(n) \in O(t_p(n))$ entonces $t_M(n) \in O(t_m(n))$
- b) $t(n) \in \Omega(t_m(n))$ implica $t(n) \in \Theta(t_M(n))$

a) $t_m(n) \in \Theta(t_p(n))$, el tiempo mejor crece igual que el tiempo promedio

$t(n) \in O(t_p(n))$, el tiempo de ejecución está en el orden de $t_p(n)$

Esto implica que, sabiendo que $t_M \geq t_p \geq t_m \rightarrow t_M = t_p = t_m$, ya que el orden del algoritmo no puede ser mayor que el de $t_p(n)$.

Entonces se cumple que $t_m(n) \in O(t_m(n))$ VERDADERO

b) $t(n) \in \Omega(t_m(n)) \rightarrow t(n) \in \Theta(t_M(n))$? FALSO

Esta implicación sólo se cumpliría en caso de que t_m, t_M , y t_p fueran iguales, en cualquier otro caso no.

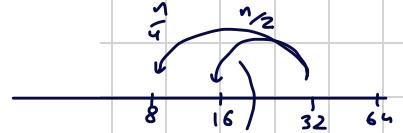
5) (3 puntos) Dada la función:

```
int g23(int Q[], int P[], int p):int
    font=0;
    k=1; while k<=p, k=k*2; endwhile
    if(p<=16)
        for i=1 to p, font++; endfor
    else
        if( mod(Q[p],4)<1)
            for i=1 to 8
                font+= g23(Q,P,p/2);
            endfor
        else
            for i=1 to 4
                font+= g23(P,Q,p/4);
            endfor
        endif
    endif
    return font;
```

determinar
caso

$$t_p(n) = \begin{cases} 3 + \log n + n, & n \leq 16 \\ 3 + \log n + \frac{1}{4} \cdot 8t(n/2) + \frac{3}{4} \cdot 4t(n/4), & n > 16 \end{cases}$$

$$\begin{aligned} n &= 2^k & t_p(n) - 2t(n/2) - 3t(n/4) &= 3 + \log n \\ k &= \log n & t'(k) - 2t'(k-1) - 3t'(k-2) &= 1^k(3+k) \\ 2 \pm \sqrt{4(1)(-3)} &\xrightarrow{\frac{x_1}{2}=3} (x^2 - 2x - 3) & &= 1^k(3+k) \\ 2 &\xrightarrow{\frac{x_2}{2}=-1} x = 3, -1, 1, 1 & & \end{aligned}$$



donde P y Q son array[1..n] de entero, y siendo la primera llamada con $g23(P, Q, n)$, estudiar los órdenes O , Ω , Θ y o-pequeña de su tiempo promedio. Para la o-pequeña, sobre su constante basta con indicar qué valores de n usar para calcularla. ¿Se podría eliminar la condición de que n sea potencia de 2?

$$t'(k) = c_1 \cdot 3^k + c_2 \cdot 1^k + c_3 \cdot k \cdot 1^k - c_4$$

$$t(n) = c_1 \cdot 3^{\log n} + c_2 + c_3 \cdot \log n - c_4$$

$$n^{\log 3}$$

$t_p(n) \in \Omega(n^{\log 3})$, $\in O(n^{\log 3})$, $\in \Theta(n^{\log 3})$

condiciones iniciales: 8, 16, 32, 64

se podría quitar la condición 2^k ?

$$\begin{cases} n^{\log 3} + \text{ENDV} \\ t(n) \text{ ENDV} \\ f(n) \text{ es } b\text{-armónica?} \end{cases}$$

$$f(b \cdot n) \rightarrow f(b \cdot n^{\log 3}) \in O(n^{\log 3})$$

MARZO 2023 2.2

1) (3 puntos) Dado el código:

```

tmp=0;
for i=1..n
    ind = T[1,i];
    k=i+1;
    while k<=n and ind!=T[k,i]
        j=1;
        while j<= n
            if(T[k,j]>tmp), tmp--; endif
            j=j++;
        endwhile
        k=k*2;
    endwhile
endfor

```

donde $T:\text{array}[1..n+1, 1..n+1]$ de enteros, estudiar los órdenes O, Ω y Θ de $t(n)$.

$$t_H(n) = \sum_{i=1}^n (2 + 2\log n \cdot \sum_{j=1}^i (3)) = n \cdot (2 + 2\log n \cdot 3n) = n^2 \cdot \log n$$

$$t(n) \in O(n^2 \cdot \log n)$$

$$t_m(n) = \sum_{i=1}^n (2) = 2n$$

$$t(n) \in \Omega(n)$$

no existe $f(n)$ tal que $t(n) \in \Theta(f(n))$

2) (2 puntos) Resolver las ecuaciones de recurrencia:

a) $t(n) = 4t(n-1) + 5t(n-2) + n2^n + n + n2^n$

b) $t(n) = 4t(n-1) + 5t(n-2) + n + n^29^{\frac{n}{2}} + 1$

a) $t(n) - 4t(n-1) - 5t(n-2) = (n+n)2^n + n \cdot 1^n$

$$(x^2 - 4x - 5) \rightarrow x: \frac{4 \pm \sqrt{16-4(-1)(-5)}}{2} = \frac{4+6}{2} = 5 \quad \text{y} \quad \frac{4-6}{2} = -1$$

$$(x+1)(x-5)(x-1)^2(x-2)^2 = 0$$

$$t(n) = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n + c_3 + c_4 \cdot n + c_5 \cdot 5^n - c_6 \cdot 1^n = 0$$

El orden será $t(n) \in O(5^n)$ si c_5 es distinto de 0.

b) $t(n) - 4t(n-1) - 5t(n-2) = (n+1) \cdot 1^n + n^2 \cdot 3^n$

$$(x+1)(x-5)(x-3)^3(x-1)^2$$

$$t(n) = c_1 \cdot 5^n + c_2 \cdot 3^n + c_3 \cdot n \cdot 3^n + c_4 \cdot n^2 \cdot 3^n + c_5 + c_6 \cdot n - c_7 \cdot 1^n = 0$$

$$t(n) \in O(5^n)$$

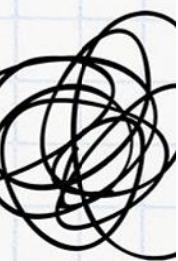
Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

X X X X X

3) (1 punto) Ordenar los siguientes órdenes de ejecución, así como los Ω y Θ correspondientes, con la relación de inclusión:

$$O(8^{\frac{n}{3}}), O(3^{0.1^n}), O((\ln n)^2 \log_7 n), O(n^2 \ln^5 n + n^{\frac{3}{2}}), O(\log_3 n^{\frac{3}{2}})$$

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

$$O(3^{0.1^n}) < O(\log^3 n) < O((\ln n)^2 \log_7 n) < O(n^2 \ln^5 n + n^{\frac{3}{2}}) < O(8^{\frac{n}{3}})$$

Ω al revés, Θ no se puede

5) (3 puntos) Dada la función:

```

int g22(int A[], int B[], int pnt):int
    cont=0;
    k=1; while k<=pnt, k++; endwhile
    if(pnt>4)
        j=0;
        if( par(A[pnt]) OR impar(B[pnt]) )
            for i=1 to 8
                cont+= g22(A,B,pnt/2);
            endfor
        else
            for i=1 to 28
                cont+= g22(B,A,pnt/4);
            endfor
        endif
    endif
    return cont;

```

donde A y B son array[1..n] de entero, y siendo la primera llamada con $g22(A,B,n)$, estudiar los órdenes O , Ω , Θ y o-pequeña de su tiempo promedio. Para la o-pequeña, sobre su constante basta con indicar qué valores de n usar para calcularla. ¿Se podría eliminar la condición de que n sea potencia de 2?

$$t_p(n) = \begin{cases} 1+n, & n \leq 4 \\ 1+n + \frac{3}{4} \cdot 8 t(n/2) + \frac{1}{4} \cdot 28 \cdot t(n/4), & n > 4 \end{cases}$$

$$t_p(n) - 6t(n/2) - 7t(n/4) = 1+n$$

$$t'(k) = 6t(k-1) - 7t'(k-2) = 1+2^k$$

$$(x-7)(x+1)(x-1)(x-2) = 0$$

$$x^2 - 6x - 7$$

$$\frac{6 \pm \sqrt{36-4(1)(-7)}}{2}$$

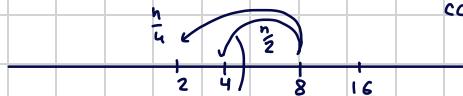
$$\frac{6+8}{2} = 7$$

$$\frac{6-8}{2} = -1$$

$$t'(k) = c_1 \cdot 7^k + c_2 \cdot 2^k + c_3 \cdot 1^k - c_4 = 0$$

$$t(n) = c_1 \cdot 7^{\log_2 n} + c_2 \cdot n + c_3 - c_4 = 0$$

$$t_p(n) \in \Omega(n^{\log_2 7}), \in O(n^{\log_2 7}), \in \Theta(n^{\log_2 7})$$



cond.ini: 2, 4, 8, 16

se puede quitar?

$$t_p(n) \in \Theta(n^{\log_2 7} | n = 2^k)$$

$$\left\{ \begin{array}{l} f(n) \text{ es END? } \checkmark \\ f(n) \text{ es END? } \checkmark \\ f \text{ es b-armónica? } \end{array} \right.$$

$$f(b \cdot n) = f(b \cdot n^{\log_2 7}) \in O(n^{\log_2 7})$$

se puede quitar

$$i = 2^{\log_2 n} = n^{\log_2 2} = n = \frac{n}{2} \quad \text{low} = 1$$

$$t(n) = 5 + \sum_{i=n/2}^{n} 4$$

$\in O(\log n)$

~~$\in O(2^{\log n})$~~

$t(n) \in \Omega(\log n)$

$t(n) \in O(\log n)$

$$t_m(n) = 5 + \sum_{i=n/2}^{n} 2$$

$\in O(\log n)$

~~$5 + \frac{1}{2} \cdot \log_2 n$~~

$t(n) \in \Theta(\log n)$

$$t_p(n) = \begin{cases} 1 + \frac{n^2}{2}, & n < 8 \\ 1 + \frac{n^2}{2} + \frac{1}{8}t(n/2) + \frac{7}{8}t(n/4) & \end{cases}$$

$$8t(n) - t(n/2) - 7t(n/4) = 1 + \frac{n^2}{2}$$

$$\frac{1 + \sqrt{1 - 4(8)(-7)}}{16} \rightarrow \frac{1 + 15}{16} = 1$$

$$\frac{1 - 15}{16} = \frac{-14}{16} = \frac{-7}{8}$$

$$(x-1)(x+7/8)(x-4)(x-1)^3$$

$$t'(k) = c_1 4^k + c_2 \cdot 1^k + c_3 \cdot k \cdot 1^k + c_4 \cdot \left(\frac{7}{8}\right)^k$$

$k \in \mathbb{N}$

$$t_m(n) = 1 + \frac{n^2}{2} + t(n/2)$$

$$t(n) - t(n/2) = 1 + \frac{n^2}{2}$$

$$t'(k) - t'(k-1) = 1 + \frac{4^k}{2}$$

$(x-1)(x-1)(x-4)$

$$t_m(k) = c_1 \cdot 1^k + c_2 \cdot k \cdot 1^k + c_3 \cdot 4^k$$

$\downarrow 4^{\log n}$

n^2

$$t_m(n) = 1 + \frac{n^2}{2} + t(n/4)$$

$t(k) - t(k-2) = 1 + \frac{4^k}{2}$

$(x-1)(x+1)(x-1)(x-4)$

$$t_m(k)$$

PROGRAMACIÓN DINÁMICA

Mochila 0/1

Tenemos:

$0 = 1, \dots, n$ objetos con p_i pesos y b_i beneficios

Mochila de peso máximo M

Objetivo: beneficio máximo

1 Definimos la recurrencia

beneficio máximo $\rightarrow B_{\max}(j, m)$, considerando objetos desde $1 \dots j$ con la capacidad disponible m . Objetos numerados de 1 a n .

- cogiendo objeto j : $b_j + B_{\max}(j-1, m-p_j)$
- no cogiéndolo: $B_{\max}(j-1, m)$

$$B_{\max}(j, m) = \max(b_j + B_{\max}(j-1, m-p_j), B_{\max}(j-1, m))$$

Los casos base son:

- $m < 0 \rightarrow -\infty$ capacidad negativa
- $j = 0 \rightarrow 0$ sin objetos
- $m = 0 \rightarrow 0$ sin capacidad

2 Dimensiones de la tabla

Será de 2 dimensiones, ya que el beneficio máximo consta de 2 parámetros: objetos y peso.

$(n+1)(M+1) \rightarrow$ dimensiones (contamos los ceros)

3 Estudiar el DAG

$$n=3 \quad M=6 \quad p=(2, 3, 4) \quad b=(1, 2, 5)$$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | $\leftarrow M$ |
|------------|-----|---|---|---|---|---|---|----------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | |
| 2 | 0 | 0 | 1 | 2 | 3 | 3 | 3 | |
| 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | |
| \uparrow | n | | | | | | | |

$$P(3, 6) = \max(\overbrace{5 + P(2, 2)}^6, \overbrace{P(2, 6)}^3)$$

$$P(2, 2) = \max(2 + P(1, 1), P(1, 2))$$

$$P(1, 2) = \max(1 + P(0, 0), P(0, 2))$$

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

aaaaah

④ Rellenar la tabla

```

def mochila_ascendente (P, M, n, B):
    array [n+1][M+1] ← inicializar a 0
    for i in 1...n+1 ← n° objeto
        for j in 1...M+1 ← capacidad
            if y - P[x-1] >= 0:
                m1 = B[x-1] + array [x-1][y - P[x-1]] → si lo cojo se suma su beneficio y se suma el
                m2 = array [x-1][y] → si no lo cojo beneficio de los demás menos
            else:
                m1 = -float("inf") → si el peso del objeto es mayor a la capacidad se vuelve inviable
                m2 = array [x-1][y]
            array [x][y] = max (m1, m2)
    return array [n][M]

```

peso ↑ capacidad ↑ n°obj ↑ beneficio

⑤ Reconstrucción de la solución

```

def mochila_ascendente_reconstruir (P, M, n, B)
    ... # construimos la tabla
    x_actual = n
    y_actual = M
    S = []
    while x_actual != 0: → si la capacidad es mayor o igual al peso del objeto anterior
        if y_actual - P[x_actual - 1] >= 0:
            m1 = B[x_actual - 1] + array [x_actual - 1][y_actual - P[x_actual - 1]]
            m2 = array [x_actual - 1][y_actual]
        else:
            m1 = -float("inf")
            m2 = array [x_actual - 1][y_actual]
        if m1 > m2:
            S.append (x_actual - 1) ← si cogemos el objeto se añade a la solución
            y_actual -= P[x_actual - 1] ← se actualiza el y_actual
        x_actual--
    return array [n][M], S

```

Cambio de monedas

| | | | | | |
|------|-----|-----|-----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0.05 | 0.1 | 0.2 | 0.5 | 1€ | 2€ |

Tenemos:

n monedas 1...n con valores v1...vn

$$P(5,5) = \min \left(\frac{5}{2} + P(4,5-\frac{5}{2}), P(4,5) \right)$$

Cantidad c

Objetivo: devolver c minimizando el nº de monedas

1 Definimos la recurrencia:

$P(j, c) \rightarrow$ nº mínimo de monedas siendo j las monedas y c la cantidad.

$$P(j, c) = \min \left(\frac{c}{v_j} + P(j-1, c - \frac{c}{v_j} \cdot v_j), P(j-1, c) \right)$$

coger la moneda implica

no coger la moneda

cogerla x veces ($\frac{c}{v_j}$) y

sumarle el mínimo de monedas

que quedan para llegar a c-

solución real

coger 1

no cogerla

$$P(j, c) = \min (1 + P(j-1, c-v_j), P(j-1, c))$$

CASOS BASE

$$c=0 \rightarrow 0 \checkmark$$

$$j < 0 \rightarrow +\infty$$

$$j=0 \rightarrow 0 \text{ y } c>0 \rightarrow \infty \text{ no solución}$$

2 Dimensiones de la tabla

2 dimensiones: una para las distintas monedas y otra para la cantidad

$$(n+1)(c+1)$$

3 Estudiar el DAG

$$V = \{1, 4, 6\} \quad C = 8$$

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $\leftarrow c$ |
|-----|---|---|----------|----------|----------|----------|----------|----------|----------|----------|----------------|
| j | 0 | 0 | ∞ | |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | |
| 2 | 0 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 2 | | |
| 3 | 0 | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 2 | | |

$$P(3,8) = \min (1 + P(3,2), P(2,8))$$

$$P(2,8) = \min (1 + P(2,4), P(1,8)) \rightarrow 1 \text{ de } 4$$

$$P(2,4) = \min (1 + P(2,0), P(1,4)) \rightarrow 1 \text{ de } 4$$

$$P(2,0) = 0 \text{ (caso base)}$$

problema 1:

8.14. En el problema de la mochila (igual que en el problema del cambio de monedas) puede existir en general más de una solución óptima para unas entradas determinadas. ¿Cómo se puede comprobar si una solución óptima es única o no, suponiendo que hemos resuelto el problema utilizando programación dinámica? Dar un algoritmo para que, a partir de las tablas resultantes del problema de la mochila, muestre todas las soluciones óptimas existentes.

| $n=3$ | $M=5$ | $p = \{2, 3, 5\}$ | $b = \{1, 2, 3\}$ | $V(k, m) = \max(V(k-1, m), b_k + V(k-1, m - p_k))$ |
|-------|-------------|-------------------|-------------------|--|
| | | 0 1 2 3 4 5 | | |
| -∞ | 0 0 0 0 0 0 | | | $n := 0 \rightarrow 0$ |
| 1 | 0 0 1 1 1 1 | | | $m := 0 \rightarrow 0$ |
| 2 | 0 0 1 2 2 3 | | | $m < 0 \rightarrow -\infty$ |
| 3 | 0 0 1 2 2 3 | | | |

Si las 2 ramas del max dan igual, significa que hay 2 soluciones óptimas.

Algoritmo :

```

    hallar_solucion(i, j, solucion[])
        Si (i == 0) imprime (solucion)
        Si no
            solcopia = solucion.clone()
            Si V[i, j] == V[i-1, j] // si no se mete el elemento
                solcopia[i] = 0
                hallar_solucion(i-1, j, solcopia)
            FinSi
            Si V[i, j] == b_i + V[i-1, j - p_i]
                solcopia[i] = 1
                hallar_solucion(i-1, j - p_i, solcopia)
            FinSi
        FinSi
    FinSi

```

Finsi

Se llamaría con $\text{hallar_solucion}(n, m, \text{solucion})$

vector vacío

④ Diseño del algoritmo:

(en C++)

problema 3:

Considerar el problema de la mochila 0/1. Calcular el nº de formas distintas de meter o no los objetos en la mochila respetando su capacidad máxima M.

Datos: n objetos, M capacidad de la mochila, $p = (p_1, p_2, \dots, p_n)$.

① Definir la recurrencia, formas distintas de coger o no coger un objeto.

$$C(j, m) = \begin{cases} 1, & j=0 \text{ y } m=0 \\ 0, & j=0 \text{ y } m>0 \\ C(j-1, m) + C(j-1, m-p_j) & \end{cases}$$

② Dimensiones tabla

$$(n+1)(M+1)$$

| | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | | | | | | |
| | 1 | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |

poner un ejemplito viene bien

③ DAG

Recorremos la tabla por filas, porque cada celda depende de 2 celdas de arriba.

ALGORITMO:

formas (n, M, pesos):

```

array[n+1][M+1]
for i : 0...n+1 :
    for j : 0...M+1 :
        if i == 0 // si no cogemos ningún elemento
            array[i][j] = 1
        elif j == 0 // si no hay capacidad
            array[i][j] = 1
        else:
            if j - p_i >= 0
                array[i][j] = array[i-1][j] + array[i-1][j - p_i]
            else // si el peso del elemento es mayor que la capacidad
                array[i][j] = array[i-1][j]
return array[n][M]
```

problema 4:

8.3. Una variante del problema de la mochila es la siguiente. Tenemos un conjunto de enteros (positivos) $A = \{a_1, a_2, \dots, a_n\}$ y un entero K . El objetivo es encontrar si existe algún subconjunto de A cuya suma sea exactamente K .

- Desarrollar un algoritmo para resolver este problema, utilizando programación dinámica. ¿Cuál es el orden de complejidad del algoritmo?
- Mostrar cómo se puede obtener el conjunto de objetos resultantes (en caso de existir solución) a partir de las tablas utilizadas por el algoritmo.
- Aplicar el algoritmo sobre el siguiente ejemplo $A = \{2, 3, 5, 2\}$, $K=7$. ¿Cómo se puede comprobar que la solución no es única?

a) $A = [a_1, a_2, a_3, \dots]$ entero K . ¿Existe un subconjunto de A que sume K ?
¿meter objeto o no?

$\text{SubsetSum}(i, K) = \begin{cases} \text{SubsetSum}(i-1, K) & \text{no se toma} \\ \text{SubsetSum}(i-1, K - a_i) & \text{se toma} \end{cases}$

$i := 0 \text{ and } K > 0 \text{ false}$

$i := 0 \text{ and } K = 0 \text{ true}$

$i := n \text{ and } K < 0 \text{ false}$

$\left. \begin{array}{l} | \\ \text{casos base} \end{array} \right\}$

La tabla será de 2 dimensiones $(n+1)(K+1) \rightarrow$ para incluir los 0

b) conjuntoSolucion (array[], K, solucion[], n):

```

i := n
while i ≥ 0:
    if K == 0:
        imprimir solucion
    else:
        if Subsetsum[i][K] == true:
            solucion.add(i)
            K := K - array[i]
        i--
    
```

c) si ambas ramas son true

$A = [2, 3, 5, 2]$, $K = 7$

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|------|-------|-------|-------|-------|-------|-------|-------|
| 0 | true | false |
| 1 | true | false | true | false | false | false | false | false |
| 2 | true | false | true | true | false | true | false | false |
| 3 | true | false | true | true | false | true | false | true |
| 4 | true | false | true | true | true | false | true | true |

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

X X X X X

problema 5:

8.20. (EX M04) En el problema de la mochila 0/1 disponemos de dos mochilas, con capacidades M_1 y M_2 . El objetivo es maximizar la suma de beneficios de los objetos transportados en ambas mochilas, respetando las capacidades de cada una. Resolver el problema mediante programación dinámica, definiendo la ecuación recurrente, las tablas usadas y el algoritmo para rellenarlas.

Datos del problema: n objetos, M_1 capacidad de la mochila 1, M_2 capacidad de la mochila 2, $p = (p_1, p_2, \dots, p_n)$ pesos de los objetos, $b = (b_1, b_2, \dots, b_n)$ beneficios de los objetos.

meter en la mochila 1, en 2 o en ninguna

$n, M_1, M_2, p = (p_1, \dots, p_n), b = (b_1, \dots, b_n)$

$$M_{12}(k, m_1, m_2) = \max (b_k + M_{12}(k-1, m_1 - p_k, m_2), b_k + M_{12}(k-1, m_1, m_2 - p_k),$$

$$M_{12}(k-1, m_1, m_2))$$

casos base:

meterlo en la 1

meterlo en la 2

$$m_1 - p_k < 0 \rightarrow -\infty$$

$$m_2 - p_k < 0 \rightarrow -\infty$$

$$m_1 := 0 \rightarrow 0$$

$$m_2 := 0 \rightarrow 0$$

$$k := 0 \rightarrow 0$$

no meterlo en ninguna

WUWUAH

8.3. Una variante del problema de la mochila es la siguiente. Tenemos un conjunto de enteros (positivos) $A = \{a_1, a_2, \dots, a_n\}$ y un entero K . El objetivo es encontrar si existe algún subconjunto de A cuya suma sea exactamente K .

- Desarrollar un algoritmo para resolver este problema, utilizando programación dinámica. ¿Cuál es el orden de complejidad del algoritmo?
- Mostrar cómo se puede obtener el conjunto de objetos resultantes (en caso de existir solución) a partir de las tablas utilizadas por el algoritmo.
- c) Aplicar el algoritmo sobre el siguiente ejemplo $A = \{2, 3, 5, 2\}$, $K=7$. ¿Cómo se puede comprobar que la solución no es única?

a) ① Definir la recurrencia

$$S(j, k) = S(j-1, K - a_j) \text{ OR } S(j-1, k)$$

Casos base:

$k := 0 \rightarrow \text{true}$

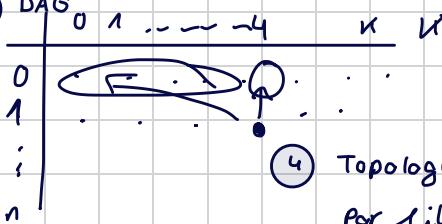
$k < 0 \rightarrow \text{false}$

$j := 0 \text{ y } k > 0 \rightarrow \text{false}$

② Dimensiones K

$$(n+1) \times (K+1)$$

③ DAG



④ Topología por filas

Suma ($A[]$, K , n)

tabla [$n+1$][$K+1$];

for i in 0...n

for j in 0...K

if ($j := 0$) tabla[i][j] = true;

if ($j > 0$ and $i := 0$) tabla[i][j] = false;

if ($j - A[i] \geq 0$):

tabla[i][j] = tabla[i-1][j-A[i]]

OR tabla[i-1][j];

else:

tabla[i][j] = tabla[i-1][j];

return tabla

b)

def reconstruir_solución (tabla [n][k], & Sol[], A[])

```
j = K;
for i in n...0
    if(tabla[i-1][j-A[i]])
        Sol.add(i);
        j -= A[i];
```

return Sol;

$$A = \{2, 3, 5, 24\} \quad k = 7$$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|-----|
| 0 | T | F | F | F | F | F | F | F |
| 1 | T | F | T | F | F | F | F | F |
| 2 | T | F | T | T | F | T | F | F |
| 3 | T | F | T | T | F | T | F | T |
| 4 | T | F | T | T | T | T | F | (T) |

```

def reconstSol(tabla[n][k], A[], sol[]):
    i := n
    j := k
    if (i == 0):
        imprimir sol
    if (tabla[i-1][j - A[i]] and tabla[i-1][j]):
        sol2 := sol.clone();
        reconstSol(tabla[i-1][j - A[i]], A, sol)
        reconstSol(tabla[i-1][j], A, sol2)
    else if (tabla[i-1][j - A[i]]):
        reconstSol(tabla[i-1][j - A[i]], A, sol)
    else:
        reconstSol(tabla[i-1][j], A, sol)
    
```

3) Reconstruir solución

```
SolVitaminas (int N, int V, array v[], array p[], array solucion[])
```

```
tabla[N+1][V+1] = Vitaminas (N,V,v[],p[])
```

```
i := N
```

```
j := V
```

```
while j != 0 :
```

```
    m0 = tabla[i-1][j]
```

```
    m1 = tabla[i-1][j-v[i] + p[i]]
```

```
    m2 = tabla[i-1][j-2*v[i]] + 2*p[i]
```

```
    m3 = tabla[i-1][j-3*v[i]] + 3*p[i]
```

```
    if (m1 == min(m0, m1, m2, m3)):
```

```
        j = j - v[i]
```

```
        solucion.add(i)
```

```
    else if (m2 == min(m0, m1, m2, m3)).
```

```
        j = j - 2*v[i]
```

```
        solucion.add(i)
```

```
        solucion.add(i)
```

```
    else if (m3 == min(m0, m1, m2, m3))
```

```
        j = j - 3*v[i]
```

```
        solucion.add(i)
```

```
        solucion.add(i)
```

```
        solucion.add(i)
```

```
i--
```

```
return solucion
```

EXAMEN GRUPO 2.2 (AÑO PASADO)

Ikrea's contest. Dado un paño de pared de M metros de longitud y N estanterías, cada una con un alto a_i y ancho b_i encontrar de cuántas formas se puede llenar la pared con estanterías (a lo alto o a lo ancho).

1) Ecuación de recurrencia + casos base

N estanterías $a[1 \dots N]$ alto $b[1 \dots N]$ ancho M metros

Formas (i, j) = Formas ($i-1, j - a[i]$) + Formas ($i-1, j - b[i]$) + Formas ($i-1, j$)
 estantería \uparrow metros restantes \uparrow meterlo a lo alto \uparrow meterlo a lo ancho \uparrow no meterlo

$i: 0 \dots N$
 $j: 0 \dots M$

Casos base:

$j < 0 \text{ || } i \leq 0 \text{ y } j > 0 \rightarrow 0$

$j := 0 \rightarrow 1$

2) Cómo son las tablas y cómo se rellenan.

Dimensiones de la tabla: $(N+1)(M+1)$

Ikrea (int N , int M , array $a[]$, array $b[]$):

```
tabla[N+1][M+1];

for i in 0 ... N:
    for j in 0 ... M:
        if j == 0:           // caso base
            tabla[i][j] = 1
        else if (j > 0 AND i <= 0): // caso base
            tabla[i][j] = 0
        else if (j - a[i] < 0): // si se pasa el alto
            if (j - b[i] < 0): // si se pasa el ancho y alto
                tabla[i][j] = tabla[i-1][j]
            else: // si sólo se pasa el alto
                tabla[i][j] = tabla[i-1][j] + tabla[i-1][j - b[i]]
        else if (j - b[i] < 0): // si sólo se pasa el ancho
            tabla[i][j] = tabla[i-1][j] + tabla[i-1][j - a[i]]
        else: // si puede usar las 3 opciones
            tabla[i][j] = tabla[i-1][j] + tabla[i-1][j - b[i]] + tabla[i-1][j - a[i]]

return tabla
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
loquito yo...

XXXXXX

3) Reconstruir solución.

Solo voy a recomponer una solución, 0 sera' no cogerlo, 1 sera' a lo alto, 2 sera' a lo ancho

```
Sol (int N, int M, array a[], array b[], solucion[])
    tabla[N+1][M+1] := lkrea(N,M,a,b)
    if (tabla[N][M]==0):
        return "No hay solución"
    for i in N...0:
        for j in M...0:
            if (tabla[i-1][j]!=0): //si hay formas no cogiéndolo
                Solucion[i]=0
            elseif(tabla[i-1][j-a[i]]!=0): //si hay formas a lo alto
                Solucion[i]=1
            else: //si hay formas a lo ancho
                Solucion[i]=2
        endif
    endfor
    return Solucion;
```

EJERCICIOS TEMA 2:

ENERO 2023:

Dentro de una secuencia S de n números indexados por $i=1 \dots n$, cuál es la subsecuencia más larga que cumple la condición de que sus elementos contiguos tienen al menos un divisor común (al margen del 1). Cuentas con la operación $\text{mcd}(a, b)$ ya implementada.

Ejemplo: $S = [1 2 4 2 6 3 7 8 9 10 12 6]$, el resultado es $[2 4 2 6 3]$, entre índices 2 y 6 de longitud 5.

Diseñar un algoritmo Divide y Vencerás que devuelva los índices (si no existe devolver 0 y 0).

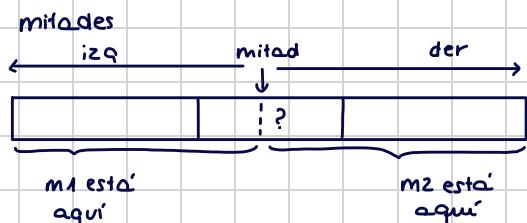
DivideYVencerás (int i, int j, S, int n):

```
if (i == j)
    return 1
else:
    return Combinar (DivideYVencerás (i, j - n/2, S, n/2), DivideYVencerás (j - n/2 + 1, j, S, n/2)
                    int i, int j, int n, S);
```

Combinar (int m1, int m2, int i, int j, int n, S):

```
int mitad := j - n/2
while (mitad < j and masdivisores):
    if (mcd (S[mitad], S[mitad + 1]) != 1):
        mitad ++
    else:
        masdivisores := false
        derecha := mitad
    mitad := j - n/2
    while (mitad >= i and masdivisores):
        if (mcd (S[mitad], S[mitad + 1]) != 1):
            mitad --
        else:
            masdivisores := false
return max (m1, m2, derecha - mitad);
```

se va recorriendo el array desde el centro hacia la derecha primero y luego a la izquierda para comprobar si la subsecuencia con divisores contiguos central es más grande que la encontrada en las mitades



JULIO 2023:

Dado un array de n enteros indexado de 1 a n y ordenado de menor a mayor, nos piden encontrar los índices de la primera y última aparición de un elemento dado x en dicho array.

Ejemplo:

$s = [1, 3, 5, 5, 5, 5, 67, 123, 125]$, $x=5$
el resultado sería 3 y 6.

La solución es un par de enteros que represento como $\text{pair}(\text{int}, \text{int})$ en pseudocódigo.

El caso base es sólo un elemento,

devolvemos su índice si es igual a x y devolvemos -1 si no.

```
DyV( s[], i, j, x, n)
    if (i == j):
        if (s[i] == x): return i,j
        else: return -1,-1
    else:
        return Combinar (DyV(s,i,j- $\frac{n}{2}$ ,x, $\frac{n}{2}$ ), DyV(s,j- $\frac{n}{2}$ +1,j,x, $\frac{n}{2}$ ),x,i,j,n)
```

Combinar (sol1(index1, index2), sol2(index1, index2), x, i, j, n):

```
if (sol1.index1 == -1)
    if (sol2.index1 == -1):
        return -1,-1
    else
        return sol2
else if (sol2.index1 == -1):
    return sol1
else
    return (sol1.index1, sol2.index2);
```

En combinar tenemos 3 casos:

- Que el elemento no esté en ninguno de los dos lados (devolvemos -1).
- Que el elemento no esté en un lado, devolvemos el otro.
- Que el elemento esté en ambos: devolvemos el primer índice de la izquierda y el último de la derecha.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
loquito yo...

XXXXXX
XXXXXXXXXX

JUNIO 2023:

2. Divide y vencerás

Subsecuencia proximal. Nos piden encontrar, dentro de una secuencia S de n números indexados por $i=1..n$, cuál es la subsecuencia más larga que cumple la condición de que sus elementos consecutivos no disten, en valor absoluto, más de 1 entre ellos.

Por ejemplo, si

$S = [1 2 1 2 3 4 7 8 9 10 12 6]$,

el resultado sería la subsecuencia $[1 2 1 2 3 4]$, entre los índices 1 y 6, con longitud 6.

DyV(S, i, j, n):

if ($i == j$):

return 1

else

return Combinar(DyV($S, i, j - \frac{n}{2}, \frac{n}{2}$), DyV($S, j - \frac{n}{2} + 1, j, \frac{n}{2}$), S, i, j, n)

Combinar($n1, n2, S, i, j, n$):

mitad = $j - \frac{n}{2}$

solcentral = 0

while (mitad < n - 1 and abs(S[mitad] - S[mitad + 1]) < 2)

mitad ++

solcentral ++

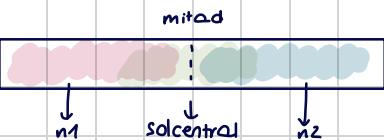
mitad := $j - \frac{n}{2}$

while (mitad > i and abs(S[mitad] - S[mitad + 1]) < 2)

mitad --

solcentral ++

return max(solcentral, n1, n2)



similar a la dinámica de otros ejercicios DyV, al combinar se recorre desde el centro hacia un lado primero y luego hacia el otro, buscando la solución central

EJERCICIOS TEMA 3:

ENERO 2023:

Celebración con tapas. Cada tapa tiene un peso asociado p_i y un n° de calorías c_i . Queremos comernos en total un peso lo más cercano a P (sin pasarnos) minimizando el total de calorías. Podemos comernos hasta 2 unidades de cada tapa.

Diseñar algoritmo voraz. ¿Garantiza la solución óptima?

Voraz (var S : Cjto solución):

```

S := ∅
pesoAct := 0
C := generarCandidatos()
while (C ≠ ∅ and !solucion(S)):
    x := seleccionar(C)
    C := C - {x}
    if (factible(x))
        insertar(S, x)
    C := generarCandidatos()
if !solucion(S)
    return "No hay solución"
else
    return S
  
```

```

generarCandidatos():
C := ∅
for i in (0...n-1):
    if (S[i] == 0):
        C.push(i)
return C;
  
```

genera como candidatos a aquellas tapas que no estén ya en la solución (donde haya un 0)

Asumo que hay n tapas, que P , $p[]$ y $c[]$ son variables globales.

El array solución S tendrá un 0, 1 o 2 en la tapa que se haya escogido.

Decisión voraz: elegir tapas cuyo c_i/p_i sea menor.

```

solucion():
return P == pesoAct
  
```

devuelve true si el peso actual iguala a P

seleccionar():

```

minC := +∞
while (C ≠ ∅):
    i := C.pop()
    proporcion := C[i] / p[i]
    if (proporcion < minC):
        minC := proporcion
        index := i
return i
  
```

selecciona usando como criterio la decisión voraz mencionada

```

factible():

    if (p[x] > P - pesoAct):
        S[x] := -1
        return false
    else:
        return true

```

```

insertar():

    if (2 * p[x] > P - pesoAct):
        S[x] := 1
        pesoAct := pesoAct + p[x]
    else:
        S[x] := 2
        pesoAct := pesoAct + 2 * p[x]

```

se coloca un -1 en S para que al generar candidatos no se incluya la tapa

JULIO 2023:

Supongamos que tenemos un robot ubicado en la posición (1,1,1) de una malla tridimensional de tamaño $N \times N \times N$. En cada celda (a, b, c) de la cuadrícula hay $P(a, b, c)$ monedas.

El robot recoge monedas por donde pasa.

Se puede mover hacia arriba ↑, hacia la derecha → y atrás ↗ es decir, si está en (a, b, c) puede ir $(a+1, b, c), (a, b+1, c), (a, b, c+1)$. Debe llegar a (N, N, N) . Maximizar cantidad de monedas.

Decisión voraz: ir a la casilla que más monedas tenga

Voraz ($P[][][], N$):

```

monedas := 0
i := 0, j := 0, k := 0
while (i < N and j < N and k < N):
    monedas := monedas + decisionVoraz(P, i, j, k)
return monedas

```

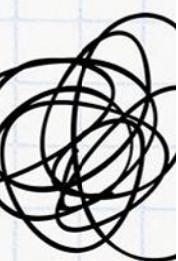
Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

X X X X X

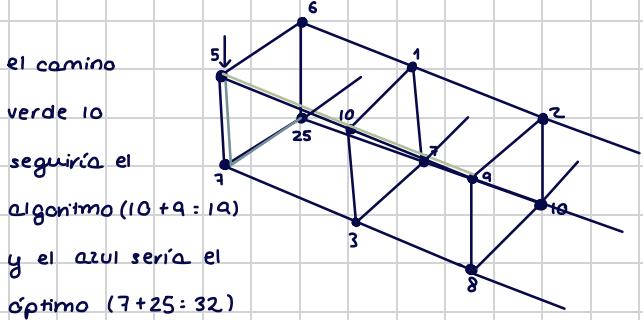
```

decisionVoraz (P[],[],[], i, j, K):
    if (i == N and j == N):
        n := P[i][j][K+1]
    else if (i == N and K == N):
        n := P[i][j+1][K]
        j := j + 1
    else if (j == N and K == N):
        n := P[i+1][j][K]
        i := i + 1
    else if (i == N):
        n1 := P[i][j+1][K]
        n2 := P[i][j][K+1]
        if (n1 >= n2):
            n := n1
            j += 1
        else :
            n := n2
            K += 1
        ...
    else:
        n1 := P[i+1][j][K]
        n2 := P[i][j+1][K]
        n3 := P[i][j][K+1]
        if (n1 == max(n1, n2, n3)):
            n := n1
            i++
        else if (n2 == max(n1, n2, n3)):
            n := n2
            j++
        else:
            n := n3
            K++
return n

```

la idea es explorar todas las casas teniendo en cuenta que i, j y K pueden ser N (nunca las 3 a la vez porque eso significa llegar al final), el nº de casas posibles es lo que hace que salga tan largo el algoritmo.

El algoritmo no va a dar la solución óptima porque al escoger ir por un camino podemos dejar otros caminos que eran mejores:



JUNIO 2023:

¿Aburrido o práctico?

Enunciado de problema común para las preguntas 3-5:

Acabamos de llegar al puesto de CEO de la *startup* para la que trabajamos en San Francisco. Queremos adoptar un *look* un poco "Zukerver", así que nos toca llenar el armario de prendas iguales, para estar a la moda. La empresa nos da D dólares como presupuesto para vestuario. Queremos maximizar el gasto, sin sobrepasar el presupuesto. Hay N tipos de prenda, cada tipo con un precio p_i por unidad de este tipo. De cada tipo podemos elegir hasta 3 unidades. No tenemos obligación de elegir prendas de todos los tipos.



Diseñar algoritmo voraz. ¿Dará la solución óptima?

Decisión voraz: empezar cogiendo prendas con el precio más alto.

Voraz ($D, p[], N$):

```
dineroGastado := 0  
S[] := Ø  
C := generarCandidatos()  
while (C ≠ Ø and !solucion())  
    x := seleccionar(C)  
    C := C - {x}  
    if (factible(x)):  
        insertar(S, x)  
return S
```

```
generarCandidatos():  
    C := Ø  
    for (i in 1...N):  
        if (p[i] < D):  
            C.add(i)  
    return C
```

```
solucion(dineroGastado, D):  
    return dineroGastado == D
```

```
seleccionar():  
    indiceMax := -1  
    valorMax := -1  
    for (i in 1....C.size()):  
        if (p[C(i)] > valorMax):  
            valorMax := p[C(i)]  
            indiceMax := C(i)  
    return indiceMax
```

```
factible(x, dineroGastado, D):  
    return p[x] ≤ D - dineroGastado
```

```
insertar(x, S, dineroGastado, D):  
    if (3*p[x] ≤ D - dineroGastado):  
        S[x] = 3  
        dineroGastado += 3*p[x]  
    else if (2*p[x] ≤ D - dineroGastado):  
        S[x] = 2  
        dineroGastado += 2*p[x]  
    else: S[x] = 1  
    dineroGastado += p[x]
```

EJERCICIOS TEMA 5

ENERO 2023:

Celebración con tapas. Cada tapa tiene un peso asociado p_i y un nº de calorías c_i . Queremos comernos en total un peso lo más cercano a P (sin pasarnos) minimizando el total de calorías. Podemos comernos hasta 2 unidades de cada tapa.

Diseñar ahora un algoritmo con backtracking.

La solución será en forma de array donde habrá un 0 si no hemos elegido la tapa, un 1 si nos comemos 1 y un 2 si podemos comernos 2.

Usaré un árbol n-ario (3 hijos por nodo porque hay 3 opciones).

Variables globales: $p[]$, $c[]$, P

`backtracking_tapas () :`

`s[] := -1, calAct := 0, pesoAct := 0`

`voo := +INF`

`nivel := 1`

`while (nivel != 0) :`

`generar(s, nivel)`

`if (solucion(s, nivel)) :`

`if (calAct < voo) :`

`voo := calAct`

`soa := copyof(s)`

`if (criterio(s, nivel)) :`

`nivel++`

`else`

`while (nivel > 1 and !manHermanos(s, nivel)) :`

`s[nivel] := 0`

`nivel--`

`pesoAct := pesoAct - s[nivel] * p[nivel]`

`calAct := calAct - s[nivel] * c[nivel]`

`if (nivel == 1 and !manHermanos(s, nivel)) nivel--`

`return soa`

`generar (s, nivel):`

`s[nivel]++;`

`solucion (s, nivel):`

`return pesoAct <= P;`

`criterio (s, nivel):`

`if (s[nivel] * p[nivel] + pesoAct <= P):`

`calAct = calAct + s[nivel] * c[nivel]`

`return true`

`else`

`return false`

`manHermanos (s, nivel)`

`return s[nivel] < 2`

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

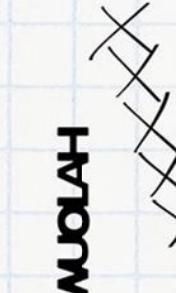
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
loquito yo...



JULIO 2023:

Supongamos que tenemos un robot ubicado en la posición $(1,1,1)$ de una malla tridimensional de tamaño $N \times N \times N$. En cada celda (a,b,c) de la cuadrícula hay $P(a,b,c)$ monedas.

El robot recoge monedas por donde pasa.

Se puede mover hacia arriba ↑, hacia la derecha → y atrás ↗ es decir, si está en (a,b,c) puede ir $(a+1, b, c), (a, b+1, c), (a, b, c+1)$. Debe llegar a (N, N, N) .

Maximizar cantidad de monedas.

La solución la representaremos mediante un array S , que tendrá indicado en cada celda el paso que ha dado el robot ($1 = \text{ARRIBA}, 2 = \text{DERECHA}, 3 = \text{ATRAS}$), se irán añadiendo pasos hasta llegar a $N \times N \times N$. El árbol empleado será n-ario con $n=3$ debido a los 3 caminos posibles.

generarNivel (nivel, s):

$s[nivel-1]++$

backtracking (PC[][], int N):

```

nivel := 1
i := 0, j := 0, k := 0
S := ∅
monedasAct := 0, soa := ∅, voa := -∞
while (nivel ≠ 0):
    generarNivel (nivel, S)
    if (solucion (nivel, S, i, j, k)):
        if (monedasAct > voa):
            voa := monedasAct
            soa := copyof(S)
        if (criterio (nivel, S)):
            nivel++
        else:
            while (nivel > 1 and !maisHermanos (nivel, S)):
                S[nivel-1] := 0
                nivel--
                monedasAct -= P[i][j][k]
            if (nivel == 1 and !maisHermanos (nivel, S)):
                nivel--
return soa;

```

solucion (nivel, S, i, j, k):

if (i == N and j == N and k == N)

monedasAct += P[N][N][N]

return true

else

return false

criterio (nivel, S):

if (i == N and j == N and k == N)

return false

else

(Aquí habrá que hacer algo parecido a lo hecho en el ejercicio del tema 3 con los casos para saber qué camino escoger teniendo en cuenta que puede haber índices que sean ya N, luego actualizamos los índices y devolvemos true)

```
masHermanos (nivel, s):
```

```
    return s[nivel-1] < 3
```

¿Aburrido o práctico?

Enunciado de problema común para las preguntas 3-5:

Acabamos de llegar al puesto de CEO de la *startup* para la que trabajamos en San Francisco. Queremos adoptar un *look* un poco "Zukerver", así que nos toca llenar el armario de prendas iguales, para estar a la moda. La empresa nos da D dólares como presupuesto para vestuario. Queremos maximizar el gasto, sin sobrepasar el presupuesto. Hay N tipos de prenda, cada tipo con un precio p_i por unidad de este tipo. De cada tipo podemos elegir hasta 3 unidades. No tenemos obligación de elegir prendas de todos los tipos.



Es un problema de buscar solución óptima. Árbol n-ario ($n=4$). La solución es un array de tamaño N , en cada celda se indican las unidades cogidas del tipo $(0, 1, 2, 3)$

backtracking ($N, D, p[]$):

```
nivel := 1
```

```
voc := -∞, sol := ∅, dineroGastado := 0
```

```
s [] := ∅
```

```
while (nivel != 0):
```

```
    generar (s, nivel)
```

```
    if (solucion (s, nivel)) :
```

```
        if (dineroGastado > voc):
```

```
            voc := dineroGastado
```

```
            sol := copyof (s)
```

```
        if (criterio (s, nivel, D, dineroGastado, p)):
```

```
            nivel++
```

```
        else
```

```
            while (nivel > 1 and !masHermanos (s, nivel)):
```

```
                s [nivel-1] = 0
```

```
                nivel--
```

```
                dineroGastado -= s [nivel-1] * p [nivel-1]
```

```
            if (nivel == 1 and !masHermanos (s, nivel))
```

```
                nivel--
```

```
return sol
```

```
generar (s, nivel):
```

```
    s [nivel-1] ++
```

```
solucion (s, nivel):
```

```
    return nivel == N
```

```
masHermanos (s, nivel):
```

```
    return s [nivel-1] < 3
```

retroceder

```
criterio (s, nivel, D, dineroGastado, p[]):  
    if (nivel == N)  
        return false  
    if (s[nivel - 1] * p[nivel - 1] <= D - dineroGastado)  
        dineroGastado += s[nivel - 1] * p[nivel - 1]  
        return true  
    else  
        return false
```