Debido a la inflación, un ahorrador desea invertir todos sus ahorros en depósitos a plazo fijo anuales. Para ello analiza la oferta de este tipo de depósitos, anotando el capital máximo que cada banco permite invertir a cada cliente y el interés anual (TAE) que ofrece.

Considere las estrategias, <primero, el de mayor interés>, <primero el de menor capital máximo> y <primero el de mayor beneficio máximo>.

Aplíquelas a este ejemplo, 44.000 € a invertir en tres depósitos con, respectivamente 1%, 2%, 4% de TAE, y capitales máximos de 30.000 €, 36.000 € y 10.000 €.

¿Garantiza un algoritmo devorador que se maximice el beneficio de su inversión con algunas de estas estrategias? Diseña un algoritmo devorador óptimo.

a) Se plantean bien los ejemplos.
b) Se explica (si existe) la opción óptima.
c) Se identifican los elementos del esquema devorador.
d) Se diseña correctamente el Algoritmo.

a) <primero el de mayor interes>

1º (4%) 44.000 − 10.000 = 34000 → 4% de 10000 = 400

2º (2%) 34.000 − 34000 = 0 (no cojo todo pq no hay suficiente dinero)

→ 2% de 34000 = 680

beneficio total = 400 + 680 = 1080

<primero el de menor capital máximo>

1º (4%) 44.000 − 10000 = 34000 → 4% de 10000 = 400

2º (1%) 34000 − 30000 = 4000 → 1% de 30000 = 300

3º (2%) 4000 − 4000 = 0 → 2% de 4000 = 80

beneficio total = 400 + 300 + 80 = 780

<primero el de mayor beneficio máximo>

1º (2%) 44000 − 36000 = 8000 → 2% de 36000 = 720

2º (4%) 8000 − 8000 = 0 → 4% de 8000 = 240

beneficio max = 1% de 30000 = 300; 2% de 36000 = 720; 4% de 10000 = 400

720 + 240 = 960

b) El algoritmo q te proporciona mayor beneficio es la 19 opción.

c) candidatos: Lista de bancos (tupla (interes, capitalmax))
F. selección: banco de mayor interés
F. objetivo beneficio
F. solución: ¿se ha invertido todo el dinero?
Objetivo maximizar
F. factibilidad: X

```
inversion (C, dinero) → S

S ← ∅

ordenarporinterés (C)                                   ] O(nlogn)
mientras C ≠ ∅ ^dinero != 0
    (i,c) ← extrae-primero (C)                          ] Θ(1)
    si dinero > c:                                                  ] O(n)
        S ← S + c·i/100
        dinero ← dinero - c
    sino
        S ← S + (dinero·i)/100
        dinero ← 0
```

Ejercicio 1. Se dispone de un buque portacontenedores que hay que cargar con tantos contenedores disponibles en la terminal de carga como sea posible, todos de idénticas dimensiones, pero cada uno con su propio peso.

El buque podría cargar todos los contenedores de estar vacíos, pero al no ser el caso, su peso total podría exceder la capacidad de carga del buque, lo que no debe suceder.

Diseñe un algoritmo devorador para intentar resolver el problema. Mejore su eficiencia con preordenación, y luego, con montículo.

a. Identificar los elementos del esquema

b. Diseñar el algoritmo

c. Analizar la eficiencia

d. Explicar cómo mejorarlo con las técnicas y analizar las mejoras

a)

Candidatos: contenedores
F. selección: más pequeños
F. objetivo: nº contenedores
F. solución: ¿se alcanza la capacidad de carga?
Objetivo: maximizar
Factibilidad: ¿se puede poner el contenedor sin hundirse?

b) Portacontenedores (C, capacarga) → S

```
S ← ∅                                   1 1
mientras C ≠ ∅ ∧ capacarga > 0          1 1
    p ← selecciona(C)                   Θ(n)
    C ← S - {p}                         2 2
    si p < capacarga
        capacarga ← capacarga - p
        S ← S ∪ {p}
```

```
selecciona(C) → aux
aux ← +∞
para todo p ∈ C
    si p < +∞
        aux ← p
```

c) $O(n^2)$

d)

```
1 ordena(C)
2 p ← extrae_primero(C)
```

```
1 crearmonticulo(C)
2 p ← extrae_primero(C)
```

Ejercicio 2. Suponga que dispone de información acerca de qué usuarios siguen a otros en una red social. Los bulos se propagan a través de cadenas de seguidores. Diseña un algoritmo que determine cuantos usuarios bastan para propagar un bulo entre los usuarios conectados por la cadena de seguidores más larga. Analice su eficiencia espaciotemporal en el peor de los casos.

Rubrica:

a) Se explica la tabla de subproblemas resueltos. → TSR (booleana)
   ↓ estructura de datos que almacena las soluciones para no tener q volver a calcularlas
b) Se diseña un algoritmo que soluciona el problema.
c) Además, se analiza su eficiencia temporal. $\Theta(n^3)$
d) Además, se analiza su eficiencia espacial. $\Theta(n^2)$
   ↓ pq usamos siempre matrices

b) Propagar_bulo (Ady, n)

```
aux ← -∞
desde i ← 1 hasta n
    desde j ← 1 hasta n
        si Ady[i,j]
            mCostes[i,j] ← 1
            Camino[i,j] ← -1
        sino
            mCostes[i,j] ← -∞
            Camino[i,j] ← ∞
```

matriz Ady → matriz costes
↓ aplicamos Floyd
$\Theta(n^2)$

```
desde i ← 1 hasta n
    mCostes[i,i] ← 0
```
$\Theta(n)$

```
desde k ← 1 hasta n
    desde i ← 1 hasta n
        desde j ← 1 hasta n
            mCostes[i,j] ← max(mCostes[i,j],
                               mCostes[i,k] + mCostes[k,j])
            Camino[i,j] ← k
```
$\Theta(n^3)$

```
desde i ← 1 hasta n
    desde j ← 1 hasta n
        si costes[i,j] > aux
            aux = costes[i,j]
            (x,y) ← (i,j)
```

$\Theta(n^2)$

S ← total ((camino, x, y)

```
Total (C, i, j) → aux

k ← C[i,j]
si k ≠ -1 (hay uno intermedio)
    aux ← 1+total (C, i, k) + total (C, k, j)
sino
    S ← 0
```

$\Theta(n)$

Ejercicio 2. Una empresa dispone de una serie de centros logísticos, una flota de vehículos permite transportar paquetes entre ellos, para lo que la empresa ha establecido un mapa con las rutas que los conectan.

Al realizar cualquier ruta entre dos centros logísticos, el vehículo asignado debe parar en todos los centros intermedios situados en ella, de forma que pueda procederse a la carga y descarga de los paquetes necesarios.

Diseñe un algoritmo que determine cuántas paradas ha de realizar como MINIMO un vehículo en ruta entre los dos centros más alejados de la red antes de llegar a su destino.

a. Diseñar tabla de subproblemas resueltos.

b. Diseñar algoritmo.

d. Analizar eficiencia temporal y espacial del algoritmo.

```
propagar-bool (Ady, n)
aux ← -∞
desde i ← 1 hasta n
    desde j ← 1 hasta n
        si Ady[i,j]
            mCostes[i,j] ← 1
            Camino[i,j] ← -1
        sino
            mCostes[i,j] ← +∞
            Camino[i,j] ← ∞

desde i ← 1 hasta n
    mCostes[i,i] ← 0

desde k ← 1 hasta n
    desde i ← 1 hasta n
        desde j ← 1 hasta n
            mCostes[i,j] ← min(mCostes[i,j],
                               mCostes[i,k] + mCostes[k,j])
            Camino[i,j] ← k
```

```
desde i ← 1 hasta n
    desde j ← 1 hasta n
        si costes(i,j) > aux
            aux = costes[i,j]
            (x,y) ← (i,j)
```

S ← total ((camino, x, y)

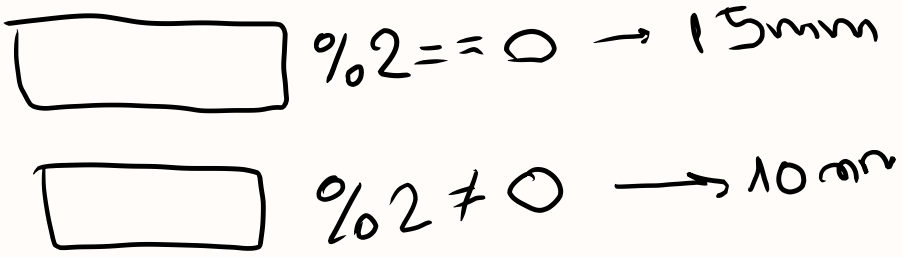
Total (C, i, j) → aux

k ← C[i,j]

si k ≠ -1 (hay uno intermedio)
    aux ← 1+total (C, i, k) + total (C, k, j)

sino
    S ← 0

Un fontanero dispone de un conjunto de trozos de tuberías T de diferentes longitudes y diámetros.
Diseñe un algoritmo eficiente, basado en una estrategia devoradora, que minimice el número de uniones necesarias para construir una única tubería de longitud exacta l, teniendo en cuenta que: el fontanero puede realizar tantos cortes en las tuberías como estime oportuno; no pueden unirse tuberías de distinto diámetro; las tuberías de longitud par tienen un diámetro igual a 10 mm; las tuberías de longitud impar tienen un diámetro igual a 15 mm. Especifique, en caso de emplear alguna de esas funciones, si la ordenación se realiza de forma creciente o decrecientemente. Un algoritmo con un orden de complejidad temporal superior al necesario se considerará incorrecto. Analice la complejidad del algoritmo en el peor de los casos. Describa los elementos que lo identifican como perteneciente al esquema general de los algoritmos voraces.


Candidatos tuberías
F.selección: coge tubería + larga
F.objetivo: nºuniones
F.solución: ¿se ha conseguido esa long?
Objetivo: minimizar
F.factibilidad: ¿cabe y es del mismo mm?

Hacemos preordenación decrecientemente

↳ tupla (long, mm)

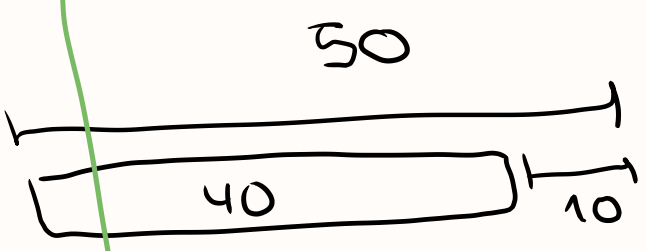
Tuberías (C, long) → S

S ← ∅

ordenar-decreciente (C) ] O(n.logn)
(p, q) ← obtiene(C)
mientras C ≠ ∅ ∧ long ≠ 0
(L, m) ← extraer_primero(C) Θ(1)
si q == m
si L > long
L ← long
long ← 0
sino
long ← long - L
S ← S ∪ {(L, m)}

O(n)

O(n log n)




long = 10
L = 20

**Diseñe un algoritmo que calcule un bosque de expansión de coste mínimo de un grafo ponderado. El algoritmo recibirá la matriz de pesos del grafo y devolverá un conjunto de árboles, cada uno de los cuales estará representado mediante un conjunto de aristas.**
**Criterios:**
**a) Se identifican correctamente todos los elementos del esquema general de los algoritmos voraces – 1**
**b) Además, se diseña un algoritmo que resuelve directamente el problema – 1.5**

candidatos: aristas
candidatos seleccionados: aristas seleccionadas
Fselección: aristas menor peso
F. objetivo: nº aristas y coste mínimo
F. solución: nº aristas = nº vertices − 1
Objetivo: minimizar
F. factibilidad: no forman ciclo

kruskal y ya está

**Una empresa de telecomunicaciones dispone del presupuesto necesario para desplegar X antenas en una ciudad.**
**La empresa ha dividido la ciudad en NxN celdas y ha estimado el número medio de habitantes en cada una de esas celdas.**
**Cada antena colocada en la celda c = <i, j> da cobertura a los habitantes que hay a una distancia d.**
**Suponga que dispone de una función dist(a, b) de complejidad temporal de orden constante, que devuelve la distancia entre las celdas a y b.**

**a) Diseñe un algoritmo que, siguiendo un enfoque voraz, determine la mejor ubicación de las antenas para conseguir dar servicio al mayor número posible de clientes. Implemente una función de factibilidad explícita.**

teleco (m, d, x, N) → S
C ← ∅
S ← ∅
desde i ← 1 hasta N
  desde j ← 1 hasta N
    C ← C ∪ {{m(i,1), i, j}}
ordena_numinhabitantes(C)
mientras C ≠ ∅ ∧ x ≠ 0
  (m(i,j), i, j) ← extraer(C)

candidatos: celdas
candidatos selec: celdas selec
Fselección: celdas + posiciones
F. objetivo: nº clientes
F. solución: ¿se han colocado todas?
Objetivo: maximizar
F. factibilidad: ¿esa celda le llega la cobertura de otra antena?