No entregar esta hoja con el examen.
La primera pregunta cuenta un 32% de la nota de teoría.
El resto de preguntas valen el 68% (1,7 puntos cada una).

1. a) Calcular O, Ω y Θ del tiempo de ejecución del siguiente algoritmo.

```
función inserciónDirecta (x, n, v)
   para i = 1 hasta n hacer
      x = v[i];
      j = i – 1;
      mientras (j ≥ 1) Y (v[j] > x) hacer
         v[j + 1] = v[j];
         j = j – 1;
      finmientras
      v[j + 1] = x;
   finpara
fin función
```

b) Resolver la siguiente ecuación de recurrencia y calcular el tiempo de ejecución y el orden exacto. No hace falta obtener las constantes, pero sí indicar los casos base.

$t(n) = a$ si $n = 1$
$t(n) = 2\ t(n/4) + \log_2 n$ si $n \geq 1$

---

2. Divide y vencerás

Disponemos de un vector A [1..*n*] que contiene elementos enteros ordenados en sentido decreciente estricto, i.e., no hay elementos repetidos. Se pretende encontrar la posición *i* tal que A[*i*] = *i*. Diseñar un algoritmo basado en el esquema *divide y vencerás* que devuelva dicha posición o el valor 0 cuando no exista.

Indicar el orden de complejidad para los casos mejor y peor.

ENUNCIADO DE PROBLEMA para las preguntas 3 a 5:

Una empresa de inversiones dispone de una tabla *Renta(1..M, 1..P)* en la que tiene registrados los porcentajes de beneficio que se obtendrán por invertir en determinados productos financieros, numerados de 1 a *P*. En concreto, *Renta(d, f)* es el porcentaje de beneficio por invertir *d* decenas de euros en el producto financiero *f*. Es importante considerar que el porcentaje de beneficio para cada producto *f* puede variar con la cantidad de euros *d* que se inviertan.

Queremos maximizar el beneficio (en euros) que se puede obtener con *D* decenas de euros (se supone que $D \leq M$).

---

3. Diseñar un algoritmo voraz que encuentre una buena forma de resolver el problema. Hay que ajustarse al esquema y desarrollar sus funciones. ¿Garantiza ese algoritmo la solución óptima? Razonarlo.

---

4. Resolver este problema mediante programación dinámica indicando la ecuación de recurrencia utilizada, los casos base, las tablas necesarias, la forma de rellenar las tablas, la forma de reconstruir la solución y el resto de pasos vistos en clase. Indicar cómo se sabe si hay varias soluciones óptimas distintas.

---

5. Resolver el problema de forma óptima por *backtracking*. Se deberán utilizar los esquemas vistos en clase. Definir la forma de representar la solución, el tipo de árbol usado, el esquema y las funciones genéricas del esquema. Seguir los pasos de desarrollo vistos en clase. Se valorarán posibles optimizaciones.

SOLUCIONES:

1. a) O(n2), Ω(n), no existe Θ

1. b)

$t(n) = a$ si $n = 1$
$t(n) = 2\,t(n/4) + \log_2 n$ si $n \geq 1$

1. $t(n) - 2t(n/4) = \log_2 n$
Cambio de variable: $n=4^k$, por lo que $k = \log_4 n$ y $t'(k):=t(n)=t(4^k)$
1'. $t'(k) - 2t'(k-1) = 2k$
2'. $(x-2)(x-1)=0$
3'. $x=2,1$
4'. $t'(k) = 2^k c_1+c_2$
4. $t(n) = 2^{\log_4 n} c_1+c_2 = n^{\log_4 2} c_1+c_2 = n^{1/2} c_1+c_2$
5. Necesitamos 2 casos base, podemos usar n=1 y n=4

2. Se puede resolver de varias formas, elijo una eficiente que descarta un subproblema, convirtiendo al algoritmo en uno de reducción en lugar de DyV, parecido a la búsqueda binaria:

```
DyV(p,q)
SI p<=q  // pequeño(p,q)
  Devolver SoluciónDirecta(p,q)
SI NO
  m = (p+q)/2  // Dividir(p,q)
  // los SI a continuación son una mezcla de combinar y llamadas recursivas a subproblemas
  SI m=A[m] devolver m // esta comprobación previa evita llamar a ambos subproblemas si se cumple
  SI m>A[m] devolver DyV(m+1,q)
  devolver DyV(p,m-1)

SoluciónDirecta(p,q)
  SI p==A[p] devolver p
  Devolver 0
```

Órdenes: O(log(n)), omega(1)

Si en lugar de este algoritmo optimizado se usa un DyV más clásico, que siempre llame a los dos subproblemas, se obtendría un orden lineal para mejor y peor caso.

(para los problemas 3 y 5, se da indicación de cómo sería la solución, no se resuelven completos)

3. Voraces:
La mejor forma de resolverlo es usar el esquema 1 visto en clase y que los candidatos sean las parejas (d, f). Se preordenarían por ratio *Renta(d, f)* / *d*. Selección sería tomar el siguiente de la lista ordenada. Factible, que d no supera la cantidad de dinero restante para invertir. Se termina el algoritmo cuando no quede dinero por invertir o cuando no queden candidatos. Siempre hay solución (incluso no invertir nada es una solución válida).

5. BT: Se trata de un problema de optimización, por lo que se usará ese esquema. El árbol sería un k-ario, donde los niveles son los productos y los hermanos son las decenas invertidas en el producto, desde 0 a *D*.
Variables auxiliares:
- bact para ir acumulando el beneficio obtenido.
- dact: presupuesto restante

Funciones
- En generar habrá que tener en cuenta restar el beneficio aportado por el hermano anterior.
- Solución: será estar en el último nivel y no haber sobrepasado el presupuesto D.
- Criterio: no estar en el último nivel y no haber sobrepasado el presupuesto D.
- MasHermanos: que lo invertido (s[nivel]) sea menor que el dinero restante por invertir (nótese que ese valor va cambiando), digamos que es un masHermanos “dinámico”, que no es lo habitual en los ejemplos vistos en clase.
- Retroceder es trivial.

4. PD:

1. Recurrencia:

   Inversión(d, f) = máximo(para i=0..d) { *Renta(i, f)* i * 10/100* + Inversión(d-i, f-1)}

   Casos base:
   Inversión(0, f) = 0
   Inversión(d, 0) = 0

2. Tabla: tabla 2D, rango f=0..P, d=0..D, nombre de la tabla, T

```
Para f=0..P,    T[f, 0]=0,      Finpara
Para d=0..D,    T[0, d]=0,      Finpara
Para f=1..P,
  Para d=1..D,
    MaxRenta = - infinito
    Para i=1..d,
      Renta = Renta(i, f)* i * 10/100 + T[d-i, f-1]}
      Si Renta>MaxRenta, MaxRenta=Renta, Finsi
    Finpara
    T[f,d]=MaxRenta
  Finpara
Finpara
```

3. Reconstruir solución:

```
X=[0,…,0] // P ceros
d=D
Para f=P..1,
  i=0
  Mientras T[f,p] != Renta[i, f]* i * 10/100 + T[d-i, f-1]}
    i++
  FinMientras
  X[f]=i
  d=d-i
Finpara
```