# Análisis y Diseño de Algoritmos

Examen Junio 2024

Francisco Javier Mercader Martínez

1. a. Calcular el tiempo de ejecución de este algoritmo y expresarlo con $O, \Omega$ y $\Theta$:

```
def Directa(x, n, v):
    for i in range(n):
        x = v[i]
        j = i - 1
        while j >= 1 and v[j] > x:
            v[j + 1] = v[j]
            j = j - 1
        v[j + 1] = x
```

   b. Determinar el orden de esta ecuación de recurrencia:

$$\begin{array}{ll} t(n) = 1 & \text{si } n < 1 \\ t(n) = 3t(n/2) + n^3 & \text{si } n \geq 1 \end{array}$$

   c. Resolver esta ecuación de recurrencia:

$$\begin{array}{ll} t(n) = 1 & \text{si } n < 1 \\ t(n) = 3t(n-1) + 2^n & \text{si } n \geq 1 \end{array}$$

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

```
def DyV(p, q):
    if p <= q: # pequeño (p,q)
        return solucion_directa(p, q)
    else:
        m = (p + q) // 2 # Dividir
        if m == A[m]:
            return m
        elif A > A[m]:
            return DyV(m + 1, q)
        else:
            return DyV(p, m - 1)

def solucion_directa(p, q):
    if p == A[p]:
        return p
    else:
        return 0
```

   Órdenes: $O(\log(n)), \Omega(1)$

ENUNCIADO DE PROBLEMA para las preguntas 3 a 5:

---

**Pares óptimos.** Dados dos vectores de números ordenados $A = [a_1, a_2, a_3, \ldots, a_m]$ y $B = [b_1, b_2, b_3, \ldots, b_n]$ con $m \geq n$, encontrar la asignación de elementos de $B$ con $A$ que minimice la diferencia en valores absolutos entre los pares $(a_i, b_j)$

resultantes. En este sentido un elemento de $B$ deberá estar asignado a un y solo un elemento de $A$ mientras que un elemento de $A$ podrá estar asignado a más de un elemento de $B$.

Ejemplo: Dados los vectores `A=[8,10,13,14]` y `B=[7,10,11]` la asignación óptima sería `[(7,8), (10,10), (11,10)]` con una diferencia total de $2\,(1+0+1)$.

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

```
import numpy as np

def pares_AV(A, B):
    S = []
    C = B.copy()

    while (len(C)>0) and not solucion(S):
        x = seleccionar(C)
        C.pop()
        if factible(S, x):
            S.append(x)
    if not solucion(S):
        return "No se puede encontrar la solución"
    return S

def solucion(S):
    return len(S) == len(B)

def seleccionar(C):
    x = [C[-1], None]
    diff = np.inf
    for a in A:
        if abs(a - x[0]) < diff:
            x[1] = a
            diff = abs(a - x[0])
    return x

def factible(S, x):
    return True
```

4. 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 (código). Seguir los pasos de desarrollo vistos en clase. Se valorará cualquier mecanismo de mejora de la eficiencia que se desarrolle o al menos explique.

```
import math
def backtracking_opt(s_inicial):

    nivel = 0
    s = s_inicial
    voa = math.inf
    soa = None

    while nivel != -1:
        s = generar(nivel, s)
        if solucion(nivel, s) and valor(s) < voa:
            voa = valor(s)
            soa = s.copy()
        if criterio(nivel, s):
            nivel += 1
        else:
            while not masHermanos(nivel, s) and nivel >= 0:
                s, nivel = retroceder(nivel, s)
    return soa, voa

def generar(nivel, s):
```

```
    s[nivel] += 1
    return s

def solucion(nivel, s):
    return nivel == (len(B) - 1)

def valor(s):
    v = 0
    for i in range(len(s)):
        v += abs(B[i]-A[s[i]])
    return v

def criterio(nivel, s):
    return nivel < (len(B) - 1)

def masHermanos(nivel, s):
    return s[nivel] < (len(A) - 1)

def retroceder(nivel, s):
    s[nivel] = -1
    return s, nivel - 1

if __name__ == '__main__':
    A = [8, 10, 13, 14]
    B = [7, 10, 11]

    s_inicial = [-1 for i in range(len(B))]
    s, v = backtracking_opt(s_inicial)
    print(s,v)
    for i in s:
        print((B[i], A[s[i]]))
```

```
[0, 1, 1] 2
(7, 8)
(10, 10)
(10, 10)
```

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