FUNDAMENTOS DE ANÁLISIS DE ALGORITMOS

GRADO EN INGENIERÍA INFORMÁTICA**.** 13 de junio de 2017. **Tiempo máximo: 120 minutos.**

ALUMNO/A		Nº HOJAS		NOTA	

EJERCICIO 1 PUNTOS: 2

- Estudiar mediante el cálculo de operaciones elementales la complejidad del algoritmo de ordenación `InsercionBinaria` por la llamada al procedimiento **`InsercionBinaria(a,1,n)`**.

El método de **Inserción Binaria** usa la búsqueda binaria recursiva para localizar dónde introducir el siguiente elemento. El procedimiento de Inserción Binaria puede ser implementado modificando el de Inserción como sigue:

```
procedimiento  InsercionBinaria(a:vector; primero,ultimo: int);
    para i:=primero+1 hasta ultimo inc 1 hacer
           x:=a[i]; k:= BinariaRc(a,primero,i,x);
           para j:=i-1 hasta k inc -1 hacer
                  a[j+1]:=a[j]
           fpara;
           a[k]:=x
    fpara
fprocedimiento InsercionBinaria;
```

- En el algoritmo se utiliza la función **`BinariaRc`** que puede ser implementado como sigue:

```
int funcion BinariaRc(V[1..n];primero,ultimo,clave:int)
    si primero > ultimo  entonces
       devolver -1
    sino
          si primero = ultimo entonces
                devolver primero
          fsi
    fsi
    mitad ←  ((ultimo - primero + 1) / 2);
    si clave = V[mitad] entonces
          devolver mitad;
    sino
          si clave < V[mitad] entonces
                devolver BinariaRc (V, primero, mitad-1, clave);
          sino
                si clave > V[mitad] entonces
                   devolver BinariaRc (V, mitad+1, ultimo, clave);
                fsi
          fsi
    fsi
ffuncion BinariaRc
```

EJERCICIO 2 PUNTOS: 2

- Usar las relaciones $\subset$ y **=** para ordenar los órdenes de complejidad, $\mathbf{O}$, $\mathbf{\Omega}$, y $\mathbf{\Theta}$, de las siguientes funciones:

$n^2, n,\ n^3, n * \log n,\ (n+3)^2,\ n * \sqrt{n},\ T(n)$, siendo $\boldsymbol{T(n)}$ la función:

$$\boldsymbol{T(n)} = \begin{cases} \mathbf{1} & \boldsymbol{si\ n \leq 1} \\ \mathbf{4} * \boldsymbol{T}\left(\frac{\boldsymbol{n}}{\mathbf{3}}\right) + \boldsymbol{n} & \boldsymbol{si\ n} > 1 \end{cases}$$

NOTAS: $\log_3 4 = 1.26184$

$$X^{\log_a Y} = Y^{\log_a X}$$

$$\sqrt{n} = n^{\frac{1}{2}} = n^{0.5}$$

FUNDAMENTOS DE ANÁLISIS DE ALGORITMOS

GRADO EN INGENIERÍA INFORMÁTICA**.** 13 de junio de 2017. **Tiempo máximo: 120 minutos.**

EJERCICIO 3	PUNTOS: 4

Dado un vector A de *n* elementos, el problema de la selección consiste en buscar el *k*-ésimo menor elemento. Una posible implementación con la estrategia Divide y Vencerás del problema es:

```
SelectRc(A,p,r,k) /* Busca el k-ésimo menor elemento en un vector A desde p hasta r */
    si (p == r) entonces
            return A[p];
    fsi
    q = Partition(A, p, r);
    i = q- p+1;          /*i es el número de elementos en el primer subvector*/
    si (k ≤ i) entonces
            return Select(A, p, q, k);          /*…buscamos en A[p..q] */
    sino
            return Select(A, q+1, r, k-i);      /*...buscamos en A[q+1..r] */
    fsi
fSelectRc
```

- Llamada inicial: **`SelectRc(A,1,n,k)`**, siendo *n* el número de elementos del vector *A*

➢ El algoritmo utiliza para su implementación la función **Partition**.

```
int función Partition (A:vector;, primero,ultimo:int)
    piv= A[primero]; i= primero-1;j= ultimo+1;
    mientras j ≥ i hacer
            mientras A[j] ≥ piv hacer
               j = j - 1;
            fmientras
            mientras A[i] ≤ piv hacer
               i = i + 1;
            fmientras
            si i < j entonces  /* A[i] ↔ A[j] */
                    temp: int; temp= A[j]; A[j]=A[i]; A[i]=temp;
            fsi
    fmientras
    return j; /* retorna el índice para la división (partición) */
ffuncion Partition;
```

➢ Se pide:

1. (1 puntos). Realizar una estimación del orden de complejidad del algoritmo dado.
2. (1 punto). Escribir posibles implementaciones, **simples y cortas**, para el problema de la selección con las siguientes estrategias:
   a. (0,75 puntos). Procedimiento iterativo, **`SelectIt.`**
   b. (0,25 puntos).procedimiento directo, mediante una simple operación elemental, **`SelectDi.`**
3. (0,50 puntos). Realizar una estimación del orden de complejidad de los dos algoritmos del apartado anterior.
4. (0,50 puntos). Comparar los órdenes de complejidad obtenidos para los tres algoritmos, estableciendo una relación de orden entre los mismos.
5. (1 punto). Realizar una traza de `SelectRc` , `SelectIt` y `SelectDi` con
   A = {31, 23, 90, 0, 77, 52, 49, 87, 60, 15} y *k* = 7.

EJERCICIO 4	PUNTOS: 2

Escribir un algoritmo voraz que dada una cantidad **C** devuelva esa cantidad usando un número **mínimo** de monedas. Se supone que se dispone de suficientes monedas de todas las cantidades consideradas. Explicar el funcionamiento del algoritmo: cuál es el conjunto de candidatos, la función de selección, la función para añadir un elemento a la solución, el criterio de finalización, etc. Suponer monedas de **1, 5, 10 y 25 €**.

- Aplicar el algoritmo para el caso que se solicite la cantidad de **207 €.**
- Demostrar, buscando contraejemplos, que el algoritmo no es óptimo si se añade un nuevo tipo de moneda de 12€ o si se elimina alguno de los tipos existentes. Demostrar que, en esas condiciones, el algoritmo puede incluso no encontrar solución alguna aunque ésta exista.