APELLIDOS, NOMBRE____ NOTA________

Ejercicio_1. (2 puntos)

- Analizar el algoritmo de la **Búsqueda del *k*-ésimo menor elemento:** dado un vector de *n* elementos, el problema de la selección consiste en buscar el *k*-ésimo menor elemento.

- Supongamos que disponemos de la siguiente definición de tipo:
  CONST n = ...;
  TYPE vector = ARRAY [1..n] OF INTEGER;
  Y supongamos que `primero` y `último` indican los límites del array (inicialmente `primero=1` y `ultimo=n`)
- Para la solución del problema utilizamos la idea del algoritmo **Partition** (utilizado en Quicksort): El vector *A*[*p*..*r*] se particiona(reorganiza) en dos subvectores *A*[*p*..*q*] y *A*[*q*+1..*r*], de forma que los elementos de *A*[*p*..*q*] son menores o iguales que el pivote(por ej. primer elemento) y los de *A*[*q*+1..*r*] mayores o iguales.

```
int función Partition (A:vector;, primero,ultimo:int)
    piv= A[primero]; i= primero;j= ultimo;
    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;
```

- El **algoritmo de la Búsqueda del *k*-ésimo menor elemento** puede ser implementado:

1. Versión **iterativa** de la **Búsqueda del *k*-ésimo menor elemento**.

```
int función SelectIterativa (A:vector;, primero,ultimo, k:int)
    mientras primero < ultimo hacer
       q = Partition(A,primero,ultimo);
       si K ≤ q entonces
           ultimo = q          /*buscamos en A[primero..q]*/
       sino
           primero = q + 1     /*buscamos en A[q + 1.. ultimo]*/
       fsi
    fmientras
    return A[primero];
 ffuncion SelectIterativa;
```

2. Versión **recursiva** de la **Búsqueda del *k*-ésimo menor elemento**.

```
int función SelectRecursiva (A:vector;, primero,ultimo, k:int)
    si (primero == ultimo)
           return A[primero];
    fsi
    q = Partition(A,primero,ultimo);
    i = q-primero+1;             /*i es el número de elementos en el primer subvector*/
    si (k ≤ i)
           return SelectRecursiva (A,primero,q,k);    /*buscamos en A[primero..q]*/
    sino
           return SelectRecursiva(A,q+1,ultimo,k-i); /*buscamos en A[q + 1.. ultimo]*/
    fsi
ffuncion SelectRecursiva
```

- **Se pide**:

a. (1 puntos). Realizar una traza de `SelectRecursiva` y `SelectIterativa` con

A = {31, 23, 90, 0, 77, 52, 49, 87, 60, 15} y *k*=7

b. (0,5 puntos).Calcular la complejidad del algoritmo **iterativo** propuesto mediante el conteo del número de operaciones elementales.

c. (0,5 puntos). Calcular la complejidad del algoritmo **recursivo** propuesto por el método de la **ecuación característica**.

Ejercicio_2. (1,5 puntos)

- La sucesión de Fibonacci se define como

  fib(0) = fib (1) = 1;

  fib(n) =fib (n – 1) + fib (n – 2) si n ≥ 2.

- **Se pide**:

a. (0,75 puntos) Escribir tres posibles implementaciones, **simples y cortas**, para el cálculo del n-ésimo número de **fib** con las siguientes estrategias:
1. procedimiento directo
2. divide y vencerás
3. programación dinámica

b. (0,75 puntos).Realizar una estimación del orden de complejidad de los tres algoritmos del apartado anterior. Comparar los órdenes de complejidad obtenidos, estableciendo una relación de orden entre los mismos.

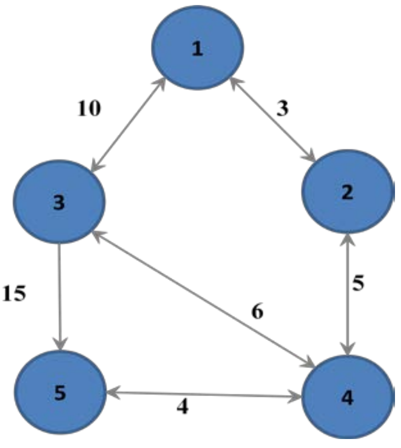
Ejercicio_3. (3 puntos)

El algoritmo de Floyd determina la ruta más corta entre dos nodos cualesquiera de la red. Una posible implementación consiste en:

- Representar la red de *n* nodos como una matriz cuadrada de orden *n*, la llamaremos **matriz *C***. De esta forma, el valor **C[i , j]** representa el **coste de ir desde el nodo *i* al nodo *j***, inicialmente en caso de no existir un arco entre ambos, el valor **C[i , j]** será infinito.
- Definir otra **matriz *D***, también cuadrada de orden *n*, cuyos elementos van a ser los **nodos predecesores en el camino hacia el nodo origen**, es decir, el valor **D[i , j] representará el nodo predecesor a *j* en el camino mínimo desde *i* hasta *j***. Inicialmente son caminos de **longitud *1***, por lo que **D[i , j]= i**
- Los pasos a dar en la aplicación del algoritmo de Floyd son los siguientes:
  1. Formar las matrices iniciales *C* y *D*.
  2. Se toma *k=1*.
  3. Se selecciona la fila y la columna *k* de la matriz *C* y entonces, para *i* y *j*, con *i≠k*, *j≠k* e *i≠j*, hacemos:
     - Si (**C[i , k]** + **C[k , j]**) < **C[i , j]** ⇒ **D[i , j]** = **D[k, j]** y **C[i , j]** = **C[i , k]** + **C[k , j]**
     - En caso contrario, dejamos las matrices como están.
  4. Si *k* ≤ *n*, aumentamos *k* en una unidad y repetimos el paso anterior(3.), en caso contrario paramos las iteraciones.

- Se pide:

a. (1.5 puntos). Aplicar el algoritmo de Floyd sobre el siguiente grafo para obtener las rutas más cortas entre cada dos nodos.



b. (1.5 puntos).Escribir posibles implementaciones de algoritmos para calcular:
1. **Distancia más corta**, Aplicar por ej, a la distancia más corta del nodo 1 al nodo 5.
2. **La ruta asociada del camino mínimo**. Aplicar por ej, entre el nodo 1 y el nodo 5

APELLIDOS, NOMBRE____ NOTA________

Ejercicio_4. (1,5 puntos)

Consideremos el problema de la mochila modificado en el que tenemos:

- **n** objetos, cada uno con un peso (**p_i**) y un valor o beneficio (**b_i**)
- Una mochila en la que podemos meter objetos, con una capacidad de peso máximo **M**.
- Cada objeto puede **meterse** dentro de la mochila, **no** meterse, o meterse la **mitad** del objeto obteniendo la mitad del beneficio.
- **Objetivo:** llenar la mochila con esos objetos, maximizando la suma de los beneficios (valores) transportados, y respetando la limitación de capacidad máxima **M**.
- Se supondrá que los objetos **se pueden partir en la mitad** ($x_i = \mathbf{0}$, $\boldsymbol{x_i} = \frac{\mathbf{1}}{\mathbf{2}}$, $\boldsymbol{x_i} = \mathbf{1}$).

➢ **Se pide**:

a) (1 punto). Diseñar un algoritmo **voraz** para resolver el problema aunque no se garantice la solución óptima. Es necesario marcar en el código propuesto a que corresponde cada parte en el esquema general de un algoritmo voraz (criterio, candidatos, función……). Si hay más de un criterio posible elegir uno razonadamente y discutir los otros. Comprobar si el algoritmo garantiza la solución óptima en este caso (la demostración se puede hacer con un contraejemplo). Calcular su tiempo de ejecución.

b) (0,5 puntos). Aplicar el algoritmo para n = 2; M = 5; p = (8, 5); b = (10, 6)

Ejercicio_5. (2 puntos)

- Dado el AFND = ({a,b,} , {p,q,r,s}, f, p, {s}) donde ***f*** viene dada por la siguiente tabla de transiciones:

	a	**b**	**λ**
→ p	**{q,s}**	**{p}**	**{q,r}**
q		**{q,r}**	**{r}**
r		**{p,s}**	**{q}**
*** s**	**{s}**	**{q,r,s}**	

➢ **Se pide**:

a. (0,5 puntos). El AFD equivalente
b. (0,5 puntos). El AFD mínimo
c. (0,5 puntos).La gramática regular equivalente al AFD obtenido en el apartado **b**
d. (0,5 puntos). Obtener una expresión regular equivalente al AFD obtenido en el apartado **b**