ANÁLISIS DE ALGORITMOS

Ejercicio 1:

Dados los siguientes vectores:

a.	6	11	8	12	1	6	14	2
b.	14	12	13	1	4	5	10	6

Realizar los pasos que realizarían los algoritmos de ordenación de Inserción, Intercambio, Selección y Ordenación Rápida para ordenarlos.

Ejercicio 2:

Dados los siguientes vectores:

a.	1	2	3	4
b.	4	3	2	1

a.- ¿Cuántos pasos realiza cada uno de los algoritmos de ordenación para proceder a su ordenado?. ¿Cuál es el mejor algoritmo en estos casos?

b.- ¿Cuántas operaciones (asignaciones y comparaciones) sobre elementos del vector realizará cada uno de los algoritmos de ordenación?

Ejercicio 3:

Indicar cuál será el número de comparaciones y asignaciones de elementos que habrá que realizar en un vector para introducir la secuencia (10, 9, 8, 7, 6, 5, 4, 3, 2, 1), considerando los dos siguientes casos:

(a) Los datos se añaden al final del vector.

(b) Los datos se insertan de manera que el vector quede ordenado.

Ejercicio 4:

Supongamos un vector con 100 elementos enteros que está ordenado. Si a ese vector le añadimos al final tres nuevos elementos, tal que el vector queda parcialmente desordenado, ¿cuál de los métodos de ordenación vistos en clase sería más eficiente para volver a ordenar el vector? ¿por qué?

Ejercicio 5:

Deseamos encontrar la mediana de un vector. Para ello nos basta con tener la mitad de sus elementos ordenados. ¿Qué método de ordenación sería conveniente modificar para determinar la mediana?

a.- El de selección. b.- El de inserción. c.- El Quick-Sort. d.- El Heap-Sort.

Razona brévemente la respuesta.

Ejercicio 6:

Supongamos que estamos ordenando un vector de ocho enteros mediante el algoritmo *quick-sort* y se ha finalizado la primera partición del proceso dejando el vector en el siguiente estado:

```
2  5  1  7  9  12  11  10
```

¿Cuál de las siguientes afirmaciones es correcta respecto del pivote utilizado?

a.- El pivote podría ser tanto el 7 como el 9

b.- El pivote podría ser el 7 pero no el 9

c.- El pivote podría ser el 9 pero no el 7

d.- El pivote no puede ser ni el 7 ni el 9

Ejercicio 7:

Supongamos que deseamos ordenar parcialmente un vector de 1000 elementos (calcular, por ejemplo, los 10 números más grandes del vector). ¿Qué algoritmo de ordenación de los vistos en clase, sería conveniente modificar para realizar esta ordenación?

Ejercicio 8:

Sea el siguiente procedimiento:

```
Procedimiento coso ( ref A: vector [1..N] de Valor; q, r: entero )
Var
   aux, x: Valor;
   s: entero;
Inicio
   Si1 ( q < r ) Entonces
      x ← A[q];
      s ← q;
      Desde i ← q+1 Hasta r Hacer
         Si2 (A[i] <= x) Entonces
            s ← s+1;
            aux ← A[i];
            A[i] ← A[s];
            A[s] ← aux;
         Fin_si2
      Fin_desde
      aux ← A[q];
      A[q] ← A[s];
      A[s] ← aux;
      coso ( A, q, s-1);
      coso ( A, s+1, r);
   Fin_si1
Fin_procedimiento
```

a.- ¿Qué mostrarán por pantalla las siguientes líneas de código si `M = [ 4, 3, 7, 1, 5]`?

```
…
coso (M, 1, 5);
Desde i ← 1 Hasta 5 Hacer
   Escribir (M[i]);
Fin_desde
…
```

b.- ¿Cuántas comparaciones sobre elementos del vector se realizan? ¿Cuál es el orden de complejidad que presenta el algoritmo?

c.- ¿Qué tarea realiza el algoritmo?

Ejercicio 9:

¿Cúal sería el coste computacional de encontrar los elementos comprendidos entre unos ciertos valores 'a' y 'b' en cada una de las siguientes estructuras:

(a) Pila
(b) Cola
(c) Árbol binario de búsqueda
(d) Árbol binario de búsqueda hilvanado
(e) Montículo de máximos

Ejercicio 10:

Indica cuál es el número de asignaciones y de comparaciones de elementos que hay que realizar para insertar el elemento *x* en la tercera posición de la secuencia (*a, b, c, d, e, f, g, h, i*), suponiendo las siguientes posibles representaciones para la secuencia:

(i) En forma de *array*.

(ii) Como una lista simplemente enlazada.

Da también el resultado para el caso hipotético en que la secuencia tuviese cien componentes.

Ejercicio 11:

Supongamos una lista enlazada de elementos. ¿Podemos utilizar en ella el método de ordenación por selección directa? En caso afirmativo escribe el algoritmo que realizaría la ordenación, en caso negativo realiza el algoritmo de ordenación por Inserción directa.

Ejercicio 12:

Las siguientes colecciones de 'n' elementos se pueden utilizar para almacenar datos. ¿Cuál es la complejidad computacional de la operación 'encontrar mínimo' en cada una de ellas?

(a) Pila

(b) Montículo de mínimos

(c) Arbol binario de búsqueda

(d) Lista ordenada con criterio ascendente

(e) Lista ordenada con criterio descendente

Ejercicio 13:

El recorrido de un árbol binario de búsqueda en orden infijo nos proporciona la secuencia ordenada de sus elementos, con lo que podemos utilizar el árbol como una estructura auxiliar para ordenar un array de 'n' enteros. ¿Cuál sería el coste de esta ordenación?

Ejercicio 14:

Queremos obtener un listado de toda la información contenida tanto en un árbol binario de búsqueda como en una lista simplemente enlazada ordenada.

a.- El coste temporal de la operación obtener listado es menor en el árbol binario de búsqueda, ya que en general, las operaciones en el árbol binario de búsqueda son del orden de ***log n***.

b.- El coste temporal de la operación obtener listado es menor en la lista ligada ordenada.

c.- El coste temporal de la operación obtener listado es el mismo en los dos casos.

d.- Es imposible obtener el listado de toda la información contenida en el árbol binario de búsqueda, ya que los procedimientos definidos sobre el árbol sólo incluyen la búsqueda, la inserción y la eliminación de elementos.

Ejercicio 15:

Entre los diferentes métodos de ordenación de vectores, en el peor de los casos:

a.- El mejor siempre es el *Quick-Sort*, porque utiliza la técnica de divide y vencerás y su coste es logarítmico.

b.- El mejor es el *Heap-Sort*, que en el peor de los casos sigue teniendo un coste 'n*lg (n)'.

c.- Depende del número de elementos. Si 'n' es pequeño, el mejor algoritmo es el de inserción.

d.- El *Heap-Sort* no se puede emplear para ordenar vectores. Sólo puede ordenar montículos.

Ejercicio 16:

Realizar un procedimiento en C++ que realice la ordenación por intercambio directo (burbuja) de una lista doblemente enlazada.

```
void Ord_Bur (ListaD & l);
```

Escribe la declaración de tipos utilizada, así como todas las operaciones auxiliares que necesites.