4.2. Métodos de ordenación logarítmicos como ejemplos de aplicación de Divide y Vencerás

4.2.1. Ordenación rápida o quick_sort()

No será necesario hacer recombinación de las soluciones parciales

Basado en el principio de intercambio, pero mejorado:

Los intercambios son más efectivos cuanta mayor sea la distancia entre elementos.

Supongamos que tenemos n elementos en orden inverso al que deben tener.

Se puede ordenar con n/2 cambios: 1 y n, 2 y n-1, 3 y n-2, ...

Pero esto sólo es posible si se conoce que están exactamente en su orden inverso.

Quick_sort (cont.)

Esto nos lleva a realizar una partición:

Tomamos un elemento X de un vector.

Buscamos dejar los elementos mayores que X a su derecha y los elementos menores a su izquierda.

Se inspecciona a la izquierda de X hasta que se encuentra un $a_i > x$, y se inspecciona a la derecha hasta que se encuentre un $a_j < x$. Entonces se intercambian.

Se procede de la misma manera hasta que ambos recorridos se encuentren (aproximadamente en la mitad del vector).

Ejemplo: Partición del siguiente vector

14	8	27	15	100	7	42

14	8	27	15	100	7	42

14	8	7	15	100	27	42

Quick_sort (cont.)

```
Algoritmo Partición (V, inf, sup) es
V: Vector [1..100] de T; {par. dato-resultado}
Inf, Sup: numérico; {par. dato}
I, J: Numérico; {var. internas}
X: T;
Inicio
    I: = Inf;
    J := Sup;
    X := V((Inf + Sup) div 2);
    Repetir
        mientras  V(I) < X hacer
                I := I + 1;
        fin mientras;
        mientras  V(J) > X hacer
                J := J - 1;
        fin mientras;
        si I <= J entonces
                intercambia (V, I, J);
                I := I + 1;
                J := J –1;
        finsi;
    hasta que I > J;
Fin
```


Quick_sort (cont.)

```
Algoritmo Partición (V, inf, sup, I, J) es
V: Vector [1..100] de T; {par. dato-resultado}
Inf, Sup: numérico; {par. dato}
I, J: Numérico; {par. resultado: necesarios para
   quick_sort}
X: T;
Inicio
    I: = Inf;
    J := Sup;
    X := V((Inf + Sup) div 2);
    Repetir
        mientras  V(I) < X hacer
                I := I + 1;
        fin mientras;
        mientras  V(J) > X hacer
                J := J - 1;
        fin mientras;
        si I <= J entonces
                intercambia (V, I, J);
                I := I + 1;
                J := J –1;
        finsi;
    hasta que I > J;
Fin
```

Quick sort (cont.)

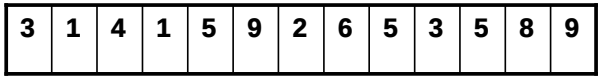
• Si utilizamos el algoritmo de partición, ¿qué nos falta para dejar ordenado el vector?

• **Divide y Vencerás**: suponiendo que solucionamos el problema en ambas mitades: quedaría ordenado (no hay recombinación de las soluciones parciales)→ hay que repetir las particiones en cada sub-mitad (si es necesario).

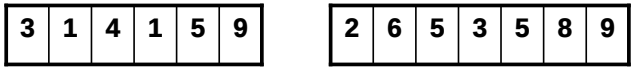
```
Algoritmo Ordena (V, N) es
V: Vector [1..100] de T; {par. dato-resultado}
N: numérico; {par. dato}
Inicio
Quick_sort (V, 1, N);
{Ordenar el vector, usando quick_sort(), entre las posiciones 1 y N}
Fin
```

```
Algoritmo Quick_sort (V, inf, sup) es
V: Vector [1..100] de T; {par. dato-resultado}
Inf, Sup: Numérico; {par. dato}
I, J: Numérico; {variables internas}
Inicio
    {Partimos V en dos mitades en torno a un elto.}
    Partición (V, inf, sup, i, j);
    Si inf < j entonces
        quick_sort (V, inf, j);
    finsi; {intenta ordenar mitad izquierda}
    Si i < sup entonces
        quick_sort (V, i, sup);
    finsi; {intenta ordenar mitad derecha}
Fin
```

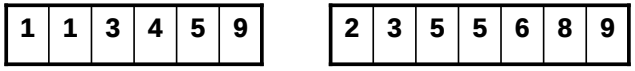

4.2.2. Ordenación por fusión (*merge sort*)
Ejemplo de recombinación no trivial



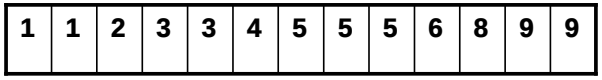
Partimos el vector en dos mitades



Ordenamos por fusión las dos mitades
(regla de diseño: suponemos que funciona)



Fusionamos (mezclamos) ambas mitades



RECOMBINAR: algoritmo de fusión de dos vectores ordenados

```
Algoritmo fusión (U, V, W) es {En general}
U: vector [1..M] de T;  {par. dato}
V: vector [1..N] de T; {par. dato}
W: vector [1..M+N] de T; {par. resultado}
INICIO
i :=1; j := 1; k := 1;
mientras (i <= M y j <= N) hacer
 si U[i] < V[j] entonces
   W[k] := U[i];
   k := k + 1;
   i := i + 1;
 sino
   si U[i] > V[j] entonces
    W[k] := V[j];
    k := k + 1;
    j := j + 1;
   sino
        W[k] := U[i]; i := i + 1;
        W[k+1] := V[j];  j := j +1;
        k := k + 2;
   finsi
 finsi
fin mientras
mientras (i <= M) hacer
   W[k] := U[i]; k := k + 1; i := i + 1;
fin mientras
mientras (j <= N) hacer
 W[k] := V[j]; k := k + 1; j := j + 1;
fin mientras
FIN
```



```
Algoritmo Mergesort (W, izd, der) es
INICIO
   si |izd - der| es suficientemente pequeño
   entonces
     ordenar (W); {sobre 1 ó 2 elementos}
   sino
    Crear U[1, (izd+der) div 2] con elementos
      de W[izd, (izd+der) div 2];
    Crear V[1, (izd+der) div 2 ] con elementos
      de W[(izd + der) div 2 + 1, der];
     Mergesort (U, 1, (izd+der) div 2);
     Mergesort (V, 1, (izd+der) div 2);
     Fusión (U, V, W);
   fin si
FIN
```

Otros ejemplos de divide y vencerás:
- multiplicación de dos enteros grandes (en binario o decimal),
- cálculo del determinante de una matriz desarrollándolo por menores