Unidad Didáctica II.
Algoritmos de recorridos de estructuras lineales y no lineales
Tema 3. Algoritmos de Clasificación

Algoritmia

Profesor: Andrés Muñoz

Escuela Politécnica

Andrés Muñoz
Universidad Católica San Antonio de Murcia - Tlf: (+34) 968 27 88 00 info@ucam.edu - www.ucam.edu

Índice

Índice

Introducción

✓ Definición de clasificación:

- Proceso mediante el cual se modifica la disposición de los elementos constituyentes de una determinada estructura lineal de forma que éstos aparezcan finalmente en un cierto *orden* (ascendente o descendente).
- En los ejemplos, sin perdida de generalidad, clasificaremos en orden *ascendente*.
- Utilizaremos vectores de enteros para los ejemplos
  - Los algoritmos de clasificación sobre cualquier tipo de datos en el que se pueda establecer un orden

Índice

- ✓ Introducción
- ✓ Clasificación en memoria principal
  - ✓ Burbuja
  - ✓ Selección directa
  - ✓ Inserción directa
  - ✓ Método Shell
  - ✓ Bucket sort (Clasficación por casilleros)
  - ✓ Ordenación rápida
  - ✓ Árboles binarios ordenados
- ✓ Clasificación en memoria secundaria

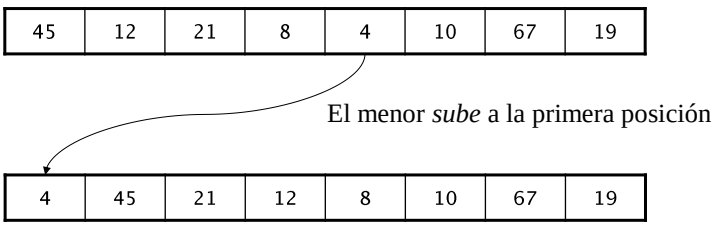
UCAM | Universidad Católica San Antonio

Índice

- ✓ Introducción
- ✓ Clasificación en memoria principal
  - ✓ Burbuja
  - ✓ Selección directa
  - ✓ Método Shell
  - ✓ Bucket sort (Clasficación por casilleros)
  - ✓ Ordenación rápida
  - ✓ Árboles binarios ordenados
- ✓ Clasificación en memoria secundaria

Clasificación M.P.: Burbuja

- ✓ La idea es imaginar que los elementos menores *suben* como burbujas hacia las primeras posiciones intercambiándose con los elementos mayores.



UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Burbuja

- ✓ Algoritmo:

```
void burbuja (int v[], int size){
       int i, j, interc;
       for (i = 0; i < size-1; i++) // Todos los elementos
              for(j = i+1; j < size; j++)// por cada elemento
                     if (v[j] < v[i]){
                            interc = v[j];
                            v[j] = v[i];
                            v[i] = interc;
                     }
}
```

Clasificación M.P.: Burbuja

✓ Ejemplo de aplicación:

Iteración	Valores							
0	44	55	12	42	94	18	6	67
1	6	55	44	42	94	18	12	67
2	6	12	55	44	94	42	18	67
3	6	12	18	55	94	44	42	67
4	6	12	18	42	94	55	44	67
5	6	12	18	42	44	94	55	67
6	6	12	18	42	44	55	94	67
7	6	12	18	42	44	55	67	94

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Burbuja

✓ Análisis de complejidad

- Para la iteración ***for*** interna se realizan:
  - n-1 comparaciones en la primera.
  - n-2 comparaciones en la segunda.
  - y así sucesivamente, siendo el número total de comparaciones:

$$t = (n-1) + (n-2) + ... + 1 = \sum_{i=1}^{n-1} i = \frac{1}{2}(n^2 - n)$$

Por tanto, el orden es $O(n^2)$

- Ojo! Tarda lo mismo tanto si el vector está ordenado como si no lo está.

Clasificación M.P.: Burbuja

- ✓ Análisis de complejidad
  - Realiza muchas comparaciones
    - ¿Cuántas en el peor caso? (Vector totalmente desordenado)
    - ¿Cuántas en el mejor caso? (Vector ordenado)
  - Realiza muchos intercambios
    - ¿Cuántas en el peor caso? (Vector totalmente desordenado)
    - ¿Cuántas en el mejor caso? (Vector ordenado)

Índice

Clasificación M.P.: Selección directa

- ✓ En cada iteración *i* busca la posición del elemento menor a partir de la posición *i* del vector .
- ✓ Cuando lo encuentra lo intercambia por el valor que hay en la posición *i* actual

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Selección directa

- ✓ Algoritmo:

```
void seleccion (int v[], int size){
       int i, j, pos, interc;
       for (i = 0; i < size-1; i++){
              pos = i;
              for(j = i+1;j<size; j++)
                     if (v[j] < v[pos])
                            pos = j;
              interc = v[pos];
              v[pos]=v[i];
              v[i] = interc;
       }
}
```

Clasificación M.P.: Selección directa

✓ Ejemplo

Iteración	Valores							
0	**44**	55	12	42	94	18	**6**	67
1	*6*	**55**	**12**	42	94	18	*44*	67
2	*6*	*12*	***55***	42	94	**18**	44	67
3	*6*	*12*	*18*	**42**	94	*55*	44	67
4	*6*	*12*	*18*	*42*	**94**	55	**44**	67
5	*6*	*12*	*18*	*42*	*44*	**55**	*94*	67
6	*6*	*12*	*18*	*42*	*44*	*55*	**94**	**67**
7	*6*	*12*	*18*	*42*	*44*	*55*	*67*	94

UCAM | Universidad Católica San Antonio

Clasificación M.P.: Selección directa

✓ Análisis:

- A diferencia del algoritmo de la burbuja, sólo se hace un intercambio por cada iteración
- Como máximo habrá *n* intercambios
  - ¿Cuántos se realizan en el mejor caso? (Vector ordenado)
- Inconveniente: Se siguen realizando muchas comparaciones, como en el algoritmo de la burbuja
- El orden de complejidad sigue siendo **$O(n^2)$**

Índice

Inserción directa

- ✓ Se basa en ir ordenando subsecuencias del array cada vez más grandes, con un recorrido de derecha a izquierda
- ✓ La idea central de este algoritmo consiste en insertar el siguiente elemento del array sin ordenar en la parte izquierda del mismo, que ya se encuentra ordenada.
- ✓ Este proceso se repite desde el segundo hasta el n-ésimo elemento.
- ✓ Conocido también por el algoritmo de la baraja, pues simula el proceso de un jugador colocando sus cartas de menor a mayor

Inserción directa

✓ Ejemplo. A: 15, 67, 8, 16, 44, 27, 12, 35

Primera pasada (i = 1)

A[1] < A[0] → 67 < 15 No hay intercambio

A: 15, 67, 8, 16, 44, 27, 12, 35

Segunda pasada (i=2)

A[2] < A[1] → 8 < 67 Sí hay intercambio *A: 15, 8, 67, 16, 44, 27, 12, 35*

A[1] < A[0] → 8 < 15 Sí hay intercambio *A: 8, 15, 67, 16, 44, 27, 12, 35*

Tercera pasada (i =3)

A[3] < A[2] 16 < 67 Sí hay intercambio *A: 8, 15, 16, 67, 44, 27, 12, 35*

A[2] < A[1] 16 < 15 No hay intercambio

fin de la iteración

Cuarta pasada (i =4) ….

UCAM | Universidad Católica San Antonio

Inserción directa

✓ Algoritmo:

```
void InsercionDirecta (int a[], int size){
       int pivote;
       int j;
       for (int i=1; i< size; i++) {
              pivote=a[i];
              j=i-1;
              while (j>=0 && pivote < a[j]) { // para cuando
                     a[j+1]=a[j];             // llega al
                     j--;                     // inicio o ya
              }                               // está orden
              a[j+1]=pivote;
       }
}
```

Inserción directa

- ✓ Caso mejor → O(n), inicialmente ordenado
- ✓ Caso peor → O(n^2), inversamente ordenado
- ✓ Nº Comparaciones del pivote
  - Mínimo 1, máximo i-1
- ✓ Nº Intercambios → Nº de comparaciones +1 (final)
- ✓ Ventajas
  - Eficiente con pocos datos y que tengan ya un cierto orden
  - Más eficiente en la práctica que burbuja, aunque su peor tiempo teórico sigue siendo O(n^2)
- ✓ Problemas
  - Pierde mucho tiempo en saber dónde insertar el pivote

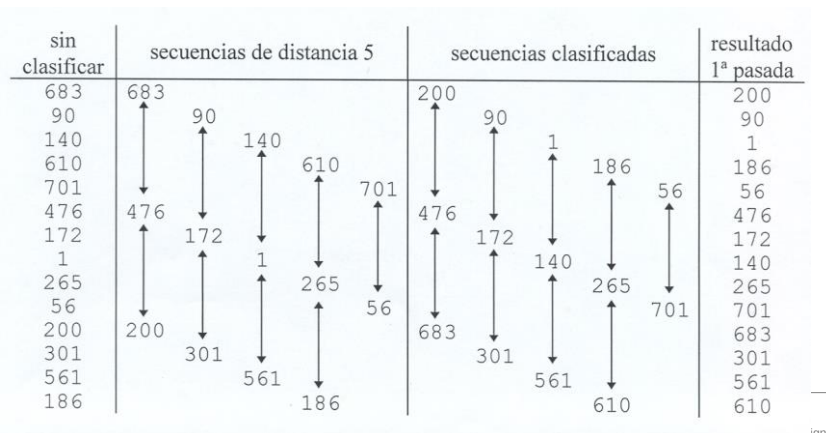
Índice

Clasificación M.P.: Método shell

- ✓ Se define una secuencia h_1, h_2, ...,h_t decreciente.
  - Ej: 5, 3, 1
- ✓ Se aplica la inserción directa para los elementos del intervalo h
  - De esta forma se reducen los intervalos de ordenamiento
  - Se realizan comparaciones entre elementos que ocupan posiciones muy distantes.
  - Se obtiene un orden parcial aprovechado posteriormente.
  - En la última iteración (h = 1) es presumible que el número de comparaciones no sea proporcional a n^2.

Clasificación M.P.: Método shell

- ✓ Ejemplo con h=(5,3,1)



Clasificación M.P.: Método shell

✓ Ejemplo con h=(5,3,1)

resultado 1ª pasada	secuencias de distancia 3			secuencias clasificadas			resultado 2ª pasada	resultado de la última pasada (con distancia 1)
200	200			172			172	1
90		90			56		56	56
1			1			1	1	90
186	186			186			186	140
56		56			90		90	172
476			476			265	265	186
172	172			200			200	200
140		140			140		140	265
265			265			301	301	301
701	701			561			561	476
683		683			610		610	561
301			301			476	476	610
561	561			701			701	683
610		610			683		683	701

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Método shell

✓ Análisis:

- Al dar la pasada final (distancia 1) el orden parcial es considerable.
  - El número de posiciones que un elemento se desplaza es pequeño.
  - Es de esperar que se realice en un tiempo menor que n^2.
- La elección de intervalos no es trivial.
  - Puede elegirse cualquiera.
  - La última será igual a 1.
  - No todas son igual de eficientes.

Clasificación M.P.: Método shell

✓ Análisis:

- Los intervalos potencia de 2 (16, 8, 4, 2, 1) implican comparar siempre los mismos elementos.
- Un trabajo posterior de Knuth evidencia que una elección razonable de incrementos es: …, 121, 40, 13, 4, 1.
- Para calcular el intervalo de Knuth tenemos:
  - $t = \lfloor \log_3 n \rfloor - 1$, siendo *n* el tamaño del vector → tamaño secuencia
  - $h_t = 1$ y
  - $h_{k-1} = 3*h_k + 1$,

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Método shell

✓ Análisis:

- Knuth también recomienda: …, 31, 15, 7, 3, 1. Para calcular esta secuencia:
  - $t = \lfloor \log_2 n \rfloor - 1$, siendo *n* el tamaño del vector.
  - $h_t = 1$ y
  - $h_{k-1} = 2*h_k + 1$
- Con esta última versión se necesita un esfuerzo que crece entre $n^{1.2}$ y $n^{1.5}$.

Clasificación M.P.: Método shell

✓ Algoritmo:

```
void shell(int array[], int size) {
   int i, j, intervalo, temp;
   intervalo = calcular_tamano_incremento(size);
   while (intervalo > 0) {
     for (i=intervalo; i < size; i++) {
          j = i;
          temp = array[i];
          while ((j >= intervalo) && (array[j - intervalo] > temp)){
                    array[j] = array[j - intervalo];
                    j = j - intervalo;
          }
          array[j] = temp;
     }
     intervalo = calcular_siguiente_incremento(size, intervalo);
   }
}
```


UCAM | Universidad Católica San Antonio

Clasificación M.P.: Método shell

✓ Algoritmo:

- `calcular_tamano_incremento(size)` calcula el primer incremento h a aplicar
- `calcular_siguiente_incremento(size, intervalo)` calcular el siguiente incremento de h a partir del incremento anterior

✓ En estas dos funciones es donde radica el buen (o mal) comportamiento del algoritmo y es el lugar a analizar para mejorar el comportamiento del algoritmo

Clasificación M.P.: Método shell

- ✓ Observad que mediante este bucle

```
for (i=intervalo; i < size; i++)
```

se va realizando la inserción directa con los saltos correspondientes y es posible que se compare un elemento del vector más de una vez

  - Ej: Para la primera iteración, se comparan los números 200 y 476, y más adelante se vuelven a comparar estos dos números con 683, que está en la primera posición.

- ✓ ¿Sería correcto empezar con la "i" en la última posición del vector para intentar realizar menos pasadas, de esta manera

```
for (i = size -1; i > (size-1)-incremento; i--)
```

¿Por qué?

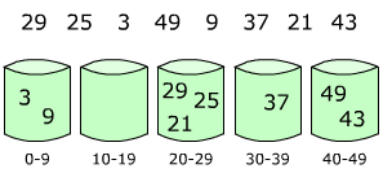
Índice

Bucket Sort

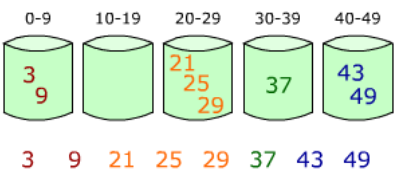
- ✓ Distribuye todos los elementos a ordenar entre un número finito de casilleros.
- ✓ Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones
  - En el caso de números, cada casillero representa un intervalo de números
- ✓ Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos.
- ✓ Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación
  - También se puede utilizar recursivamente el algoritmo Bucket Sort

Bucket Sort

- ✓ Distribución inicial



- ✓ Tras ordenar cada cubo y mezclarlos en orden



Bucket Sort

✓ Pasos del algoritmo:

1. Crear una colección de casilleros vacíos
2. Colocar cada elemento a ordenar en un único casillero
3. Ordenar individualmente cada casillero
4. Devolver los elementos de cada casillero concatenados por orden

Bucket Sort

✓ Pseudocódigo

función bucket-sort(elementos, n)
casilleros ← colección de k listas
para i = 1 **hasta** longitud(elementos) **hacer**
c ← buscar el casillero adecuado para *elementos[i]*
insertar *elementos[i]* en casillero[c]
fin para
para i = 1 **hasta** k **hacer**
ordenar(casilleros[i])
fin para
devolver la concatenación de casilleros[1],..., casilleros[k]

Bucket Sort

✓ Complejidad

- Peor caso: Todos los elementos en el mismo cubo → $O(n^2)$, donde *n* es el número de elementos a ordenar
  - 1 pasada para distribuir los N números en los cubos → O(n)
  - Ordenar los n números del cubo → $O(n^2)$
  - Total = $O(n) + O(n^2) = O(n^2)$

- Caso medio: Elementos dispersos en diferentes cubos → O(n+k), donde k es el número de cubos
  - 1 pasada para distribuir los N números en los cubos → O(n)
  - Ordenar cada cubo → $C_i(n\backslash k)$
  - Ordenar todos los cubos → $O(k *(C_i(n\backslash k)))$
  - Total = $O(n) + O((k *(C_i(n\backslash k))) = O(n + k *(C_i(n\backslash k))$ → O(n+k)

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Bucket Sort

✓ Complejidad

- Ejemplo caso medio, con n = 100, k = 10
  - Ordenar cada cubo tiene una complejidad de O(100/10) = O(10).
  - Ordenar 10 elementos con burbuja puede costar 100 iteraciones (10^2) en el peor caso, o quizá menos si se aplican mejoras al algoritmo.
  - Incluso puede haber cubos vacios
  - Si suponemos que cada cubo cuesta 100 iteraciones, tenemos un coste total

    $O(n) + O((k *(C(n\backslash k))) = O (100) + O(10 * 100)$

  - Observar que no puedo eliminar 'k' porque está multiplicando y dividiendo, ya que afecta a factores diferentes!!!
    - Si fuera así, tendríamos $O(n) + O((k *(O(n\backslash k))) = O(n) + O(n) = O(200)$
    - NO ES CORRECTO!!!

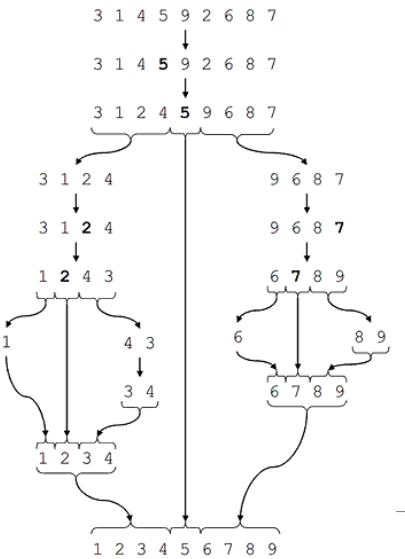
Índice

Clasificación M.P.: Ordenación rápida

- ✓ Conocido también como *Quick Sort*
- ✓ Es uno de los más rápidos conocidos
  - Sigue la estrategia *divide y vencerás*
- ✓ Algoritmo recursivo de 4 pasos:
  - Si número de elementos de S (secuencia a ordenar) es 0 ó 1, terminar
  - Escoger un elemento cualquiera *v* de S. Ese elemento se denomina pivote
  - Hacer una partición, tal que elementos >*v* forman la partición D y elementos <*v* forman la partición I.
  - Devolver el resultado de quicksort (I), seguido de v y seguido del resultado de quicksort (D)

Clasificación M.P.: Ordenación rápida

✓ Ejemplo
(en negrita los pivotes)



UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Ordenación rápida

✓ ¿Cómo elegir el pivote?
- La elección del pivote es heurística.
  - Óptimo = divide en dos listas de tamaño idéntico.
- La *suerte* de elegir uno u otro determinará la eficiencia de nuestro algoritmo.
- Estudiemos el peor y el mejor caso.

✓ ¿Cómo se realiza la partición?
- Siguientes diapositivas

Clasificación M.P.: Ordenación rápida

- ✓ **Elección del pivote**
  - Primer elemento: Elección incorrecta
    - Puede ser válida para entradas aleatorias
    - Si vector ordenado o inversamente ordenado: peor partición (estamos ante el caso peor)
  - (Pos.Inicial+Pos.final)/2: Elección segura
    - Vector ordenado: Ideal.
    - Puede darse el caso de un vector de entrada en el cual se comporte como el caso peor; muy pocas probabilidades de que ocurra
  - Al azar: A ciegas
    - No requiere cálculo adicional
    - Puede llevar al peor caso en algunas entradas

Clasificación M.P.: Ordenación rápida

- ✓ **Elección del pivote (II)**
  - Mediana: Mejor partición
    - La mediana del vector completo puede llevar mucho tiempo
    - Se ha probado que con una muestra de 3 elementos se obtiene el mejor caso de mediana en promedio
    - ¿Qué 3 elementos se utilizan para obtener el pivote?
      - ➢ **El primer, último y elemento central del vector**
      - ➢ **Se toma el valor del medio como pivote**
    - Ej: 3 1 4 5 9 2 5 8 7

      Pivote → mediana (3, 7, 5) → **5**

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Se llevan a cabo los siguientes pasos

1. Intercambiar el pivote con el último elemento
2. Usamos dos punteros, *i* y *j*.
   i = pos_ini y *j* = pos_ultimo -1.
3. Bucle:
   - 3.1. Movemos i de izquierda a derecha hasta que s[i]>pivote o i > j
   - 3.2. Movemos j de derecha a izquierda hasta que s[j]<pivote o j < i
   - 3.3 Si i<j, intercambiar elementos s[i] , s[j], luego hacer i++ y j-- y continuar, si no dar por concluido el bucle.
4. Intercambiar el elemento de posición i con el pivote

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo: 5-3-7-4-2-1-6
  - Tomamos 4 como el pivote → 5-3-6-**4**-2-1-6

1. Intercambiamos el pivote por el último elemento
   5-3-7-**6**-2-1-**4**
2. i = 0, j = 5
3. Bucle :
   i j
   5-3-7-6-2-1-4
   - 3.1. s[i] > pivote? → Sí, parar *i*
   - 3.2. s[j] < pivote? → Sí, parar *j*
   - 3.3. i <j ? → Sí, intercambiar posiciones

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo(II): 5-3-4-6-2-1-7

3 .Bucle :

1-3-7-6-2-5-4

3.3. i++, j --

i j
1-3-7-6-2-5-4

3.1. s[i] > pivote? → No, i++

i j
1-3-7-6-2-5-4

3.1. s[i] > pivote? → Sí, parar *i*

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo(III): 5-3-4-6-2-1-7

3 .Bucle :

i j
1-3-7-6-2-5-4

3.2. s[j] < pivote? → No, j--

i j
1-3-7-6-2-5-4

3.2. s[j] < pivote? → Sí, parar *j*

3.3. i <j ? → Sí, intercambiar posiciones

i j
1-3-2-6-7-5-4

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo(IV): 5-3-4-6-2-1-7

3 .Bucle :

j

1-3-2-6-7-5-4

3.3. i++, j - -

j

1-3-2-6-7-5-4

3.1. s[i] > pivote? → Sí, parar *i*

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo(V): 5-3-4-6-2-1-7

3 .Bucle :

j

1-3-2-6-7-5-4

3.2. s[j] < pivote? → No, j--

1-3-2-6-7-5-4

j i

3.2 . *j* < *i*, parar *j*

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- Ejemplo(y VI): 5-3-4-6-2-1-7

3 .Bucle :

j i
1-3-2-6-7-5-4

3.3 . *i >j ?* → No, parar bucle

4. Intercambiar *s[i]* por *pivote*

1-3-2-**4**-7-5-6

Ahora hay que repetir el algoritmo para la parte izquierda y para la parte derecha de 4

Clasificación M.P.: Ordenación rápida

✓ **Partición**

- ¿Qué ocurre con los elementos de s iguales al *pivote*?
  - Los punteros de i y de j deben seguir para obtener un buen rendimiento
  - Si sólo para uno, se perjudica la parte controlada por el otro puntero:
    - Ej: i para y j no → todos los elementos iguales al pivote acaban en la parte derecha

Clasificación M.P.: Ordenación rápida

✓ **Código** (tomando (pos_ini+pos_fin)/2 como pivote)

```
public QuickSort(int a[], int ini, int fin) {
      int pivote;
      int i,j;

     // Caso vector con 1 solo número --> No hacer nada
     if(ini >= fin) return;

     // Caso vector con 2 números --> Comprobar si es necesario ordenarlos
     if(ini+1 == fin){
       if(a[ini] > a[fin])  // No están ordenados los dos números, intercambiar
           intercambio(a, ini, fin);
       return;
     }

     // Resto de casos --> vector con 3 números o más

     // Pivote --> La posición de mitad del vector
     int medio = (ini+fin)/2;
```


✓ **Código (cont)** (tomando (pos_ini+pos_fin)/2 como pivote)

```
public QuickSort(int a[], int ini, int fin) {
     // Intercambiar pivote por el último elemento
     intercambio(a, medio, fin);
     pivote=a[fin];
     // empezamos la particion
     for(i=ini,j=fin-1;;){
        while((i <= fin-1) && (a[i]<=pivote)) i++;
        while((j >= ini) && (pivote <=a[j])) j--;

        if(i<j){  // Todavía no se han intercambiado los índices, intercambiar números
            intercambio(a,i,j);
            i++; j--;
        }
        else  // Se han intercambiando los índices, fin de la particion
            break;
     }
     //colocación del pivote en su sitio
     intercambio (a, i, fin);
     //termina particion; //llamadas recursivas
     QuickSort(a, ini, i-1);  // Vector de la izquierda del pivote
     QuickSort(a, i+1, fin);  // Vector de la derecha del pivote
}
```

Clasificación M.P.: Ordenación rápida

✓ **Complejidad**

- Peor caso: Particiones totalmente desiguales
  - Ej: Partición izquierda con un elemento y derecha con n-1 elementos
  - Denotaremos T_n el tiempo necesario para clasificar *n* elementos

$$T_n = c_1 n + T_1 + T_{n-1}$$

$$T_1 = c_2$$

donde:

- c_1 = tiempo en cada comparación y/o recolocación de elementos.
- c_2 = tiempo invertido en cada recursión para clasificar un solo elemento.

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Ordenación rápida

✓ **Complejidad**

- Peor caso: Particiones totalmente desiguales
  - Desarrollando la fórmula $T_n = c_1 n + T_1 + T_{n-1}$

$$T_{n-1} = c_1(n-1) + T_1 + T_{n-2}$$

$$T_n = c_1(n) + c_1(n-1) + 2c_2 + T_{n-2}$$

si continuamos llegamos a que el peor caso para n elementos es…

$$T_n = c_2 n + [c_1 n + c_1(n-1) + \ldots + c_1 2 + c_1] = c_2 n + c_1 * \sum_{i=1}^{n} i = c_2 n + \frac{c_1}{2}(n^2 + n)$$

Y por tanto, **$O(n^2)$**

Clasificación M.P.: Ordenación rápida

✓ **Complejidad**

- Mejor caso: Particiones iguales
  - Suponemos que *n* es potencia de 2

    $T_1 = c_2$

    $T_n = c_1 n + T_{n/2} + T_{n/2} = c_1 n + 2T_{n/2}$

  - Al ser *n* potencia de 2, tenemos que

$$T_{2^i} = c_1 2^i + T_{2^{(i-1)}} + T_{2^{(i-1)}} = c_1 2^i + 2T_{2^{(i-1)}}$$

Clasificación M.P.: Ordenación rápida

✓ **Complejidad**

- Mejor caso: Particiones iguales
  - Desarrollando la fórmula anterior:

$$T_{2^{(i-1)}} = c_1 2^{(i-1)} + 2T_{2^{(i-2)}}$$

$$T_{2^i} = c_1 2^i + 2c_1 2^{(i-1)} + 4T_{2^{(i-2)}} = c_1 2^i + c_1 2^i + 4T_{2^{(i-2)}}$$

…

$$T_{2^i} = ic_1 2^i + 2^i c_2$$

Clasificación M.P.: Ordenación rápida

✓ **Complejidad**

- Mejor caso: Particiones iguales
  - Finalmente, la fórmula $T_{2^i} = ic_1 2^i + 2^i c_2$

  se puede transformar a $T_n = c_1 n \log_2(n) + c_2 n$

Por lo tanto, el orden es **O(n·log(n))**, que es una fuerte reducción con respecto a $O(n^2)$

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Ordenación rápida

✓ **Uso de Quick Sort**

- Ventajas:
  - Rápido para valores n > 20
  - Ejecución en paralelo de cada partición
- Desventajas
  - Utiliza recursividad (sobrecarga en memoria)
  - Excesiva complejidad para n < 20. En ese caso usar inserción directa

Clasificación M.P.: Ordenación rápida

✓ **Comparación con burbuja**

http://www.youtube.com/watch?v=aXXWXz5rF64

✓ **Ejercicio.** Ordenar por Quick Sort

3-4-9-5-6-5-9-2-4-7-5-9-4-9

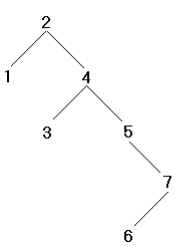
Índice

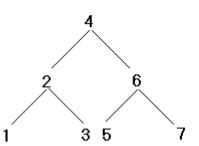
Índice

Clasificación M.P.: Árbol de búsquedas

✓ Un árbol binario de búsqueda(ABB) es un árbol binario con las siguientes propiedades:

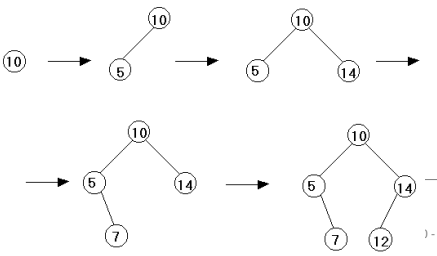
- Todos los elementos almacenados en el subárbol izquierdo de cualquier nodo *x* son menores que el elemento almacenado en *x* ,
- Todos los elementos almacenados en el subárbol derecho de *x* son mayores que el elemento almacenado en *x*.





Clasificación M.P.: Árbol de búsquedas

- ✓ Un recorrido in-order nos dará la lista ordenada de elementos
  - Dado un nodo, se recorre primero el subárbol izquierdo, luego el nodo actual, y después el subárbol derecho
- ✓ El nodo raíz del árbol actúa como el pivote de QuickSort
- ✓ La principal ventaja de los ABB es que permite la inserción ordenada de elementos
  - Ej: {10,5,14,7,12}



UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.P.: Árbol de búsquedas

- ✓ Complejidad
  - Para obtener la lista ordenada, se recorre cada nodo una sola vez en inorden → **O(n)**
  - Insertar cada elemento en el ABB tiene una complejidad **O($\log_2$ n)**
  - **Por tanto, insertar todos los elementes es O(n* $\log_2$ n)**
  - **Total: O(n) + O(n* $\log_2$ n)**

Índice

Clasificación M.S.: Introducción

✓ La clasificación en memoria secundaria se aplica:
- Para colecciones de datos muy grande.
- No es posible cargar en memoria principal toda la colección

✓ Se suelen hacer operaciones de mezcla.
- Mezclar: obtener una secuencia ordenada a partir de otras dos (o más).
- En memoria principal sólo tendremos un elemento de cada secuencia

Clasificación M.S.: Introducción

✓ Ejemplo de operación de mezcla:

- Tenemos dos secuencias f1 y f2

f1	6	14	37	65	83
f2	7	10	73	90	94

Compara: 6 y 7. El menor es el 6. Lo graba en la salida. Y coge el siguiente de f1 ya que ha sacado de esa secuencia

Salida	6

Clasificación M.S.: Introducción

✓ Ejemplo de operación de mezcla:

- Tenemos dos secuencias f1 y f2

f1	~~6~~	14	37	65	83
f2	7	10	73	90	94

Compara: 14 y 7. El menor es el 7. Lo graba en la salida. Y coge el siguiente de f2 ya que ha sacado de esa secuencia

Salida	6	7

Clasificación M.S.: Introducción

✓ Ejemplo de operación de mezcla:
- Tenemos dos secuencias f1 y f2

f1	~~6~~	14	37	65	83
f2	~~7~~	10	73	90	94

Compara: 14 y 10. El menor es el 10. Lo graba en la salida. Y coge el siguiente de f2 ya que ha sacado de esa secuencia

Salida	6	7	10

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Introducción

✓ Ejemplo de operación de mezcla:
- Tenemos dos secuencias f1 y f2

f1	~~6~~	14	37	65	83
f2	~~7~~	~~10~~	73	90	94

Compara: 14 y 73. El menor es el 14. Lo graba en la salida. Y coge el siguiente de f1 ya que ha sacado de esa secuencia

Salida	6	7	10	14	…	…	…	…	…	…

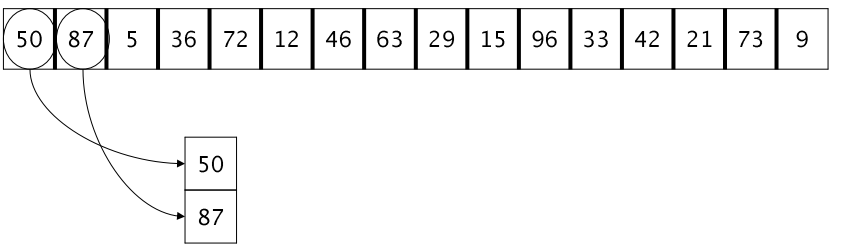
Índice

Clasificación M.S.: Mezcla directa

- ✓ Se realizan pasadas de mezcla tratando de obtener secuencias ordenadas cada vez más largas.
- ✓ Al final se obtiene una única secuencia ordenada.

Clasificación M.S.: Mezcla directa

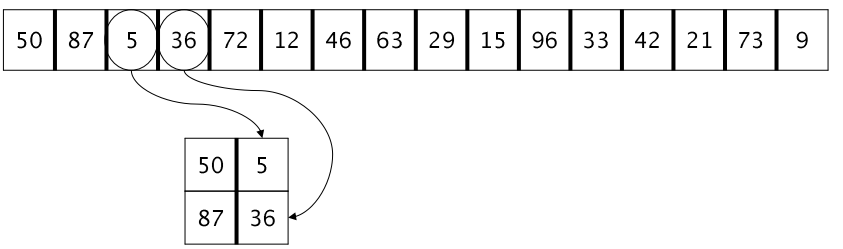
- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias de longitud 1
  - Repartimos cada parte en dos secuencias



UCAM | Universidad Católica San Antonio

Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias de longitud 1
  - Repartimos cada parte en dos secuencias



Clasificación M.S.: Mezcla directa

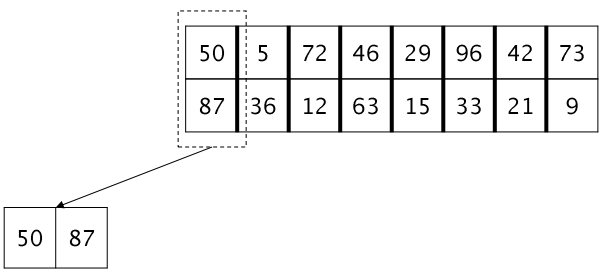
- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias de longitud 1
  - Repartimos cada parte en dos secuencias

50	87	5	36	72	12	46	63	29	15	96	33	42	21	73	9

50	5	72	46	29	96	42	73
87	36	12	63	15	33	21	9

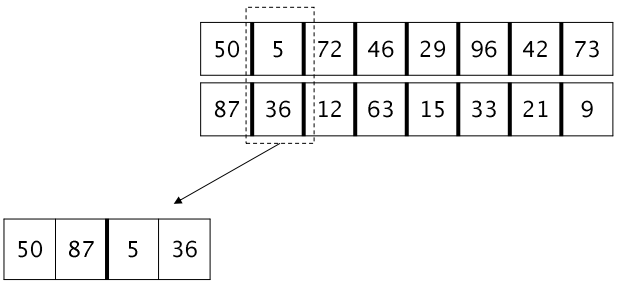
Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(II):
  - Combinamos las subsecuencias aplicando una mezcla ordenada de los elementos de cada subsecuencia



Clasificación M.S.: Mezcla directa

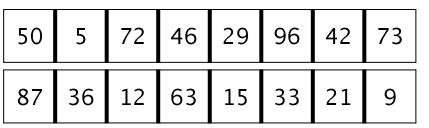
- ✓ Funcionamiento(II):
  - Combinamos las subsecuencias aplicando una mezcla ordenada de los elementos de cada subsecuencia

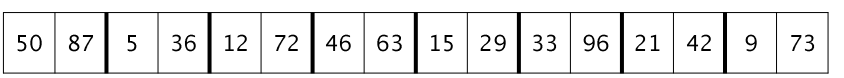


UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla directa

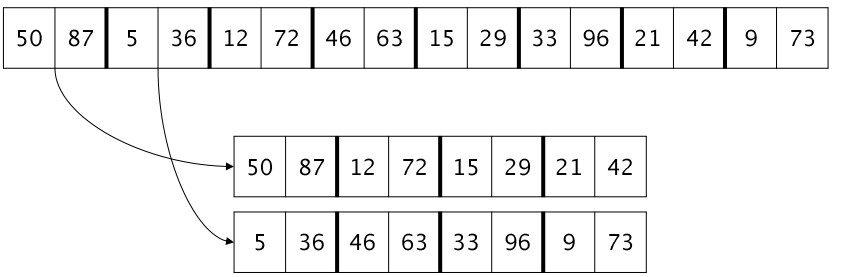
- ✓ Funcionamiento(II):
  - Combinamos las subsecuencias aplicando una mezcla ordenada de los elementos de cada subsecuencia





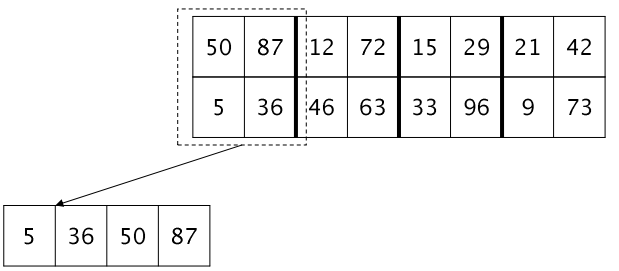
Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(III):
  - Volvemos a dividir la secuencia en subsecuencias, ahora de longitud 2
  - Repartimos cada parte en dos secuencias



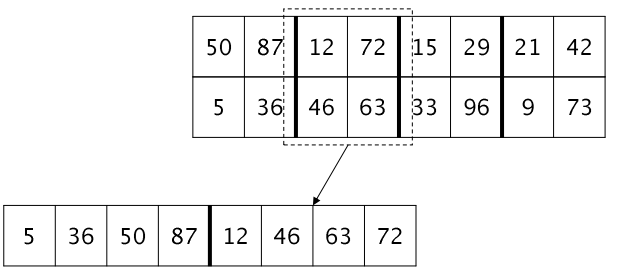
Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(IV):
  - Combinamos las subsecuencias aplicando mezcla



Clasificación M.S.: Mezcla directa

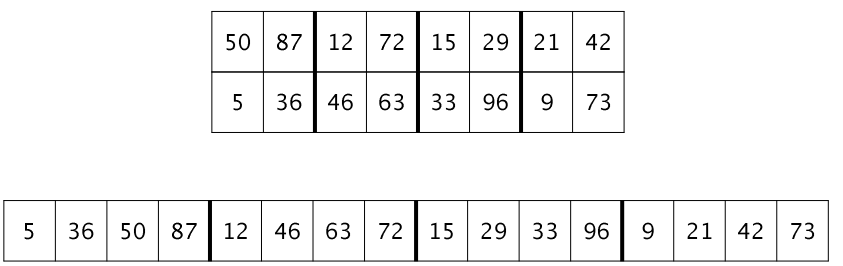
- ✓ Funcionamiento(IV):
  - Combinamos las subsecuencias aplicando mezcla



UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(IV):
  - Combinamos las subsecuencias aplicando mezcla



Clasificación M.S.: Mezcla directa

- ✓ Funcionamiento(V):
  - Dividimos la secuencia en subsecuencias de tamaño 4
  - Las ordenamos con mezcla
  - Dividimos la secuencia en subsecuencias de tamaño 8
  - Las ordenamos
  - …

5	9	12	15	21	29	33	36	42	46	50	63	72	73	87	96

Clasificación M.S.: Mezcla directa

- ✓ Código

```
void mezclaDirecta(FILE *a, int n) {   // n es el tamaño de a
        p=1;
        int size_b, size_c;
        size_b = ((n/2)% 2 == 0) ? n/2 : (n/2)+1;
        size_c = n/2;
        // Crear FILE b y c con sus tamaños
        while (p<n) {
                        /*distribuimos los elementos de 'a' de 'p' en
                          'p' en las  dos subsecuencias de 'b' y 'c'*/
                distribuir (a, b, c, p, n);
                        /*Mezclamos las subsecuencias 'b' y 'c' de
                  longitud 'p' sobre 'a'*/
                mezcla(a, b, c, size_b, size_c);
                p=p*2;
        }
        // Eliminar archivos b y c
}
```

Clasificación M.S.: Mezcla directa

✓ Código

```
void distribuir(FILE *a, FILE *b, FILE *c, int p, int n) {

        int i = 0;
        while (i < n){
                //Copiar siguientes 'p' elementos de 'a' en 'b'
                i+= p;
                if (i < n) {
                      //Copiar siguientes 'p' elementos de 'a' en 'c'
                      i+ = p;
                }
        }
}
```


UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla directa

✓ Código

```
void mezcla(FILE *a, FILE *b, FILE *c, int size_b, int size_c) {

        int p_b = 0, p_c = 0;
        while (p_b < size_b || p_c < size_c){
                  if ((p_b < size_b) && (b[p_b] <= c[p_c])){
                        a[p_b + p_c]= b[p_b];
                        p_b++;
                  }
                  if ((p_c < size_c) && (b[p_b] > c[p_c])) {
                        a[p_b + p_c] = c[p_c];
                        p_c++;
                  }

        }
}
```

Clasificación M.S.: Mezcla directa

- ✓ Complejidad
  - El bucle principal de distribuir y ordenar se ejecuta $\lceil \log_2(n) \rceil + 1$ veces
  - distribuir() tiene complejidad O(n)
  - mezcla() tiene complejidad O(n)

- ✓ Complejidad total: $[O(n)+O(n)]*O(\log_2 n)=O(2n*\log_2 n)=$ **$O(n*\log_2 n)$**

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla directa

- ✓ Ejercicio
  - Aplicar mezcla directa a la siguiente secuencia

9	7	2	1	4	5	3	6	8

Índice

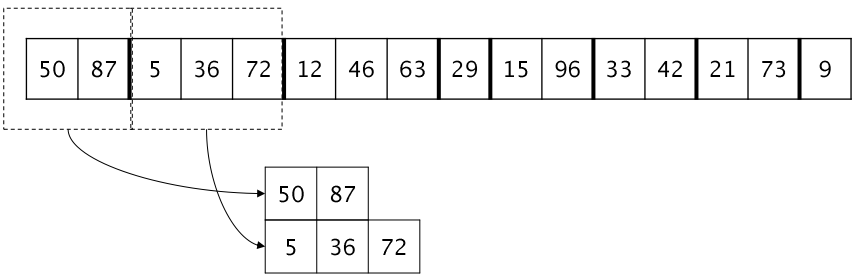
Clasificación M.S.: Mezcla natural

- ✓ La idea principal es no romper las subsecuencias ordenadas que se encuentran inicialmente en la secuencia original

- ✓ Se realizan pasadas de mezcla tratando de obtener secuencias ordenadas cada vez más largas.

Clasificación M.S.: Mezcla natural

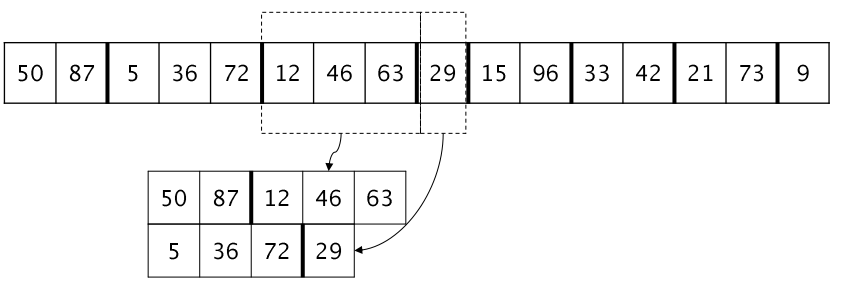
- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias que ya estén ordenadas.
  - Repartimos cada parte en dos secuencias



UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla natural

- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias que ya estén ordenadas.
  - Repartimos cada parte en dos secuencias



Clasificación M.S.: Mezcla natural

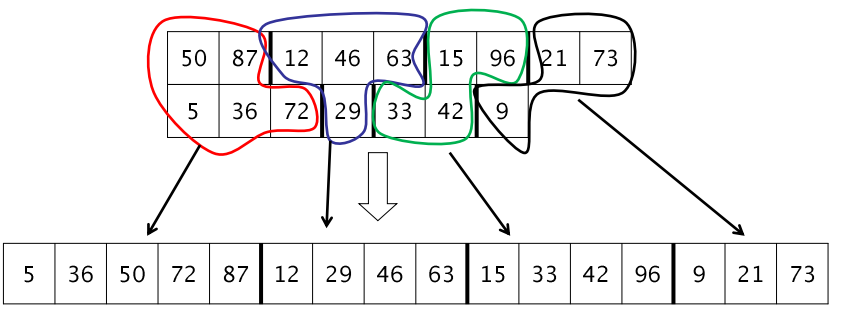
- ✓ Funcionamiento(I):
  - Dividimos la secuencia en subsecuencias que ya estén ordenadas.
  - Repartimos cada parte en dos secuencias

50	87	5	36	72	12	46	63	29	15	96	33	42	21	73	9

50	87	12	46	63	15	96	21	73
5	36	72	29	33	42	9		

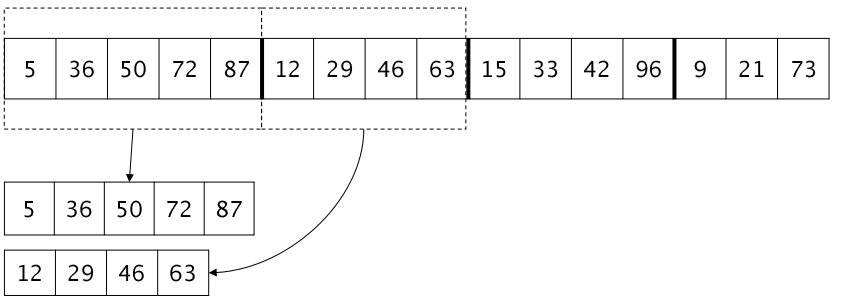
Clasificación M.S.: Mezcla natural

- ✓ Funcionamiento(II):
  - Combinamos las dos subsecuencias aplicando mezcla
  - Se mezclan por subsecuencias ordenadas



Clasificación M.S.: Mezcla natural

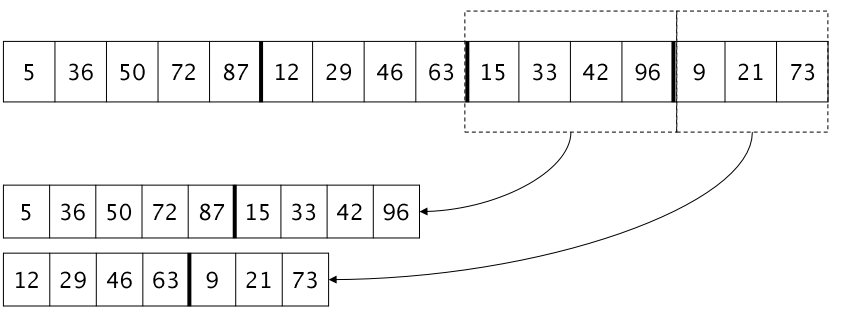
- ✓ Funcionamiento(III):
  - Repetimos el paso anterior I: Dividimos la secuencia en subsecuencias que ya estén ordenadas y repartimos cada parte en dos secuencias



UCAM | Universidad Católica San Antonio

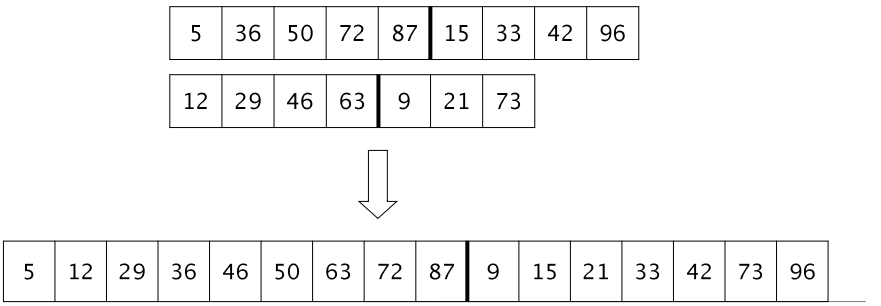
Clasificación M.S.: Mezcla natural

- ✓ Funcionamiento(III):
  - Repetimos el paso anterior I: Dividimos la secuencia en subsecuencias que ya estén ordenadas y repartimos cada parte en dos secuencias



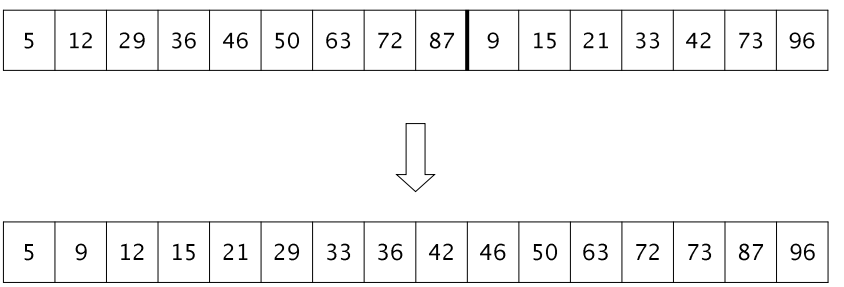
Clasificación M.S.: Mezcla natural

- ✓ Funcionamiento(IV):
  - Repetimos el paso II: Combinamos las dos subsecuencias aplicando mezcla
  - Se mezclan por subsecuencias ordenadas



Clasificación M.S.: Mezcla natural

- ✓ Funcionamiento(V):
  - Como ya sólo quedan dos subsecuencias, se repite la mezcla y acaba el proceso



Clasificación M.S.: Mezcla natural

- ✓ Complejidad
  - El bucle principal para obtener las subsecuencias y mezclar se ejecuta $\lceil \log_2(T) \rceil+1$ veces, donde ***T*** es el número de subsecuencias (tramos) ordenados que hay inicialmente en la secuencia original
  - distribuir() tiene complejidad O(n)
  - mezcla() tiene complejidad O(n)

- ✓ Complejidad total: $[O(n)+O(n)]*O(\log_2(T))=O(2n*\log_2(T))=$ **$O(n*\log_2(T))$**

UCAM | UNIVERSIDAD CATÓLICA SAN ANTONIO

Clasificación M.S.: Mezcla natural

- ✓ Complejidad
  - **Caso peor:** Vector inversamente ordenado
    - Inicialmente *n* tramos → igual que mezcla directa
  - **Caso mejor:** Vector ordenado: Inicialmente tenemos un solo tramo → Complejidad O(n)

- ✓ Ventajas:
  - Mejora la complejidad de la Mezcla Directa para vectores inicialmente ordenados
  - Se comporta mejor que MezclaDirecta al aprovechar la fusión de tramos.

Clasificación M.S.: Mezcla natural

- ✓ Ejercicio
  - Aplicar mezcla natural a la siguiente secuencia

9	7	2	1	4	5	3	6	8

Índice

Clasificación M.S.: Índice

- ✓ Realmente no es un método de clasificación.
- ✓ La idea es almacenar la información de manera que no sea preciso ordenarla cuando se recupera.
- ✓ La construcción de índices es frecuente en BBDD
- ✓ Es útil cuando se quieren buscar registros por uno de los campos
  - Dicho campos se denomina *clave*.

UCAM | Universidad Católica San Antonio

Clasificación M.S.: Índice

- ✓ Supongamos que en disco tenemos la siguiente información

posición	Nombre	Dirección	DNI	Teléfono
1	Sánchez López, Demetrio	Vinadel 4, 4º C	26934123	226533
2	Frutos Dólera, Fuensanta	Gran Vía 9, 3º A	22123004	223567
3	Albacete Ródenas, Hilario	San Antón 23, 1º A	53100894	290640
4	Imbernón Jover, Fulgencio	Simón García 9, 3º B	32103477	230981
5	Carrión Gómez, Sebastián	Mayor 73	15965445	831100
6	Cánovas Vera, Eulalia	Salvador Aledo 4, 1ºA	29234904	426989
7	Zamora Losada, Blas	Cartagena 42, 5º B	11087302	229870
8	Martínez Balsalobre, Pío	Juan Carlos I 25, 2º C	28462511	264218

Posición representa la posición relativa dentro del archivo donde se encuentra cada estructura (registro). No está grabado sino que está implícito

Clasificación M.S.: Índice

- ✓ Si queremos hacer una búsqueda por un campo, deberíamos tener la estructura ordenada por dicho campo.
- ✓ Si el fichero es muy grande hacer una ordenación previa es inviable.
- ✓ Si el fichero es pequeño una búsqueda secuencial puede ofrecer resultados aceptables.
- ✓ **Una solución intermedia consistiría en ordenar una tabla con sólo dos columnas:**
  - Posición relativa
  - Campo clave a buscar.

Clasificación M.S.: Índice

- ✓ Para hacer una búsqueda por DNI con índice tendríamos:

1	26934123
2	22123004
3	53100894
4	32103477
5	15965445
6	29234904
7	11087302
8	28462511

que tras ser ordenada por la segunda columna produciría →

7	11087302
5	15965445
2	22123004
1	26934123
8	28462511
6	29234904
4	32103477
3	53100894

Posición en el fichero **ÍNDICE**

Clasificación M.S.: Índice

✓ Con el índice:

- Se realiza una búsqueda binaria por la segunda columna **(por el valor del campo clave).**
- Tras encontrar el dato se toma de la primera columna la posición relativa **(posición del índice)**.
- Se localiza la posición en el fichero.

Clasificación M.S.: Índice

✓ **Complejidad:**

- El coste de este proceso es proporcional a **$O(n \cdot \log_2(n))$**
  - Construir la tabla recorriendo toda la tabla: → O(n).
  - Ordenarla → $O(n \cdot \log_2(n))$
  - Realizar una búsqueda binaria: $O(\log_2(n))$

- Una forma de reducir este coste sería mantener el índice según se insertan datos en el archivo
  - No tener que crear el índice de cero a partir de un archivo ya creado reduce el coste a $O(\log_2(n))$

Clasificación M.S.: Mezcla natural

- ✓ Ejercicio
  - Construir el índice para el campo Nombre y explicar su funcionamiento para localizar el registro con Nombre = “Frutos Dólera”

posición	Nombre	Dirección	DNI	Teléfono
1	Sánchez López, Demetrio	Vinadel 4, 4º C	26934123	226533
2	Frutos Dólera, Fuensanta	Gran Vía 9, 3º A	22123004	223567
3	Albacete Ródenas, Hilario	San Antón 23, 1º A	53100894	290640
4	Imbernón Jover, Fulgencio	Simón García 9, 3º B	32103477	230981
5	Carrión Gómez, Sebastián	Mayor 73	15965445	831100
6	Cánovas Vera, Eulalia	Salvador Aledo 4, 1ºA	29234904	426989
7	Zamora Losada, Blas	Cartagena 42, 5º B	11087302	229870
8	Martínez Balsalobre, Pío	Juan Carlos I 25, 2º C	28462511	264218

Bibliografía