Algoritmos de ordenación

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Describir algunos algoritmos de ordenación

¿Qué es un array?

- Es usual en los programas la necesidad de almacenar una lista de valores para después procesarlos.
- Una posibilidad es asociar a cada valor una variable, pero esto sería ineficiente y engorroso.
- Un **array** es una variable que almacena una lista de valores del mismo tipo.
- El array se almacena en posiciones continuas de memoria y el acceso a los elementos se realiza mediante índices.

Algoritmos de ordenación

- Ordenación o clasificación es el proceso de reordenar un conjunto de objetos en un orden específico.
- El propósito de la ordenación es facilitar la búsqueda de elementos en el conjunto ordenado.
- Existen muchos algoritmos de ordenación, siendo la diferencia entre ellos la eficiencia en tiempo de ejecución.
- Los métodos de ordenación se pueden clasificar en dos categorías: ordenación de ficheros o externa y *ordenación de arrays* o interna.

Algoritmos comunes - Ordenación

- Formalmente el problema del ordenamiento se expresa como:
  - Dados los elementos: $a_1, a_2, \ldots, a_n$
  - Ordenar consiste en permutar esos elementos en un orden: $a_{k_1}, a_{k_2}, \ldots, a_{k_n}$ tal que dada una función de ordenamiento f: $f(a_{k_1}) \leq f(a_{k_2}) \leq \ldots \leq f(a_{k_n})$
- Normalmente, la función de ordenamiento se guarda como un componente explícito (campo) de cada item (elemento). Ese campo se llama la *llave del item*.
- Un método de ordenamiento es *estable* si el orden relativo de elementos con igual llave permanece inalterado por el proceso de ordenamiento.

Algoritmos comunes - Ordenación

- Los métodos de ordenación buscan un uso eficiente de la memoria por lo que las permutaciones de elementos se hará *in situ* (uso del array original).
- Existen varios métodos de ordenación: burbuja, agitación, selección, inserción, quicksort, etc.

https://www.toptal.com/developers/sorting-algorithms

Método de Ordenación: burbuja

- Es un método caracterizado por la *comparación* e *intercambio* de pares de elementos hasta que todos los elementos estén ordenados.
- En cada iteración se coloca el elemento más pequeño (orden ascendente) en su lugar correcto, cambiándose además la posición de los demás elementos del array.
- La complejidad del algoritmo es $O(n^2)$.

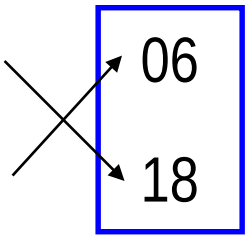
Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94							
18							
06 →	06						
67 →	67						

67 < 06 no hay intercambio

Método de Ordenación: burbuja

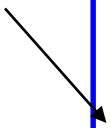
Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94							
18	06						
06	18						
67	67						



06 < 18 hay intercambio

Método de Ordenación: burbuja

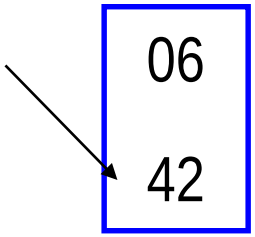
Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94	06						
18	94						
06	18						
67	67						



06 < 94 hay intercambio

Método de Ordenación: burbuja

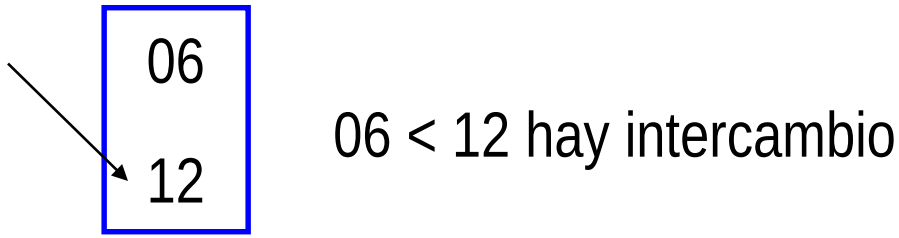
Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42	06						
94	42						
18	94						
06	18						
67	67						



06 < 42 hay intercambio

Método de Ordenación: burbuja

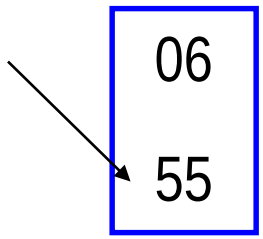
Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12	06						
42	12						
94	42						
18	94						
06	18						
67	67						



06 < 12 hay intercambio

Método de Ordenación: burbuja

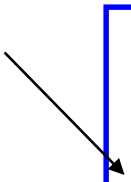
Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55	06						
12	55						
42	12						
94	42						
18	94						
06	18						
67	67						



06 < 55 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06						
55	44						
12	55						
42	12						
94	42						
18	94						
06	18						
67	67						



06 < 44 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06					
55	44	12					
12	55	44					
42	12	55					
94	42	18					
18	94	42					
06	18	94					
67	67	67					

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06	06				
55	44	12	12				
12	55	44	18				
42	12	55	44				
94	42	18	55				
18	94	42	42				
06	18	94	67				
67	67	67	94				

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06	06	06	06	06	06
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
06	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

Generación de datos random para ordenación

```
Private Sub B_ordto_Click(sender As Object, e As EventArgs)
  Handles B_ordto.Click
        Dim time As Stopwatch = Stopwatch.StartNew
        Dim N As Integer
        'numero de datos
        N = CInt(TB_numlista.Text)
        Dim items(N) As Integer
        Dim itemso(N) As Integer

        L_teb.Text = " "
        L_teq.Text = " "
        L_lista.Text = ""
        'generacion de lista de numeros aleatoria
        For i As Integer = 0 To N - 1
            items(i) = CInt(Int((N * Rnd()) + 1))
```

Generación de datos random para ordenación

```
'           L_lista.Text = L_lista.Text & items(i) & " "
        Next
        itemso = items
        time.Reset()
        time.Start()
             Bubblesort(itemso, N - 1)
        time.Stop()
        L_teb.Text = "Time bubble sorting: " &
   time.ElapsedMilliseconds.ToString & " msec"
        L_ordbur.Text = ""
        For i As Integer = 0 To N - 1
            '   L_ordbur.Text = L_ordbur.Text & items(i) & " "
        Next
    End Sub
```

Método de Ordenación: burbuja

```
Sub Bubblesort(Array() As Integer, Length As Integer)
        Dim i As Integer
        Dim j As Integer
        Dim Temp As Integer

        For i = 0 To Length - 1
            For j = Length - 1 To i Step -1
                If Array(j) > Array(j + 1) Then  ' Orden asc e int
                    Temp = Array(j)
                    Array(j) = Array(j + 1)
                    Array(j + 1) = Temp
                End If
            Next
        Next
    End Sub
```

Método de Ordenación: inserción

- Método usado para ordenar una mano de naipes.
- Los elementos están divididos conceptualmente en una secuencia destino y una secuencia fuente.
- En cada paso, comenzando con i=2 e incrementando i en uno, el elemento i-ésimo de la secuencia fuente se toma y se transfiere a la secuencia destino insertándolo en el lugar adecuado.
- Este algoritmo puede mejorarse fácilmente si vemos que la secuencia destino $a_1, a_2, \ldots, a_{i-1}$ está ordenada, por lo que usamos una búsqueda binaria para determinar el punto de inserción.
- La complejidad del algoritmo es $O(n^2)$. Es estable.

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2								
3								
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3								
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3	12	42	44	55	94	18	06	67
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3	12	42	44	55	94	18	06	67
4	12	42	44	55	94	18	06	67
5	12	18	42	44	55	94	06	67
6	06	12	18	42	44	55	94	67
7	06	12	18	42	44	55	67	94

Método de Ordenación: selección

- En éste método, en el i-ésimo paso seleccionamos el elemento con la llave de menor valor, entre a[i],…, a[n] y lo intercambiamos con a[i].
- Como resultado, después de i pasadas, el i-ésimo elemento menor ocupará a[1],…, a[i] en el lugar ordenado.
- La complejidad del algoritmo es $O(n^2)$.

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2								
3								
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3								
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3	06	12	18	42	94	55	44	67
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3	06	12	18	42	94	55	44	67
4	06	12	18	42	94	55	44	67
5	06	12	18	42	44	55	94	67
6	06	12	18	42	44	55	94	67
7	06	12	18	42	44	55	67	94

Método de Ordenación: Quicksort

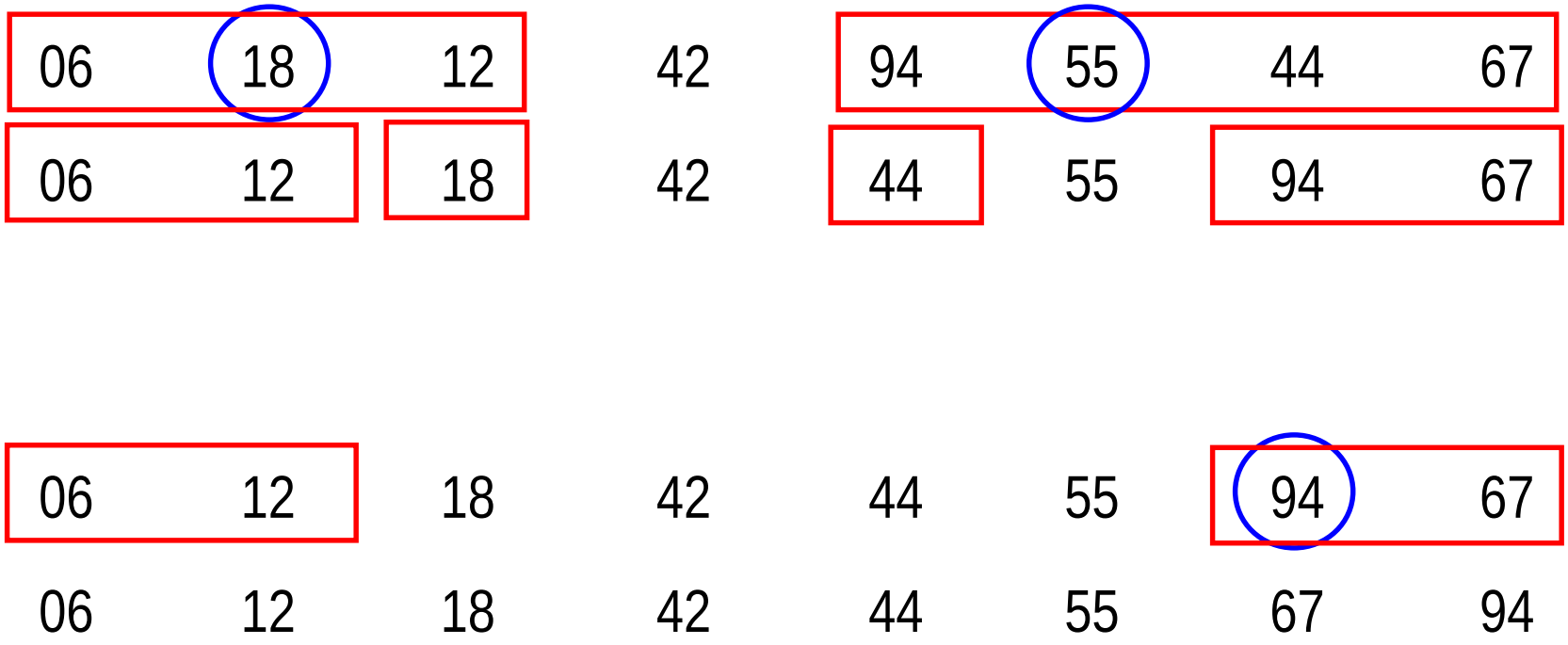
- Se basa en el hecho que los intercambios deben ser realizados preferentemente sobre distancias grandes.
- El algoritmo (técnica de dividir y vencer) simplificado es:
  - Seleccionar un elemento del array (elemento pivote, p.e. el que se encuentra en la mitad).
  - Todos los elementos menores al pivote se colocan en un array y los mayores en otro.
  - Se aplica el mismo procedimiento de forma *recursiva*, sobre los subarrays hasta que solo exista un elemento.
- La complejidad del algoritmo es O(n·logn).

Método de Ordenación: Quicksort

- División del array

44	55	12	42	94	18	06	67
44	55	12	42	94	18	06	67
06	55	12	42	94	18	44	67
06	18	12	42	94	55	44	67
06	18	12	42	94	55	44	67

Método de Ordenación: Quicksort



Método de Ordenación: Quicksort

```
Sub QuickSort(List() As Integer, min As Integer, max As Integer)
        Dim med_value, temp As Long
        Dim i, j As Integer
        i = min
        j = max
        'Elemento en la mitad como pivote
        med_value = List(min + (max - min) / 2)
        'Intercambio y formacion de dos grupos
        Do While i <= j
            Do While List(i) < med_value
                i += 1
            Loop
            Do While List(j) > med_value
                j -= 1
            Loop
```

Método de Ordenación: Quicksort

```
            If i <= j Then
                temp = List(i)
                List(i) = List(j)
                List(j) = temp
                i += 1
                j -= 1
             End If
        Loop
        ' Recursion sobre los grupos
        If min < j Then
            QuickSort(List, min, j)
        End If
        If i < max Then
            QuickSort(List, i, max)
        End If
    End Sub
```