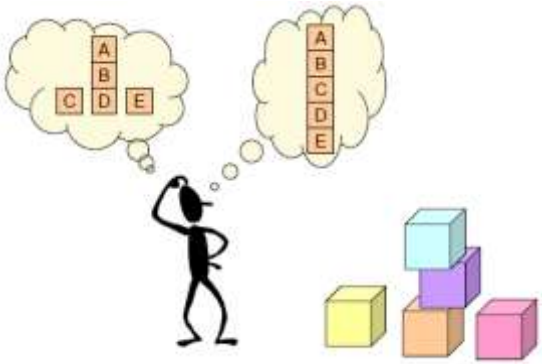
UNER Facultad de Ciencias de la Administración

Algoritmos y Programación

Métodos de Ordenamiento

Métodos de Ordenamiento

- La necesidad de datos ordenados.
- Funciones o estrategias incluidas en los diseños de los lenguajes de programación.
  - Librerías
- Algunas Metodologías de ordenamiento:



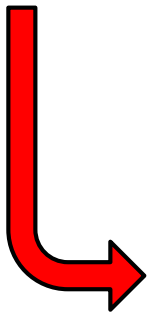
a) Burbuja
b) Shell
c) Inserción
d) Quick sort

Universidad Nacional de Entre Ríos

Métodos de Ordenamiento

¿ Qué métodos vamos a analizar ?

BURBUJA



a) Burbuja Simplificado

b) Burbuja con Acotamiento

c) Burbuja con Retroceso

Universidad Nacional de Entre Ríos

Métodos de Ordenamiento - Burbuja

El procedimiento opera sobre el mismo array donde se encuentran los datos que se requiere ordenar. No necesita arrays auxiliares y se basa en la comparación de elementos contiguos.

Procedimiento:

1. *Se compara el elemento ubicado en la 1ra. posición con el de la 2da. posición. Si no se encuentran en el orden que se requiere (ascendente o descendente, según el caso) se los intercambia de lugar.*

2. *Se compara el elemento ubicado en la 2da. posición con el de la 3ra. posición. Si no se encuentran en el orden que se requiere (ascendente o descendente, según el caso) se los intercambia de lugar.*

3. *El procedimiento se repite hasta comparar el elemento ubicado en la penúltima posición con el de la última, intercambiándose si es necesario.*

Métodos de Ordenamiento – Burbuja

Rutina Elemental:

Sea un arreglo unidimensional "V" de "n" elementos, se requiere ordenarlos de menor a mayor:

Es necesario codificar una rutina para recorrer el arreglo comparando sus elementos contiguos e intercambiarlos si corresponde

¿ RESUELVE EL PROBLEMA ESTA RUTINA ?

```
Para i = 1, n-1 , 1
    Si V(i) > V(i + 1)
        Aux  := V(i)
        V(i) :=  V(i + 1)
        V(i + 1) := Aux
    Finsi
FinPara
```

¿Porqué se recorre el array hasta el elemento "n-1" ?

Porque si fuera hasta "n", en esta comparación, cuando "i" sea igual a "n" la comparación sería inválida (n+1 ?).

Universidad Nacional de Entre Ríos

Ordenamiento Burbuja

Ordenamiento Burbuja

SEGUNDA RECORRIDA

ARRAY						PASOS
1	2	3	4	5	6	
2	5	0	6	4	9	1. Compara 1er. con 2do. Elemento
No Cambia						
2	5	0	6	4	9	2. Compara 2do. con 3er. Elemento
	Cambia					
2	0	5	6	4	9	3. Compara 3er. con 4to. Elemento
		No Cambia				
2	0	5	6	4	9	4. Compara 4to. con 5to. Elemento
			Cambia			
2	0	5	4	6	9	5. Compara 5to. con 6to. Elemento
				No Cambia		
2	0	5	4	6	9	¿ Quedó el array ordenado ?

Ordenamiento Burbuja

TERCERA RECORRIDA

ARRAY						PASOS
1	2	3	4	5	6	
2	0	5	4	6	9	1. Compara 1er. con 2do. Elemento
Cambia						
0	2	5	4	6	9	2. Compara 2do. con 3er. Elemento
	No Cambia					
0	2	5	4	6	9	3. Compara 3er. con 4to. Elemento
		Cambia				
0	2	4	5	6	9	4. Compara 4to. con 5to. Elemento
			No Cambia			
0	2	4	5	6	9	5. Compara 5to. con 6to. Elemento
				No Cambia		
0	2	4	5	6	9	¿ Quedó el array ordenado ?

Ordenamiento Burbuja

Luego: ¡ Necesidad de varias recorridas !

```
Begin
Mientras  ?
        Para i = 1, n-1, 1
            Si V(i) > V(i + 1)
                Aux  := V(i)
                V(i) :=  V(i + 1)
                V(i + 1) := Aux
            Finsi
        FinPara
FinMientras
End
```

Como es necesario contemplar varias recorridas (ya que no hay certeza de que el arreglo quede ordenado), la rutina anterior debe incluirse dentro de otro esquema repetitivo.

Ordenamiento Burbuja

Ante Necesidad de varias recorridas :

↓

¿ CUÁNTAS ?

↓

Surge el problema de saber cuando salir de dicho esquema repetitivo Mientras-Finmientras (que es cuando queda ordenado el arreglo).

La estrategia consiste en determinar cuando en una recorrida completa no se hace ningún intercambio y para saber esto se recurre a la utilización de un campo de tipo booleano.

Universidad Nacional de Entre Ríos

Métodos Ordenamiento - Burbuja

Una Solución posible:

```
Begin
B := False
Mientras   B = False
        B := True
    Para i = 1, n-1, 1
        Si V(i) > V(i + 1)
                Aux  := V(i)
                V(i) :=  V(i + 1)
                V(i + 1) := Aux
                B := False
        Finsi
    FinPara
FinMientras
End
```

Para indicar que faltan recorridas

Universidad Nacional de Entre Ríos

Ordenamiento Burbuja

Ventajas :

- Implementación sencilla.
- No requiere almacenamiento adicional temporal.

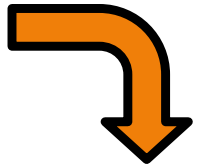
Desventajas :

- Lento para arrays con muchos elementos.
- Requiere una recorrida adicional cuando los elementos ya están ordenados.

Ordenamiento Burbuja

Verdaderamente... ¿ No es posible saber cuantas repeticiones aseguran que el array quede ordenado ?

Análisis del peor caso:

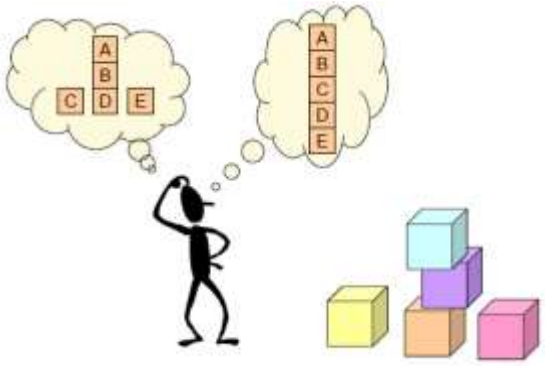


Si el array tiene sus elementos ordenados de mayor a menor y se desea ordenarlos de menor a mayor, la cantidad de repeticiones necesarias es igual a la cantidad de elementos disminuida en una unidad.

Métodos Ordenamiento - Burbuja

Burbuja Simplificado

```
Program Burbuja.
Var
   V : array [1..n] of integer;
    i, Aux : integer;
    B : boolean;
```




```
Begin
B := False
Mientras   B = False
        B := True
     Para i = 1, n-1, 1
        Si V(i) > V(i + 1)
            Aux  := V(i)
            V(i) :=  V(i + 1)
            V(i + 1) := Aux
            B := False
        Finsi
     FinPara
FinMientras
End
```

Métodos Ordenamiento - Burbuja

PASCAL

```
PROGRAM Ordena;
VAR
     lista : array [1..500] of string;
     i,j:= integer;
     aux : string

BEGIN
  for i := 1 to 499 do
    Begin
    for j := 1 to 499 do
         Begin
         if lista[j] > lista[j+1]
             Begin
                 aux := lista[j];
                 lista[j] := lista[j+1];
                 lista[j+1] := aux;
             end;
         end;
     end;
END.
```

JAVA

```
public static void burbuja(int[] A) {
      int i, j, aux;
      for (i = 0; i < A.length - 1; i++) {
         for (j = 0; j < A.length - i - 1; j++) {
            if (A[j + 1] < A[j]) {
               aux = A[j + 1];
               A[j + 1] = A[j];
               A[j] = aux;
            }
         }
      }
}
```

Universidad Nacional de Entre Ríos

Métodos de Ordenamiento - Burbuja

Rutina para arreglos tipo registro

En estos casos se compara por el campo por el cual el arreglo debe quedar ordenado y en el intercambio de posiciones, cuando es necesario, puede asignarse el elemento compuesto haciendo referencia al nombre común.

Ejemplo:

Dado un arreglo de 80 elementos, cada uno de los cuales contiene el documento y el nombre de alumnos inscriptos en una asignatura, ordenarlos por número de documento.

Estructura del Arreglo:

```
Type
    Registro  =  Record
        Nudo : integer;
        Nomb : string
    end;
Var
    Alumno  :  array [1..80] of Registro;
```

Universidad Nacional de Entre Ríos

Métodos de Ordenamiento - Burbuja

Ordenamiento de arreglos tipo registros:

```
Begin
B := False
Mientras B = False
        B := True
        Para i = 1, n-1, 1
            Si Alumno(i).Nudo > Alumno(i+1).Nudo
                Aux := Alumno(i)
                Alumno(i) := Alumno(i + 1)
                Alumno(i + 1) := Aux
                B := False
            Finsi
        FinPara
FinMientras
End
```

Se compara por el campo Nudo de ALUMNO

Se transfieren todos los campos del elemento.

Métodos de Ordenamiento - Burbuja

¿ De qué tipo debe ser la variable auxiliar ?

```
Type
        Alumnado  =  Record
            Nudo : integer;
            Nomb : string
        end;

Var
        Alumno  :  array [1..80] of Alumnado;

        AUX : Alumnado;
```

Del tipo registro

Universidad Nacional de Entre Ríos

Método de Ordenamiento - Burbuja

Primer Caso de mejora: Burbuja con Acotamiento

PRIMERA RECORRIDA

1	2	3	4	5	6
5	2	6	0	9	4
2	5	6	0	9	4
2	5	6	0	9	4
2	5	0	6	9	4
2	5	0	6	9	4
2	5	0	6	4	9

Posiciones

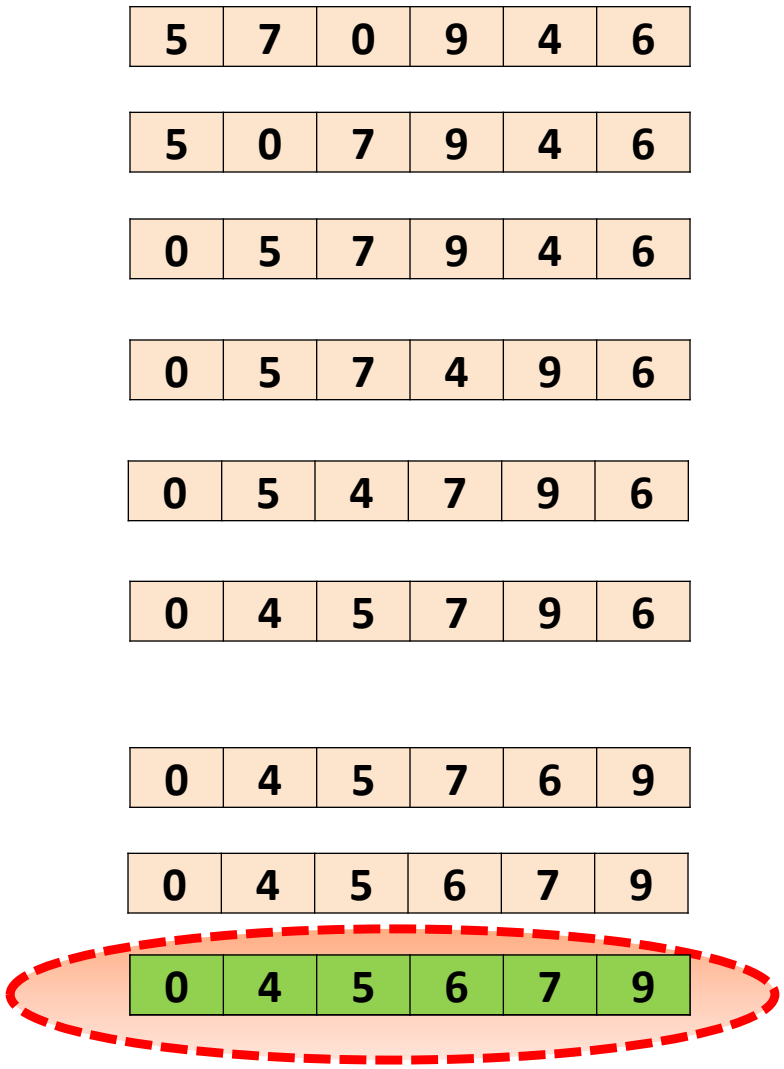
SEGUNDA RECORRIDA

1	2	3	4	5	6
2	5	0	6	4	9
2	5	0	6	4	9
2	0	5	6	4	9
2	0	5	6	4	9
2	0	5	4	6	9
2	0	5	4	6	9

Dado que en la recorrida el elemento de mayor valor (en caso de estar ordenando de menor a mayor) queda siempre en la última posición , en la segunda recorrida el mayor (con excepción del anterior) queda en la anteúltima posición y así sucesivamente, es posible mejorar la rutina acortando las sucesivas recorridas en un elemento.

Métodos de Ordenamiento

Segundo caso de mejora: Una sola recorrida con retroceso.



Se compara: 5 con 7: No Cambia
Se compara: 7 con 0 : Cambia

Retroceso:
Se compara: 5 con 0 Cambia.

Retroceso: Fin Retroceso. **Retoma**
Se compara: 7 con 9 : No cambia.
Se compara: 9 con 4 : Cambia.

Retroceso:
Se compara: 7 con 4 : Cambia.

Retroceso:
Se compara: 5 con 4 : Cambia.

Retroceso:
Se compara: 0 con 4 : No Cambia.

Retoma:
Se compara: 9 con 6 : Cambia.

Retroceso:
Se Compara: 7 con 6 : Cambia

Se compara : 5 con 6 : No cambia

Retoma:
Fin de Recorrida.

Métodos de Ordenamiento - Burbuja

Burbuja con retroceso.

Consiste en hacer una sola recorrida en el array, pero cuando se efectúa un intercambio se debe retroceder y comparar los elementos anteriores para asegurar el orden de los elementos.

El proceso de retroceso se realiza mientras se efectúen intercambios y se interrumpe por dos motivos:

a) en una comparación no se efectúa intercambio y
b) se llegó al inicio del array.

Al detenerse el retroceso, se retoma la comparación en el punto en que inició la recorrida hacia atrás.

Métodos de Ordenamiento - Shell

Donald Shell
Marzo 1924 – Noviembre 2015
Universidad de Michigan

Características:

- Recorrida del arreglo comparando elementos y cambiando de lugar si no están en el orden requerido.
- Se realizan varias recorridas.
- No siempre se compara elementos contiguos.
- El procedimiento finaliza cuando se realizo una recorrida completa comparando elementos contiguos.

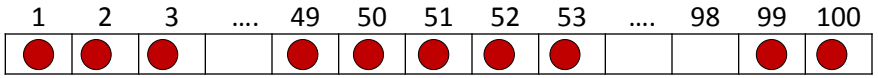
Métodos de Ordenamiento - Shell

Procedimiento:

1. Se calcula la distancia de los elementos a comparar. Para la primera recorrida dicho valor se obtiene dividiendo por dos la cantidad de elementos del vector.

2. Se inicia una recorrida del vector comparando los dos elementos que están separados por una distancia igual al valor calculado en el punto 1. Si no están en el orden requerido, se los cambia de lugar.

Ejemplo: Array de 100 elementos
la distancia es: 50

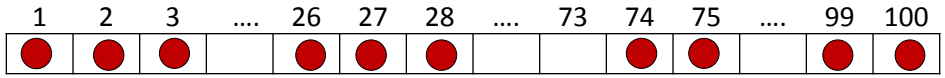


Ultima comparación: Cuando i = 50 [Es decir: Longitud del arreglo – Distancia (100-50)

IMPORTANTE

3. Se recalcula el valor correspondiente a la distancia de los elementos a comparar dividiendo al valor utilizado en la recorrida anterior por 2. Se obtiene así un nuevo valor que resulta igual a la mitad del utilizado en la recorrida anterior. (se toma la parte entera cuando el cociente no sea exacto).
   Se inicia una nueva recorrida repitiendo desde el paso 2.

Distancia anterior: 50
Nueva distancia es (50/2): 25



Ultima comparación: Cuando i = 75
[Es decir: Longitud del arreglo – Distancia (100-25)]

IMPORTANTE

UNER

Métodos de Ordenamiento - Shell

Pero falta algo…..

Cuando en cualquier recorrida, es necesario realizar un intercambio, el método exige que se 'retroceda', restituyendo las posiciones de la comparación inmediata anterior para volver a comparar los elementos e intercambiarlos si corresponde.

Este retroceso debe continuar hasta que:

a) la comparación efectuada indique que los elementos están en el orden correspondiente.

b) cuando se llegue al inicio del vector

Una vez finalizado el proceso de retroceso, se debe continuar la recorrida desde el punto en que se inició dicho retroceso.

El procedimiento termina cuando finaliza una recorrida en la que se compararon e intercambiaron elementos ubicados en posiciones contiguas.

Métodos de Ordenamiento - Shell

Recorrida: 01

Distancia : (8/2) = 4

Valor de "i"	Comparación	Acción
01	01 - 05	Cambia
Retroceso: (01-04) = **-3** No se puede		
		Retoma
02	02 - 06	Cambia
Retroceso: (02-04) = **-02** No se puede		
		Retoma
03	03 - 07	No Cambia
04	04 - 08	No Cambia

77	15	27	0	**63**	8	42	51
1	2	3	4	5	6	7	8

Valor

Posición

63	**15**	27	0	77	**8**	42	51
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

63	8	**27**	**0**	77	15	**42**	**51**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

FIN RECORRIDA 1

Métodos de Ordenamiento - Shell

Recorrida: 02

Distancia : (4/2) = 2

Valor de "i"	Comparación	Accion
01	01 - 03	Cambia
Retroceso: (01-02) = **-01** (No se puede)		
02	02 - 04	Cambia
Retroceso: (02-02) = **0** (No se puede)		
03	03 - 05	No Cambia
04	04 - 06	No Cambia
05	05 - 07	Cambia
Retroceso: (05-02) = **03**		
03	03 - 05	Cambia
Retroceso: (03-02) = **01**		
01	01 - 03	No Cambia
		Retoma
06	06 - 08	No Cambia

FIN RECORRIDA 2

63	8	27	0	77	15	42	51
1	2	3	4	5	6	7	8

27	8	63	0	77	15	42	51
1	2	3	4	5	6	7	8

27	0	63	8	77	15	42	51
1	2	3	4	5	6	7	8

27	0	63	8	42	15	77	51
1	2	3	4	5	6	7	8

27	0	42	8	63	15	77	51
1	2	3	4	5	6	7	8

Métodos de Ordenamiento - Shell

Recordando:

- Se realizan varias recorridas.

- No siempre se compara elementos contiguos.
  - -Primer Recorrida: Distancia = Cantidad de elementos / 2
  - - Siguientes recorridas = Distancia recorrida anterior / 2

- Cuando se intercambian elementos se debe retroceder.

- Cada recorrida finaliza cuando la variable que se utiliza (por ejemplo “i”) es igual Cantidad Elementos menos distancia

-El procedimiento finaliza cuando se realizo una recorrida completa comparando elementos contiguos.
  - - Finalizó la recorrida con distancia igual a 1
  - - Al recalcular la distancia se divide (1 / 2 = 0,5), como se toma la parte entera (cero) Termina el procedimiento quedando el array ordenado.

Ordenamiento - Shell

Construcción de la rutina para ordenar

```
INICIO
distancia := n / 2
Mientras distancia > 0
    Para i = 1, (n – distancia), 1
        z := i
        Mientras z > 0 and then V(z) > V(z + distancia)
            Aux := V(z)
            V(z) := V(z + distancia)
            V(z + distancia) := Aux
            z := z - distancia
        Finmientras
    FinPara
    distancia := distancia / 2
FinMientras
FIN
```

Como hay varias recorridas, la rutina Anterior debe repetirse varias veces.

En cada recorrida se debe finalizar cuando "i" tenga valor igual a longitud del arreglo menos la distancia.

Y en la comparación debiera ser entre el elementos de la posición "z" con el elemento de la posición "z+distancia".

Como la primer recorrida la distancia debe ser igual a la mitad de la longitud.

Cuando termina una recorrida hay que Recalcular el valor de la distancia.

¿ Cuando terminan todas las recorridas ?. Cuando al recalcular el valor de la distancia el resultado (entero) es cero.

UNER

Métodos de Ordenamiento - Shell

Luego: Solución Algorítmica:

Ordenar los datos de un arreglo "V" de "n" elementos:

```
Inicio
Distancia := n / 2
Mientras Distancia > 0
    Para i = 1, (n – Distancia), 1
        z := i
        Mientras z > 0 and then  V(z) > V(z + Distancia)
                AUX := V(z)
                V(z)  := V(z + Distancia)
                V(z + Distancia) := AUX
                z := z - Distancia
        Finmientras
    Finpara
    Distancia := Distancia / 2
Finmientras
FIN
```

UNER

Métodos de Ordenamiento - Shell

Caso:

Dado un arreglo que contiene los datos de los alumnos de la FCAd se desea ordenarlos en forma creciente por número de documento. El arreglo contiene el número de documento y el nombre de cada alumno.

Se parte de considerar que la cantidad de elementos del arreglo se encuentra en una variable LON.

```
Program Ordena
Type
        Registro  =  Record
             NDoc : Integer;
             Nomb :  string
        end;

  Var
        Alumno  :  array [1..100] or Registro;
        i, j, Dis : Integer;
        Aux : Registro;
```

UNER

Métodos de Ordenamiento - Shell

Solución Algorítmica:

```
INICIO
Dis := 100 / 2
Mientras Dis > 0
   Para i = 1, (100 – Dis), 1
      z := i
      Mientras z > 0 and then Alumno(z).NDoc > Alumno(z+Dis).NDoc
            AUX := Alumno(z)
            Alumno(z)  := Alumno(z+Dis)
            Alumno (z+Dis) := AUX
            z := z – Dis
      Finmientras
   Finpara
   Dis := Dis / 2
Finmientras
FIN
```

Métodos de Ordenamiento - Shell

Ejemplo Pascal

```
program ShellSort;
type
    rango = 1..500;
    Tlista = array [rango] of integer;
var
    lista : Tlista;
```

```
procedure intercambiar(var x, y :   integer);
var
    aux : integer;
begin
    aux := x;
    x := y;
    y := aux;
end;
```

```
begin {programa principal}
clrscr;
writeln();
write(' Ordenacion por ShellSort: ');
ShellSort(lista, 500);
write(' FIN Ordenacion por ShellSort: ');
end.
```

```
procedure ShellSort (var list : Tlista; num : integer);
var
    Distancia, i, j, k : integer;
begin
    Distancia := num div 2;
    while (Distancia > 0) do
    begin
        for i := (Distancia + 1) to num do
        begin
            j := i - Distancia;
            while (j > 0) do
            begin
                k := j + Distancia;
                if (list[j] <= list[k]) then
                    j := 0
                else
                    intercambiar(list[j], list[k]);
                    j := j - Distancia;
            end; {while j > 0}
        end; {for}
        Distancia := Distancia div 2;
        end; {while (Distancia > 0)}
end;
```

Métodos de Ordenamiento - Shell

Solución en JAVA

```
int[] ShellSort(int[] array){
   int gap = array.length / 2;
   while (gap > 0) {
      for (int i = 0; i < array.length - gap; i++) {
         int j = i + gap;
         int tmp = array[j];
         while (j >= gap && tmp > array[j - gap]) {
            array[j] = array[j - gap];
            j -= gap;
         }
         array[j] = tmp;
      }
      if (gap == 2) {
         gap = 1;
      }else{
         gap /= 2.2;
      }
   }
   return array;
}
```

UNER

UNER Facultad de Ciencias de la Administración

Algoritmos y Programación

Métodos de Ordenamiento

FIN DE LA CLASE