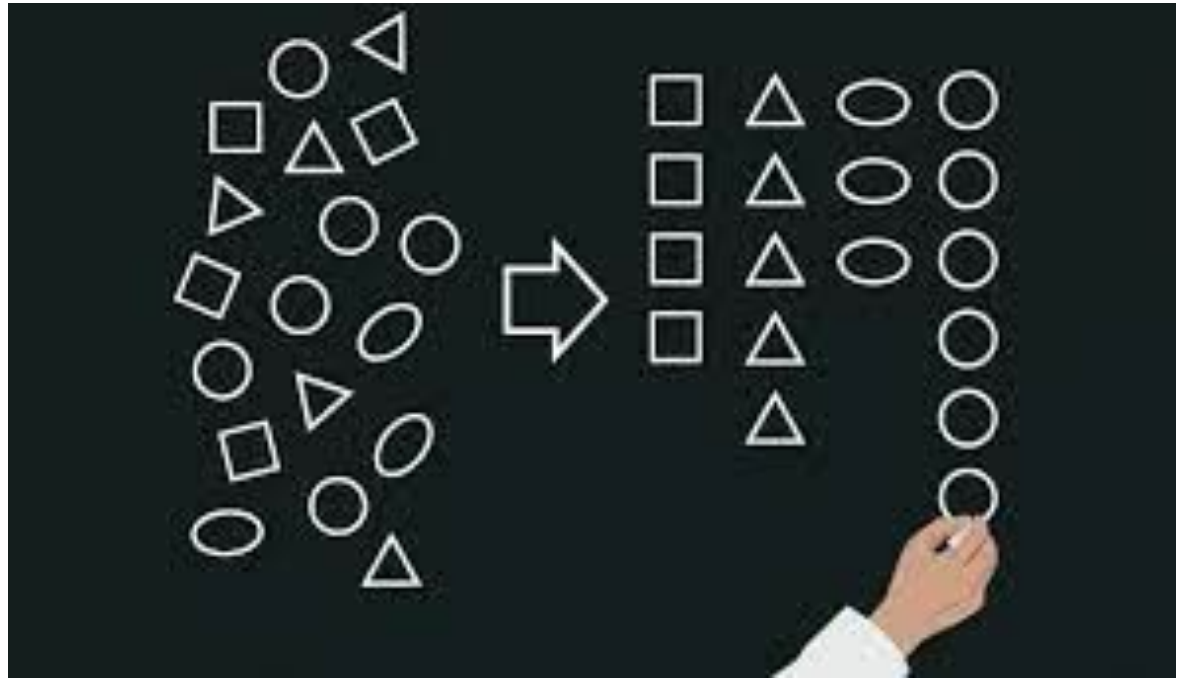


Lenguaje de Programación

1

Ordenación y Búsqueda

- Bibliografía: “Estructura de Datos. Algoritmos, abstracción y objetos”. Aguilar y Martínez. McGraw Hill 1998. Capítulo 15.



ORDENAMIENTO DE DATOS

Ordenación

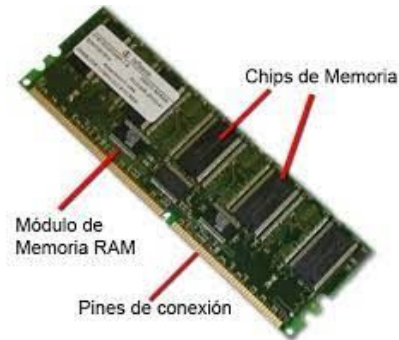
- La ordenación de los datos consiste en disponer un conjunto de datos (o una estructura) en algún orden con respecto a alguno de sus campos.



- **Orden**: Relación de una cosa con otra.
- **Clave**: Campo por el cual se ordena.

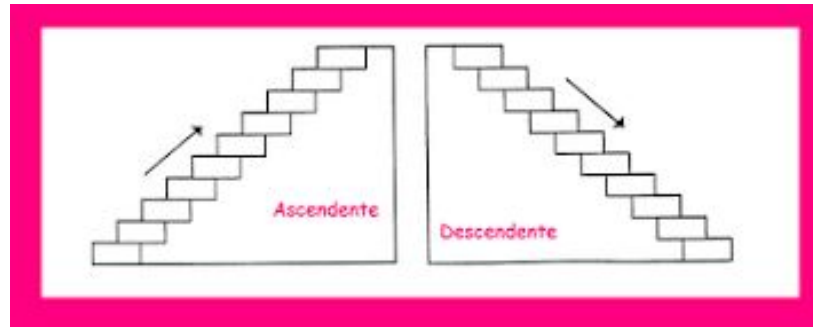
Ordenación

- Según donde estén almacenados los datos a Ordenar, podemos decir que la Ordenación es:
 - Interna: Arrays, listas o árbol. Típicamente en RAM.
 - Externa: En Archivos en discos o cintas.



Orden

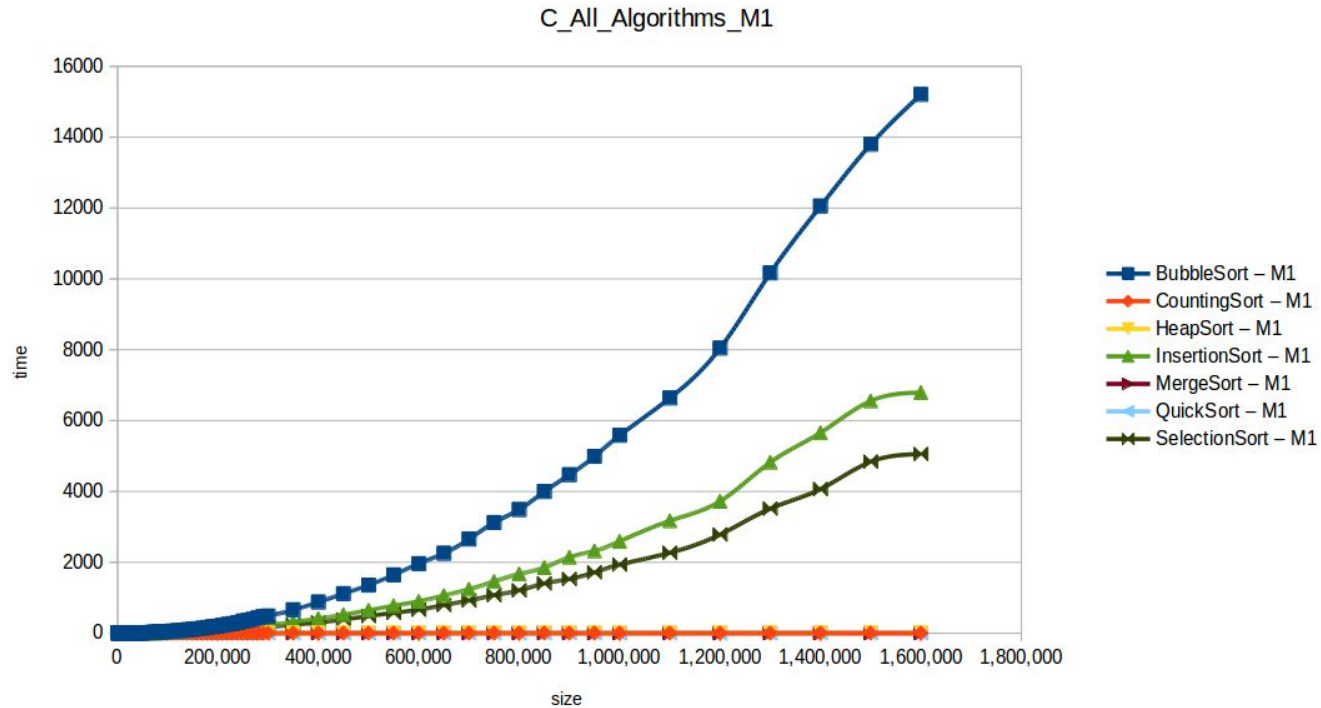
- Una lista de datos está ordenada por la clave k si la lista está en orden con respecto a la clave anterior.
- Este Orden puede ser:
 - **Ascendente**: $(i < j)$ entonces $(k[i] \leq k[j])$
 - **Descendente**: $(i < j)$ entonces $(k[i] \geq k[j])$



Orden

- Hay Numerosos Métodos de Ordenamiento que difieren en **eficiencia**.
 - Análisis de algoritmo orientado a las comparaciones realizadas por cada uno.
 - Las comparaciones serán función de “n”. Siendo “n” el tamaño del vector a Ordenar.

Orden Métodos de Ordenamiento



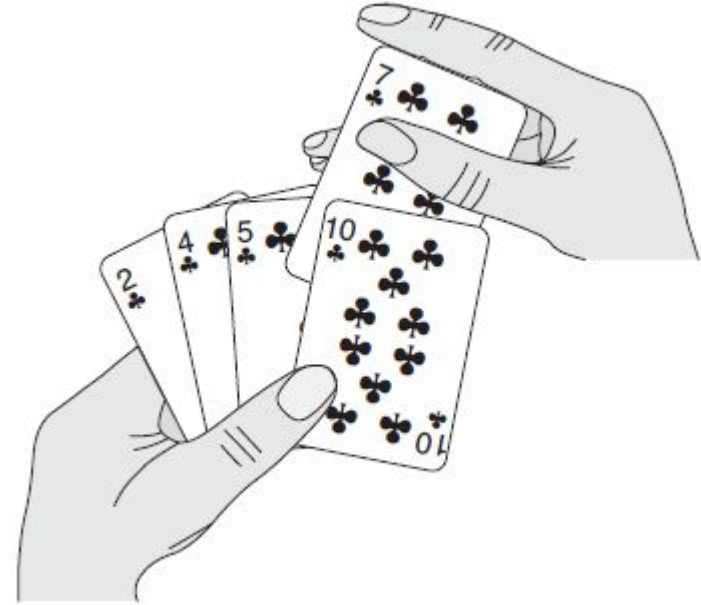
<https://github.com/xergioalex/analysisOfSortAlgorithms>

Clasificación de Métodos de Ordenación

- Todos los métodos se verán con **Orden Ascendente**.
- Analizaremos los siguientes métodos:
 - **Básicos**: Son eficaces en Listas pequeñas
 - *Burbuja e Intercambio*: simple pero Ineficiente.
 - *Selección e inserción*: Recomendados.
 - **Avanzados**: Son eficaces en Listas grandes.
 - Shell: muy extendido.

Ordenación Por **Intercambio**

- El más sencillo de todos.
- Se basa en:
 - La lectura sucesiva de la lista a ordenar,
 - Comparando el elemento inferior de la lista con todos los restantes
 - Efectúa el intercambio de posiciones cuando el orden resultante no sea correcto.
- Siendo n la cantidad de elementos, Realizará al menos $n-1$ pasadas.




Ejemplo


Ordenación por Intercambio

Pasada 0: Se compara $a[0]$ con todos, así primero se cambia $a[0]$ con $a[1]$ pues $a[0] > a[1]$ y debe ser Ascendente, es decir $a[0] < a[1]$...y por último $a[0]$ con $a[3]$

$a[0]$	$a[1]$	$a[2]$	$a[3]$
8	4	6	2



$a[0]$	$a[1]$	$a[2]$	$a[3]$
4	8	6	2



$a[0]$	$a[1]$	$a[2]$	$a[3]$
2	8	6	4

Pasada 1: El elemento más pequeño está en $a[0]$ y se analiza la sublista restante. Al cabo de la pasada, el segundo más chico está en $a[1]$

Pasada 2:

$a[0]$	$a[1]$	$a[2]$	$a[3]$
2	6	8	4

$a[0]$	$a[1]$	$a[2]$	$a[3]$
2	4	8	6

$a[0]$	$a[1]$	$a[2]$	$a[3]$
2	4	6	8

Pasada i: Al cabo de la pasada i , el elemento de orden i , está en $a[i]$

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

CUANTAS VECES SE EJECUTA ESTE BLOQUE?

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

CUANTAS VECES SE EJECUTA ESTE BLOQUE? ***n-1 veces***

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

¿CUÁNTAS VECES SE EJECUTA ESTE BLOQUE?

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

¿CUÁNTAS VECES SE EJECUTA ESTE BLOQUE? para cada i se ejecuta **$n-i-1$**

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

i=0: j=1,.....,n-1 = n-1
i=1: j=2,.....,n-1 = n-2
i=2: j=3,.....,n-1 = n-3
.
.
.
i=n-2: j=n,.....,n-1 = 0

$0 + 1 + 2 + \dots + n - 1 =$
 $n*(n-1)/2$

¿CUÁNTAS VECES SE EJECUTA ESTE BLOQUE? para cada i se ejecuta **$n-i-1$**

Codificación C:

Ordenación por Intercambio

```
void ordIntercambio (int a[], int n) {  
    int i, j, aux; /* se realizan n-1 pasadas, a[0] ... a[n-2] */  
    for (i = 0 ; i <= n-2 ; i++) { /* coloca mínimo de a[i+1]...a[n-1] en a[i] */  
        for (j = i+1 ; j <= n-1 ; j++) {  
            if (a[i] > a[j]) {  
                aux = a[i];  
                a[i] = a[j];  
                a[j] = aux ;  
            }  
        }  
    }  
}
```

Complejidad $n*(n-1)/2$
Del Orden $F(n)=n^2$.

Ordenación Por Selección

- Realiza sucesivas pasadas que
 - Busca el elemento más pequeño de la lista a y lo escribe al frente de la lista $a[0]$.
 - Considera las posiciones restantes, $a[1]...a[n]$
 - Finaliza cuando ya no hay Posiciones Restantes.
- En la pasada i
 - Está Ordenado: desde $a[0]$ hasta $a[i-1]$.
 - Está Desordenado: Desde $a[i]$ hasta $a[n]$.
- El proceso continua $n-1$ vueltas.

Ejemplo

Ordenación por Selección

Lista Original

a[0]	a[1]	a[2]	a[3]
39	21	90	80

Pasada 0: Lista entre 0 y 3. Selecciona el 21 y lo pasa al a[0]

a[0]	a[1]	a[2]	a[3]
21	39	90	80

Pasada 1: Lista entre 1 y 3. Selecciona el 39 y lo pasa al a[1]

a[0]	a[1]	a[2]	a[3]
21	39	90	80

Pasada 2: Lista entre 2 y 3. Selecciona el 80 y lo mueve al a[2].

a[0]	a[1]	a[2]	a[3]
21	39	80	90

Codificación C:

Ordenamiento por Selección

```
void ordSeleccion (double a[], int n){
    int indiceMenor, i, j;
    double aux;
    for (i = 0; i < n-1; i++){          /* ordenar a[0]..a[n-2] y a[n-1] en cada pasada */
        indiceMenor = i;                /* comienzo de la exploración en índice i */
        for (j = i+1; j < n; j++){      /* j explora la sublista a[i+1]..a[n-1] */
            if (a[j] < a[indiceMenor]){
                indiceMenor = j;
            }
        }
        if (i != indiceMenor){          /* sitúa el elemento más pequeño en a[i] */
            aux = a[i];
            a[i] = a[indiceMenor];
            a[indiceMenor] = aux ;
        }
    }
}
```

Codificación C:

Ordenamiento por Selección

```
void ordSeleccion (double a[], int n){
    int indiceMenor, i, j;
    double aux;
    for (i = 0; i < n-1; i++){          /* ordenar a[0]..a[n-2] y a[n-1] en cada pasada */
        indiceMenor = i;                /* comienzo de la exploración en índice i */
        for (j = i+1; j < n; j++){      /* j explora la sublista a[i+1]..a[n-1] */
            if (a[j] < a[indiceMenor]){
                indiceMenor = j;
            }
        }
        if (i != indiceMenor){          /* sitúa el elemento más pequeño en a[i] */
            aux = a[i];
            a[i] = a[indiceMenor];
            a[indiceMenor] = aux ;
        }
    }
}
```

Complejidad $n*(n-1)/2$
Del Orden $F(n)=n^2$.

Ordenación Por Inserción

- Similar al proceso de ordenar tarjetas en un tarjetero por orden alfabético:
 - Consiste en insertar un elemento en su posición correcta, dentro de una lista que ya está Ordenada.
- Algoritmo:
 - El 1er elemento $a[0]$ se lo considera ordenado
 - Se inserta $a[1]$ en la posición correcta, delante o detrás del $a[0]$, según sea mayor o menor
 - Por cada bucle i (desde $i=1$ hasta $n-1$) se explora la sublista $a[0]..a[i-1]$ buscando la posición correcta de inserción del elemento $a[i]$
 - Al dejar vacío la posición $a[i]$ se impone un desplazamiento de todo el vector, desde el lugar de inserción.

Ordenación Por Inserción

54	26	93	17	77	31	44	55	20	Se asume que 54 es una lista ordenada de 1 ítem
26	54	93	17	77	31	44	55	20	Se inserta 26
26	54	93	17	77	31	44	55	20	Se inserta 93
17	26	54	93	77	31	44	55	20	Se inserta 17
17	26	54	77	93	31	44	55	20	Se inserta 77
17	26	31	54	77	93	44	55	20	Se inserta 31
17	26	31	44	54	77	93	55	20	Se inserta 44
17	26	31	44	54	55	77	93	20	Se inserta 55
17	20	26	31	44	54	55	77	93	Se inserta 20

Codificación en C: Ordenación por Inserción

```
void ordInsercion (int [] a, int n){  
    int i, j, aux;  
    for (i = 1; i < n; i++){  
        j = i;  
        aux = a[i]; /* se localiza el punto de inserción explorando hacia abajo */  
        while (j > 0 && aux < a[j-1]){ /* desplazar elementos hacia arriba para hacer espacio */  
            a[j] = a[j-1];  
            j--;  
        }  
        a[j] = aux;  
    }  
}
```

Complejidad $n*(n-1)/2$
Del Orden $F(n)=n^2$.

Ordenación Por Burbuja

- Los elementos burbujan:
 - Los más grandes, caen al fondo del array (posición n)
 - Los más chicos suben a la cima (posición 0).
- Estudia parejas de elementos Adyacentes
 - $a[0]$ y $a[1]$, $a[1]$ y $a[2]$... $a[i]$ y $a[i+1]$... $a[n-2]$ y $a[n-1]$.
 - Si $a[i+1] < a[i]$ Entonces Los INTERCAMBIA
- Algoritmo:
 - Pasada 0: considera desde ($a[0]$, $a[1]$) hasta ($a[n-2]$, $a[n-1]$).
 - En $a[n-1]$ está el elemento más grande.
 - Pasada 1: considera desde ($a[0]$, $a[1]$) hasta ($a[n-3]$, $a[n-2]$).
 - En $a[n-2]$ está el segundo elemento más grande.
 - Pasada i : considera desde ($a[0]$, $a[1]$) hasta ($a[n-i-2]$, $a[n-i-1]$).
 - En $a[n-i-1]$ está el elemento de orden i .
 - El proceso termina con la pasada $n-1$
 - El elemento más pequeño está en $a[0]$.

Ordenación Por Burbuja

54	26	93	17	77	31	44	55	20
26	54	93	17	77	31	44	55	20
26	54	93	17	77	31	44	55	20
26	54	17	93	77	31	44	55	20
26	54	17	77	93	31	44	55	20
26	54	17	77	31	93	44	55	20
26	54	17	77	31	44	93	55	20
26	54	17	77	31	44	55	93	20
26	54	17	77	31	44	55	20	93

primera pasada

Codificación C

Ordenación por Burbuja

```
void ordBurbuja (long a[], int n){  
    int interruptor = 1;  
    int pasada, j;  
    for (pasada = 0; pasada < n-1 && interruptor; pasada++){  
        long aux; /* bucle externo controla la cantidad de pasadas */,  
        interruptor = 0;  
        for (j = 0; j < n-pasada-1; j++){  
            if (a[j] > a[j+1]){ /* elementos desordenados, es necesario intercambio */  
                interruptor = 1;  
                aux = a[j];  
                a[j] = a[j+1];  
                a[j+1] = aux;  
            }  
        }  
    }  
}
```

- *Mejor Caso:* en una lista ordenada, hará una sola pasada: $F(n)=n$
- *Peor Caso:* $F(n)=n^2$

Ordenación Shell

- Debe su nombre a su inventor D.L.Shell.
 - Modifica los saltos contiguos del método de burbuja a saltos variables que se achican.
 - Inicia con Intervalo del orden $n/2$ y luego tiende a 1.
- Algoritmo
 - Dividir lista original en $n/2$ grupos de 2 elementos,
 - Intervalo entre los elementos: $n/2$.
 - Clarificar cada grupo por separado, comparando parejas de elementos.
 - Si No Están Ordenados Entonces CAMBIARLOS.
 - Dividir lista original en $n/4$ grupos de 4 elementos,
 - Intervalo entre los elementos: $n/4$.
 - Continuar sucesivamente hasta que el intervalo==1.

Ejemplo: Ordenación Shell

- Lista Original n=6.
- Intervalo Inicial: $n/2=6/2=3$
 - Intervalos Siguientes=IntervaloAnterior/2
- Se compara $a[i]$ con $a[i+Intervalo]$

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
6	1	5	2	3	4	0

Paso	Intervalo	Si No Están Ordenados Entónces Cambiarlos	La Lista Queda
1	3	(6,2)= 2, 1, 5,6, 3, 4, 0 (5,4)= 2, 1, 4,6, 3,5, 0 (6;0)=2, 1, 4,0, 3,5, 6	2, 1, 4,0, 3,5, 6
2	3	(2, 0)=0, 1, 4,2, 3,5, 6	0, 1, 4,2, 3,5, 6
3	3	Ninguno	0, 1, 4,2, 3,5, 6
4	3/2=1	(4, 2)=0, 1, 2,4, 3,5, 6 (4, 3)= 0, 1, 2,3,4,5, 6	0, 1, 2,3, 4,5, 6
5	1	Ninguno	Lista Ordenada

Comunicación en C: Ordenamiento

Shell

```
void ordenacionShell(double a[], int n)
{
    int i, j, k, intervalo = n / 2;
    double temp;
    while (intervalo > 0)
    {
        for (i = intervalo; i ≤ n; i++)
        {
            j = i - intervalo;
            while (j ≥ 0)
            {
                k = j + intervalo; //queda k=i;
                if (a[j] ≤ a[k]) j = -1; /*termina el bucle, par ordenado */
                else{
                    temp = a[j];
                    a[j] = a[k];
                    a[k] = temp;
                    j -= intervalo;
                }
            }
        }
        intervalo = intervalo / 2;
    }
}
```

El 1er while: $\log_2 n$

El for: n

$$F(n) = n * \log_2(n)$$

Búsqueda

- Encontrar una CLAVE específica dentro de un Almacén, donde existe un campo con la clave que se referencia.
 - Si está en la lista, informa su posición.
- Si el Almacén Está Ordenado la búsqueda puede ser más eficiente.

Búsqueda

- Dos Tipos de Búsqueda:
 - **Secuencial:** Busca la clave explorando un elemento después de otro.
 - Es la única forma de encontrarlo cuando la Lista no está Ordenada por la Clave de Búsqueda.
 - Eficiencia del Orden de $F(n)=n$.



Búsqueda

- Dos Tipos de Búsqueda:
 - **Secuencial:** Busca la clave explorando un elemento después de otro.
 - Es la única forma de encontrarlo cuando la Lista no está Ordenada por la Clave de Búsqueda.
 - Eficiencia del Orden de $F(n)=n$.



Búsqueda

□ Dos Tipos de Búsqueda:

- **Binaria:** En listas Ordenadas por la clave de Búsqueda es el mejor método.
 - Se sitúa la lectura al centro de la lista y se lo comprueba contra la clave.
 - Si clave < a[central]: Buscar entre inferior y central-1
 - Si clave > a[central]: Buscar entre central+1 y superior
 - Eficiencia del Orden $F(n)=\log_2 n$



Búsqueda

- Dos Tipos de Búsqueda:
 - **Binaria:** En listas Ordenadas por la clave de Búsqueda es el mejor método.
 - Se sitúa la lectura al centro de la lista y se lo comprueba contra la clave.
 - Si clave < a[central]: Buscar entre inferior y central-1
 - Si clave > a[central]: Buscar entre central+1 y superior
 - Eficiencia del Orden $F(n)=\log_2 n$



Codificación C

Búsqueda Binaria Iterativa

```
/* búsqueda binaria. devuelve el índice del elemento buscado, o bien -1 caso de fallo */
int busquedaBin(int lista[], int n, int clave){
    int central, bajo=0, alto=n-1, valorCentral;

    while (bajo <= alto){
        {   central = (bajo + alto)/2;    /* índice de elemento central */
            valorCentral = lista[central]; /* valor del índice central */
            if (clave == valorCentral) return central; /* devuelve posición */
            else if (clave < valorCentral) alto = central - 1; /*sublista inferior*/
            else bajo = central + 1; /* ir a sublista superior */
        }
    }
    return -1; /* elemento no encontrado */
}
```

Lenguaje de Programación

1

Ordenación y Búsqueda

- Bibliografía: “Estructura de Datos. Algoritmos, abstracción y objetos”. Aguilar y Martínez. McGraw Hill 1998. Capítulo 15.