Búsqueda,

Ordenamiento

y Mezcla

en Arreglos Lineales

Operaciones con arrreglos

S **SEARCH – BUSQUEDA :**

Secuencial , binaria (para arreglos ordenados)

M **MERGE- MEZCLA -INTERCALACIÓN:**

De dos arreglos ordenados

S **SORT – ORDENAMIENTO :**

Algoritmos básicos y avanzados

Búsqueda secuencial (vector no ordenado)

```
//Esta funcion devuelve la ubicacion de clave en la lista
//Se devuelve un -1 si no se encuentra el valor
int busquedaLineal(int lista[], int tamanio, int clave){
    int i;

    for (i = 0; i < tamanio; i++){
        if (lista[i] == clave)
            return i;
    }
    return -1;
}
```

Al ser una función, el retorno corta el for
También se podría diseñar con un while

Búsqueda binaria (vector ordenado)

```
int busquedabinaria(const int vec [], const int tl, const int valor){
    // busqueda binaria en vector ordenado
    int central, bajo, alto;
    bajo=0;
    alto=tl-1;
    //pos=-1;

    while (bajo <= alto){
        central = (bajo + alto) / 2;

        if (vec[central] == valor)
            return central;
        else
            if (vec[central] > valor)
                alto = central-1;
            else
                bajo = central+1;
    }
    return -1;
}
```

Búsqueda binaria recursiva (vector ordenado)

Llamada

```
res = busquedabinariar(vector, 0, TL-1, elem);
if (res>=0 && res<=TL-1)
    cout << endl << "El elemento se encuentra en la posicion " << res;
else
    cout << endl << "El elemento no se encuentra en el arreglo";
```

```
int busquedabinariar(const int vec [], int inicio, int fin, int valor){
    // busqueda binaria recursiva en vector ordenado
    int medio;

    if (inicio > fin)
        return -1;

    medio = (inicio + fin) / 2;
    if (vec[medio] == valor)
        return medio;
    else
        if (vec[medio] > valor)
            return busquedabinariar(vec, inicio, medio-1, valor);
        else
            return busquedabinariar(vec, medio+1, fin, valor);
}
```

Búsqueda binaria recursiva (vector ordenado)

```
int busquedabinariar(const int vec [], int inicio, int fin, int valor){
    // busqueda binaria recursiva en vector ordenado
    int medio;

    if (inicio > fin)
        return -1;

    medio = (inicio + fin) / 2;
    if (vec[medio] == valor)
        return medio;
    else
        if (vec[medio] > valor)
            return busquedabinariar(vec, inicio, medio-1, valor);
        else
            return busquedabinariar(vec, medio+1, fin, valor);
}
```

Operación: Merge de 2 vectores ordenados

```
int main(){
    int vector1[TF], vector2[TF], vector3[TF*2];
    int cant1, cant2, cant3;

    cout << "Tamanio Logico del vector 1? ";
    cin >> cant1;
    cargarvector(vector1, cant1); //ordenado
    mostrarvector(vector1, cant1);

    cout << "Tamanio Logico del vector 2? ";
    cin >> cant2;
    cargarvector(vector2, cant2); //ordenado
    mostrarvector(vector2, cant2);

    intercalar(vector1, cant1, vector2, cant2, vector3, cant3);
    mostrarvector(vector3, cant3);
```

Operación: Merge de 2 vectores ordenados

```
Vector 1
   1   3   5   7
TL = 4, TF=1000

 Vector 2
   2   4   6   6
TL = 4, TF=1000

 Vector intercalado
   1   2   3   4   5   6   6   7
TL = 8, TF=1000

Presione una tecla para continuar . . .


<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>_
```

Operación: Merge de 2 vectores ordenados

```
void intercalar(int vector1[], const int cant1, int vector2[], const int cant2,
                int vector3[], int& cant3){

    int i=0, j=0, k=0;

    while (i<cant1 && j<cant2){
        if (vector1[i]<vector2[j]){
            vector3[k]=vector1[i];
            i++; }
        else
            { vector3[k]=vector2[j];
              j++; }
        k++;
    }
    while (i<cant1){
        vector3[k]=vector1[i];
        i++; k++; }

    while (j<cant2){
        vector3[k]=vector2[j];
        j++; k++; }

    cant3=k;
}
```

Iniciar posiciones

Intercalar mientras hay elementos en los 2 arreglos

Pasar los que restan en un arreglo cuando el otro terminó

Concepto de Ordenación

- **Ordenar:** significa reagrupar o reorganizar un conjunto de datos en algún determinado orden con respecto a uno de los campos del conjunto. El campo por el cual se ordena un conjunto de datos se denomina *clave*.

- Formalmente, se define la **ordenación** de la siguiente manera:

Sea A una lista de N elementos: A1, A2, A3 . ..... An

Ordenar significa *permutar* estos elementos de tal forma que los mismos queden de acuerdo con un orden preestablecido.

Ascendente: A1<= A2<= A3.....<= An

Descendente: A1>= A2>= A3.....>= An

Concepto de Ordenación

- **Ordenación interna:**

Los datos se encuentran almacenados en memoria (ya sean arrays, listas, etc), y son de acceso aleatorio o directo (se puede acceder a un determinado campo sin pasar por los anteriores).

- **Ordenación externa:**

Los datos están en un dispositivo de almacenamiento externo (archivos), y es probable que no tengan acceso directo. Su ordenación es más lenta que la interna.

Arreglos Lineales: Ordenación Interna

Métodos simples

- Selección directa
- Inserción directa
- Burbuja
- Burbuja mejorado con señal

Métodos avanzados

- Merge sort

Método de Selección Directa

- **Descripción:**

Se debe encontrar el elemento más pequeño del arreglo
e intercambiarlo por el elemento de la primera posición,
luego encontrar el segundo elemento más pequeño
e intercambiarlo con el elemento de la segunda posición
y continuar de esta manera hasta que se haya ordenado el arreglo entero.

Los pasos del algoritmo son:

- Seleccionar el menor elemento del arreglo. Intercambiarlo con el primer elemento A[0].
- Considerar el resto de las posiciones del arreglo (A[1], A[2],….)
  Seleccionar el menor elemento e intercambiarlo con el elemento A[1].
- Repetir los pasos anteriores con los elementos restantes del arreglo, seleccionando el elemento más pequeño e intercambiándolo adecuadamente.

Método de Selección Directa

- **Codificación**:

```
void seleccion(int vec[], const int tl){
     int i,j,min;

     for (i=0; i<tl; i++) {
          min=i;
          for (j=i+1; j<tl; j++)
               if (vec[j] < vec[min]) min=j;
          intercambio(vec[i],vec[min]);
     }
}
```

```
void intercambio( int& a, int& b ){
     int aux = a;
     a = b;
     b = aux;
}
```

Ejemplo de Selección directa

Ejemplo: Consideremos un arreglo B con 8 valores enteros:

25	3	12	19	2	1	9	6
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 1: Seleccionar 1 e intercambiarlo por B[0]

1	3	12	19	2	25	9	6
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 2: Seleccionar 2 e intercambiarlo por B[1]

1	2	12	19	3	25	9	6
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 3: Seleccionar 3 e intercambiarlo por B[2]

1	2	3	19	12	25	9	6
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 4: Seleccionar 6 e intercambiarlo por B[3]

1	2	3	6	12	25	9	19
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 5: Seleccionar 9 e intercambiarlo por B[4]

1	2	3	6	9	25	12	19
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 6: Seleccionar 12 e intercambiarlo por B[5]

1	2	3	6	9	12	25	19
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 7: Seleccionar 19 e intercambiarlo por B[6]

1	2	3	6	9	12	19	25
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]

Pasada 8: **Arreglo ordenado**

Método de Inserción Directa

- **Descripción:**

  En este algoritmo consideramos un elemento a la vez y lo movemos hasta su lugar entre aquellos que ya han sido considerados (manteniéndolos ordenados).

  El elemento se inserta moviendo los elementos superiores una posición a la derecha y ocupando la posición vacante.

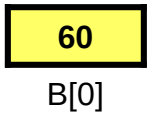
- **Codificación:**

```
void insercion(int vec[], const int tl){
    int i,j;

    for (i=1; i<tl; i++) {
        j=i;
        while ((j>0) && (vec[j] < vec[j-1])) {
            intercambio(vec[j],vec[j-1]);
            j = j-1;
        }
    }
}
```

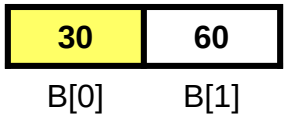
Ordenación: Inserción directa

Ejemplo: Consideremos un arreglo B con 4 valores enteros: 60, 30, 50, 45



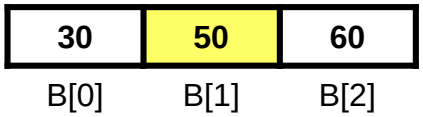
Comienza con 60.

Procesar 30



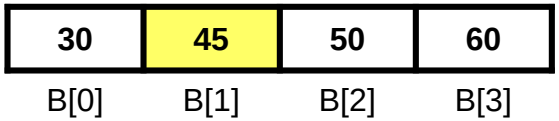
Se inserta 30 en la posición 0. Se mueve 60 a la posición 1.

Procesar 50



Se inserta 50 en la posición 1. Se mueve 60 a la posición 2.

Procesar 45



Se inserta 45 en la posición 1. Se desplazan a la derecha los valores de la derecha.

→ **Arreglo ordenado**

Método de la Burbuja

- **Descripción:**

Lleva los elementos más pequeños hacia la parte izquierda del arreglo.

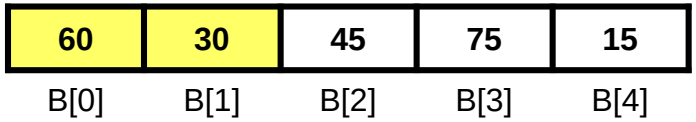
Realiza repetidamente el intercambio de pares de elementos adyacentes hasta que estén todos ordenados.

Se hacen varias pasadas a través del arreglo. En cada pasada se comparan pares de elementos. Si están en orden creciente (o los valores son idénticos) se los deja como está, sino se los intercambia.

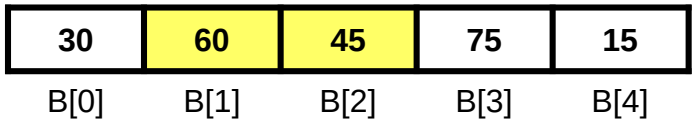
Ordenación: Burbuja

Ejemplo: Consideremos un arreglo B con 5 valores enteros: 60, 30, 45, 75, 15.

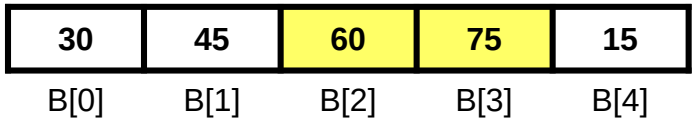
Pasada 1



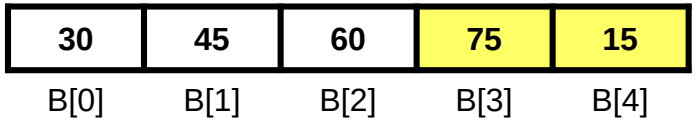
Se intercambia 60 y 30.



Se intercambia 60 y 45.

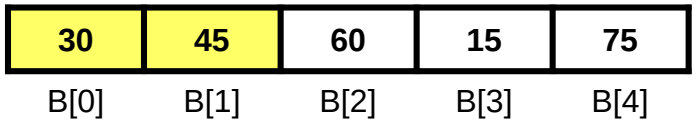


60 y 75 están ordenados.

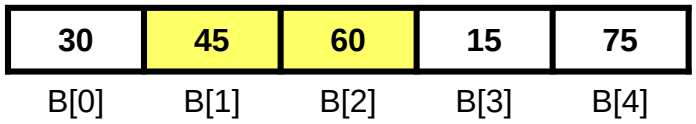


Se intercambia 75 y 15.

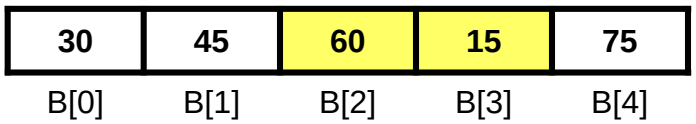
Pasada 2



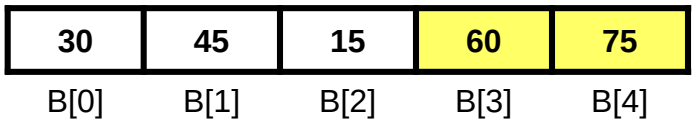
30 y 45 están ordenados.



45 y 60 están ordenados.



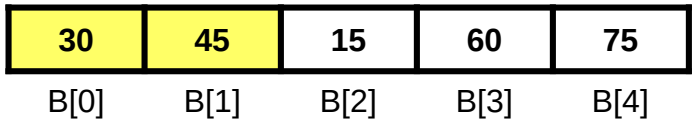
Se intercambia 60 y 15.



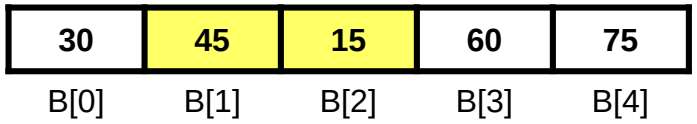
60 y 75 están ordenados.

Ordenación: Burbuja (continuación)

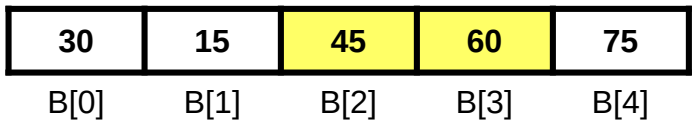
Pasada 3



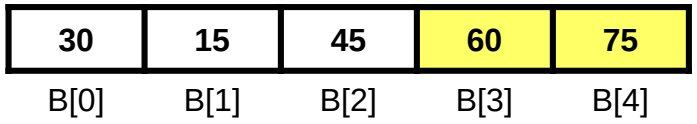
30 y 45 están ordenados.



Se intercambia 45 y 15.

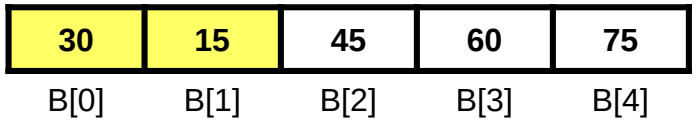


45 y 60 están ordenados.

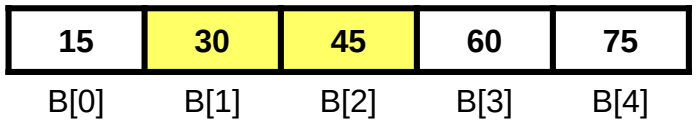


60 y 75 están ordenados.

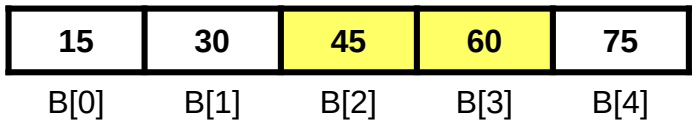
Pasada 4



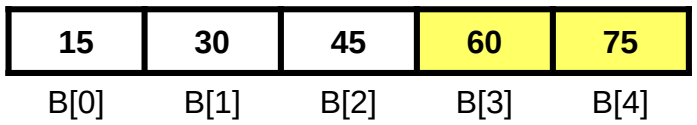
Se intercambia 30 y 15.



30 y 45 están ordenados.



45 y 60 están ordenados.



60 y 75 están ordenados → **Arreglo ordenado**

Método de la Burbuja

- **Descripción:**

  Lleva los elementos más pequeños hacia la parte izquierda del arreglo.

  Realiza repetidamente el intercambio de pares de elementos adyacentes hasta que estén todos ordenados.

  Se hacen varias pasadas a través del arreglo. En cada pasada se comparan pares de elementos. Si están en orden creciente (o los valores son idénticos) se los deja como está, sino se los intercambia.

- **Codificación:**

```
void burbuja(int vec[], const int tl){
    int pasada, k;

    for ( pasada = 1; pasada < tl; pasada++ )

        for ( k = 0; k < tl - 1; k++ )
            if ( vec[ k ] > vec[ k + 1 ] )
                intercambio( vec[ k ], vec[ k + 1 ] );
}
```

Método de la Burbuja Mejorado (v2)

- **Descripción:**
  Es una versión mejorada del algoritmo anterior, donde se realiza una comparación menos por cada pasada (aprovechando el hecho de que los elementos en el extremo derecho van quedando ordenados y no tiene sentido volver a compararlos).

- **Codificación:**

```
void burbujamejorado(int vec[], const int tl){
    int pasada, k;

    for ( pasada = 1; pasada < tl; pasada++ )
        for ( k = 0; k < tl - pasada; k++ )
            if ( vec[ k ] > vec[ k + 1 ] )
                intercambio( vec[ k ], vec[ k + 1 ] );
}
```

Método de la Burbuja mejorado con Centinela

El algoritmo finaliza cuando no se produce intercambio alguno entre los elementos del arreglo, pues esto indica que ya quedó ordenado.

- **Descripción:**
  Este algoritmo utiliza una marca o señal para indicar que no se ha producido ningún intercambio en una pasada. Al final de cada pasada se evalúa si no se han hecho intercambios y en ese caso se termina el ciclo externo.
- **Codificación:**

```
void burbujacentinela(int vec[], const int tl){
    int pasada, k, intercambios=1;
    pasada=1;
    while ( pasada < tl && intercambios ){
        intercambios=0;
        for ( k = 0; k < tl - pasada; k++ )
            if ( vec[ k ] > vec[ k + 1 ] ){
                intercambio( vec[ k ], vec[ k + 1 ] );
                intercambios=1;
            }
        pasada = pasada + 1;
    }
}
```

Análisis del algoritmo de selección directa

Este es uno de los algoritmos más simples de ordenamiento.

Cada elemento se mueve como máximo una vez.

El número de intercambios realizados es menor que en el **burbuja.**

Este algoritmo realiza muchas menos operaciones intercambio() que el de la burbuja (n-1 en total).

Una desventaja respecto al burbuja con centinela es que no mejora su rendimiento cuando los datos ya están ordenados o parcialmente ordenados. En el caso del burbuja con centinela se requiere una única pasada para detectar que el vector ya está ordenado y finalizar. En la ordenación por selección se realiza el mismo número de pasadas independientemente de si los datos están ordenados o no.

Este método es recomendable para un número pequeño de elementos.

Análisis del algoritmo de inserción directa

En el **mejor de los casos**, el arreglo estará inicialmente ordenado, entonces el bucle interno (while) sólo ejecuta el paso de comparación, que siempre será falso. Esto implica que en el mejor de los casos el número de comparaciones sera **n-1**.

En el **peor de los casos**, el arreglo está inversamente ordenado y el número de comparaciones a realizar será el máximo.

En el **caso promedio**, los elementos aparecen en el arreglo en forma aleatoria, y puede ser calculado mediante la suma del mejor y peor caso dividido entre 2.

A pesar de ser un método ineficiente y recomendable solo para un número pequeño de elementos, es intuitivo y de muy fácil implementación.

Análisis de burbuja con centinela

Cuál es la eficiencia del burbuja con centinela?

- Si el **arreglo está completamente desordenado** (peor caso), el ciclo while se comporta como el ciclo for de las versiones anteriores, y la cantidad de comparaciones e intercambios realizados es similar.
- Si el **arreglo queda completamente ordenado** en alguna pasada intermedia, nos ahorramos todas las comparaciones de las pasadas restantes.
- Si el **arreglo está inicialmente ordenado**, sólo se ejecuta una pasada sobre el mismo (realizando n-1 comparaciones) y ningún intercambio.

Merge Sort

- **Descripción:**

  En éste método se unen dos estructuras ordenadas para formar una sola ordenada correctamente.

  - **Consiste en dividir en dos partes iguales el vector a ordenar, ordenar por separado cada una de las partes, y luego mezclar ambas partes, manteniendo el orden, en un solo vector ordenado**

- **Tiene la ventaja de tener complejidad logarítmica: n log (n).**

- **Su desventaja radica en que se requiere de un espacio extra para el procedimiento.**

Merge Sort

```
Mergesort (a, primero, ultimo)

   si (primero < ultimo) Entonces

        central = (primero+ultimo)/2

        mergesort(a, primero, central);  ordena primera mitad
        mergesort(a, central+1, ultimo); ordena segunda mitad
        mezcla(a, primero, central, ultimo); fusiona las dos sublistas

 fin_si
fin
```

Merge Sort

Ejemplo : **9 1 3 5 10 4 6**

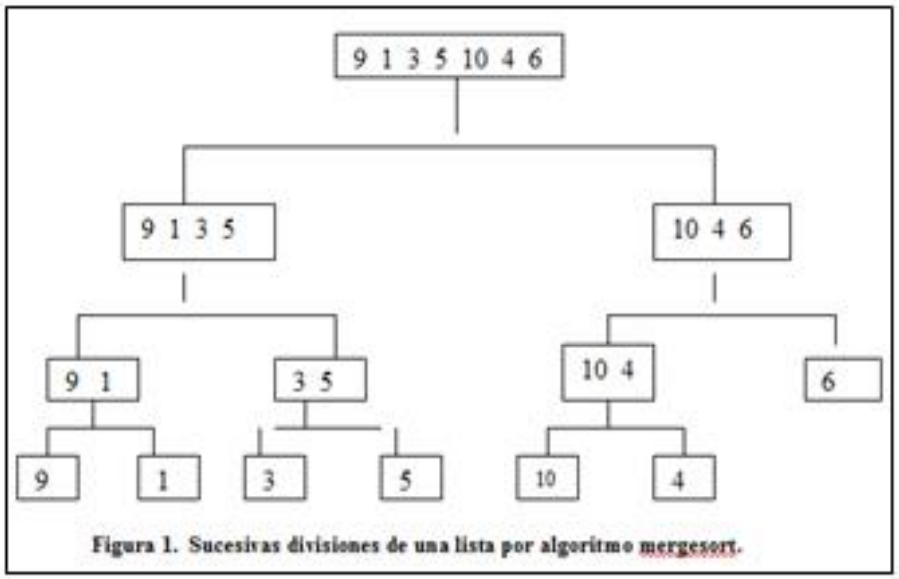


Figura 1. Sucesivas divisiones de una lista por algoritmo mergesort.

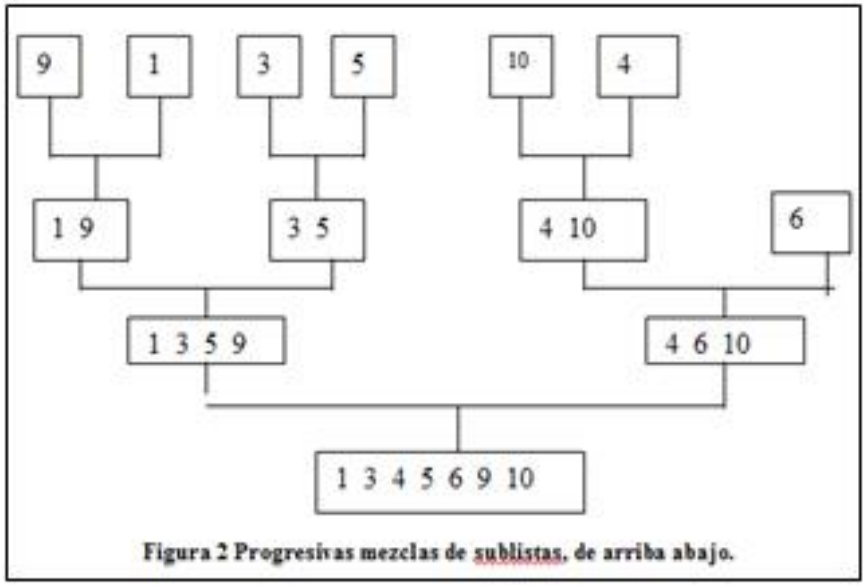


Figura 2 Progresivas mezclas de sublistas, de arriba abajo.

Merge Sort

```cpp
#include <iostream>
using namespace std;

void mergesort(int v[], int inicio , int final);
void merge(int v[], int inicio1, int final1, int inicio2, int final2);

int main (void){
    int c [8];
    int i;

    cout << endl << "Ingrese un dato por renglon:" << endl;
    for (i=0; i<8; i++)
        cin >> c[i];

    mergesort(c,0,7);

    cout << endl << "Se muestran los datos ordenados:" << endl;

    for (i=0; i<8; i++)
        cout << c[i] << "  ";
    return 0;
}

void mergesort(int v[], int inicio , int final) {
    if (final - inicio == 0)
        return;
    else {
        mergesort ( v, inicio , (inicio+final) / 2);
        mergesort ( v, (inicio+final) / 2 + 1, final );
        merge( v, inicio, (inicio+final) / 2, (inicio+final) / 2 + 1, final);
    }
}
```

Merge Sort

```
void mergesort(int v[], int inicio , int final) {
    if (final - inicio == 0)
        return;
    else {
        mergesort ( v, inicio , (inicio+final) / 2);
        mergesort ( v, (inicio+final) / 2 + 1, final );
        merge( v, inicio, (inicio+final) / 2, (inicio+final) / 2 + 1, final);
    }
}
```

Merge Sort

```
void merge(int v[], int inicio1, int final1, int inicio2, int final2) {
    int i, j, k;
    int c[8];

    i = inicio1;
    j = inicio2;
    k = 0;

    while (i<=final1 && j<=final2)
        if (v[i] < v[j])
            c[k++] = v[i++];
        else
            c[k++] = v[j++];

    while (i<=final1)
        c[k++] = v[i++];

    while (j<=final2)
        c[k++] = v[j++];

    for (k=0; k<final2-inicio1+1; k++)
        v[inicio1+k] = c[k];
}
```

```
Ingrese un dato por renglon:
12
4
6
23
99
56
4
0

Se muestran los datos ordenados:
0  4  4  6  12  23  56  99

<< El programa ha finalizado: codigo
<< Presione enter para cerrar esta ve
```

Comportamiento de los métodos de ordenación

Criterios

ESTABILIDAD-es "estable" si valores iguales en el arreglo guardan su orden relativo. Ej: [4_a,4_b,1,7] resulta [1, 4_a,4_b,7]

NATURALIDAD: es "natural" si se tiene en cuenta la posible ordenación del arreglo aunque sea parcial.

EFICIENCIA

ORDEN DEFINITIVO: es aquel que en cada iteración, se pone un elemento en su sitio definitivo, es decir la parte ordenada del arreglo ya es definitiva. Este tipo de métodos sirve para el caso de necesitar ordenar sólo los k (k<n) primeros elementos

Comportamiento de los métodos de ordenación

Criterio / Método	Estable	Natural	Orden definitivo	Eficiencia
Selección Directa	SI (ver)	No	Si	cuadrático
Inserción Directa	SI (ver)	Si	No	cuadrático
Burbuja	SI	No	Si	cuadrático
Merge Sort	SI	No	No	N logN

Videos de Ordenamiento

- **Burbuja**
- http://www.youtube.com/watch?v=YKlDz1J3TSw

- **Selección Directa**
- http://www.youtube.com/watch?v=TbwkDXtHgOU&feature=related

- **Inserción Directa**
- http://www.youtube.com/watch?v=GcmY-Tho1tc
- https://www.youtube.com/watch?v=Md39DDUN_EY

- **Merge Sort**
- http://www.youtube.com/watch?v=L-E6KKmXR0o
- http://www.youtube.com/watch?v=INHF_5RIxTE
- http://www.youtube.com/watch?v=cDNqk4tdvqQ

https://visualgo.net/en/sorting