ALGORITMOS DE ORDENAMIENTO Y BÚSQUEDA EN C

1. Introducción

Los algoritmos de ordenamiento permiten organizar los elementos de un vector según un criterio determinado, normalmente de menor a mayor.

En este material estudiaremos:

- **Bubble Sort** — ordenamiento por burbuja.
- **Selection Sort** — ordenamiento por selección.
- **Insertion Sort** — ordenamiento por inserción.
- **Búsqueda binaria** — búsqueda eficiente de un elemento en un vector ordenado.

Ejemplo de vector desordenado:

```
8 3 5 1 9 2
```

Orden ascendente:

```
1 2 3 5 8 9
```

---

2. Bubble Sort

2.1 ¿Qué hace?

Bubble Sort compara elementos vecinos y los intercambia cuando están en el orden incorrecto.

La idea es que, después de cada recorrido, el elemento mayor va desplazándose hacia el final del vector, como si “subiera” una burbuja.

Ejemplo:

```
5 3 8 2
```

Comparamos:

```
5 y 3
```

Como:

```
5 > 3
```

se intercambian:

3 5 8 2

Después:

5 y 8

No se intercambian:

3 5 8 2

Después:

8 y 2

Se intercambian:

3 5 2 8

El 8 ya quedó en la última posición.

2.2 Código Bubble Sort

```
#include <stdio.h>

void bubbleSort(int v[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
         for (j = i + 1; j < n; j++) {
             if (v[j] <v[i]) {
                temp = v[j];
                v[j] = v[i];
                v[i] = temp;
                }
            }
        }
    }

void mostrar(int v[], int n)
{
    for (int i = 0; i < n; i++)
```

```
    {
        printf("%d ", v[i]);
    }

    printf("\n");
}

int main()
{
    int v[] = {5, 3, 8, 2, 9, 1};
    int n = 6;

    printf("Vector original:\n");
    mostrar(v, n);

    bubbleSort(v, n);

    printf("Vector ordenado:\n");
    mostrar(v, n);

    return 0;
}
```

Resultado:

```
Vector original:
5 3 8 2 9 1

Vector ordenado:
1 2 3 5 8 9
```

2.3 ¿Cómo funciona el código?

La parte principal es:

```
for (i = 0; i < n - 1; i++)
```

Este ciclo controla las pasadas.

El segundo:

```
for (j = 0; j < n - 1 - i; j++)
```

compara elementos consecutivos.

La comparación es:

```
if (v[j] > v[j + 1])
```

Si el elemento izquierdo es mayor que el derecho, se intercambian.

El intercambio se realiza utilizando una variable temporal:

```
temp = v[j];
v[j] = v[j + 1];
v[j + 1] = temp;
```

2.4 Ejemplo de una pasada

Vector:

```
5 3 8 2 9 1
```

Primera comparación:

```
5 > 3
```

Resultado:

```
3 5 8 2 9 1
```

Segunda:

```
5 > 8
```

No cambia:

```
3 5 8 2 9 1
```

Tercera:

```
8 > 2
```

Resultado:

```
3 5 2 8 9 1
```

Cuarta:

```
8 > 9
```

No cambia.

Quinta:

```
9 > 1
```

Resultado:

```
3 5 2 8 1 9
```

El mayor, 9, quedó al final.

---

3. Selection Sort

3.1 ¿Qué hace?

Selection Sort divide conceptualmente el vector en dos partes:

```
PARTE ORDENADA | PARTE NO ORDENADA
```

En cada paso busca el elemento más pequeño de la parte no ordenada y lo coloca en la posición correspondiente.

3.2 Ejemplo

Vector:

```
5 3 8 2 9 1
```

Buscamos el menor de todo el vector:

```
1
```

Lo colocamos al principio:

```
1 3 8 2 9 5
```

Ahora buscamos el menor desde la posición 1:

```
2
```

Lo colocamos:

```
1 2 8 3 9 5
```

Después:

```
1 2 3 8 9 5
```

Luego:

```
1 2 3 5 9 8
```

Finalmente:

```
1 2 3 5 8 9
```

3.3 Código Selection Sort

```c
#include <stdio.h>

void selectionSort(int v[], int n)
{
    int i, j, menor, temp;

    for (i = 0; i < n - 1; i++)
    {
        menor = i;

        for (j = i + 1; j < n; j++)
        {
            if (v[j] < v[menor])
            {
                menor = j;
            }
        }

        temp = v[i];
        v[i] = v[menor];
        v[menor] = temp;
    }
}

void mostrar(int v[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", v[i]);
    }

    printf("\n");
}

int main()
{
    int v[] = {5, 3, 8, 2, 9, 1};
    int n = 6;

    printf("Vector original:\n");
    mostrar(v, n);

    selectionSort(v, n);

    printf("Vector ordenado:\n");
    mostrar(v, n);
```

```
    return 0;
}
```

3.4 ¿Cómo funciona?

Inicialmente:

```
menor = i;
```

Esto supone que la posición `i` contiene el menor valor.

Después se recorre el resto:

```
for (j = i + 1; j < n; j++)
```

Si encontramos uno menor:

```
if (v[j] < v[menor])
{
    menor = j;
}
```

se actualiza la posición del menor.

Cuando termina la búsqueda, se intercambia:

```
temp = v[i];
v[i] = v[menor];
v[menor] = temp;
```

---

4. Insertion Sort

4.1 ¿Qué hace?

Insertion Sort funciona de manera parecida a ordenar cartas en la mano.

Supongamos que tenemos:

```
5 3 8 2
```

Consideramos que `5` ya está ordenado.

Tomamos `3` y buscamos dónde colocarlo:

```
3 5
```

Luego tomamos 8:

```
3 5 8
```

Finalmente tomamos 2:

```
2 3 5 8
```

4.2 Código Insertion Sort

```c
#include <stdio.h>

void insertionSort(int v[], int n)
{
    int i, j, actual;

    for (i = 1; i < n; i++)
    {
        actual = v[i];
        j = i - 1;

        while (j >= 0 && v[j] > actual)
        {
            v[j + 1] = v[j];
            j--;
        }

        v[j + 1] = actual;
    }
}

void mostrar(int v[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", v[i]);
    }

    printf("\n");
}

int main()
{
    int v[] = {5, 3, 8, 2, 9, 1};
    int n = 6;

    printf("Vector original:\n");
    mostrar(v, n);
```

```
    insertionSort(v, n);

    printf("Vector ordenado:\n");
    mostrar(v, n);

    return 0;
}
```

4.3 ¿Cómo funciona?

La variable:

```
actual = v[i];
```

guarda el elemento que queremos insertar.

Luego:

```
j = i - 1;
```

comenzamos a comparar con los elementos anteriores.

Mientras:

```
j >= 0 && v[j] > actual
```

los elementos mayores se desplazan hacia la derecha:

```
v[j + 1] = v[j];
```

Cuando encontramos la posición correcta:

```
v[j + 1] = actual;
```

insertamos el elemento.

---

5. Comparación de los tres algoritmos

Algoritmo	Idea principal
Bubble Sort	Compara vecinos e intercambia
Selection Sort	Busca el menor y lo coloca
Insertion Sort	Toma un elemento y lo inserta donde corresponde

Una forma sencilla de recordarlos:

```
BUBBLE
→ comparar vecinos

SELECTION
→ seleccionar el menor

INSERTION
→ insertar en la posición correcta
```

---

6. Búsqueda binaria

6.1 ¿Qué es?

La búsqueda binaria permite encontrar rápidamente un valor dentro de un vector **ordenado**.

Esta condición es fundamental:

> La búsqueda binaria solamente funciona correctamente si los datos están ordenados.

Por ejemplo:

```
1 3 5 7 9 12 15 20
```

Queremos buscar:

```
12
```

En lugar de revisar uno por uno, analizamos el elemento central.

---

7. Idea de la búsqueda binaria

Tenemos:

```
1 3 5 7 9 12 15 20
```

Posiciones:

```
0 1 2 3 4  5  6  7
```

Inicialmente:

```
inicio = 0
fin = 7
```

Calculamos:

```
medio = (inicio + fin) / 2
```

Entonces:

```
medio = 3
```

El valor es:

```
7
```

Buscamos:

```
12
```

Como:

```
12 > 7
```

sabemos que el elemento debe estar a la derecha.

Por tanto:

```
inicio = medio + 1
```

Ahora:

```
inicio = 4
fin = 7
```

Calculamos nuevamente:

```
medio = 5
```

Tenemos:

```
v[5] = 12
```

Encontramos el elemento.

---

8. Código de búsqueda binaria

```c
#include <stdio.h>

int busquedaBinaria(int v[], int n, int buscado)
{
    int inicio = 0;
    int fin = n - 1;
    int medio;
```

```c
    while (inicio <= fin)
    {
        medio = (inicio + fin) / 2;

        if (v[medio] == buscado)
        {
            return medio;
        }

        if (buscado > v[medio])
        {
            inicio = medio + 1;
        }
        else
        {
            fin = medio - 1;
        }
    }

    return -1;
}

int main()
{
    int v[] = {1, 3, 5, 7, 9, 12, 15, 20};
    int n = 8;

    int buscado;

    printf("Ingrese el valor a buscar: ");
    scanf("%d", &buscado);

    int posicion = busquedaBinaria(v, n, buscado);

    if (posicion != -1)
    {
        printf("Elemento encontrado en la posicion %d\n", posicion);
    }
    else
    {
        printf("Elemento no encontrado\n");
    }

    return 0;
}
```

---

9. ¿Por qué retorna `-1`?

La función:

```
return -1;
```

se utiliza para indicar:

El elemento no fue encontrado.

Una posición válida del vector normalmente comienza en:

```
0
```

Por eso `-1` permite diferenciar:

```
0, 1, 2, 3...
```

de:

```
-1 → no encontrado
```

---

10. Análisis de la búsqueda binaria

La parte más importante es:

```
while (inicio <= fin)
```

Mientras exista un intervalo donde buscar, continuamos.

Calculamos:

```
medio = (inicio + fin) / 2;
```

Comparamos:

```
if (v[medio] == buscado)
```

Si son iguales:

```
return medio;
```

Si buscamos un valor mayor:

```
inicio = medio + 1;
```

Si buscamos un valor menor:

```
fin = medio - 1;
```

En cada paso descartamos aproximadamente la mitad del vector.

---

11. ¿Por qué la búsqueda binaria es más rápida?

Supongamos que tenemos 1.000 elementos.

La búsqueda secuencial puede necesitar revisar muchos elementos.

La búsqueda binaria divide:

```
1000
↓
500
↓
250
↓
125
↓
62
↓
31
↓
15
↓
7
↓
3
↓
1
```

En muy pocas comparaciones podemos reducir enormemente el espacio de búsqueda.

Por eso la búsqueda binaria es mucho más eficiente que revisar uno por uno cuando el vector está ordenado.

---

12. Condición indispensable

Incorrecto:

```
8 2 10 4 7 1
```

Intentar aplicar búsqueda binaria directamente sobre este vector no es válido.

Correcto:

```
1 2 4 7 8 10
```

Primero debemos ordenar.

Por ejemplo:

```
bubbleSort(v, n);
```

y después:

```
posicion = busquedaBinaria(v, n, buscado);
```

Podemos combinar ambos algoritmos:

```
VECTOR DESORDENADO
        ↓
     ORDENAR
        ↓
VECTOR ORDENADO
        ↓
BÚSQUEDA BINARIA
        ↓
 ENCONTRAR ELEMENTO
```

---

13. Ejemplo integrando ordenamiento y búsqueda

```c
#include <stdio.h>

void bubbleSort(int v[], int n)
{
    int i, j, temp;

    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - 1 - i; j++)
        {
            if (v[j] > v[j + 1])
            {
                temp = v[j];
                v[j] = v[j + 1];
                v[j + 1] = temp;
            }
        }
    }
}

int busquedaBinaria(int v[], int n, int buscado)
{
```

```c
    int inicio = 0;
    int fin = n - 1;

    while (inicio <= fin)
    {
        int medio = (inicio + fin) / 2;

        if (v[medio] == buscado)
        {
            return medio;
        }
        else if (buscado > v[medio])
        {
            inicio = medio + 1;
        }
        else
        {
            fin = medio - 1;
        }
    }

    return -1;
}

int main()
{
    int v[] = {25, 10, 40, 5, 30, 15};
    int n = 6;
    int buscado;

    bubbleSort(v, n);

    printf("Vector ordenado:\n");

    for (int i = 0; i < n; i++)
    {
        printf("%d ", v[i]);
    }

    printf("\n\nIngrese el valor a buscar: ");
    scanf("%d", &buscado);

    int posicion = busquedaBinaria(v, n, buscado);

    if (posicion != -1)
    {
        printf("Encontrado en la posicion %d\n", posicion);
    }
```

```
    else
    {
        printf("No encontrado\n");
    }

    return 0;
}
```

---

14. Diferencia entre búsqueda secuencial y binaria

Búsqueda secuencial

Busca uno por uno:

```
10 → ¿es?
25 → ¿es?
30 → ¿es?
40 → ¿es?
```

No requiere que el vector esté ordenado.

Búsqueda binaria

Compara con el elemento central y elimina la mitad del espacio de búsqueda.

Requiere:

```
VECTOR ORDENADO
```

---

15. Comparación general

Algoritmo	Propósito	Requiere orden previo
Bubble Sort	Ordenar	No
Selection Sort	Ordenar	No
Insertion Sort	Ordenar	No
Búsqueda binaria	Buscar	Sí

---

16. Complejidad básica

Para un primer acercamiento a la eficiencia:

Algoritmo	Complejidad aproximada
Bubble Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Búsqueda binaria	O(log n)

Esto permite que el estudiante comprenda una idea importante:

> Ordenar puede ser relativamente costoso, pero una vez que los datos están ordenados, buscar mediante búsqueda binaria puede ser muy eficiente.

---

17. ¿Cómo diferenciarlos rápidamente?

Bubble Sort

Pregunta:

> ¿Estoy comparando vecinos?

Sí:

```
v[j] > v[j + 1]
```

Entonces probablemente es Bubble Sort.

Selection Sort

Pregunta:

> ¿Estoy buscando la posición del menor?

Sí:

```
menor = i;
```

y después:

```
if (v[j] < v[menor])
```

Entonces es Selection Sort.

Insertion Sort

Pregunta:

¿Estoy tomando un elemento y desplazando los mayores para insertarlo?

Sí:

```
actual = v[i];
```

y:

```
v[j + 1] = v[j];
```

Entonces es Insertion Sort.

Búsqueda binaria

Pregunta:

¿Estoy utilizando `inicio`, `fin` y `medio`?

Sí:

```
medio = (inicio + fin) / 2;
```

Entonces se trata de una búsqueda binaria.
