Algoritmos de Búsqueda y Ordenamiento

Algoritmos de Búsqueda

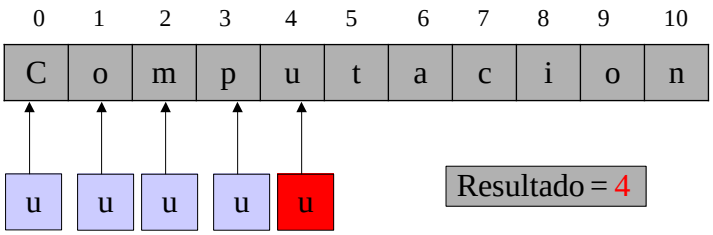
- Los procesos de búsqueda involucran recorrer un array completo con el fin de encontrar algo. Lo más común es buscar el menor o mayor elemento (cuando es puede establecer un orden), o buscar el índice de un elemento determinado.
- Para buscar el menor o mayor elemento de un array, podemos usar la estrategia, de suponer que el primero o el último es el menor (mayor), para luego ir **comparando con cada uno de los elementos**, e ir actualizando el menor (mayor). A esto se le llama Búsqueda Lineal.

Algoritmos de Búsqueda

- Definición:
  - Para encontrar un dato dentro de un array, para ello existen diversos algoritmos que varían en complejidad, eficiencia, tamaño del dominio de búsqueda.
- Algoritmos de Búsqueda:
  - Búsqueda Secuencial
  - Búsqueda Binaria

Búsqueda Secuencial

- Consiste en ir comparando el elemento que se busca con cada elemento del array hasta cuando se encuentra.
- Busquemos el elementos **'u'**



Búsqueda Secuencial

- Búsqueda del menor

```
menor = a[0];
for (i=1;i<n;i++) {
        if ( a[i]<menor )
                menor=a[i]; }
```

- Búsqueda del mayor

```
mayor= a[n-1];
for (i=0;i<n-1;i++) {
                if ( a[i]>mayor )
                  mayor=a[i]; }
```

- Búsqueda de elemento

```
 encontrado=-1;
for (i=0;i<n;i++) {
        if ( a[i]==elemento_buscado )
           encontrado=i; }
```

Ejemplo

Desarrollar un programa que posea una función que reciba como parámetro un array de 10 enteros, y un entero, y retorne la posición del entero si es que se encuentra, de lo contrario devolver –1.

```
int encuentra(int[] A, int b) { int k=0,
        result=-1;

        do{
                if (A[k]== b)
                        result =k;
                else
                        k++;
        }while ((result==-1)&&(k<10)); return
        result;
}

public static void main(String[] args) {  int i,
        int[] x = new int[10];
        Scanner   sc   =   new   Scanner(System.in);
        for(i=0;i<10;i++)
                x= sc.nextInt();
                i   =   encuentra(   x,   10);
        System.out.println("resultado: "+i);
}
```

Eficiencia y Complejidad

- Considerando la Cantidad de Comparaciones
  - Mejor Caso:

1

    - El elemento buscado está en la primera posición. Es decir, se hace una sola comparación
  - Peor Caso:

n

    - El elemento buscado está en la última posición. Necesitando igual cantidad de comparaciones que de elementos el arreglo
  - En Promedio:

n/2

    - El elemento buscado estará cerca de la mitad. Necesitando en promedio, la mitad de comparaciones que de elementos
- Por lo tanto, la velocidad de ejecución depende linealmente del tamaño del array

Búsqueda Binaria

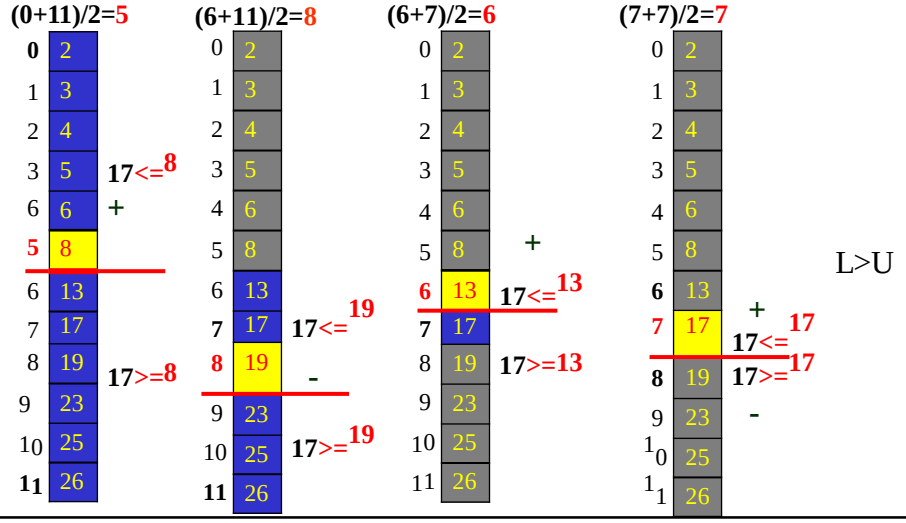
En el caso anterior de búsqueda se asume que los elementos están en cualquier orden. En el peor de los casos deben hacerse **n** operaciones de comparación.

Una búsqueda más eficiente puede hacerse sobre un array ordenado. Una de éstas es la Búsqueda Binaria.

La Búsqueda Binaria, compara si el valor buscado está en la mitad superior o inferior. En la que esté, subdivido nuevamente, y así sucesivamente hasta encontrar el valor.

Búsqueda Binaria

- Supuesto: Arreglo con datos ordenados en forma ascendente: i<k → a[i]<a[k]
- Estamos buscando la posición del valor 17



Algoritmo de Búsqueda Binaria

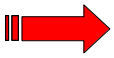
```
public static void main(String[] args) {
    int b,i,j,k;
    int[] v = new int[12];
    Scanner sc= new Scanner(System.in);
    for(i=0;i<12;i++)
        v[i]=sc.nextInt();
    System.out.println("fin del llenado");
    System.out.println("ingrese numero a buscar
    ");
    b=sc.nextInt();
    i= 0;
    j= 12-1;
    do {
        k=  (i+j)/2;
        if (v[k]<=b )
            i=k+1;
        if (v[k]>=b )
            j= k-1;
    } while (i<=j);
    System.out.println("elemento "+d+"esta en
    "+v[k]+k)
}
```

```
i= 0;
j= tamaño-1;
do {
    k=  (i+j)/2;
    if (v[k]<=b )
        i=k+1;
    if (v[k]>=b )
        j= k-1;
} while (i<=j);
```

Algoritmo de Búsqueda Binaria

```
public static void main(String[] args) {
        int b,i,j,k;
        int[] v = new int[12];
        Scanner sc= new Scanner(System.in)
        for(i=0;i<12;i++)
                v[i]=sc.nextInt();
        System.out.println("fin del llenado");
        System.out.println("ingrese numero a buscar
        ");
        b=sc.nextInt();
        i= 0;
        j= 12-1;
        do {
                k=  (i+j)/2;
                if (v[k]<=b )
                        i=k+1;
                if (v[k]>=b )
                        j= k-1;
        } while (i<=j);
        System.out.println("elemento "+d+"esta en
        "+v[k]+" "+k)

}
```



```
i= 0;
j= tamaño-1;
do {
        k=  (i+j)/2;
        if (v[k]<=b )
                i=k+1;
        if (v[k]>=b )
                j= k-1;
 } while (i<=j);
```

Complejidad y Eficiencia

- Contando Comparaciones
  - Mejor Caso:

1

    - El elemento buscado está en el centro. Por lo tanto, se hace una sola comparación
  - Peor Caso:

log(n)

    - El elemento buscado está en una esquina. Necesitando $\log_2(n)$ cantidad de comparaciones

log(n/2)

  - En Promedio:
    - Serán algo como $\log_2(n/2)$
- Por lo tanto, la velocidad de ejecución depende logarítmicamente del tamaño del array

Ordenamiento de Componentes

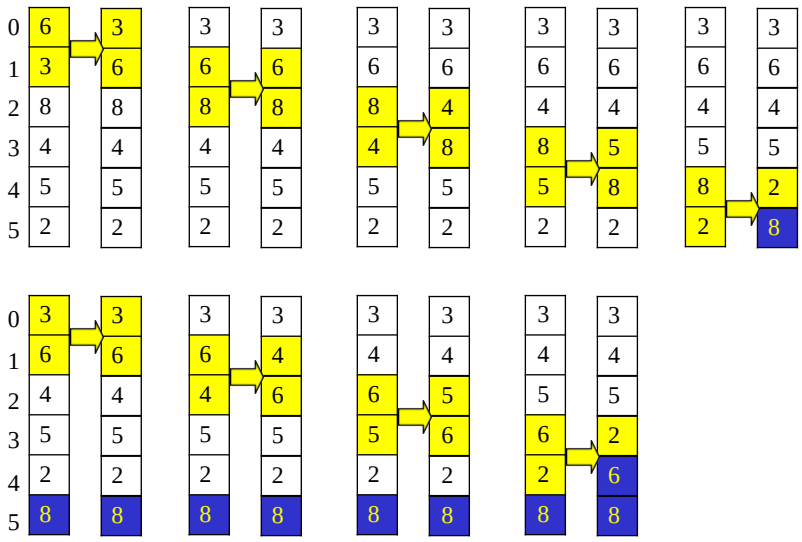
Ordenamiento Ascendente

- Existen numerosos algoritmos para ordenar. A continuación se verán algunos algoritmos de ordenamiento.
- Ordenamiento Burbuja (bublesort):

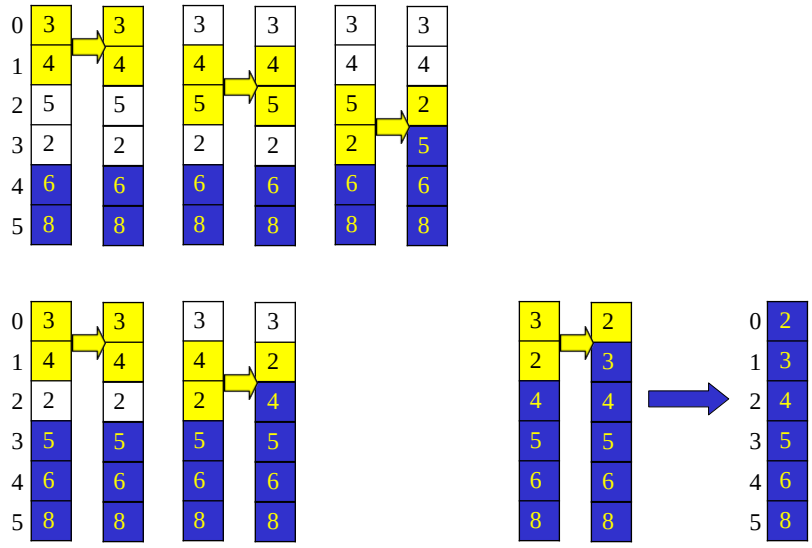
  Idea: vamos comparando elementos adyacentes y empujamos los valores más livianos hacia arriba (los más pesados van quedando abajo). Idea de la burbuja que asciende, por lo liviana que es.

0	3
1	6
2	8
3	4
4	5
5	2

Ordenamiento Burbuja



Ordenamiento Burbuja



Algoritmo Ordenamiento Burbuja

```
void intercambia(int[] f,int[] g)
    { //SE INTERCAMBIAN
¿COMO?
"Arrays", for…
 }




public static void main(String[] args) {
    int i,j,
    int[] v ={3,4,5,2,6,8};
    for (i=6-1;i>1;i--)
            for (j=0;j<i;j++)
                    if (v[j]>v[j+1])
            intercambia(v[j],v[j+1]);
    for (i=0;i<N;i++)
            System.out.println(v
    [i]);}
```

```
for (i=N-1;i>0;i--)
    for (j=0;j<i;j++)
      if (V[j]>V[j+1])
          Intercambia(A[j],A[j+1]);
```

Complejidad y Eficiencia

- Cantidad de Comparaciones:
  - Constante: n*(n+1)/2
- Cantidad de Intercambios:
  - Mejor Caso:

0

    - Arreglo ordenado. Por lo tanto, no se hace ni un solo swap
  - Peor Caso:

n*(n+1)/2

    - Arreglo ordenado inversamente. Se necesitarán n*(n+1)/2 cantidad de swaps
  - En Promedio:

n*(n+1)/4

    - Serán algo como n*(n+1)/4 swaps
- Por lo tanto, la velocidad de ejecución depende cuadráticamente del tamaño del arreglo