Análisis de búsquedas

Búsqueda secuencial

Búsqueda secuencial

- Consiste en buscar el elemento comparándolo secuencialmente (de ahí su nombre) con cada elemento del arreglo o conjunto de datos hasta que se encuentre, o hasta que se llegue al final del arreglo.

- La existencia se puede asegurar desde el momento que el elemento es localizado, pero no podemos asegurar la no existencia hasta no haber analizado todos los elementos del arreglo.

- Se utiliza cuando el contenido del arreglo no se encuentra o no puede ser ordenado.

Ejemplo. Búsqueda secuencial de un elemento dentro de un *array* de longitud len.

```
int busquedaSecuencial(int arr[], int len, int v)
{
   int i = 0;
   while( i<len && arr[i]!=v )
   {
         i = i + 1;
   }
   return i<len?i:-1;
}
```

Ejemplo. Búsqueda secuencial de un elemento dentro de un *array* de longitud len.

```
int busquedaBinaria(int arr[], int len, int v)
{
    int i = 0;                          1
    while( i<len && arr[i]!=v )         4
    {
        i = i + 1;
    }
    return i<len?i:-1;                  2
}
```

1 + 4 + 2 = **7**

Ejemplo. Búsqueda secuencial de un elemento dentro de un *array* de longitud len.

```
int busquedaBinaria(int arr[], int len, int
    v)
{
    int i = 0;
    while( i<len && arr[i]!=v )
    {
        i = i + 1;
    }
    return i<len?i:-1;
}
```

4

2

Ciclo While

Donde ***n*** es la cantidad de iteraciones que realiza el *while*

Instrucción	**i = i + 1**	**i<len && arr[i]!v**
Unidades de tiempo	2n	4n

Ejemplo. Búsqueda secuencial de un elemento dentro de un *array* de longitud len.

$$f(n) = 6n + 7$$

- Si el elemento que buscamos se encuentra en la primera posición del *array* entonces el algoritmo no ingresará al while y solo requerirá 7 unidades de tiempo. **Mejor de los casos**.

- **Peor de los casos.** Si el valor que buscamos no existe en el *array* o cuando se encuentre en la última posición.

  7 unidades de tiempo fijas + 6*n*, siendo *n* igual a *len*

Búsqueda binaria

Algoritmo de búsqueda binaria

- Después de ordenar un conjunto numérico en una secuencia creciente o decreciente, el algoritmo de búsqueda binaria empieza desde la parte media de la secuencia.

- Si el punto de prueba es igual al punto medio de la secuencia, finaliza el algoritmo.

- En caso contrario, dependiendo del resultado de comparar el elemento de prueba y el punto central de la secuencia, de manera recurrente se busca a la izquierda o a la derecha de la secuencia.

Primera iteración:
i=0 y j=len

Segunda iteración:
Se procesará la mitad del array ya que habremos desplazado algunos de los índices i, j.

len/2 elementos

Tercera iteración:
Se procesará la mitad de la mitad del array, es decir, len/2/2, lo que equivale a decir:

$len/2^2$ elementos

Cuarta iteración:
$len/2/2/2 = len/2^3$

```
int busquedaBinaria(int arr[], int len, int v)
{
        int i = 0;
        int j = len;
        int enc = 0; //false

        while(i<=j && !enc)
        {
                int k = (i+j)/2;
                if(arr[k]==v)
                        enc = 1; //true
                else
                {
                        if(arr[k]<v)
                                i = k + 1;
                        else
                                j = k - 1;
                }
        }
        return enc?k:-1;
}
```

- Generalizando, en la n-ésima iteración estaremos procesando $\frac{len}{2^n}$ elementos.
- El peor de los casos se dará cuando $\frac{len}{2^n}$ sea igual a 1 ya que en esta situación aún no habremos encontrado el elemento que buscamos y no podremos seguir dividiendo el array.
- Ahora bien,

si: $1 = \frac{len}{2^n}$ Entonces

Pasamos multiplicando 2^n : $2^n = len$ Luego,

Aplicando $\log_2$ en ambos lados: $\log_2(2^n) = \log_2(len)$

Resolvemos: $n = \log_2(len)$

$$\log_b(x^y) = y\log_b(x)$$

$$\log_b b = 1$$

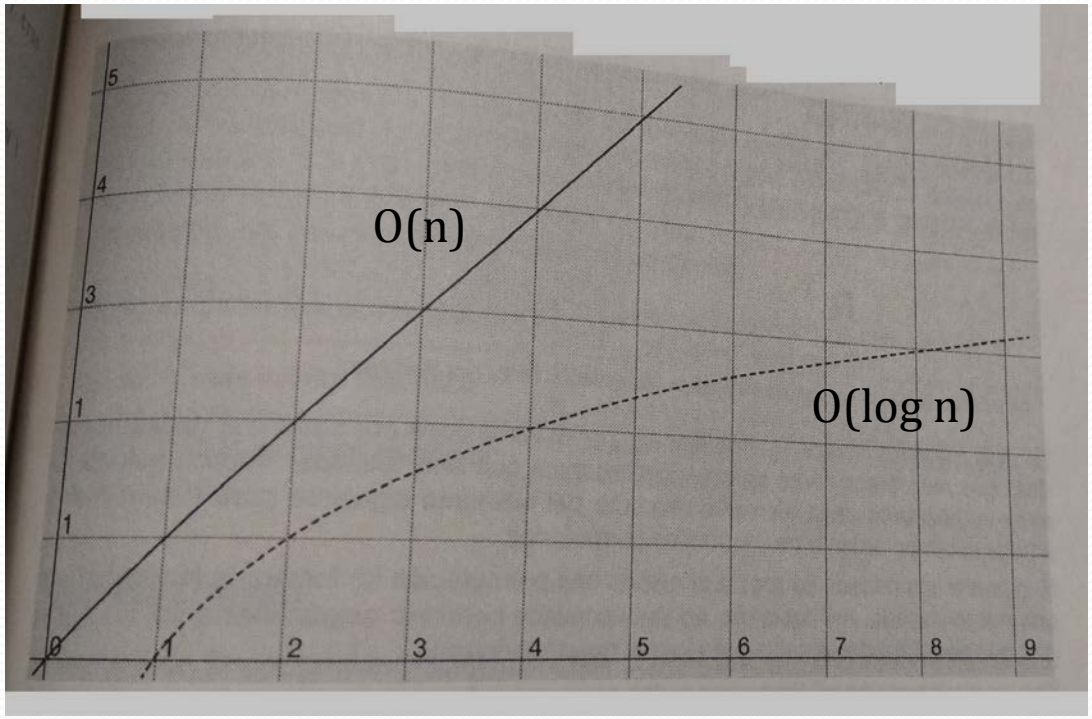
$$\log_2(2^n) = n\log_2(2)$$

$$\log_2(2^n) = n(1) = n$$

Recordemos que *n* es la cantidad de iteraciones que, en el peor de los casos, realiza el *while*; por lo tanto, la función de complejidad de la búsqueda binaria es $\log_2(k)$, siendo *k=len*.

Para simplificarlo diremos que el algoritmo tiene una complejidad logarítmica.

- Comparemos las gráficas de las funciones de la búsqueda binaria y la búsqueda secuencial.



Como vemos, la función de la búsqueda binaria tiene un crecimiento mucho más atenuado que la función de la búsqueda secuencial.

Esto permite determinar que a medida que crece el tamaño del *array* (eje de las equis) requerirá realizar una menor cantidad de instrucciones.

Búsqueda secuencial
COMPLEJIDAD LINEAL
O(n)

Búsqueda binaria
COMPLEJIDAD LOGARÍTMICA
O(log n)