Búsqueda secuencial

Características

* funciona bien con arreglos pequeños o para arreglos no ordenados.
* Es un método sumamente simple que resulta útil cuando se tiene un conjunto de datos pequeños (Hasta aproximadamente 500 elementos)
* Es fácil adaptar la búsqueda secuencial para que utilice una lista enlazada ordenada, lo que hace la búsqueda más eficaz.
* Si los datos buscados no están en orden es el único método que puede emplearse para hacer dichas búsquedas
* Este método tiende hacer muy lento.
* En el peor de los casos se recorre el array completo y el valor no se encuentra o se recorre el array completo si el valor buscado está en la última posición del array.

Complejidad

– Mejor Caso: El elemento buscado está en la primera posición. Es decir, se

hace una sola comparación

– Peor Caso: El elemento buscado está en la última posición. Necesitando

igual cantidad de comparaciones que de elementos el arreglo

– Caso medio: 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 arreglo

Ejemplo

i=0 Valor buscado=3 tamaño=5

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 5 | 3 | 18 | 20 |

i=1 Valor buscado=3 tamaño=5

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 5 | 3 | 18 | 20 |

i=2 Valor buscado=3 tamaño=5

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 5 | 3 | 18 | 20 |

Elemento buscado entrado en la iteración 2

Búsqueda secuencial ordenada

características

* El algoritmo anterior puede ser mejorado si el vector esta ordenado
* El recorrido termina en cuanto encuentra el valor buscado
* En ocasiones no recorre todo el vector
* si durante la búsqueda se alcanza un componente con mayor valor que ‘elemento buscado’, podremos asegurar que no se encuentra dentro del vector

complejidad

* mejor caso: El elemento buscado está en la primera posición o es menor que todos los elementos del vector.
* Peor caso: cuando el elemento a buscar es el último del vector o mayor que todos los elementos del vector
* Caso medio: el elemento a buscar se encuentra en medio del vector, el algoritmo tendría que emplear n/2 iteraciones

Binaria

características

* La lista debe estar ordenada, en un orden especifico de acuerdo al valor de la clave.
* Debe conocerse el número de elementos
* Se puede aplicar tanto a datos en listas lineales como en árboles binarios de búsqueda.
* Es el método más eficiente para encontrar elementos en un arreglo ordenado.
* si nos encontramos con arreglos que no están en orden, este método, no nos ayudaría en nada.
* Es mas eficiente que el método de búsqueda secuencial, debido a que el numero de comparaciones se reduce a la mitad por cada iteración del método

Complejidad

– Mejor Caso: El elemento buscado está en el centro. Por lo tanto, se

hace una sola comparación

– Peor Caso: El elemento buscado está en una esquina. Necesitando

log2(n) cantidad de comparaciones

– En Promedio: Serán algo como log2(n/2)

• Por lo tanto, la velocidad de ejecución depende

logarítmicamente del tamaño del arreglo

Algoritmo

* Se compara la llave buscada con la llave localizada al centro del arreglo.
* Si la llave analizada corresponde a la buscada fin de búsqueda si no.
* Si la llave buscada es menor que la analizada repetir proceso en mitad superior, sino en la mitad inferior.
* El proceso de partir por la mitad el arreglo se repite hasta encontrar el registro o hasta que el tamaño de la lista restante sea cero, lo cual implica que el valor de la llave buscada no está en la lista.

Burbuja

Caracteristicas

* Este algoritmo sólo requiere de una variable adicional para realizar los intercambio
* requiere n al cuadrado de pasos de procesamiento para cada n número de elementos a ser ordenados.
* Funciona comparando elementos de dos en dos en un ciclo, intercambiándolos según sea el caso ascendente o descendente.
* Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios.

Complejidad

* – Mejor Caso: Arreglo ordenado. Por lo tanto, no se hace ni un solo cambio
* – Peor Caso: Arreglo ordenado inversamente. Será necesario ordenar todo