

Contenido

[QuickSort (algoritmo de ordenamiento) 3](#_Toc183299980)

[Algoritmo de búsqueda Binary Search 10](#_Toc183299981)

[Pasos del algoritmo: 11](#_Toc183299982)

[Ejemplos Prácticos: 13](#_Toc183299983)

[Referencia 14](#_Toc183299984)

# QuickSort (algoritmo de ordenamiento)

Este es un método de ordenamiento rápido actualmente el más eficiente y veloz de los métodos de ordenación interna. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de QuickSort por la velocidad con que ordena los elementos del arreglo.

Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar “n” elementos en un tiempo proporcional a “n Log n”.

**Descripción del algoritmo.**

Elegir un elemento de la lista de elementos a ordenar, al que llamaremos “pivote”.

La idea central de este algoritmo consiste en lo siguiente:

* Se toma un elemento “x” de una posición cualquiera del arreglo. Se trata de ubicar a “x”

en la posición correcta del arreglo, de tal forma que todos los elementos que se

encuentran a su izquierda sean menores o iguales a “x” y todos los elementos que se

encuentren a su derecha sean mayores o iguales a “x”.

* Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se

encuentran a la izquierda y a la derecha de la posición correcta de “x” en el arreglo.

* Reubicar los demás elementos de la lista a cada lado del pivote, de manera que a un

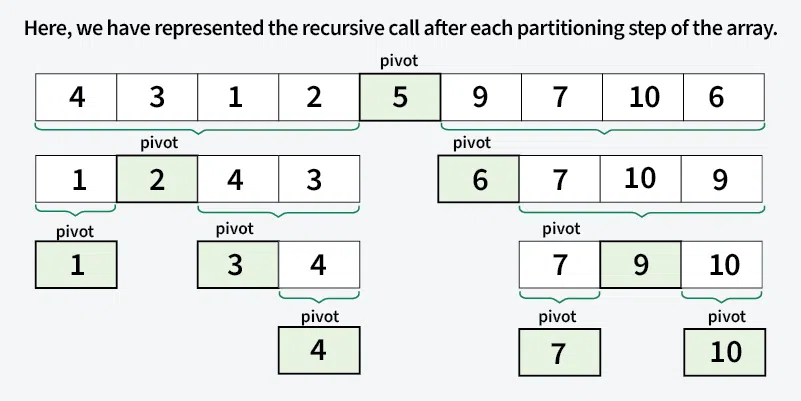
lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote

ocupa exactamente el lugar que le corresponderá en la lista ordenada.

* Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan

más de un elemento. Una vez terminado este proceso todos los elementos estarán

ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el

pivote elegido.

Ejemplo:

A screenshot of a video game

Description automatically generated

* Tomamos un valor de nuestro arreglo como pivote

A screenshot of a video game

Description automatically generated

* Se ordenan todos los elementos menores que el pivote a la izquierda y los mayores a la derecha

A screenshot of a video game

Description automatically generated

* Para elegir el pivote se puede hacer de muchas formas, un ejemplo de cómo elegirlo.

A screenshot of a computer

Description automatically generated

* En este caso el elegimos el ultimo elemento como pivote y luego definimos como apuntador el elemento más pequeño, inicializándolo -1, luego se recorrería el arreglo de principio a fin y se comparara con el pivote si este es más grande no se realiza nada, pero si es más pequeño se intercambian a un lado del elemento más pequeño.
* Una vez que lleguemos al 4 se realizara el intercambio

A screenshot of a computer

Description automatically generated

* El 7 y el 4 cambian de posiciones

A screenshot of a computer

Description automatically generated

* Una vez recorrido todo el arreglo, se hace un último intercambio entre el pivote y el elemento que está a la derecha del último elemento más pequeño, el 6 queda en su posición final.

A screenshot of a computer

Description automatically generated

* Esto deja con dos sub-arreglos, los cuales se resuelven de forma recursiva con el algoritmo de QuickSort.

A screenshot of a computer

Description automatically generated



**Ventajas de QuickSort**

1. Es un algoritmo de divide y vencerás que facilita la resolución de problemas.
2. Es eficiente con conjuntos de datos grandes.
3. Tiene un bajo consumo de recursos, ya que solo requiere una pequeña cantidad de memoria para funcionar.
4. Es amigable con la caché, ya que trabaja directamente en el mismo arreglo sin copiar datos a arreglos auxiliares.
5. Es el algoritmo más rápido de propósito general para conjuntos de datos grandes cuando no se requiere estabilidad.
6. Es recursivo por cola, lo que permite optimizaciones en las llamadas de cola.

Un ejemplo podría ser, digamos que tenemos un gran conjunto de datos de 1 millón de registros que deben ordenarse para una aplicación de listado. Con Quick Sort, podemos lograr una clasificación eficiente en un período de tiempo razonable. Por otro lado, si utilizáramos Bubble Sort o Insertion Sort, el proceso de clasificación sería significativamente más lento, lo que podría provocar un rendimiento deficiente en la aplicación de listado.

La estrategia de dividir y conquistar de Quick Sort, junto con su eficiente complejidad temporal, lo convierte en la opción preferida para optimizar las aplicaciones de listado. Al utilizar este algoritmo, los desarrolladores pueden asegurarse de que sus aplicaciones puedan ordenar grandes conjuntos de datos con facilidad y brindar una experiencia de usuario perfecta.

**Desventajas de QuickSort**

1. Tiene una complejidad temporal en el peor caso de **O(n²)**, que ocurre cuando el pivote se elige de forma inadecuada.
2. No es una buena opción para conjuntos de datos pequeños.
3. No es un algoritmo estable, lo que significa que, si dos elementos tienen la misma clave, su orden relativo no se conservará en el resultado ordenado, ya que los elementos se intercambian según la posición del pivote sin considerar sus posiciones originales.

# Algoritmo de búsqueda Binary Search

La búsqueda binaria es un tipo de algoritmo de búsqueda que se utiliza para encontrar rápidamente un elemento de destino dentro de una matriz ordenada. Es una forma rápida y efectiva de encontrar un elemento en una matriz, ya que el elemento de destino se puede identificar en menos comparaciones cada vez (Time complexity O(log2)), donde n es el número de elementos en la matriz. Log2, ya que la cantidad de área donde se busca se reduce por dos en cada iteración.

Podemos pensar en la búsqueda binaria como un diccionario, la razón por la cual una búsqueda de diccionario es sencilla es que sabemos de antemano si una palabra se encuentra al inicio, al medio o al final con solo saber con qué letra comienza. Si nuestra palabra comienza con A o con Z será bastante fácil saber si comenzar por el inicio o el final del diccionario. Sobre este concepto se basa la implementación. Ir reduciendo nuestra área de búsqueda y descartar el contenido donde sabemos que no vamos a encontrar a nuestro candidato. Por ejemplo, si buscáramos la palabra “Mango” podríamos comenzar por la mitad. Y conservar únicamente la mitad que sabemos que contiene a las palabras con la letra “M”. Separar dos mitades (binario) nos quedamos con la mitad que esté en el rango y descartamos la otra, luego hay que repetir el proceso hasta encontrar al elemento. Esa es la idea detrás de la búsqueda binaria.

A diagram of a search engine

Description automatically generated with medium confidence

## Pasos del algoritmo:

**1. Ordenar la matriz:**

Es importante tener en cuenta que la matriz debe ordenarse en orden creciente o decreciente, según el elemento de destino.

**2. Inicializar el área de búsqueda:**

Una vez determinado el elemento objetivo, se debe inicializar el área de búsqueda. Esto se puede hacer estableciendo los límites izquierdos (low) y derecho (high) de la matriz. El límite izquierdo debe establecerse en el índice del primer elemento de la matriz y el límite derecho debe establecerse en el índice del último elemento de la matriz.

**3. Calcular el punto medio:**

Una vez que se han establecido los límites, se puede calcular el punto medio. Esto se puede hacer tomando el promedio de los límites izquierdo y derecho. Este punto medio se usará para determinar en qué mitad de la matriz buscar en el siguiente paso.

**4. Comparar el elemento objetivo y el elemento en el punto medio:**

Una vez que se calcula el punto medio, el elemento objetivo se puede comparar con el elemento en el punto medio. Si el elemento de destino es igual al elemento en el punto medio, entonces la búsqueda está completa y se ha encontrado el elemento de destino.

**6. Ajustar el área de búsqueda:**

Si el elemento de destino no es igual al elemento en el punto medio, entonces se debe ajustar el área de búsqueda. Dependiendo de si el elemento de destino es más grande o más pequeño que el elemento en el punto medio, el límite izquierdo o derecho debe ajustarse en consecuencia. Si el elemento de destino es más grande, el límite izquierdo debe establecerse en el punto medio + 1. Si el elemento de destino es más pequeño, el límite derecho debe establecerse en el punto medio — 1.

**7. Repetir los pasos 3 a 6:**

Los pasos para calcular el punto medio, comparar el elemento objetivo y el elemento en el punto medio y ajustar el área de búsqueda deben repetirse hasta que el objetivo. Esto se puede hacer de forma iterative con un loop while o una funciona que se manda a llamar a si misma (recursivo)



Las siguientes razones hacen que la búsqueda binaria sea una mejor opción para utilizar como algoritmo de búsqueda:

* La búsqueda binaria funciona de manera eficiente con datos ordenados sin importar el tamaño de los datos.
* En lugar de realizar la búsqueda revisando los datos en una secuencia, el algoritmo binario accede aleatoriamente a los datos para encontrar el elemento requerido. Esto hace que los ciclos de búsqueda sean más cortos y precisos.
* La búsqueda binaria realiza comparaciones de los datos ordenados basándose en un principio de ordenamiento en lugar de utilizar comparaciones de igualdad, que son más lentas y en su mayoría inexactas.
* Después de cada ciclo de búsqueda, el algoritmo divide el tamaño de la matriz a la mitad, por lo que en la siguiente iteración funcionará solo en la mitad restante de la matriz.

## Ejemplos Prácticos:

[FSJ25/Tarea\_algoritmos at main · Karlach-a/FSJ25](https://github.com/Karlach-a/FSJ25/tree/main/Tarea_algoritmos)

## Referencia

<https://youtu.be/SLauY6PpjW4?si=kf9obpF-cpeiPF-H>

<https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf>

[Divide and Conquer - Quicksort](https://algorithm-visualizer.org/divide-and-conquer/quicksort)

[Quick Sort | Ordenamiento Rápido | Explicado con Cartitas!](https://www.youtube.com/watch?v=UrPJLhKF1jY)

[Quick Sort - GeeksforGeeks](https://www.geeksforgeeks.org/quick-sort-algorithm/)

[Algoritmos de clasificacion optimizacion de las aplicaciones de listado para lograr eficiencia - FasterCapital](https://fastercapital.com/es/contenido/Algoritmos-de-clasificacion--optimizacion-de-las-aplicaciones-de-listado-para-lograr-eficiencia.html)

[Búsqueda binaria (artículo) | Algoritmos | Khan Academy](https://es.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search#:~:text=La%20b%C3%BAsqueda%20binaria%20es%20un,ubicaciones%20posibles%20a%20solo%20una.)