Trabajo práctico - Estructura de Datos
Comisión: Mañana

Integrantes:
-Garegnani, Luciano Ezequiel
-Sarubbi, Mateo Alejandro

Algoritmos De Ordenamiento

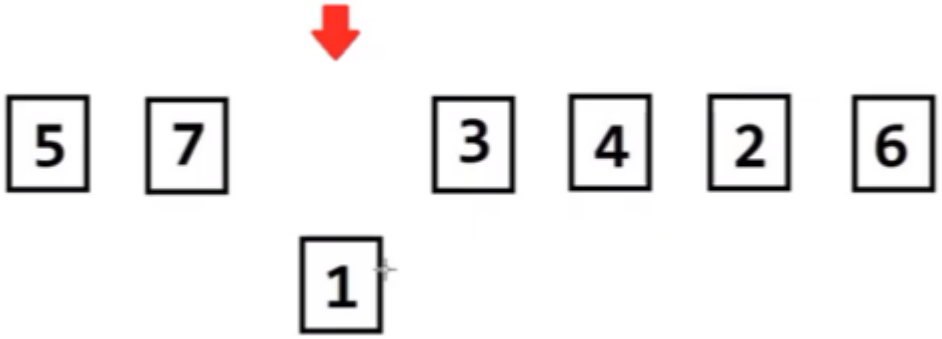
Ordenamiento por inserción:

El ordenamiento por inserción, como lo dice el nombre, ordena los elementos de un arreglo desordenado, pero ¿cómo lo hace?.
Generalmente para explicar este algoritmo se usa un ejemplo de un mazo de cartas desordenado.

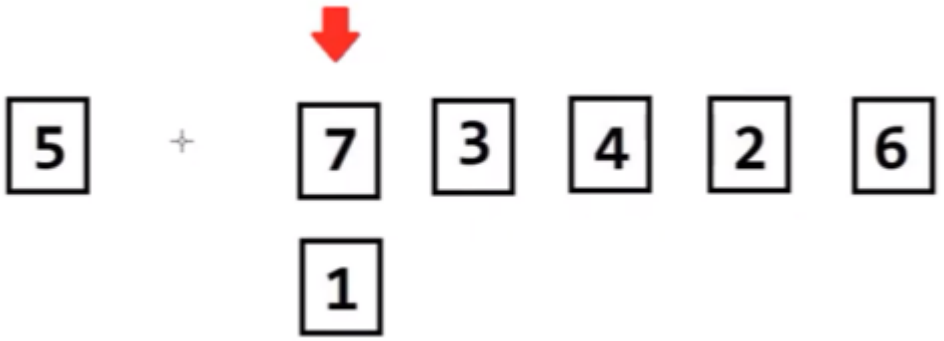
Tenemos este mazo de cartas:

Como se puede ver se encuentra desordenado, primero tenemos que apoyarnos sobre el índice 1, para así poder comparar si el índice 1, qué en este caso el el 7, es menor que el índice 0, que es el 5.

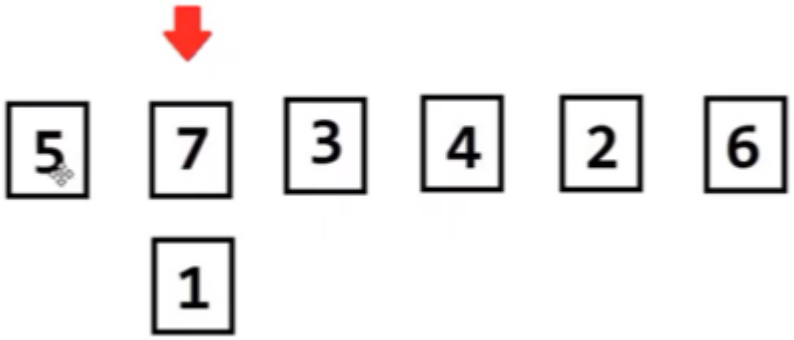
Como podemos observar el 7 es mayor que el 5, así que no hace falta modificar nada, por ende hasta este momento se encuentra todo ordenado, así que pasamos al próximo índice 2, que es el 1.



En este caso al comparar el 1 con el 7, podemos notar que el 1 es menor que el 7, así que se encuentra desordenado, para ordenarlo primero colocamos el 7 en el lugar del 1.

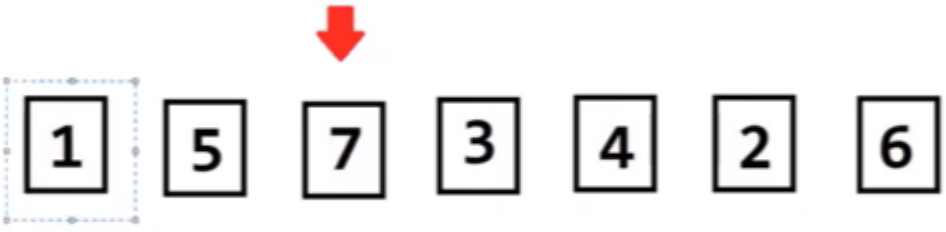


Como podemos ver, todavía hay un número más a la izquierda del 1, que es el 5, comparamos nuevamente para ver si el 1 es menor al 5 y resulta que lo es, por ende ponemos el número 5 donde iba el 7.

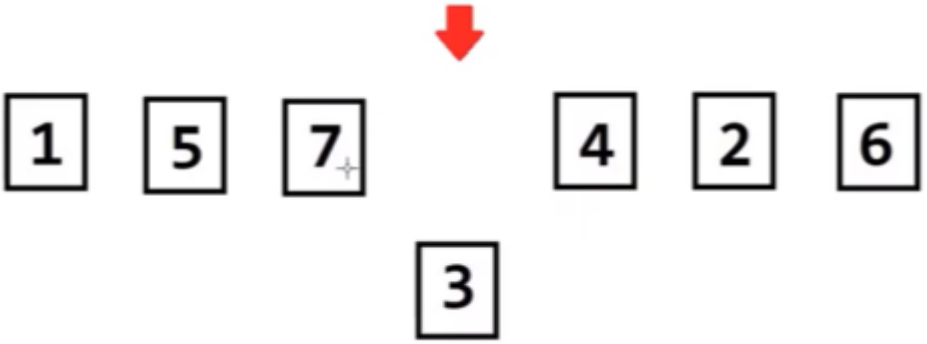


Ahora solo nos queda colocar el 1, como no tiene a nadie más a la izquierda, se convierte en el número más chico por ahora, ya que no se recorrió todo el vector.

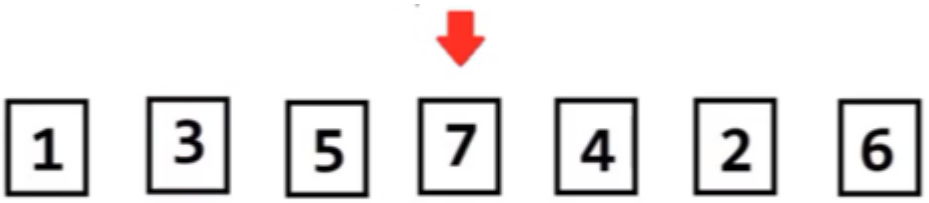
Quedaría así:



Ahora vamos a hacer algo que ya hicimos con anterioridad, ir al siguiente índice, que sería el 3.



Los pasos son exactamente los mismos que antes, pero qué pasa cuando el tres se encuentra en el índice uno?
Simplemente se compara el número 3 con el 1, y como el 3 es mayor que el 1, se queda en ese lugar.



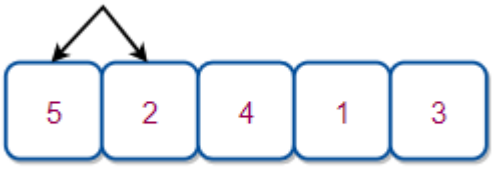
Se hace lo mismo con la lista entera hasta que se encuentre totalmente ordenada.

Complejidad computacional:

Este algoritmo requiere de $O(n^2)$ operaciones para ordenar un vector de N elementos, en el caso promedio.

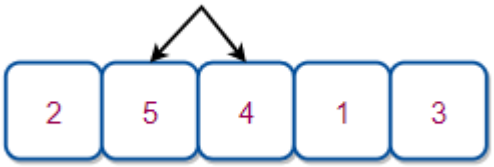
Ordenamiento de burbuja:

Se parte del siguiente arreglo:

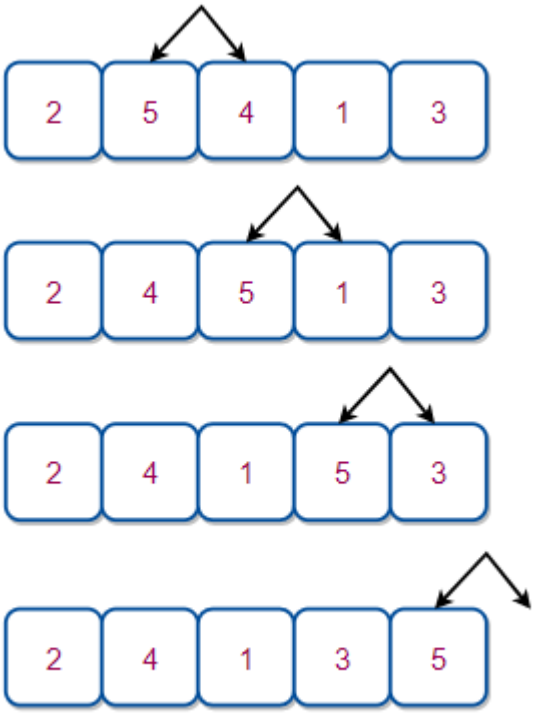


Con “burbuja” nos referimos a las dos flechas que apuntan a dos elementos contiguos en el arreglo.

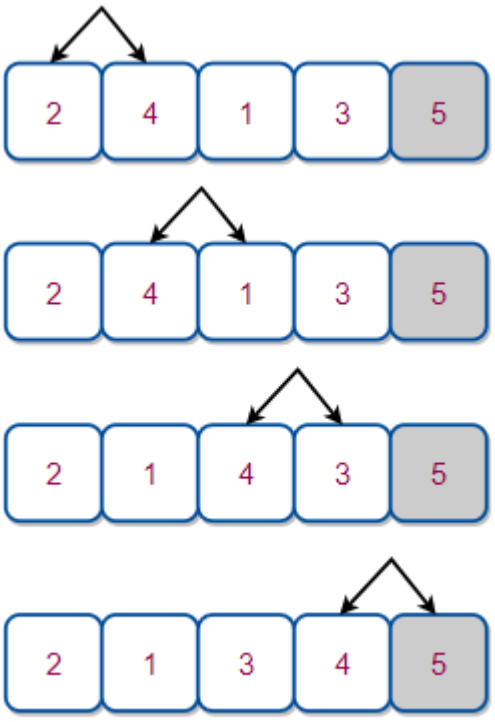
Comparamos los elementos de la burbuja y los intercambiamos según el criterio de ordenamiento, en este caso en orden ascendente. Luego desplazamos la burbuja una posición.



Repetimos la misma operación hasta llegar al final

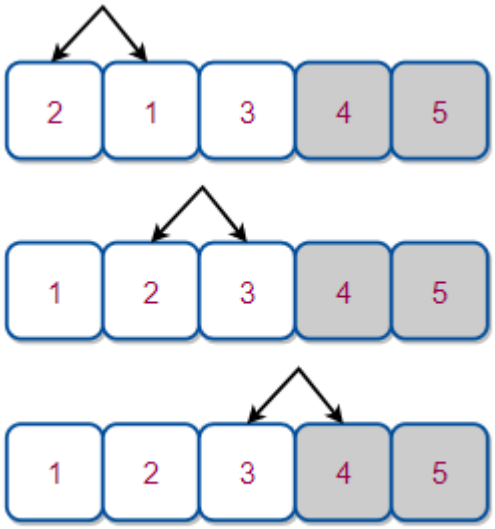


Una vez terminado el recorrido, llevamos la burbuja de nuevo al primer elemento y repetimos el proceso en esta segunda iteración. Pero como la iteración anterior nos dejó el elemento más grande al final, en esta recorreremos una posición menos que en la anterior.

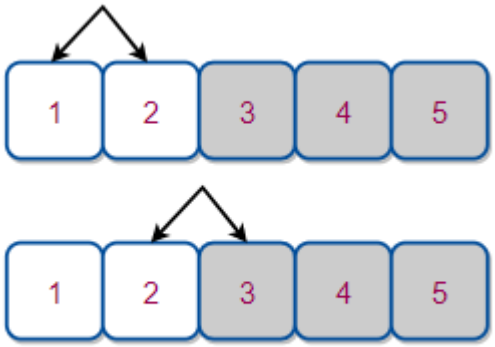


(Cuando la burbuja sea [4][5] no se considera el caso)

Lo mismo vuelve a suceder en esta iteración. Por ende repetimos hasta que solo tengamos que considerar los dos primeros elementos.



Aunque el arreglo ya esté ordenado, aún nos queda la última iteración:



Complejidad computacional:

Su complejidad es de $O(n^2)$ para el caso promedio.

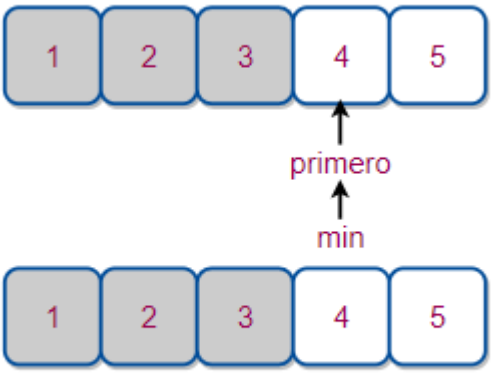
Ordenamiento por selección:

Dado un arreglo [5,3,4,2,1] a ordenar, recorremos el arreglo en busca del elemento mínimo mediante un sencillo algoritmo que guarda en memoria el valor del primer elemento y a medida que se recorre el arreglo lo intercambia si encuentra un menor valor.

Una vez encontrado el mínimo elemento, lo intercambiamos con el primero:

Repetimos el proceso en una segunda iteración, pero esta vez sin considerar el primer elemento del arreglo.

Seguimos hasta que nos quede considerar un solo elemento:



Complejidad computacional:

Su complejidad es de $O(n^2)$ para el caso promedio.

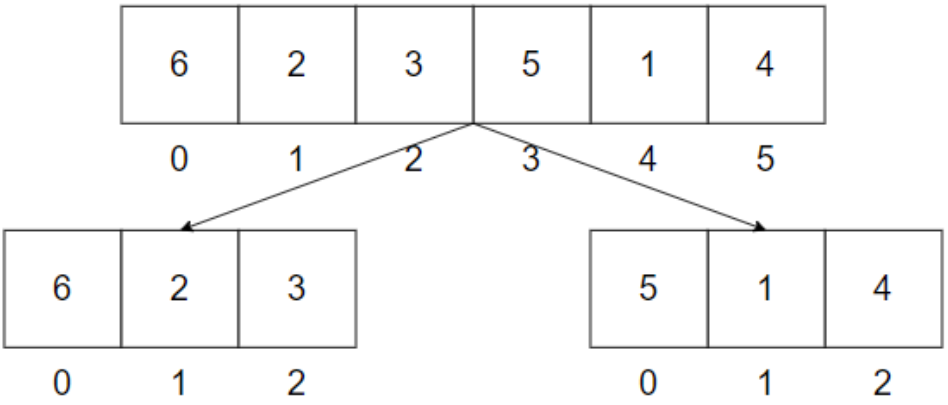
Ordenamiento por mezcla:

El ordenamiento por mezcla o *mergesort* consiste en dividir un arreglo de **n** elementos en dos partes aproximadamente iguales y repetir el proceso para cada subarreglo hasta obtener **n** subarreglos de un elemento. A partir de este punto, se realiza el proceso inverso de **mezclar** los subarreglos ordenando los elementos, en este caso de menor a mayor, hasta llegar al arreglo original ya ordenado.

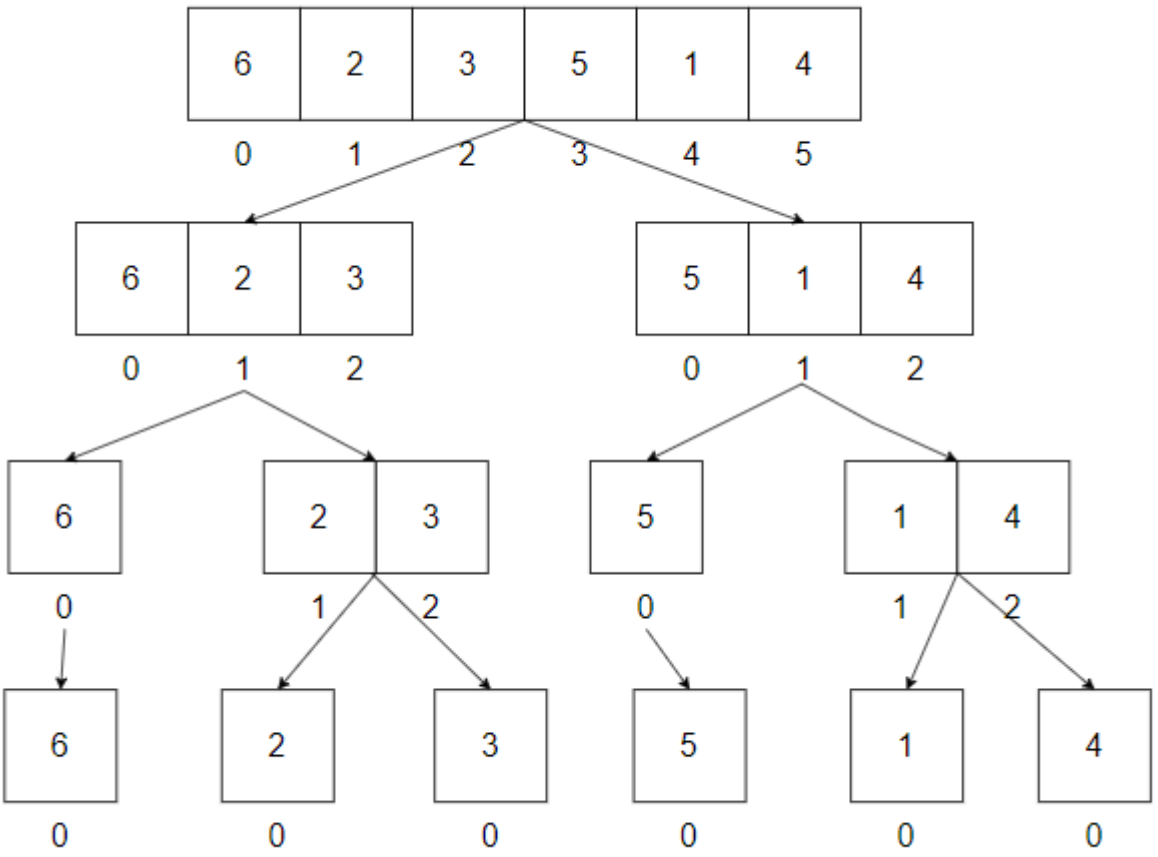
Por ejemplo, dado un arreglo desordenado de 6 elementos:

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

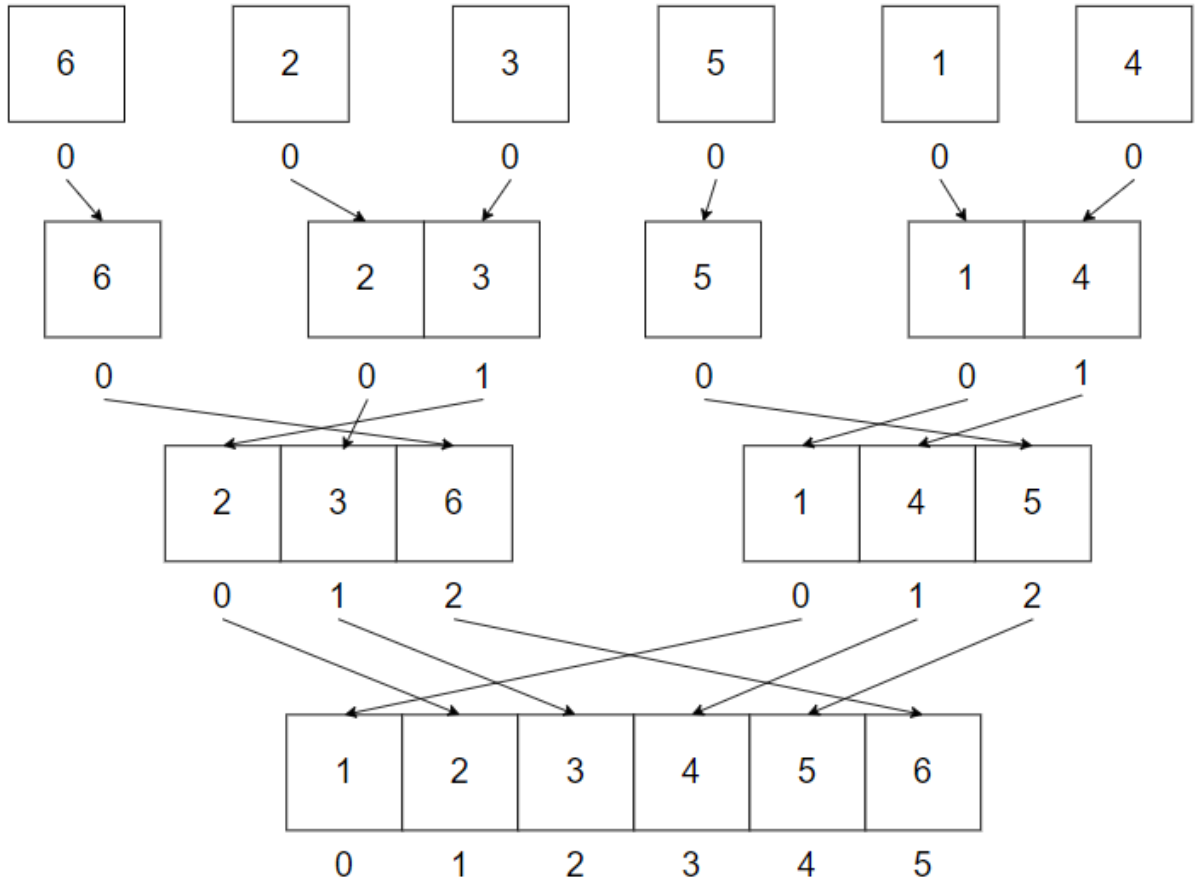
El punto de división es igual a N // 2, osea 6 // 2 = 3. Entonces obtenemos dos subarreglos arreglo[0:3] y arreglo[3:6]



Aplicando recursividad, repetimos el proceso hasta llegar al caso base de N arreglos de un elemento.



Una vez que estamos en el caso base, vamos comparando los elementos de los subarreglos recién divididos para identificar el menor de los dos y reescribir el arreglo de donde provienen.



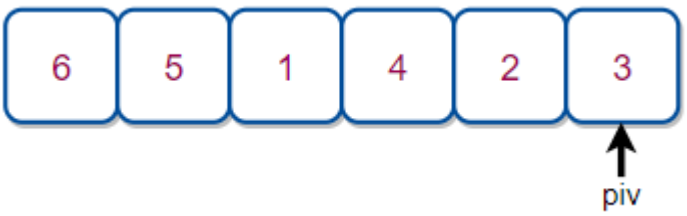
Complejidad computacional:

La complejidad del algoritmo es de O(n log n) en el caso medio.

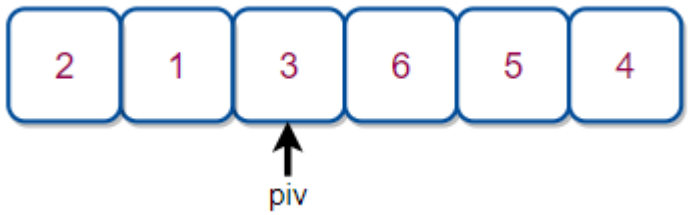
Ordenamiento rápido:

Al igual que el mergesort, el quicksort es un algoritmo recursivo de ordenamiento. También se divide en dos grandes pasos: dividir y combinar. Pero en el caso del quicksort, el ordenamiento se produce en el paso de dividir.

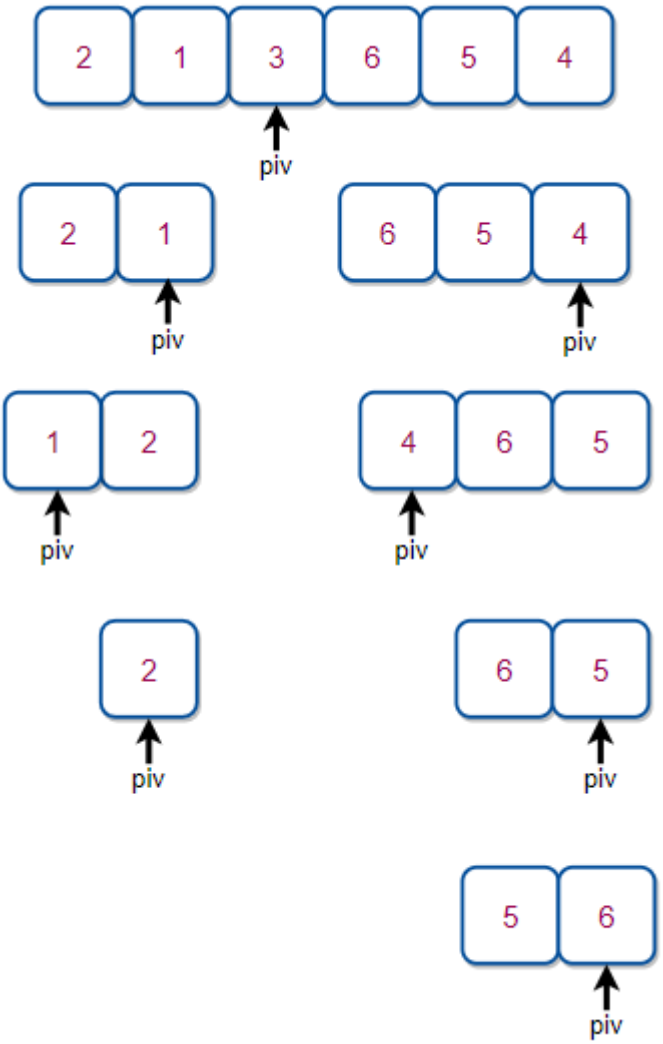
Por ejemplo, dado el siguiente arreglo:



Se elige un elemento cualquiera del arreglo como elemento pivote. En esta implementación, se elige siempre el último. El siguiente paso es, mediante un recorrido, reescribir todos los elementos del arreglo a los lados del pivote. Los que sean menores a la izquierda y los mayores a la derecha. Para ello se compara cada elemento con el pivote y se intercambian según el criterio de ordenación.



Ahora tenemos el elemento pivote en su lugar correspondiente. Luego, dividimos el arreglo en dos subarreglos, uno con los elementos a la izquierda y otro con los de la derecha, y los ordenamos recursivamente.



Como resultado final:

Complejidad computacional:

La complejidad del algoritmo es de O(n log n) en el caso promedio.

Algoritmos De Búsqueda

Búsqueda Secuencial:

Es el algoritmo de búsqueda más sencillo pero menos eficiente.

Dada una matriz y un elemento buscado , se recorre la matriz con un índice **i** para las filas y un índice **j** para las columnas. En cada iteración se lee el elemento en la posición **i**, **j** y se compara con el elemento buscado. Si es igual, la función devuelve los índices **i**, **j**. Si no es igual se continúa el recorrido.

Si se sale de los bucles sin haber devuelto nada, se devuelve -1, -1 y se interpreta que **e** no se encontró.

Complejidad computacional:

Su complejidad es de $O(n^2)$ operaciones para una matriz de n filas y n columnas.

Búsqueda Binaria por vector:

El algoritmo de búsqueda binaria, consta en calcular el centro del vector e ir dividiendo en partes hasta encontrar el dato que buscamos, en un vector ordenado.

Primero vemos el vector, vamos a buscar el número 16.

MIN = 0 CEN = 4 MAX = 7

Primero encontramos el centro, sumando el índice mínimo, con el índice máximo y después dividiéndolo por 2.

Ahora vemos si el centro es el dato que buscamos, como no lo es, vemos si el número del centro es mayor o menor que el dato que buscamos.
Si es menor, entonces nos vamos a fijar la parte izquierda de la lista solamente, ya que sabemos que el dato va a estar de ese lado, si es mayor, nos fijamos la parte de la derecha.

El 10 es menor que el 16, por ende sabemos que el 16 va a estar a la derecha.
Así que para hacer esto, el índice mínimo pasa a ser el centro más uno. y calculamos nuevamente el centro como lo hicimos anteriormente.

MIN=5 CENTRO=6 MAX=7

Ahora hacemos lo mismo, el mínimo pasa a ser el centro más uno, ya que el 14 es menor que el 16, y como podemos ver, el centro pasa a ser el 16.
Nuevamente vemos si el centro es el dato que buscamos y llegamos al caso base.

MIN=7
MAX=7
CENTRO=7

Complejidad computacional:

Este algoritmo requiere de $O(\log(n))$ operaciones para un vector de N elementos en el caso promedio.

Búsqueda Binaria por matriz:

En este caso, es una idea parecida, teniendo una matriz buscamos el medio de la misma, el caso base es cuando la fila del centro es la que tiene el dato que buscamos, en otro caso, si el min de la fila del centro es mayor al dato que buscamos, entonces “eliminamos” la parte inferior de la matriz, en el otro caso, si el min de la fila del centro es menor al dato que buscamos, eliminamos la parte superior de la matriz, con una lógica casi igual a la búsqueda binaria por vector.
Cuando lleguemos a la fila donde sabemos que está el dato que buscamos, aplicamos la búsqueda binaria por vector.

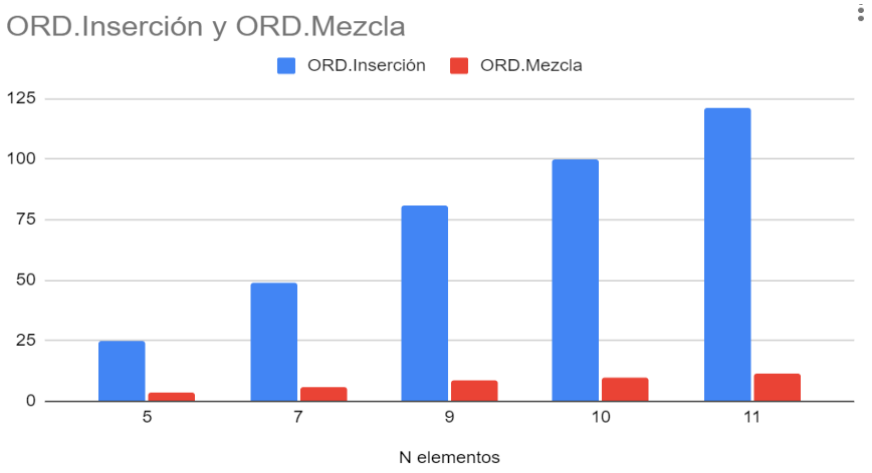
Complejidad computacional:

Este algoritmo requiere de $O(\log(n) * \log(n))$ operaciones para una matriz de N filas y N columnas.

COMPARACIÓN DE COMPLEJIDAD ENTRE LOS ALGORITMOS.

Ordenamiento por Inserción y ordenamiento de Mezcla

N elementos	ORD.Inserción	ORD.Mezcla
5	25	3,494850022
7	49	5,91568628
9	81	8,588182585
10	100	10
11	121	11,45531954



Búsqueda secuencial y búsqueda binaria

OPERACIONES	BUS.Secuencial	BUS.Binaria
5	25	0,488559067
7	49	0,7141906972
9	81	0,9105787668
10	100	1
11	121	1,084498725