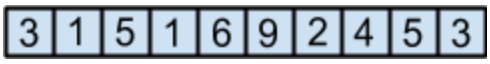
Algoritmos recursivos de ordenamiento

Aunque actualmente existen varios algoritmos recursivos de ordenamiento y varias versiones de cada uno, nosotros sólo estudiaremos las versiones básicas de dos de ellos: Quick sort y Merge sort. Estos algoritmos son, por mucho, los más utilizados hoy en día.

Quick sort

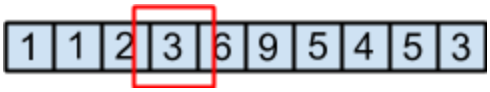
En esencia, quick sort ordena un arreglo a[1], a[2], …, a[n] seleccionando el valor v de una llave en el arreglo como el elemento *pivote*, alrededor del cual se organizarán el resto de los elementos en el arreglo. Idealmente, el pivote estará cerca al valor medio de las llaves en el arreglo, tal que sea precedido y seguido por aproximadamente la mitad de los registros. Se intercambian los elementos en el arreglo de tal forma que para alguna j, todos los registros con llaves menores a v aparezcan en a[1], a[2], …, a[j-1] y todos los registros con llaves mayores o iguales a v se ubiquen en a[j+1], a[j+2], …, a[n]. Entonces quick sort se aplica recursivamente a a[1], a[2], …, a[j-1] y a a[j+1], a[j+2], …, a[n] para ordenar ambos subarreglos.Debido a que las llaves de los registros en el primer grupo preceden a las llaves de los registros en el segundo grupo, el arreglo en su totalidad estará ordenado.

Veamos una demostración del funcionamiento de quick sort. Consideremos un arreglo a como el que se muestra a continuación:



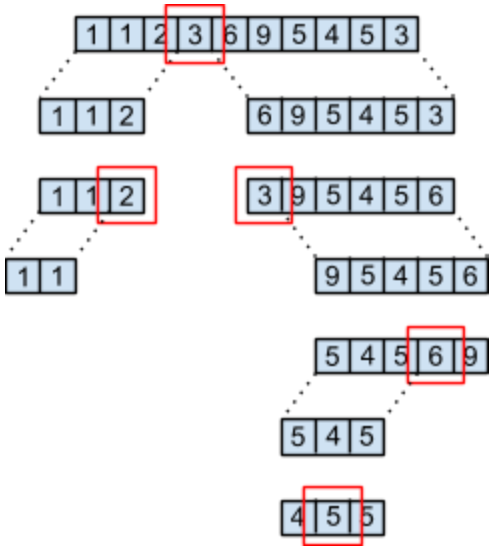
El primer paso de este algoritmo es definir una posición cuyo valor servirá como pivote. Nosotros utilizaremos la posición n del arreglo, es decir la última posición. En este ejemplo, la última posición contiene un 3.

El siguiente paso se refiere a agrupar a todos aquellos números menores al pivote en las primeras j-1 posiciones y los números mayores o iguales al pivote se agrupan a partir de las posición j+1. La siguiente figura ilustra este caso.

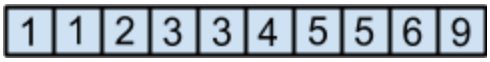


En este momento, tenemos que los números menores a 3 están ubicados antes de la cuarta posición y los mayores o iguales a 3 a partir de la quinta posición.

El siguiente, es el paso recursivo, en donde el algoritmo se aplica tanto para el subarreglo definido por los índices 1 y 3, como para el subarreglo definido por los índices 5 y 10. Si hacemos las llamadas recursivas sobre los subarreglos subsecuentes hasta que su tamaño sea 1, tenemos el siguiente resultado:



Y así, finalmente, obtenemos el siguiente arreglo ordenado:



El siguiente es el algoritmo del método de ordenamiento quick sort. Nótese que este método está compuesto por dos funciones: una que es la función recursiva quick sort y otra que hacer la partición del arreglo.

```
función quickSort (RegistroT *a, entero p, entero r)
comienza
      si p < r entonces
            q ← particionar (a, p, r)
            quickSort (a, p, q-1)
            quickSort (a, q+1, r)
      fin si
fin

función particionar (RegistroT *a, entero p, entero r)
      tipoLlave pivote
      entero último, i
comienza
      // a[r] es el registro elegido cuya llave será el pivote
      pivote ← a[r].llave

      // indica el índice del último registro cuya llave es menor a x
      último ← p - 1
```

```
      // Se agrupan los elementos menores a pivote en las primeras
      // posiciones del arreglo a
      para i ← p hasta r − 1 hacer
            si a[i].llave < pivote  entonces
                  último ← último + 1
                  intercambiar (a[último], a[i])
            fin si
      fin para

      // Y se mueve al pivote a su posición correcta
      intercambiar (a[último+1], a[r])
      regresar último+1
fin
```

Regresemos al ejemplo anterior para analizar la ejecución de quick sort con detalle. Originalmente tenemos el arreglo

```
a = {3, 1, 5, 1, 6, 9, 2, 4, 5, 3}
```

La primera llamada a la función quick sort, enviará quickSort (a, 1, 10), es decir, p = 1 y r = 10.

```
a = {3, 1, 5, 1, 6, 9, 2, 4, 5, 3}
     ↑                          ↑
     p                          r
```

Como p < r, procedemos a hacer la partición del arreglo. En este caso, pivote toma el valor de 3 y último se inicia en 0. Recordemos que las llaves de todos los registros en las posiciones 1 a último son menores a pivote.

Se comienza el ciclo que recorrerá el arreglo desde la posición 1 hasta la 9, verificando si la llave el i-ésimo registro es menor a pivote. En nuestro ejemplo, la llave del registro en i = 2 es 1, que es menor a 3, el pivote. Aquí se incrementa último, que ahora vale 1, y se intercambian los registros en las posiciones 1 (último) y 2 (i-ésima).

```
a = {1, 3, 5, 1, 6, 9, 2, 4, 5, 3}
     ↑  ↑
     ú  i
```

Continuamos recorriendo el arreglo hasta la posición 4, en donde encontramos otro registro cuya llave 1 es menor a 3 (pivote). Nuevamente se incrementa último, ahora vale 2, y se intercambian los registros en las posiciones 2 (último) y 4 (i-ésima).

```
a = {1, 1, 5, 3, 6, 9, 2, 4, 5, 3}
         ↑     ↑
         ú     i
```

Se sigue con el recorrido hasta la posición 7, en donde la llave 2 es menor a 3 (pivote). Se incrementa último a 3 y se intercambian los registros 3 y 7.

```
a = {1, 1, 2, 3, 6, 9, 5, 4, 5, 3}
            ↑           ↑
            ú           i
```

El recorrido se termina en la posición 9 y no se encuentran más registros con llave menor a 3. Entonces se intercambian los registros 10, que contiene el pivote, y el posterior a último.

```
a = {1, 1, 2, 3, 6, 9, 5, 4, 5, 3}
               ↑                 ↑
             ú+1                 r
```

Se hace una llamada recursiva para cada subarreglo: de la posición 1 a la 3 y de la posición 5 a la 10. La posición 4 ya no se incluye puesto que con el recorrido anterior garantizamos que ese registro está en su ubicación correcta.

```
a = {1, 1, 2}, 3, {6, 9, 5, 4, 5, 3}
     ↑     ↑       ↑              ↑
     p     r       p              r
```

Recordemos que la llave del registro en la última posición r de cada subarreglo se considerará el pivote. Al recorrer el subarreglo izquierdo nos damos cuenta que las llaves de los registros 1 y 2 son menores a 2, el pivote, por lo tanto el intercambio se realiza en la misma posición, es decir, no hay cambios.

```
a = {1, 1, 2}, 3, {6, 9, 5, 4, 5, 3}
     ↑
    ú,i
```

```
a = {1, 1, 2}, 3, {6, 9, 5, 4, 5, 3}
        ↑
       ú,i
```

Finalmente, el registro que contiene el pivote, r, se intercambio con el posterior a último. En nuestro ejemplo, el subarreglo no sufrió modificaciones después de esta operación. La siguiente llamada recursiva envía 1 y 2, como p y r respectivamente. Y el 1 será considerado el pivote. Dado que el otro registro también tiene llave 1, este subarreglo no sufrirá cambios y al

hacer una nueva llamada recursiva, p y r tendrán el mismo valor, 1, lo que implica el fin de la recursión para el subarreglo inicial.

Volvamos ahora al subarreglo definido por las posiciones 5 y 10. En este caso, el pivote es 3, correspondiente a la llave del registro en la posición r. Al hacer el recorrido, no se encuentran registros con llaves menores a 3, por lo que el recorrido termina y como paso final se intercambian los registros en las posiciones último+1 y r, es decir, 5 (porque último se inicia en 4) y 10.

```
a = 1, 1, 2, 3, {3, 9, 5, 4, 5, 6}
                  ↑              ↑
                 ú+1             r
```

Se hace una llamada recursiva para el subarreglo delimitado por las posiciones 6 y 10. Ahora el pivote es 6.

```
a = 1, 1, 2, 3, 3, {9, 5, 4, 5, 6}
                     ↑           ↑
                     p           r
```

Iniciamos el recorrido del subarreglo y en la posición 7 encontramos una llave menor a 6, que es 5. Entonces este registro se intercambia por el ubicado en último, que ahora vale 6.

```
a = 1, 1, 2, 3, 3, {5, 9, 4, 5, 6}
                     ↑  ↑
                     ú  i
```

Nuevamente, en las posiciones 8 y 9, existen registros cuyas llaves son menores a 6, entonces éstos se intercambian con el registro en la posición último.

```
a = 1, 1, 2, 3, 3, {5, 4, 9, 5, 6}
                        ↑  ↑
                        ú  i

a = 1, 1, 2, 3, 3, {5, 4, 5, 9, 6}
                           ↑  ↑
                           ú  i
```

Finalmente, se intercambian los registros 9 y 10.

```
a = 1, 1, 2, 3, 3, {5, 4, 5, 6, 9}
                              ↑  ↑
                            ú+1  r
```

Aquí se hacen dos llamadas recursivas, una para el subarreglo definido por las posiciones 6 y 8 y otra para el subarreglo delimitado por la posición 10. En este último caso, dado que p y r valen 10, se termina la recursión. Enfoquémonos entonces en el primer caso.

```
a = 1, 1, 2, 3, 3, {5, 4, 5}, 6, 9
                     ↑     ↑
                     p     r
```

El valor del pivote es 5 y, al recorrer el subarreglo, vemos que en la posición 7 hay un registro cuya llave es 4, por lo tanto se intercambia con el registro en la posición 6 (último).

```
a = 1, 1, 2, 3, 3, {4, 5, 5}, 6, 9
                     ↑  ↑
                     ú  i
```

Termina el recorrido y se hacen llamadas recursivas para subarreglos con un solo registro, lo que provoca que finalice la recursión. Cuando las llamadas recursivas regresan a la llamada original, vemos que el arreglo está ordenado:

```
a = {1, 1, 2, 3, 3, 4, 5, 5, 6, 9}
```

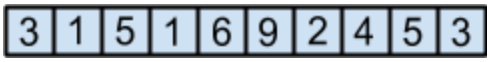
Tarea: Aplicar el algoritmo quick sort para ordenar los siguientes arreglos.

1. a = {fresa, uva, naranja, papaya, melón, sandía, zarzamora, frambuesa, manzana, pera}
2. a = { $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, π, e, $log_2 3$, $2cos(1)$, $ln\ 2$, $\frac{1+\sqrt{5}}{2}$, $\sqrt[3]{2}$ }

Merge sort

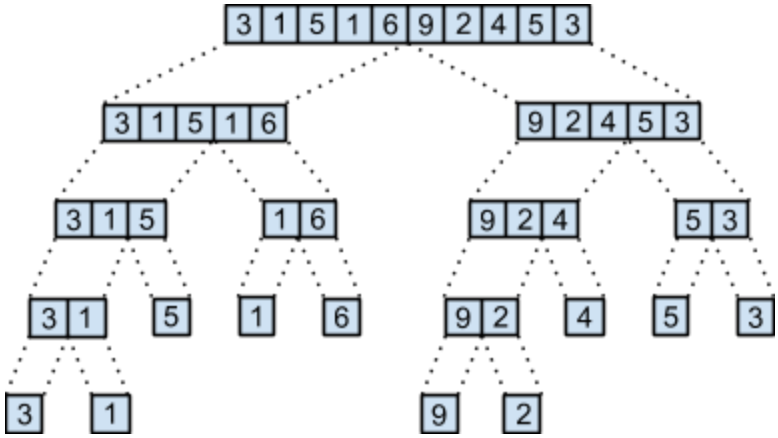
A diferencia de quick sort, merge sort utiliza la técnica divide y vencerás para ordenar un arreglo de registros a[1], a[2], …, a[n]. El arreglo es dividido en dos subarreglos de tamaño similar a[1], a[2], …, a[n/2] y a[n/2+1], a[n/2+2], …, a[n], lo que contrasta con la partición que hace quick sort. A continuación, se ordena recursivamente cada uno de los subarreglos por separado. Las llamadas recursivas se detienen en cuanto el tamaño del subarreglo es 1 y es precisamente cuando el subarreglo está ordenado. Posteriormente, para reconstruir el arreglo original de tamaño n, se fusionan los dos subarreglos ordenados.

Veamos una demostración del funcionamiento de merge sort. Consideremos nuevamente el arreglo ejemplo de quick sort:

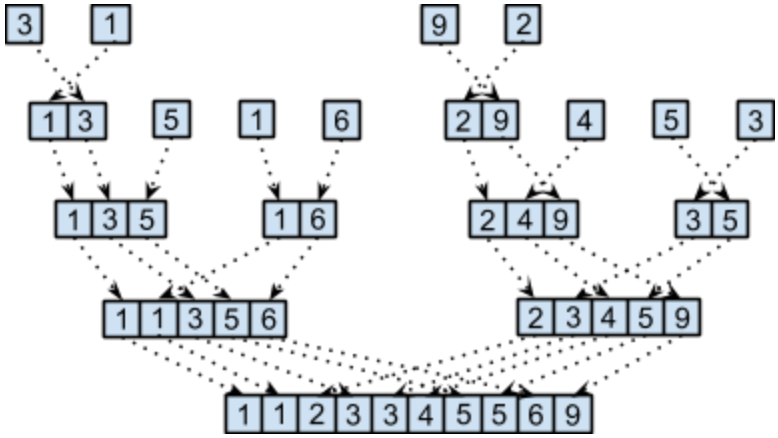


El arreglo se divide en dos y se hace la llamada recursiva sobre las dos mitades:

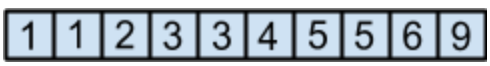
Eventualmente vamos a obtener las siguientes subdivisiones:



Cuando tenemos que las divisiones han dado como resultado subarreglos de tamaño 1, comenzamos a combinar los elementos de estos subarreglos de forma ordenada como se muestra a continuación:



Y así, finalmente, obtenemos el siguiente arreglo ordenado:



El siguiente es el algoritmo del método de ordenamiento merge sort. Este método, al igual que quick sort, está compuesto por dos funciones: una que es la función recursiva merge sort y otra

que hace la combinación de los subarreglos.

```
función mergeSort (RegistroT *a, entero inicio, entero fin)
comienza
      si inicio < fin entonces
            medio ← (inicio + fin)/2
            mergeSort (a, inicio, medio)
            mergeSort (a, medio+1, fin)
            combinar (a, inicio, medio, fin)
      fin si
fin

función combinar (RegistroT *a, entero inicio, entero medio,
                  entero fin)
      RegistroT aux[fin - inicio + 1]
      entero h, i, j, k
comienza
      h ← inicio      // Índice del arreglo auxiliar
      i ← inicio      // Índice de la primera mitad
      j ← medio + 1   // Índice de la segunda mitad

      // Se recorren ambos subarreglos y se van combinando sus
      // elementos ordenadamente hasta que se haya recorrido
      // totalmente uno de ellos
      mientras (i ≤ medio) ⋀ (j ≤ fin) hacer
            si a[i].llave ≤ a[j].llave entonces
                  aux[h] ← a[i]
                  i ← i+1
            si no
                  aux[h] ← a[j]
                  j ← j+1
            fin si
            h ← h+1
      fin mientras

      // Si se copiaron todos los elementos de la primera mitad,
      // se copia el resto de la segunda mitad
      si i > medio entonces
            para k ← j hasta fin hacer
                  aux[h] ← a[k]
                  h ← h+1
            fin para
```

```
      // Si no, se copia el resto de la primera mitad
      si no
            para k ← i hasta medio hacer
                  aux[h] ← a[k]
                  h ← h+1
            fin para
      fin si

      // Finalmente, se copia el contenido del arreglo auxiliar
      para k ← inicio hasta fin hacer
            a[k] ← aux[k]
      fin para
fin
```

Regresemos a nuestro ejemplo y analicemos la ejecución de merge sort detenidamente. El arreglo original es

```
a = {3, 1, 5, 1, 6, 9, 2, 4, 5, 3}
```

La primera llamada a la función merge sort, enviará mergeSort (a, 1, 10), es decir, inicio = 1 y fin = 10 y la posición medio será 5.

```
a = {3, 1, 5, 1, 6, 9, 2, 4, 5, 3}
     ↑           ↑              ↑
     i           m              f
```

Entonces la primera llamada recursiva considera la primera mitad del arreglo, es decir inicio = 1 y fin = 5, entonces medio será 3.

```
a = {3, 1, 5, 1, 6}, 9, 2, 4, 5, 3
     ↑     ↑     ↑
     i     m     f
```

Nuevamente consideramos la primera mitad del subarreglo y eventualmente obtendremos las siguientes llamadas recursivas

```
a = {3, 1, 5}, 1, 6, 9, 2, 4, 5, 3
     ↑  ↑  ↑
     i  m  f

a = {3, 1}, 5, 1, 6, 9, 2, 4, 5, 3
     ↑  ↑
    i,m f
```

En este momento, se hacen nuevas llamadas recursivas, pero el tamaño de los subarreglos es 1, entonces se combinan los elementos de forma ordenada

```
a = {1, 3}, 5, 1, 6, 9, 2, 4, 5, 3
     ↑  ↑
     i  f
```

Ya que está ordenada esta sección del subarreglo, se hacen llamadas recursivas sobre la segunda mitad

```
a = 3, 1, {5}, 1, 6, 9, 2, 4, 5, 3
           ↑
         i,m,f
```

Como el tamaño del subarreglo es 1, queda intacto y se combina con los elementos de la primera mitad

```
a = {1, 3, 5}, 1, 6, 9, 2, 4, 5, 3
     ↑     ↑
     i     f
```

Se hace la llamada recursiva sobre la segunda mitad del subarreglo y después se combinan los elementos con los de la primera mitad

```
a = 1, 3, 5, {1, 6}, 9, 2, 4, 5, 3
              ↑  ↑
             i,m f
```

```
a = {1, 1, 3, 5, 6}, 9, 2, 4, 5, 3
     ↑           ↑
     i           f
```

En este momento, la primera mitad del arreglo original está ordenada, ahora se hacen las llamadas recursivas sobre la segunda mitad. En este caso, inicio = 6 y final = 10, entonces la posicion medio = 8

```
a = 1, 1, 3, 5, 6, {9, 2, 4, 5, 3}
                    ↑     ↑     ↑
                    i     m     f
```

De aquí se hacen nuevas llamadas recursivas, y eventualmente se tendrán los siguientes resultados sobre la primera mitad del subarreglo

```
a = 1, 1, 3, 5, 6, {9, 2, 4}, 5, 3
                    ↑  ↑  ↑
                    i  m  f

a = 1, 1, 3, 5, 6, {9, 2}, 4, 5, 3
                    ↑  ↑
                   i,m f

a = 1, 1, 3, 5, 6, {2, 9}, 4, 5, 3
                    ↑  ↑
                    i  f

a = 1, 1, 3, 5, 6, 2, 9, {4}, 5, 3
                          ↑
                       i,m,f

a = 1, 1, 3, 5, 6, {2, 4, 9}, 5, 3
                    ↑     ↑
                    i     f
```

En este momento, la primera mitad del subarreglo está ordenada, entonces se hacen llamadas recursivas sobre la segunda mitad del subarreglo

```
a = 1, 1, 3, 5, 6, 2, 4, 9, {5, 3}
                             ↑  ↑
                            i,m f

a = 1, 1, 3, 5, 6, 2, 4, 9, {3, 5}
                             ↑  ↑
                             i  f

a = 1, 1, 3, 5, 6, {2, 3, 4, 5, 9}
                    ↑           ↑
                    i           f
```

Aquí la segunda mitad del arreglo original está ordenada, entonces se procede a combinar sus elementos con los de la primera mitad, que también ya están ordenados, para obtener el arreglo original ordenado

```
a = {1, 1, 2, 3, 3, 4, 5, 5, 6, 9}
     ↑                          ↑
     i                          f
```

En este momento, termina la recursión y finalmente obtenemos el arreglo

```
a = {1, 1, 2, 3, 3, 4, 5, 5, 6, 9}
```

Tarea: Aplicar el algoritmo merge sort para ordenar los siguientes arreglos.

1. a = {fresa, uva, naranja, papaya, melón, sandía, zarzamora, frambuesa, manzana, pera}
2. a = { $\sqrt{2},\ \sqrt{3},\ \sqrt{5},\ \pi,\ e,\ log_2 3,\ 2cos(1), ln\ 2,\ \frac{1+\sqrt{5}}{2},\ \sqrt[3]{2}$ }