*REPORTE: ALGORITMOS DE ORDENAMIENTO*

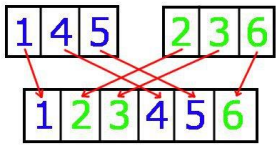
*Luis Daniel Honorato Hernández*

*Licenciatura en Matemáticas*

*1729448*

**Algoritmo Merge**

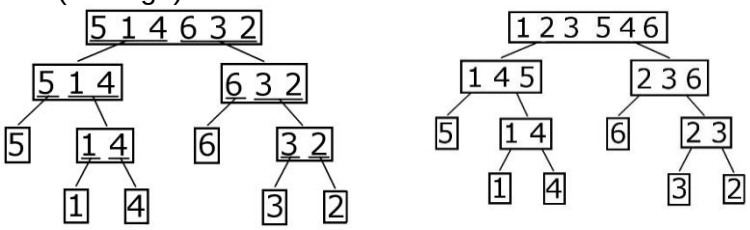
Es un método de ordenamiento por mezcla, la idea general es que: Dados dos conjuntos ordenados, A y B, si los mezclamos entre ambos, tomando los valores de ellos en orden, entonces nos va quedar el conjunto ordenado C, con los elementos de A y B.




Partiendo de este punto, aplicaremos la estrategia Divide y Vencerás. La idea es, que, si inicialmente tenemos la lista desordenada, y la dividimos a la mitad, nos quedaremos con 2 sub-listas desordenadas, entonces, realizamos otra vez la misma acción: dividimos las sub-listas resultantes en 4 nuevas sub-listas, y así sucesivamente.

Esta operación se realizará hasta que lleguemos a una sub-lista con 1 o 0 elementos en ella, que por defecto va a estar ordenada, y como dicha sub-lista ya está ordenada, la mezclamos con la de al lado, que está ordenada también, y así continuamente vamos ordenando las sub-listas hacia arriba para llegar al caso base.

Si representamos esas operaciones en una imagen (ver Fig2), puedes notar que se forma un árbol, donde, partiendo de las hojas (son los elementos terminales del árbol), se van a ordenar los nodos padres, por medio de la mezcla entre sus dos hijos, hasta llegar al nodo raíz (ver Fig3).




Fue desarrollado en 1945 por John Von Neumann

Conceptualmente el ordenamiento de mezcla funciona de la siguiente manera

> Si la longitud de la lista es 0 o 1 entonces ya está ordenada sin embargo en otro caso:

>Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño

>Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla

>Mezclar las dos sublistas en una sola lista ordenada

El pseudocódigo del algoritmo sería:

```
status intercalar (Tabla una, Tabla otra,
                   Tabla final, dim NU, dim NO)
```

```
dim NF = NU + NO;
iuno = iotro = ifinal = 0;
mientras (iuno <= NU y iotro <= NO)
  if uno[iuno] < otro[iotro]
    final[ifinal++] = uno[iuno++]
  else
    final[ifinal++] = otro[iotro++]
//
mientras (iuno <= NU)
   final[ifinal++] = uno[iuno++]
mientras (iotro <= NO)
   final[ifinal++] = otro[iotro++]
return ok;
```

Si se contempla el caso en el que puede haber elementos repetidos, hay que modificar ligeramente el pseudocódigo.

**Algoritmo Quicksort**

Desde que existe la ciencia de la computación, uno de los mayores problemas con los que los ingenieros se encontraban en su día a día, era el de ordenar listas de elementos. Por su causa, diversos algoritmos de ordenación fueron desarrollados a lo largo de los años y siempre existió un intenso debate entre los desarrolladores sobre cuál de todos los algoritmos de ordenación era el más rápido.

El debate finalizó abruptamente en 1960 cuando Sir Charles Antony Richard Hoare, nativo de Sri Lanka y ganador del premio Turing en 1980, desarrolló el algoritmo de ordenación Quicksort casi por casualidad mientras ideaba la forma de facilitar la búsqueda de palabras en el diccionario.

El algoritmo Quicksort se basa en la técnica de "divide y vencerás" por la que, en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.

En la práctica, es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O (n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable. El hecho de que sea más rápido que otros algoritmos de ordenación con tiempo promedio de O (n log (n)) (como SmoothSort o HeapSort) viene dado por que Quicksort realiza menos operaciones ya que el método utilizado es el de partición

Los pasos que realiza este algoritmo son:

Selecciona un valor del arreglo como pivote es decir un numero por el cual todos los elementos van a ser comparados.

Se realizan dos búsquedas: una de izquierda a derecha, buscando un elemento mayor que el pivote, y otra de derecha a izquierda, buscando un elemento menor que el pivote.

Cuando se han encontrado los dos, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran.

Luego se organizan los subarreglos que quedaron a mano derecha y izquierda.

Ejemplo:

Tenemos un arreglo que está definido con los valores {22,40,4,10,12,35} los pasos en Quicksort para arreglarlo son los siguientes:

Se toma como pivote el numero 22 recuerden puede ser cualquier número.

La búsqueda de izquierda a derecha encuentra el valor 40 que es mayor a pivote y la búsqueda de derecha a izquierda encuentra el valor 35 no lo toma porque es mayor a el numero pivote (recuerden que la búsqueda de derecha a izquierda busca los menores) entonces continua y encuentra a 12 que es menor que el pivote y se intercambian el resultado sería {21,12,4,10,40,35}

Si seguimos la búsqueda la primera encuentra el valor 40, y la segunda el valor 10, pero ya se han cruzado, así que paramos. Para terminar la división, se coloca el pivote en su lugar el numero encontrado por la segunda búsqueda, el 10 quedando: {10,12,4,22,40,35}

Ahora tenemos dividido el arreglo en dos arreglos más pequeños que son {10,12,4} y el {40,35}, y en ellos se repetirá el mismo proceso.

```
// Inicialización de variables
1. elem_div = lista[sup];
2. i = inf - 1;
3. j = sup;
4. cont = 1;

// Verificamos que no se crucen los límites
5. if (inf >= sup)
6.     retornar;
```

```
//  Clasificamos la sublista
7. while (cont)
8.      while (lista[++i] < elem_div);
9.      while (lista[--j] > elem_div);
10.      if (i < j)
11.           temp = lista[i];
12.           lista[i] = lista[j];
13.           lista[j] = temp;
14.      else
15.           cont = 0;

// Copiamos el elemento de división
// en su posición final
16. temp = lista[i];
17. lista[i] = lista[sup];
18. lista[sup] = temp;

// Aplicamos el procedimiento
// recursivamente a cada sublista
19. OrdRap (lista, inf, i - 1);
20. OrdRap (lista, i + 1, sup);
```

**Algoritmo Bubble**

Es el algoritmo de ordenamiento más sencillo de todos, conocido también como método del intercambio directo, el funcionamiento se basa en la revisión de cada elemento de la lista que va a ser ordenada con el elemento siguiente, intercambiando sus posiciones si están en el orden equivocado, para esto se requieren varias revisiones hasta que ya no se necesiten más intercambios, lo que indica que la lista ha sido ordenada.

El origen del nombre de este algoritmo proviene de la forma con la que suben por la lista los elementos durante los intercambios, tal y como si fueran "burbujas", el algoritmo fundamental de este método es la simple comparación de elementos siendo así el más fácil de implementar.

la siguiente, es una implementación en Pseudocódigo, donde A es un arreglo de N elementos

BURBUJA (A, N)

{I, J y AUX son variables de tipo entero}

1. Repetir con I desde 2 hasta N

    1.1 Repetir con J desde N hasta I

        1.1.1 Si A[J-1] > A[J] entonces

            Hacer AUX? A[J-1]

            A[J-1]? A[J]

            A[J]? AUX

        1.1.2 {Fin del condicional del paso 1.1.1}

    1.2 {Fin del ciclo del paso 1.1}

2. {Fin del ciclo del paso 1

**Algoritmo Selection**

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo

Explicación:
Su funcionamiento es el siguiente:

Buscar el mínimo o máximo elemento de la lista

Intercambiarlo con el primero

Buscar el mínimo o el máximo en el resto de la lista

Intercambiarlo con el segundo

Y en general:

Buscar el mínimo o máximo elemento entre una posición i y el final de la lista

Intercambiar el mínimo o máximo con el elemento de la posición i.

1 función Seleccion (V: vector de natural; n: natural) devuelve vector de natural

variable i,j,menor : natural ;

para i ← 1 hasta n - 1 hacer

```
        menor ← i;
        para j ← i + 1 hasta n hacer
            si (V[j] < V[menor]) entonces
                menor ← j;
            fsi
        fpara
        Intercambiar (V, i, menor);
fpara
devuelve V;
13 ffuncion
```