Ordenamiento avanzado

MergeSort – QuickSort - HeapSort

Merge Sort

Sigue el enfoque divide y vencerás.

Se basta en el algoritmo de mezcla

Pasos:

1. El arreglo original se divide en mitades recursivamente hasta que queden subarreglos de 1 sólo elemento.
2. Los subarreglos individuales se combinan en pares y se ordenan mediante el uso del algoritmo de mezcla.
3. Se obtiene un arreglo ordenado con la fusión de todos los subarreglos.

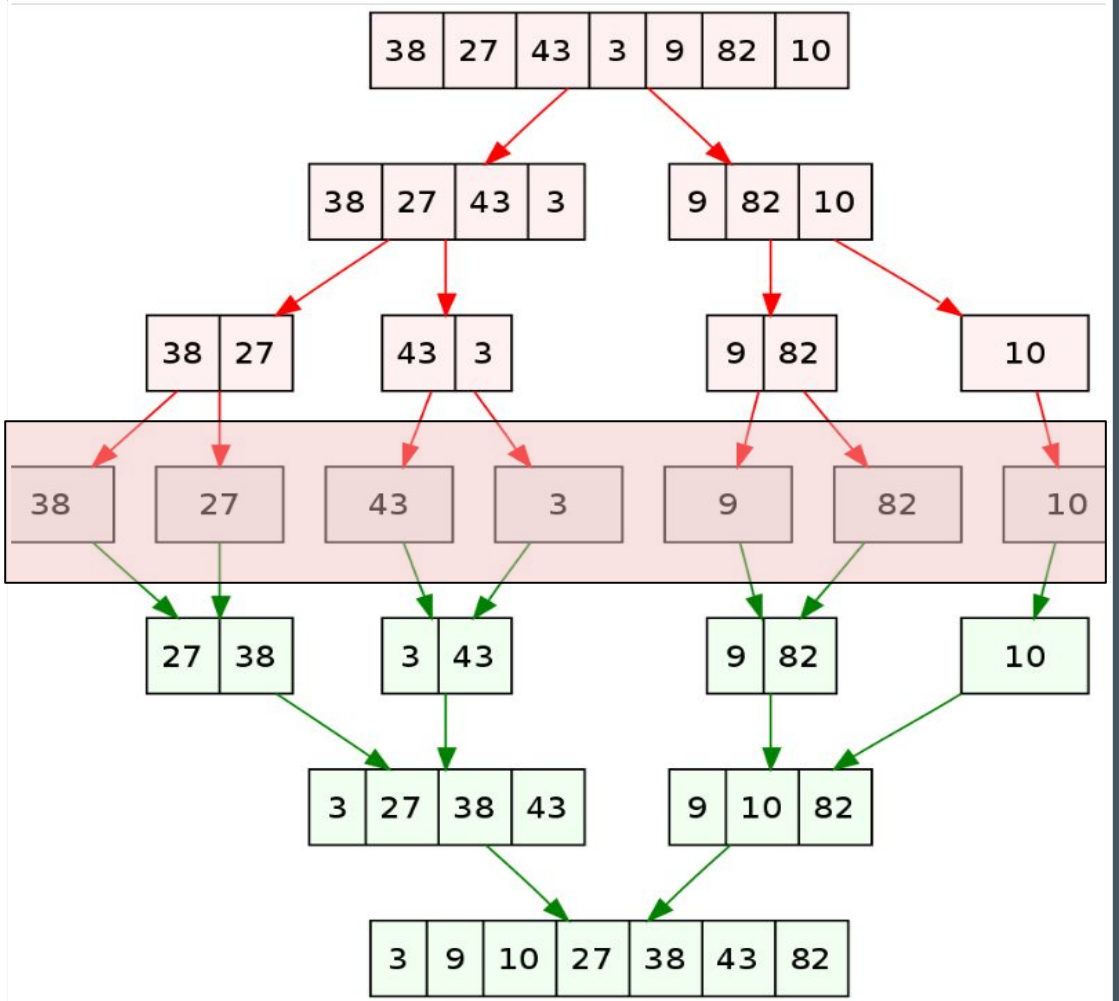
Merge Sort

Comienza a dividir el arreglo

Cuando no puede más, o sea, hay vectores de 1 sólo elemento, comienza a ordenar.

Se utiliza el algoritmo de mezcla de vectores, el cual se basa en que ambos vectores están ordenados.

Representación gráfica



Merge Sort

- Complejidad **O(n log n)**, ordena eficientemente grandes cantidades de datos.
- Conserva el orden relativo de los elementos iguales, lo que lo hace **estable**.
- Requiere espacio adicional.

Ej:

[8, 3, 6, 3, 2, 9, 6]

[2, 3, 3, 6, 6, 8, 9]

[4, 7, 2, 4, 1, 7, 3]

[1, 2, 3, 4, 4, 7, 7]

```
Procedure OrdMezcla( Var V: TArreglo; Prim,Ult : integer);
var mitad: Integer;
Begin
    IF (prim < ult) Then // puedo seguir dividiendo?
        Begin
        //
        mitad := ( prim + ult) DIV 2; // 4 - 2 - 1
        //
        OrdMezcla( V, prim, mitad);//1 y 4 - 1  2 - 1 y 1
        //
        OrdMezcla( V, mitad + 1 , ult);//5, 8 - 2 - 2
        //
        Mezclar( V, prim, mitad, mitad +1, ult)
        End
End.
...
ordMezc(arreglo, 1, 8);
```

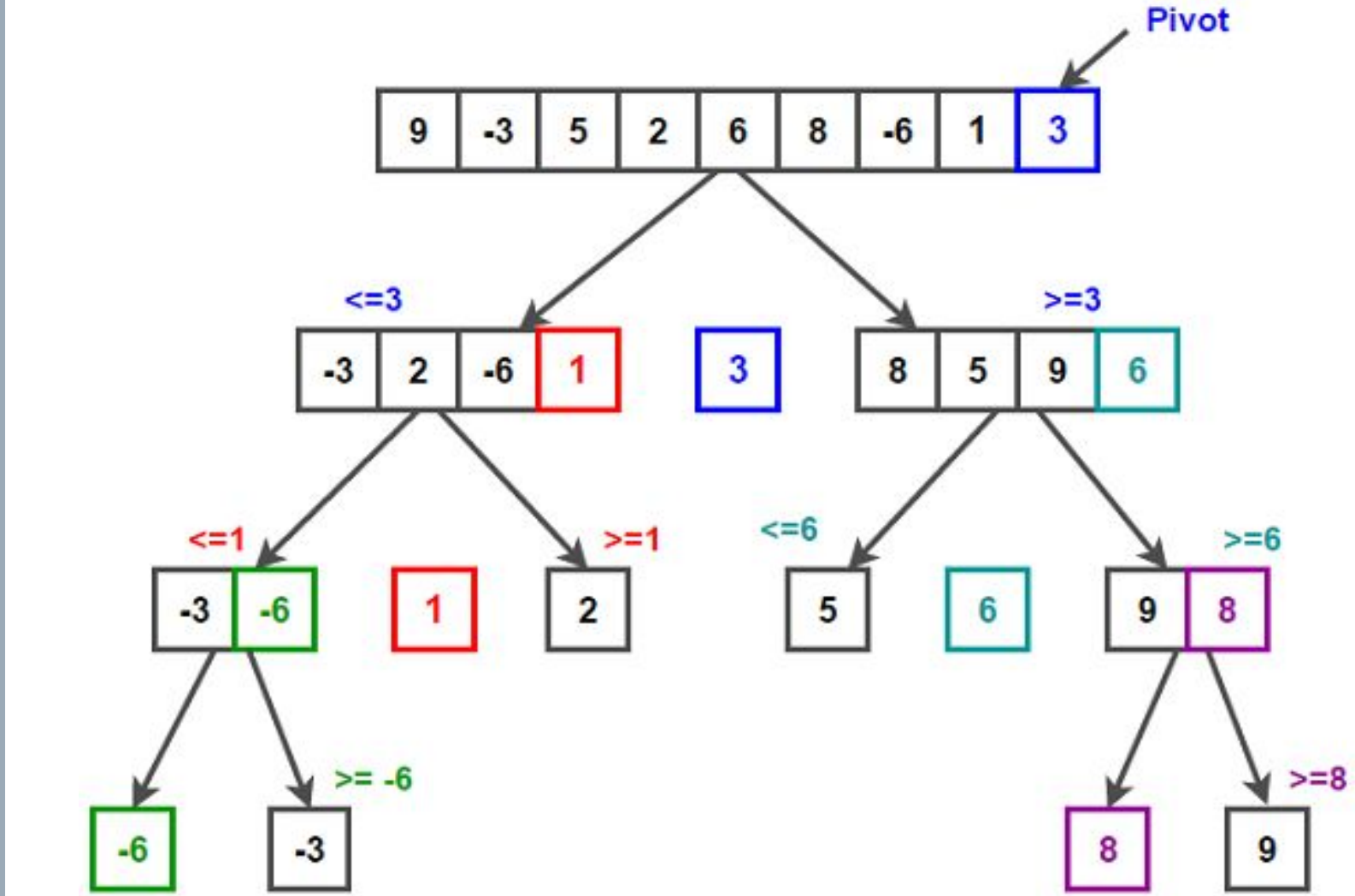
Quick Sort

Sigue el enfoque divide y vencerás.

Pasos:

1. Se elige un **pivote**. Hay varias formas, aunque es común tomar alguno de los extremos.
2. Se particiona. Se reorganizan los elementos más pequeños a la izquierda del pivot y los más grandes a su derecha. El pivote queda en su posición.
3. De forma recursiva, se vuelve elegir un pivote en cada partición y hacer 1 y 2.

Representación gráfica



Quick Sort

- Complejidad promedio **O(n log n)**, ordena eficientemente grandes cantidades de datos.
- No requiere de espacio adicional.
- En el peor caso puede tener un **$O(n^2)$**, ocurre cuando el pivote elegido es el más grande o el más pequeño de los elementos.

```
Function dividir(pri, ult: integer): integer;
var izq, der, pivot: integer;
begin
pivot := arreglo[pri];
izq := pri + 1;
der := ult;
//
WHILE izq <= der DO
BEGIN
     WHILE (izq <= ult) AND (arreglo[izq] < pivot) DO
          izq := izq + 1;
     WHILE (der > pri) AND (arreglo[der] >= pivot) DO
          der := der - 1;
   IF izq < der THEN
          intercambiar(arreglo[izq], arreglo[der]);
END;
//
intercambiar(arreglo[pri], arreglo[der]);
dividir := der;
end;
```

```
…
IF pri < ult THEN
BEGIN
   division := dividir(pri, ult);
   QuicksortRecur(pri, division-1);
   QuicksortRecur(division+1, ult);
END
…
```

Heap Sort

!No vimos Heap como estructura…

Heap :: Montículo

Existen dos tipos de montículos binarios: el montículo máximo y el montículo mínimo.

Se debe clumplir:

- Propiedad de orden: En un montículo máximo, para cada nodo, el valor del nodo padre es "mayor" o igual que los valores de sus nodos hijos.
- Estructura de árbol binario completo: todos los niveles del árbol están completamente llenos con excepción del último nivel que podrían faltarle nodos.

Se utiliza el método Heapify (recursivo, desde la raiz)…

Se usan en HeapSort y PriorityQueue

Bibliografía

- Data Structures. Nalle Dale.
- Estructuras de datos en C. A, Tenenbaum - Y, Langsam – M, Augenstein Un libro
- Estructuras de datos y algoritmos. M, Weiss
- https://es.wikipedia.org/wiki/C._A._R._Hoare