

Módulo 8: Recursión

Parte 3

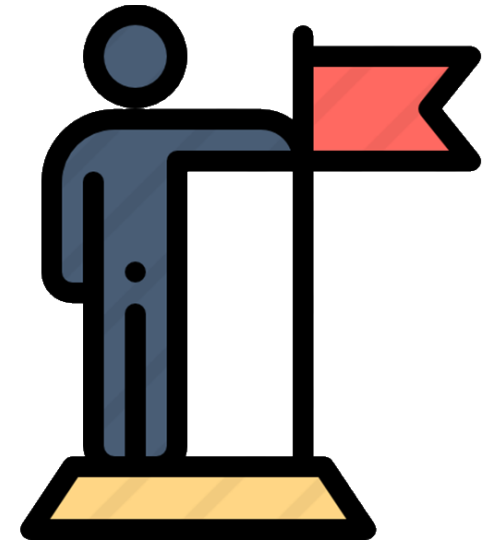
Capítulo 11



Merge Sort

Merge Sort es un algoritmo de ordenamiento más eficiente que los otros algoritmos de ordenamiento que hemos visto anteriormente.

Es un algoritmo recursivo que utiliza un patrón de diseño de tipo “Divide and Conquer”.



Método divide()

Diseña un método que permita partir un arreglo a la mitad, retornando las dos mitades en arreglos distintos.

```
private static void divide(int[] from, int[] firstHalf, int[] lastHalf) {  
    //copy first half of array  
    for(int i=0; i<firstHalf.length; i++) {  
        firstHalf[i] = from[i];  
    }  
    //copy second half of array  
    for(int i=0; i<lastHalf.length; i++) {  
        lastHalf[i] = from[i+firstHalf.length];  
    }  
}
```

Método merge()

Diseña un método que permita combinar dos arreglos, retornando el arreglo combinado.

firstHalf				
2	5	6	8	10
×	×	×	×	×

lastHalf				
1	3	3	6	7
×	×	×	×	×

1	2	3	3
---	---	---	---

5	6	6	7
---	---	---	---

8	10
---	----


```
private static void merge(int[] newArray, int[] firstHalf, int[] lastHalf) {
    int firstHalfIndex = 0, lastHalfIndex = 0, newArrayIndex = 0;

    //Compare and copy the smallest element
    while (firstHalfIndex < firstHalf.length && lastHalfIndex < lastHalf.length) {
        if (firstHalf[firstHalfIndex] < lastHalf[lastHalfIndex]) {
            newArray[newArrayIndex] = firstHalf[firstHalfIndex];
            firstHalfIndex++;
        } else {
            newArray[newArrayIndex] = lastHalf[lastHalfIndex];
            lastHalfIndex++;
        }
        newArrayIndex++;
    }
    //Copy remaining objects from firstHalf
    while(firstHalfIndex < firstHalf.length) {
        newArray[newArrayIndex++] = firstHalf[firstHalfIndex++];
    }
    //Copy remaining objects from lastHalf
    while(lastHalfIndex < lastHalf.length) {
        newArray[newArrayIndex++] = lastHalf[lastHalfIndex++];
    }
}
```

Merge Sort

El Merge Sort está compuesto por 2 pasos:

1. Divide
2. Merge

Merge Sort

Caso base

1. El tamaño del arreglo `data` es menor o igual a 1. (El arreglo está ordenado).

Caso Recursivo

2. Copiar la primera mitad del arreglo `data` a `firstHalf`.
3. Copiar el resto de los elementos de `data` a `lastHalf`.
4. Ordenar `firstHalf` usando una llamada recursiva a `mergeSort()`.
5. Ordenar `lastHalf` usando una llamada recursiva a `mergeSort()`.
6. Unir `firstHalf` y `lastHalf`.

1 5 3 4 2 6

1 5 3

4 2 6

1

5

3

4

2

6

1

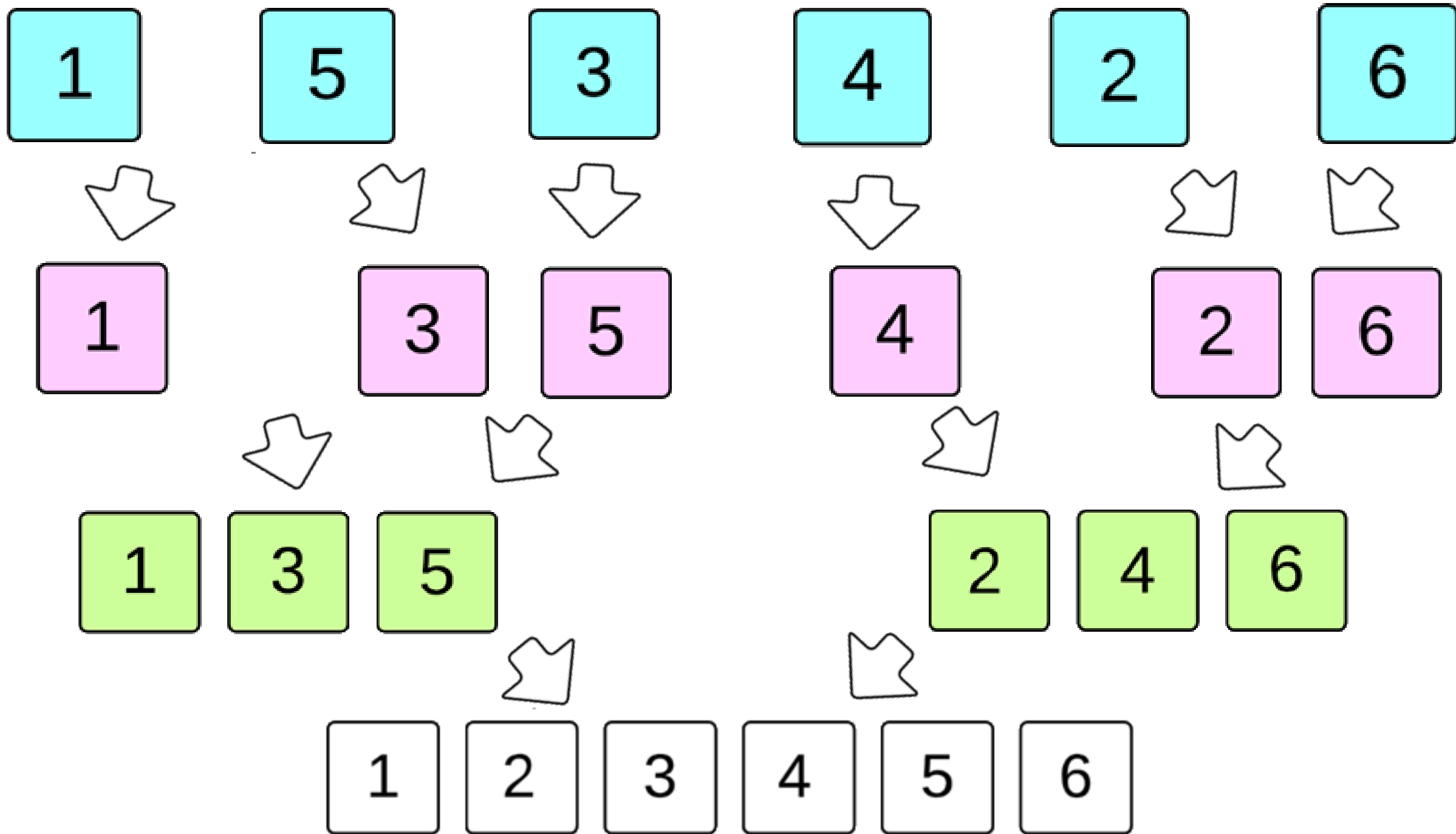
5

3

4

2

6



Merge Sort

<https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/visualize/>

Merge Sort

```
public static void mergeSort(int[] data) {  
    //base case -> 1 element array is always sorted  
    if (data.length <= 1)  
        return;  
  
    //divide  
    int[] firstHalf = new int[data.length / 2];  
    int[] lastHalf = new int[data.length - firstHalf.length];  
    divide(data, firstHalf, lastHalf);  
  
    //recursive call -> sort halves  
    mergeSort(firstHalf);  
    mergeSort(lastHalf);  
  
    //merge two halves  
    merge(data, firstHalf, lastHalf);  
}
```

Merge Sort

Ordena los siguientes arreglos utilizando el algoritmo merge sort.

-5	5	10	15	-10
----	---	----	----	-----

77	8	5	63	1	2	6	5	4	8
----	---	---	----	---	---	---	---	---	---