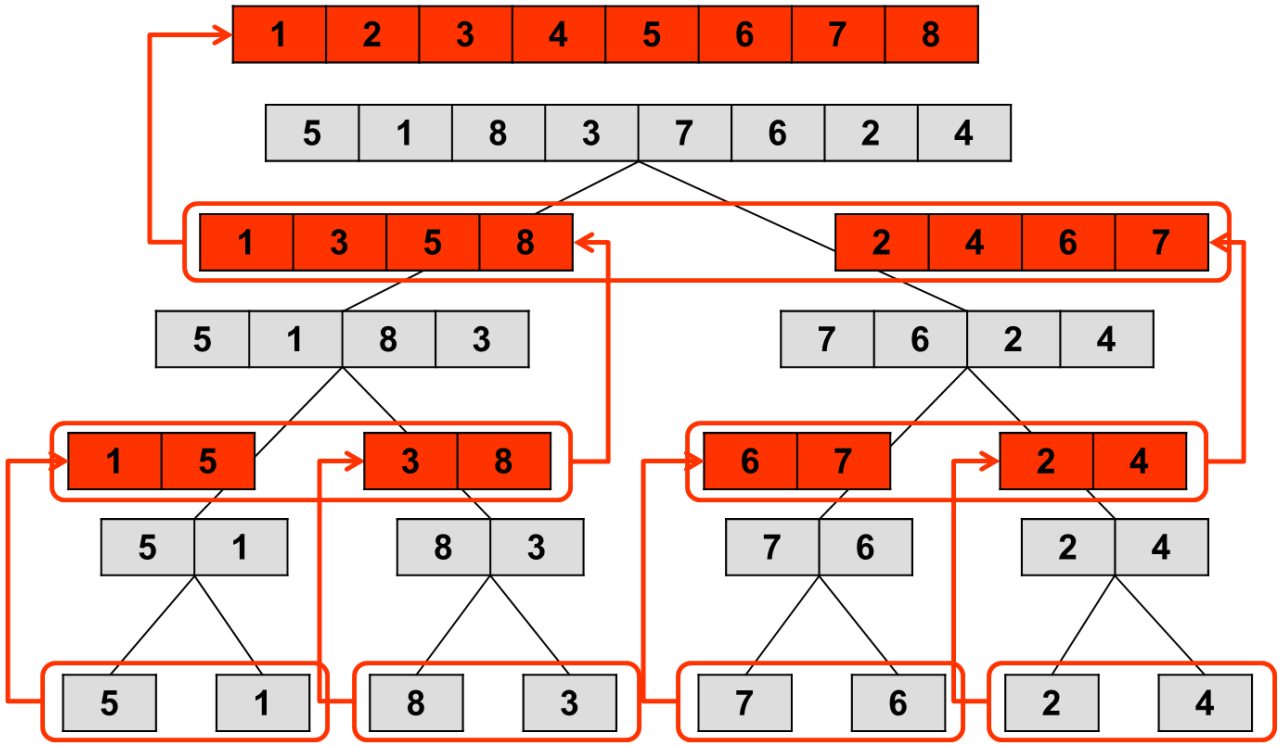
QuickSort, parte 1

QuickSort

Ya vimos que la complejidad del MergeSort respecto a ejecución es O(N*log(N)), pero ¿Qué pasa con la complejidad respecto a la memoria?



Pues resulta que también es O(*N.log(N)*)

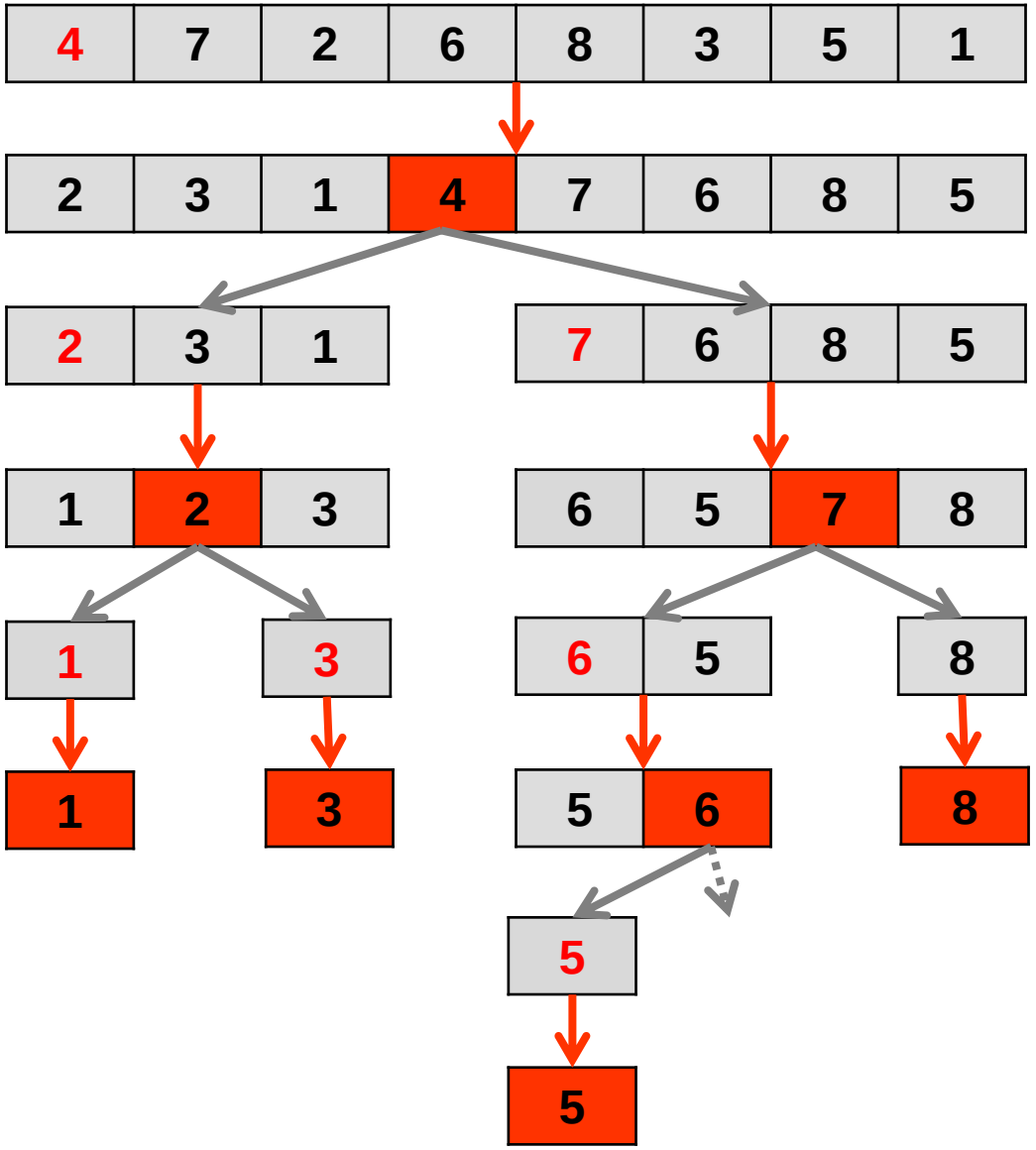
Aunque una versión modificada sería O(*N*)

QuickSort

Postulado por Hoore en 1961, funciona de la siguiente manera:

- Elegir un elemento del arreglo llamado *pivote* (el primero por ejemplo).
- Partir el arreglo: Mover los demás elementos del arreglo a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores.
- El arreglo queda separado en dos sub-arreglos, uno formado por los elementos a la izquierda del pivote, y otro por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sub-arreglo mientras éstos contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

QuickSort



QuickSort

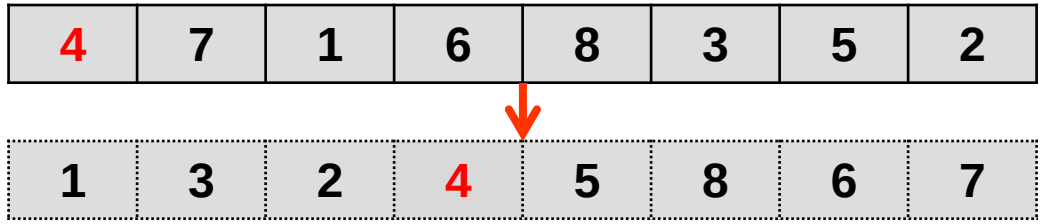
```
function quickSort(X, N):
    quickSort(X, 0, N-1)

function quickSort(X, a, b):
    if b-a > 0:
      p = choosePivot(a, b)
      h = partition(X, a, b, p)
      quickSort(X, a, h-1)
      quickSort(X, h+1, b)
```

QuickSort

Particionamiento, primera alternativa: *N* memoria adicional (temporal)

```
function partition(X, a, b, p):
    swap(Xa, Xp)
    i = a
    j = b
    c = a+1
    while c <= b:
        if Xc < Xp
            Yi = Xc
             i ++
        else
            Yj = Xc
             j --
        c ++
    Yi = Xa
    return i
```



QuickSort

Particionamiento, primera alternativa: *1* memoria adicional

```
function partition(X, a, b, p):
   swap(X_a, X_p)
   i = a
   for j=a+1 to b:
      if (X_j < X_a){
         i++
         swap(X_i, X_j)
   swap(X_i, X_a)
   return i
```

En un determinado j: