Merge Sort: Shortcomings

- Merging A and B creates a new array C
  - *No obvious way to efficiently merge in place*
- Extra storage can be costly
- Inherently recursive
  - *Recursive call and return are expensive*

Alternative approach

- Extra space is required to merge
- Merging happens because elements in left half must move right and vice versa
- [2,4,6] [1,3,5]
- Can we divide so that everything to the left is smaller than everything to the right?
  - *No need to merge!*
  - *[2,1,3] [6,4,5]*

Divide and conquer without merging

- **Quick Sort (1960): Tony Hoare,** Turing Award (1980)
- Suppose the median {0, 0, 0, 1, 1, 2, 10, 10} value in A is m
- Move all values ≤ m to left half of A
  - *Right half has values > m*
  - *This shifting can be done in place, in time O(n)*
- Recursively sort left and right halves
- A is now sorted! No need to merge
  - *O(n log n) : Time Complexity wise same, no extra array for merge*

Divide and conquer without merging

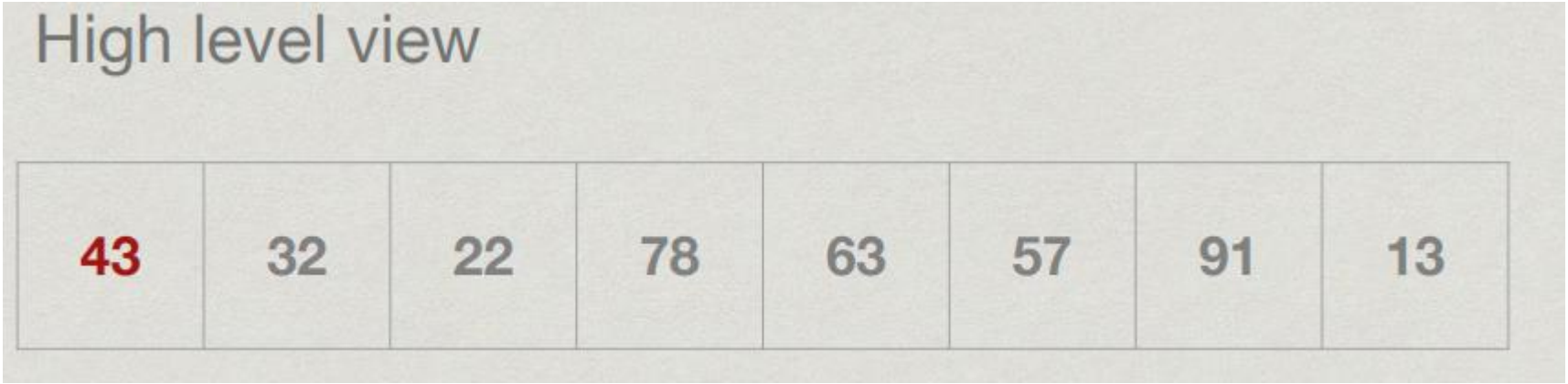
- How do we find the median?
- Sort and pick up middle element
- But our aim is to sort! (Chicken Egg Problem)

- Instead, pick up some value in A — **pivot**
- Split A with respect to this pivot element

Quick Sort

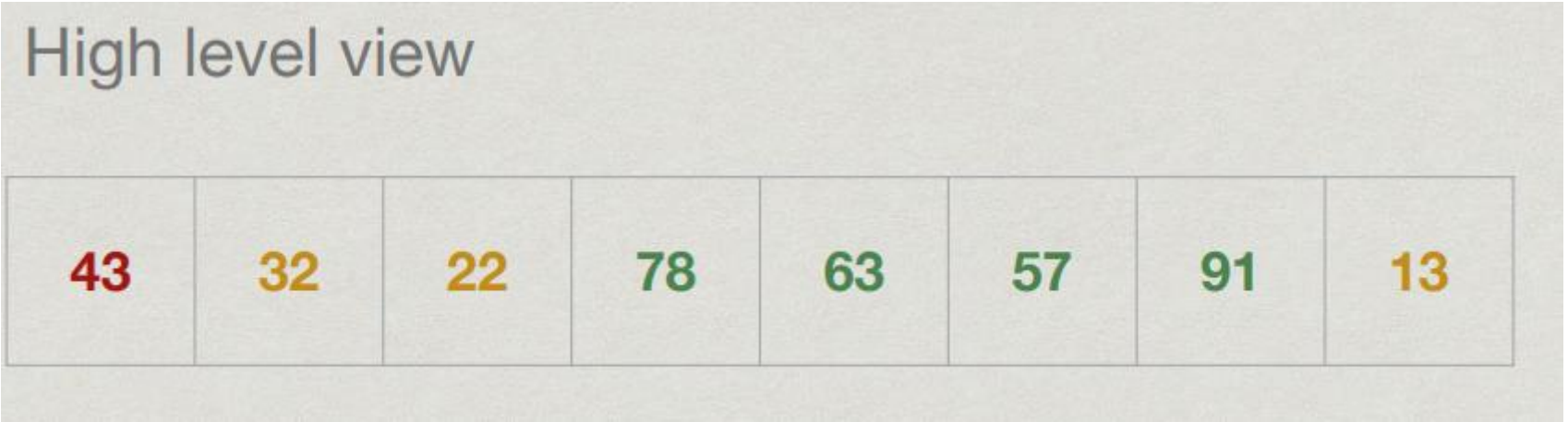
- Choose a pivot element
  - *Typically the first value in the array*
- Partition A into lower and upper parts with respect to pivot
- Move pivot between lower and upper partition
- Recursively sort the two partitions

Quicksort

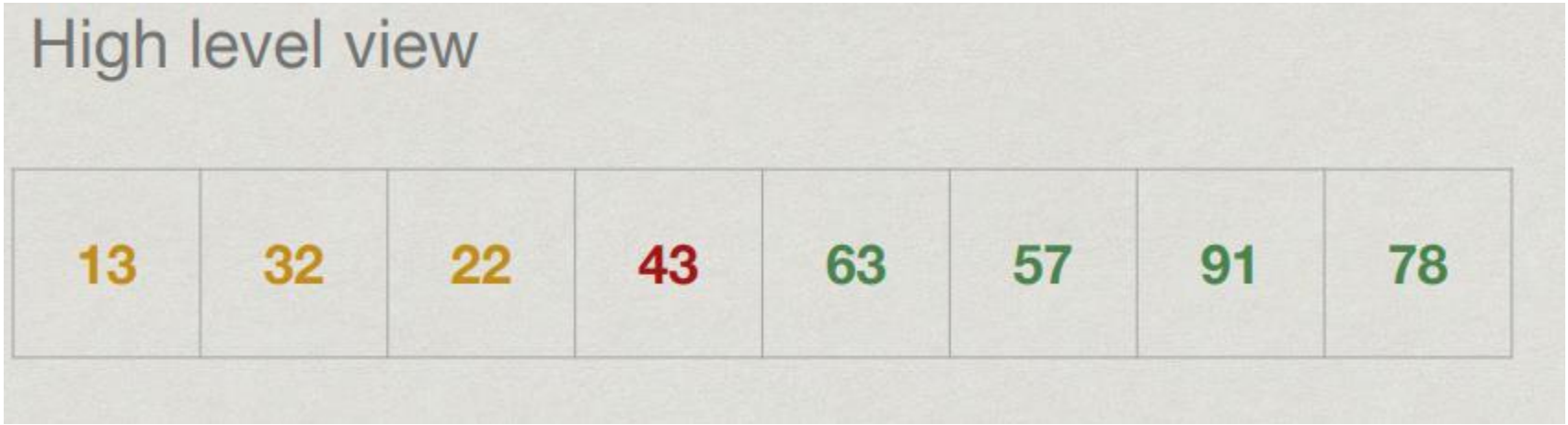


Quicksort

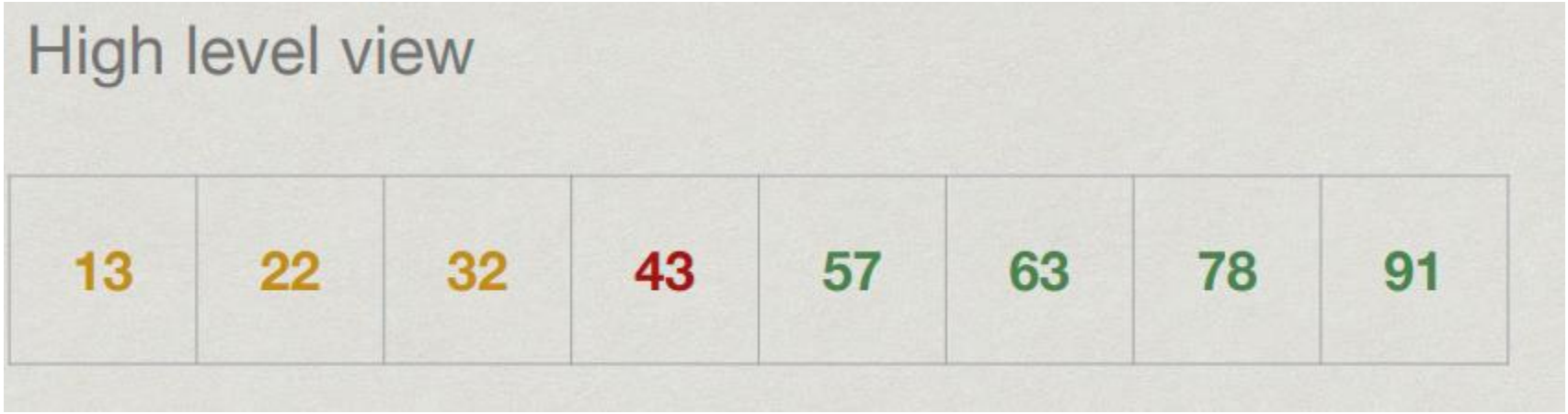
High level view



Quicksort

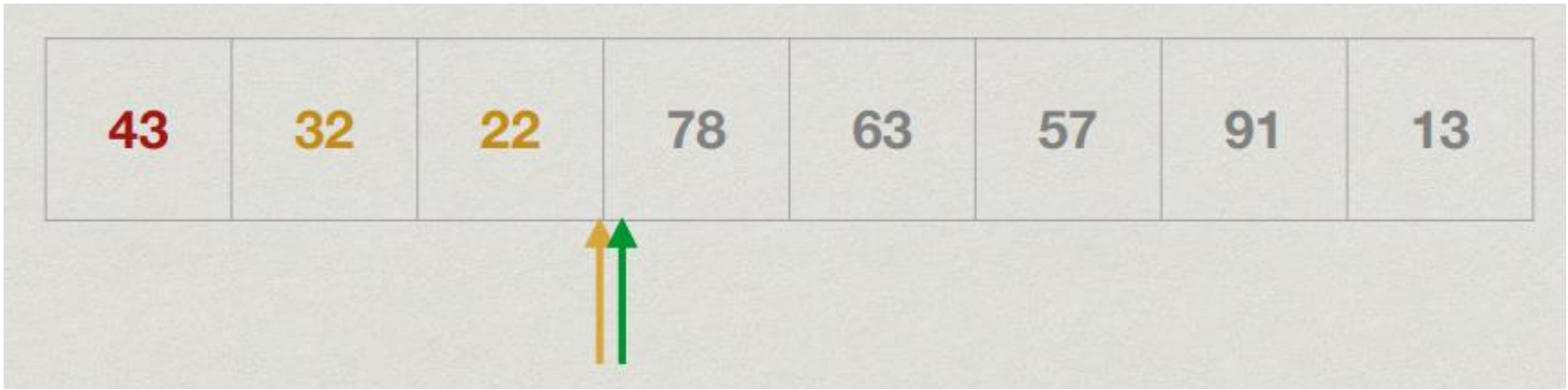


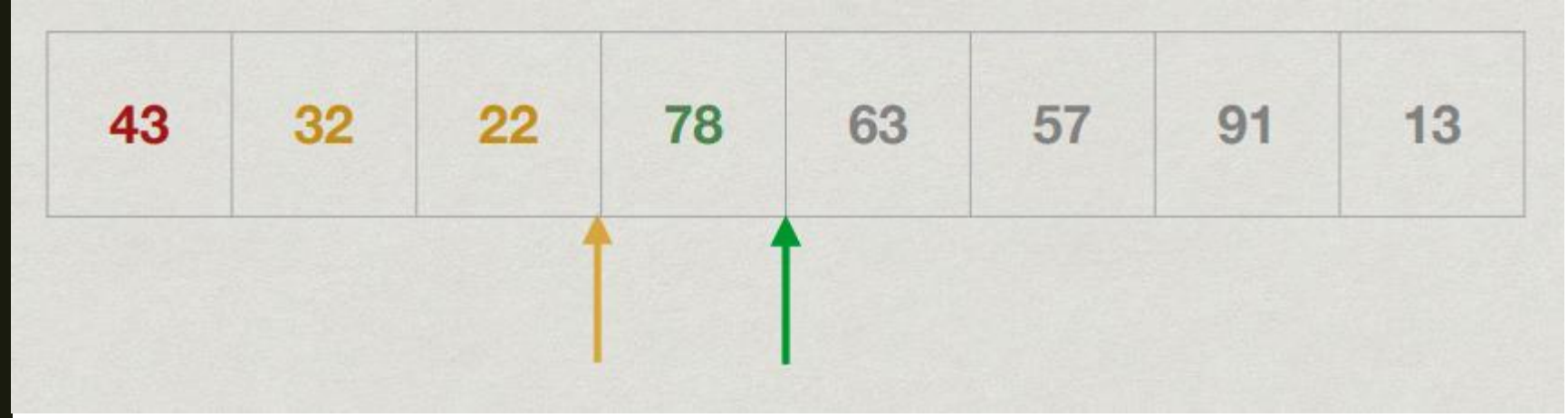
Quicksort



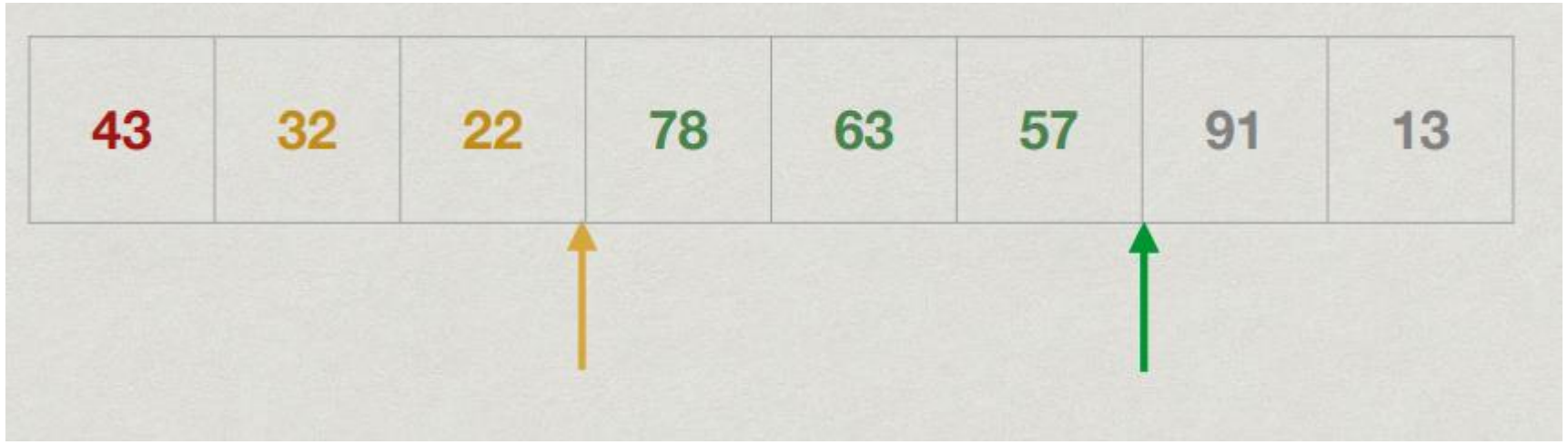

43
32
22
78
63
57
91
13

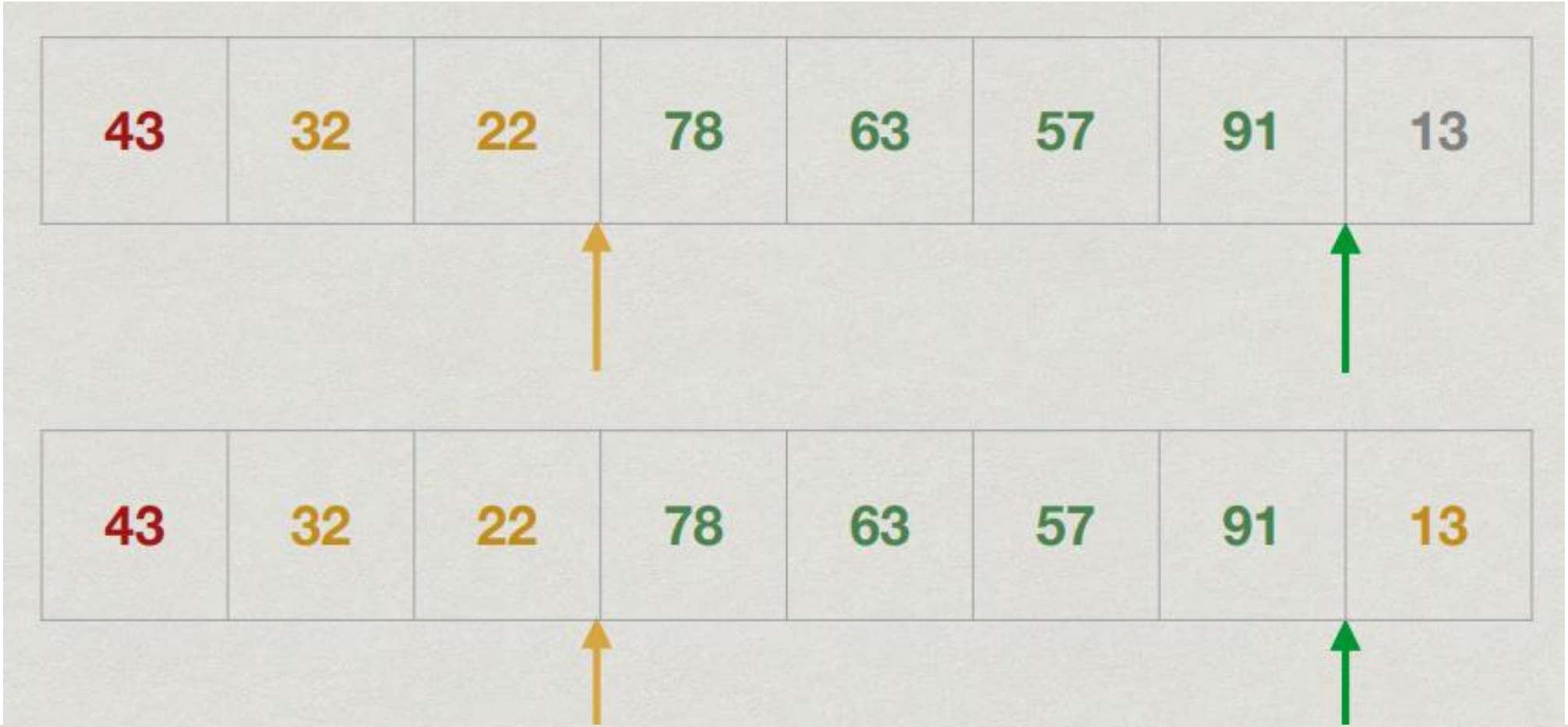

43
32
22
78
63
57
91
13

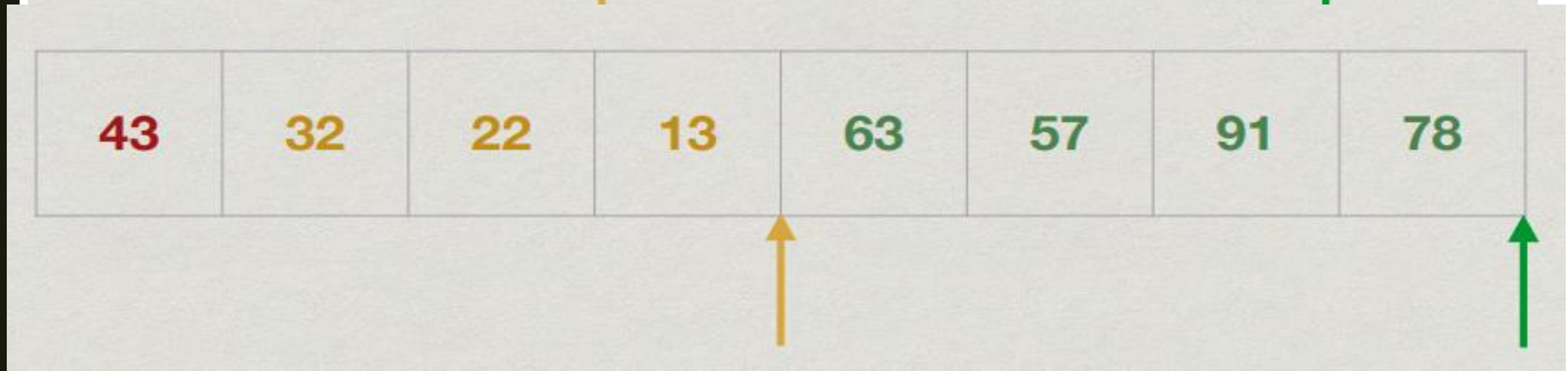

43
32
22
78
63
57
91
13

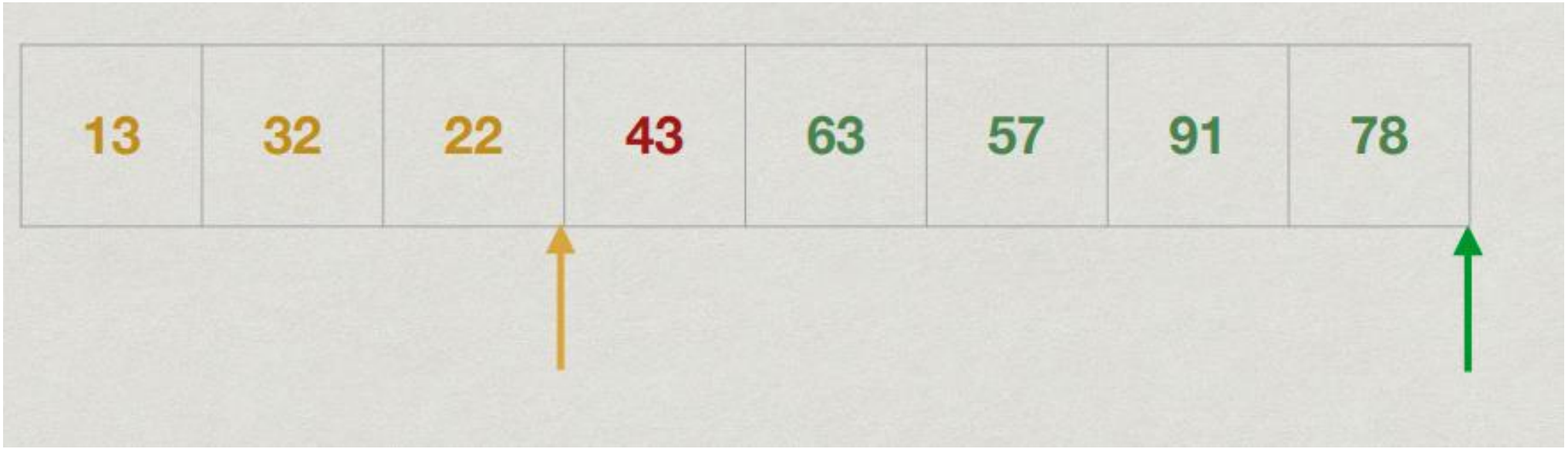

43
32
22
78
63
57
91
13


43
32
22
78
63
57
91
13


43
32
22
78
63
57
91
13


43 32 22 78 63 57 91 13
43 32 22 78 63 57 91 13


43 32 22 13 63 57 91 78


13
32
22
43
63
57
91
78


13
22
32
43
57
63
78
91

Quicksort: Implementation

```
Quicksort(A,l,r) // Sort A[l..r-1]
if (r - l <= 1)) return; // Base case
// Partition with respect to pivot, a[l]
yellow = l+1;
for (green = l+1; green < r; green++)
if (A[green] <= A[l])
swap(A,yellow,green);
yellow++;
swap(A,l,yellow-1); // Move pivot into place
Quicksort(A,l,yellow); // Recursive calls
Quicksort(A,yellow+1,r);
```

Analysis of Quicksort

- Worst case
  - *Pivot is maximum or minimum*
  - *One partition is empty*
  - *Other is size n-1*
- Already sorted array is worst case input O(n2)