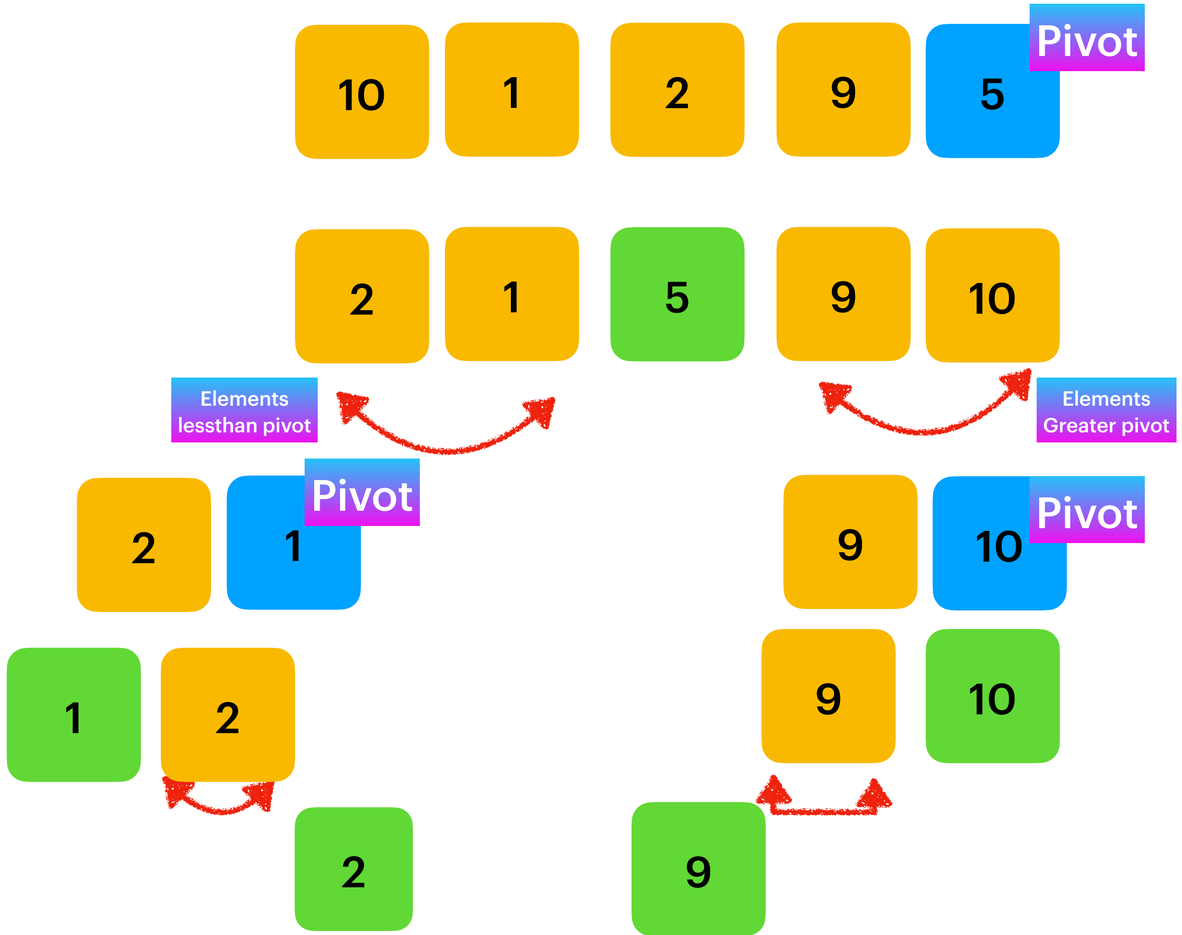
Quick Sort

Quick Sort uses the divide & conquer approach.

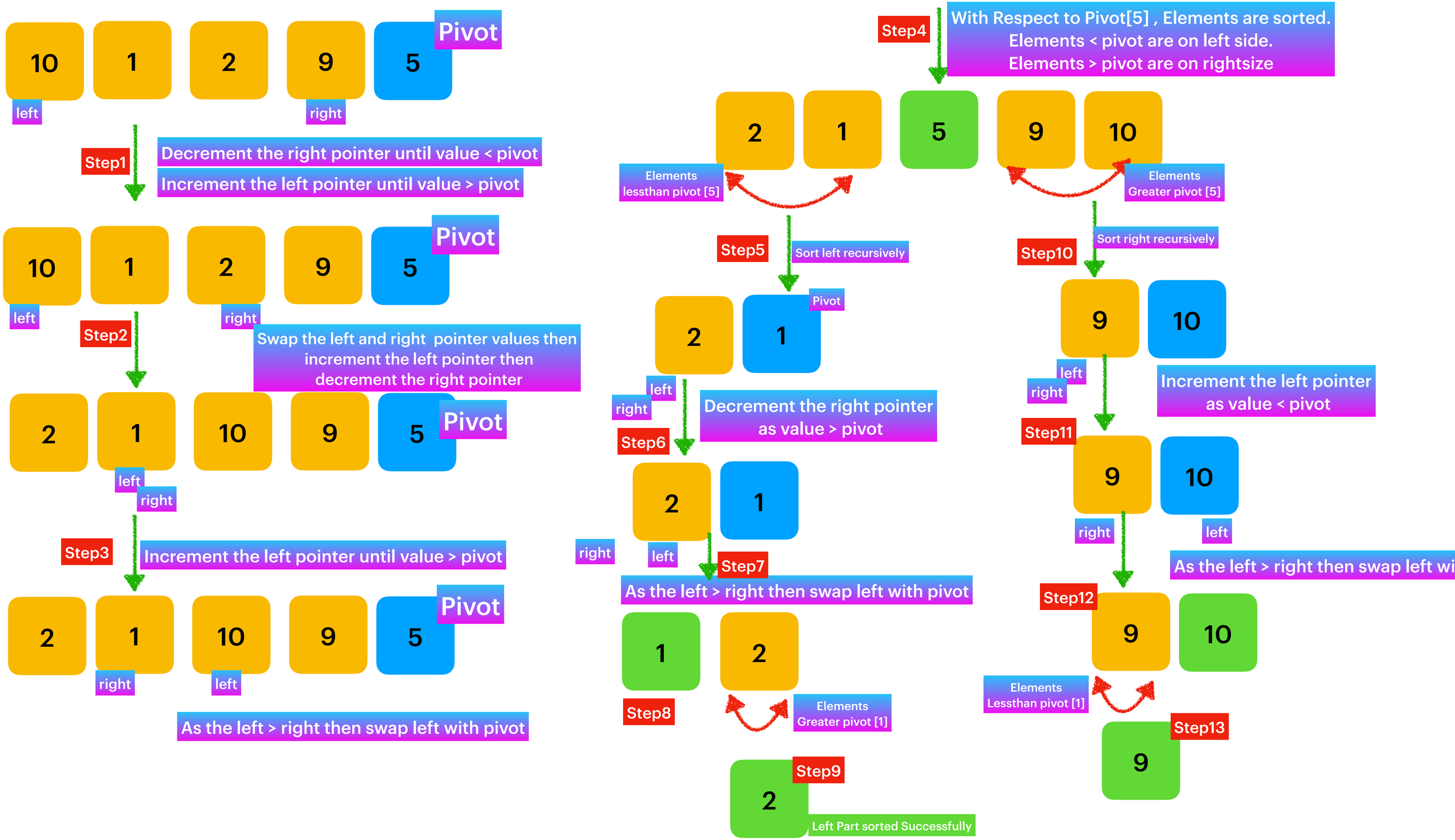
Quick Sort chooses the pivot element within a collection of elements then do the sorting such that elements less than the pivot moved to left part and the elements greater than pivot moved to right part.

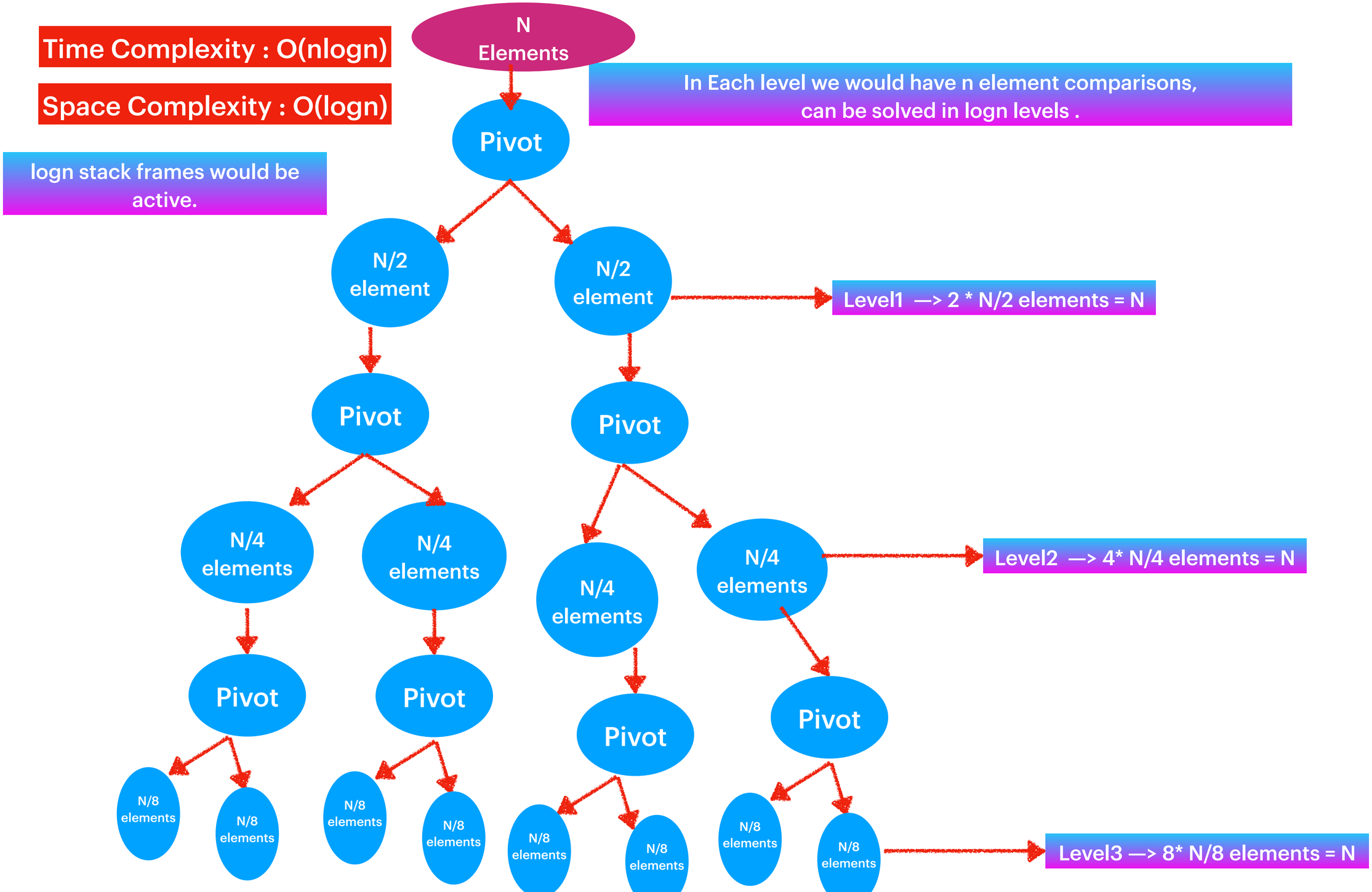
Same logic applies on left Part and Right Part recursively.

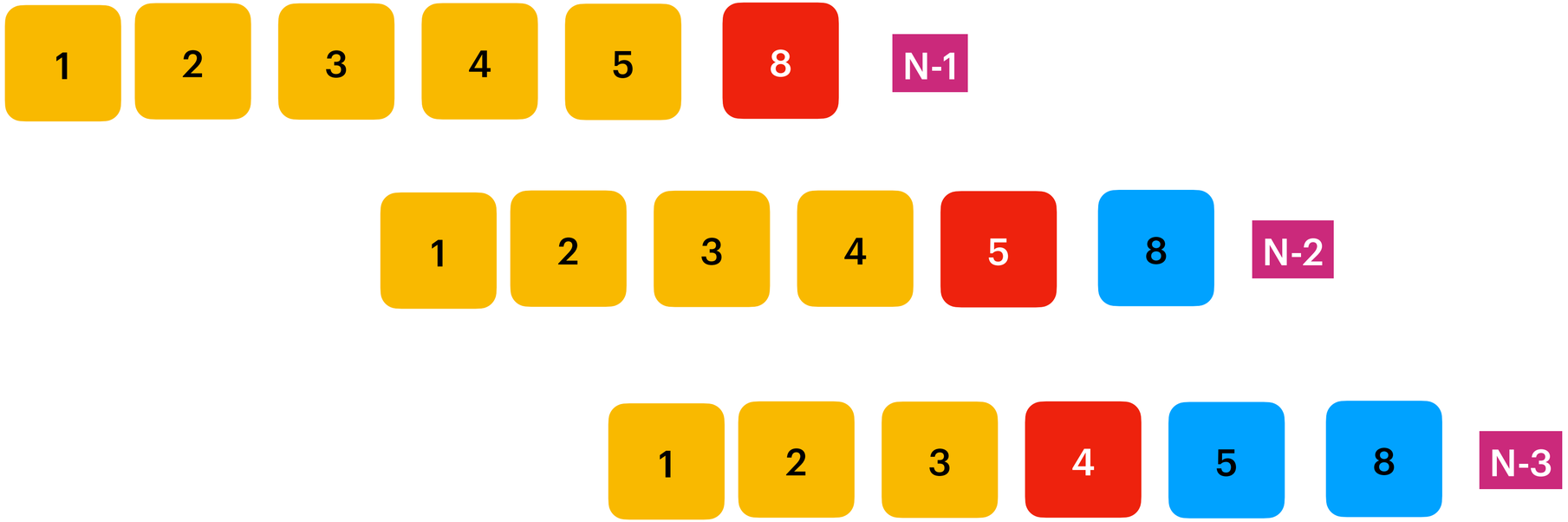


In Ideal practice we consider either the first or the last element as pivot.

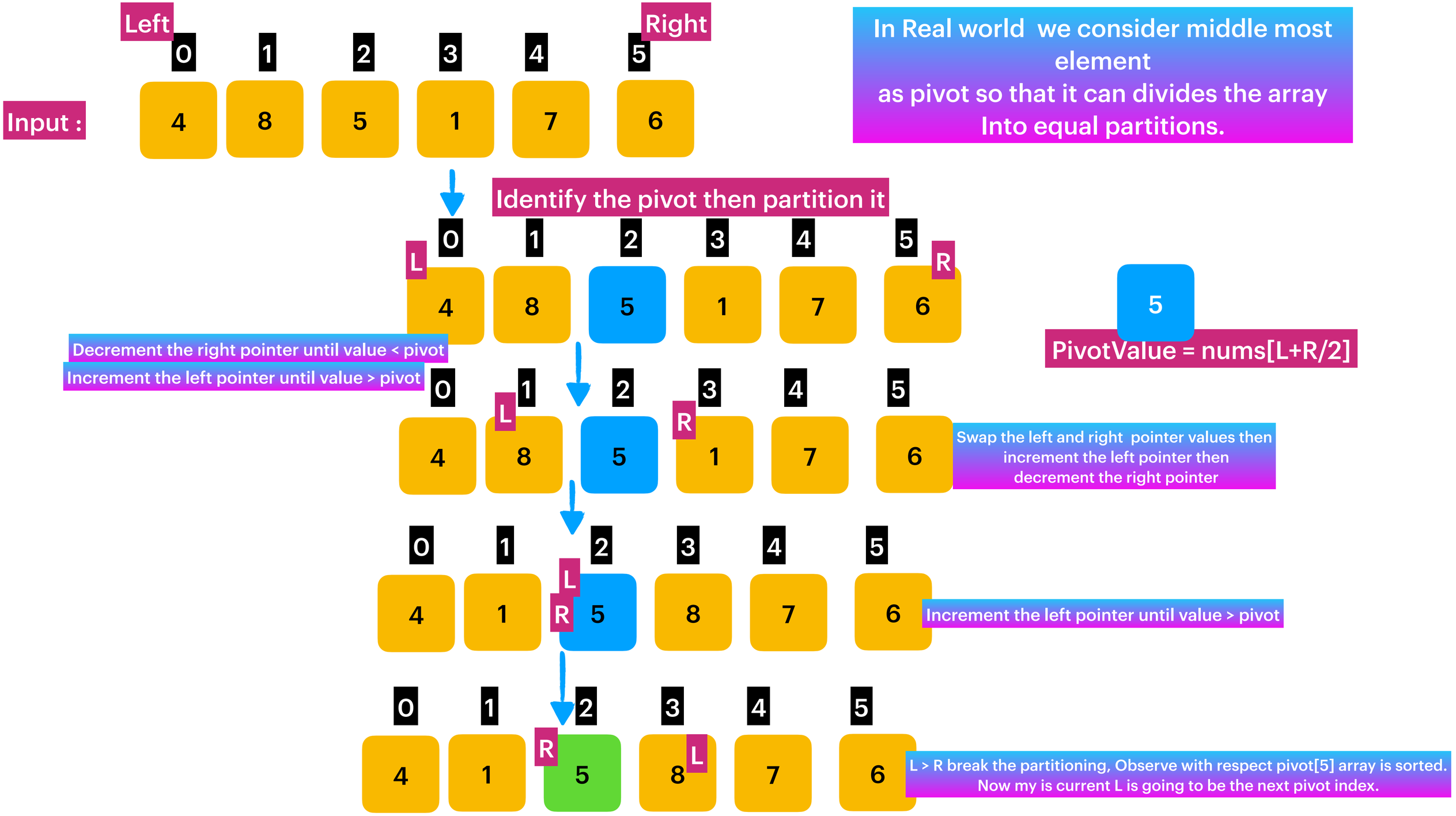
In Real world we consider Middle most element as pivot So that it can divides the array Into equal partitions.

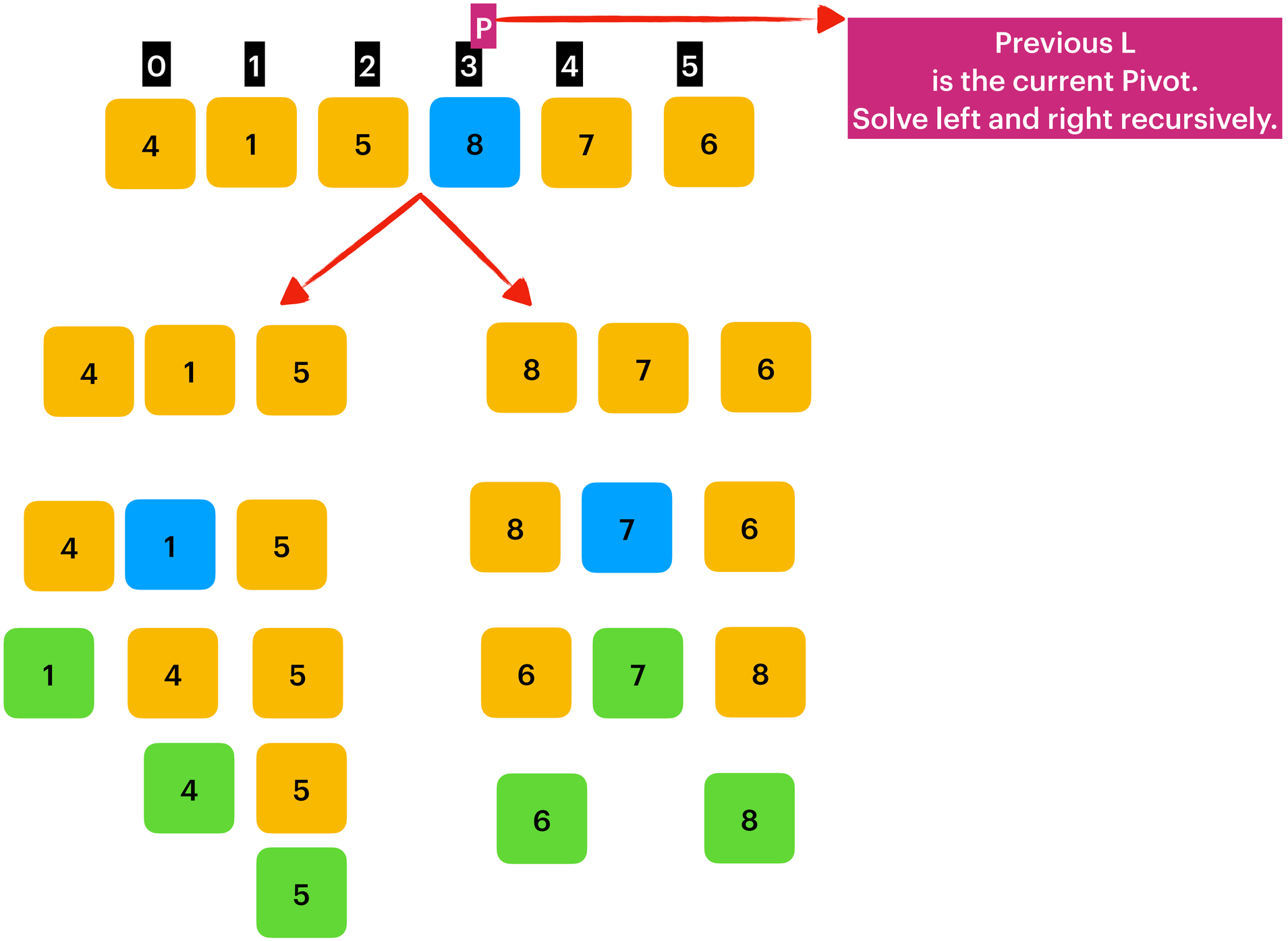

10
1
2
9
5
Pivot
left
right
Step1
Decrement the right pointer until value < pivot
Increment the left pointer until value > pivot
Step2
Swap the left and right pointer values then increment the left pointer then decrement the right pointer
Step3
Increment the left pointer until value > pivot
As the left > right then swap left with pivot
Step4
With Respect to Pivot[5] , Elements are sorted.
Elements < pivot are on left side.
Elements > pivot are on rightsize
Elements lessthan pivot [5]
Elements Greater pivot [5]
Step5
Sort left recursively
Step6
Decrement the right pointer as value > pivot
Step7
As the left > right then swap left with pivot
Step8
Elements Greater pivot [1]
Step9
Left Part sorted Successfully
Step10
Sort right recursively
Step11
Increment the left pointer as value < pivot
Step12
Elements Lessthan pivot [1]
Step13


Time Complexity : O(nlogn)
Space Complexity : O(logn)
logn stack frames would be active.
N Elements
In Each level we would have n element comparisons, can be solved in logn levels .
Pivot
N/2 element
N/2 element
Level1 —> 2 * N/2 elements = N
Pivot
Pivot
N/4 elements
N/4 elements
N/4 elements
N/4 elements
Level2 —> 4* N/4 elements = N
Pivot
Pivot
Pivot
Pivot
N/8 elements
N/8 elements
N/8 elements
N/8 elements
N/8 elements
N/8 elements
N/8 elements
N/8 elements
Level3 —> 8* N/8 elements = N



We should not consider pivot either as last or first index results to n^2.
If the elements are sorted .


Left
Right
0 1 2 3 4 5
Input :
4 8 5 1 7 6
In Real world we consider middle most element
as pivot so that it can divides the array
Into equal partitions.
Identify the pivot then partition it
L
R
Decrement the right pointer until value < pivot
Increment the left pointer until value > pivot
5
PivotValue = nums[L+R/2]
Swap the left and right pointer values then
increment the left pointer then
decrement the right pointer
Increment the left pointer until value > pivot
L > R break the partitioning, Observe with respect pivot[5] array is sorted.
Now my is current L is going to be the next pivot index.


P
Previous L
is the current Pivot.
Solve left and right recursively.
0
1
2
3
4
5
4
1
5
8
7
6
4
1
5
8
7
6
4
1
5
8
7
6
1
4
5
6
7
8
4
5
6
8
5

QuickSort

Time Complexity : O(nlog(n))

Comparison Time : O(nlog(n))

Space Complexity : O(logn)

InPlace / OutPlace : In Place

Stability : YES
[After sort maintains insertion order]

External Sorting : NO