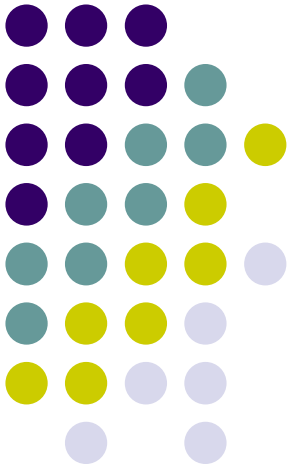
Sorting Algorithms

Quick Sort



Quick Sort

- **Divide**:
  - Pick any element **p** as the **pivot**, e.g, the first element
  - Partition the remaining elements into

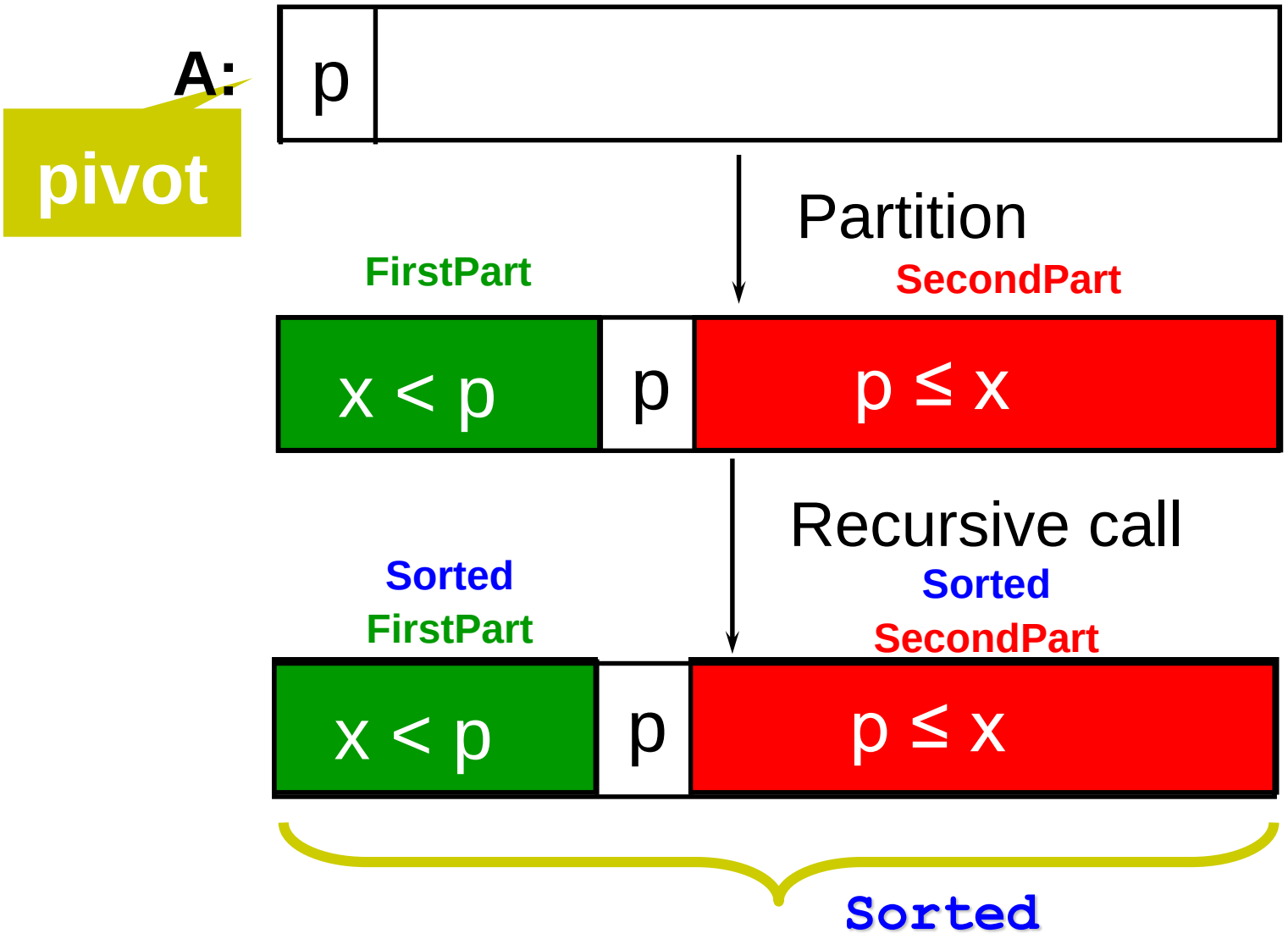
    FirstPart, which contains all elements < p

    SecondPart, which contains all elements ≥ p

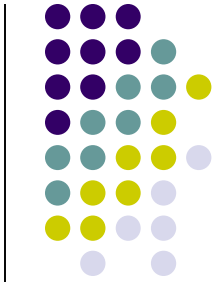
- **Recursively sort** the FirstPart and SecondPart

- **Combine**: no work is necessary since sorting is done in place

Quick Sort

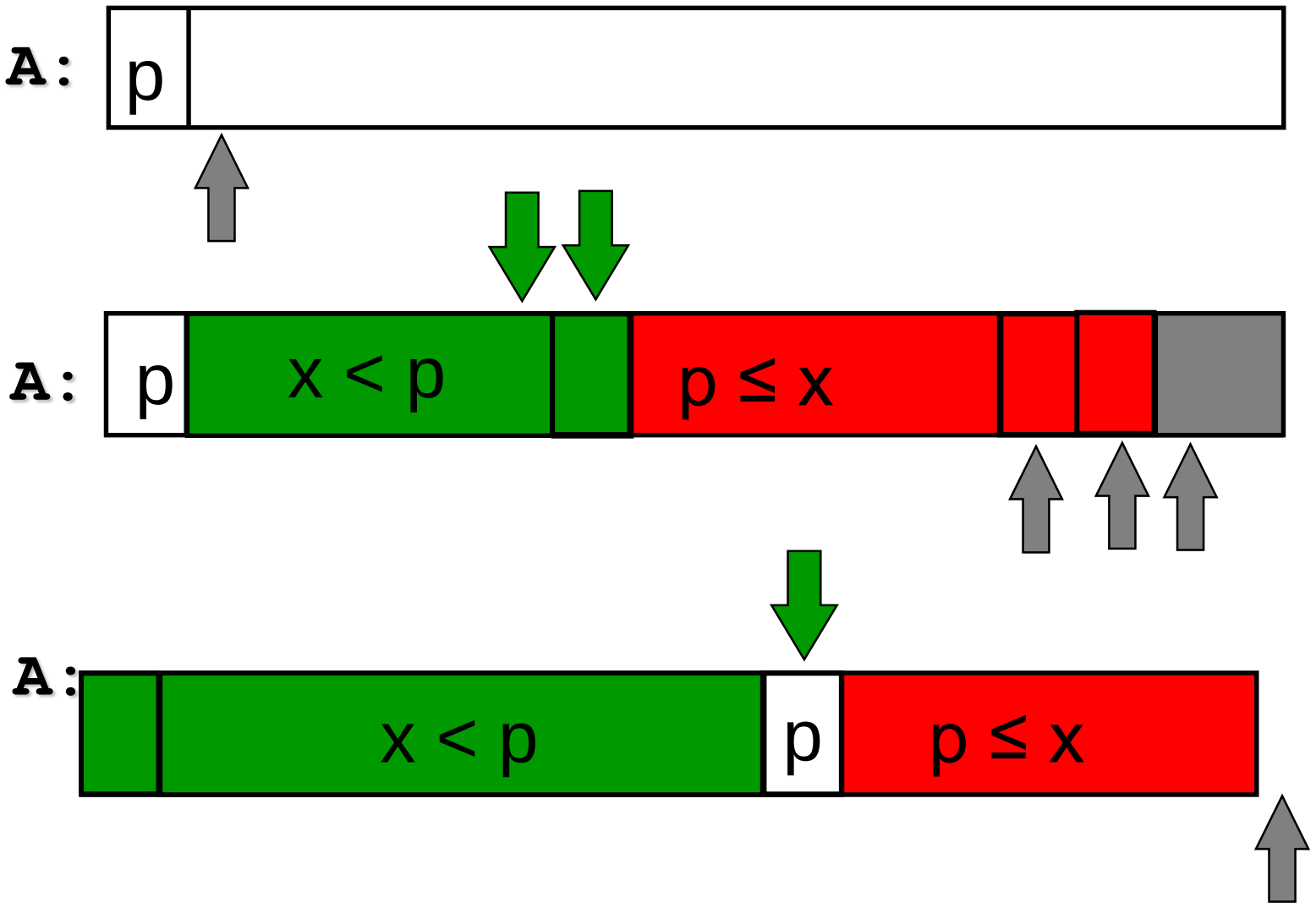

A:
pivot
p
Partition
FirstPart
SecondPart
x < p
p
p ≤ x
Recursive call
Sorted
FirstPart
Sorted
SecondPart
x < p
p
p ≤ x
Sorted

Quick Sort

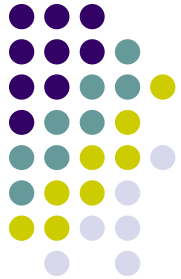


```
Quick-Sort(A, left, right)
  if    left ≥ right  return
  else
        middle ← Partition(A, left, right)
        Quick-Sort(A, left, middle-1 )
        Quick-Sort(A, middle+1, right)
  end if
```

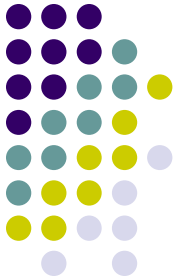
Partition

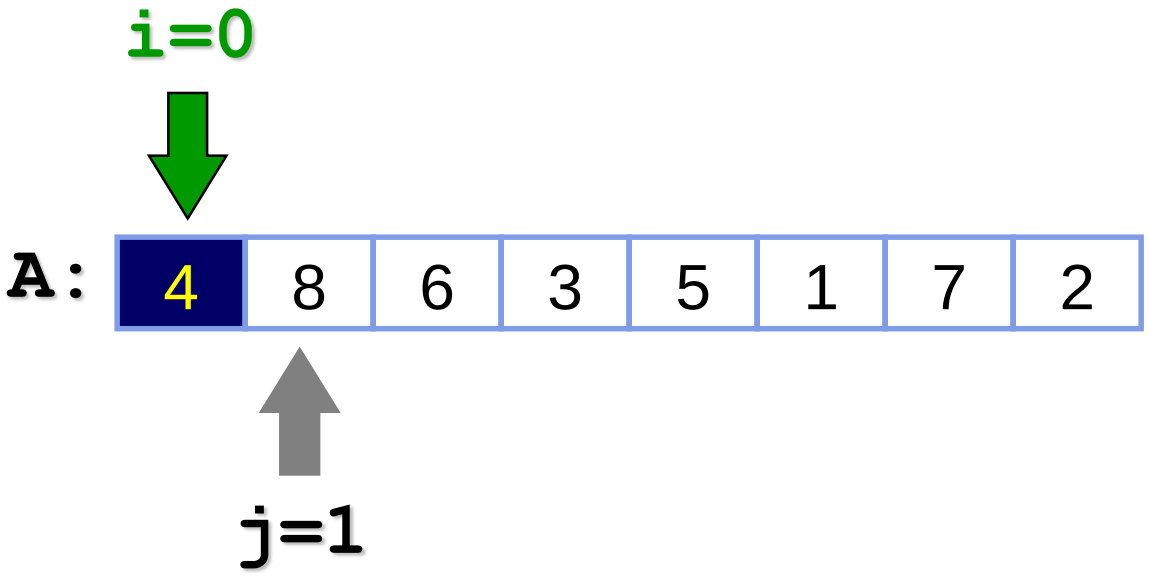


Partition Example

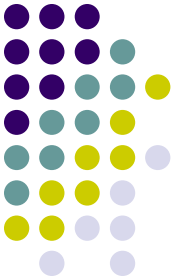


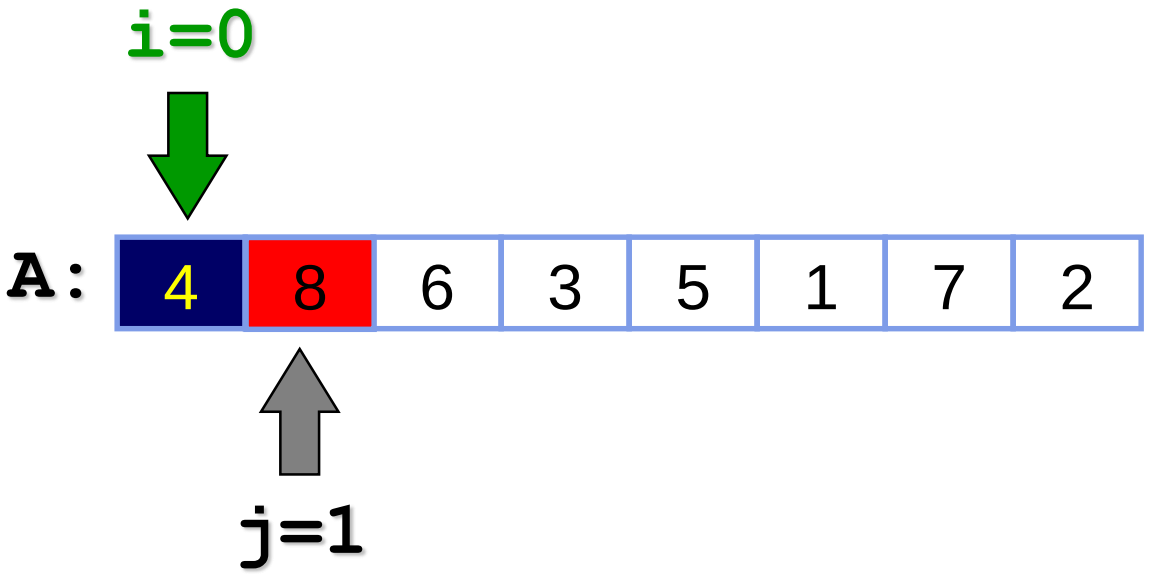
Partition Example



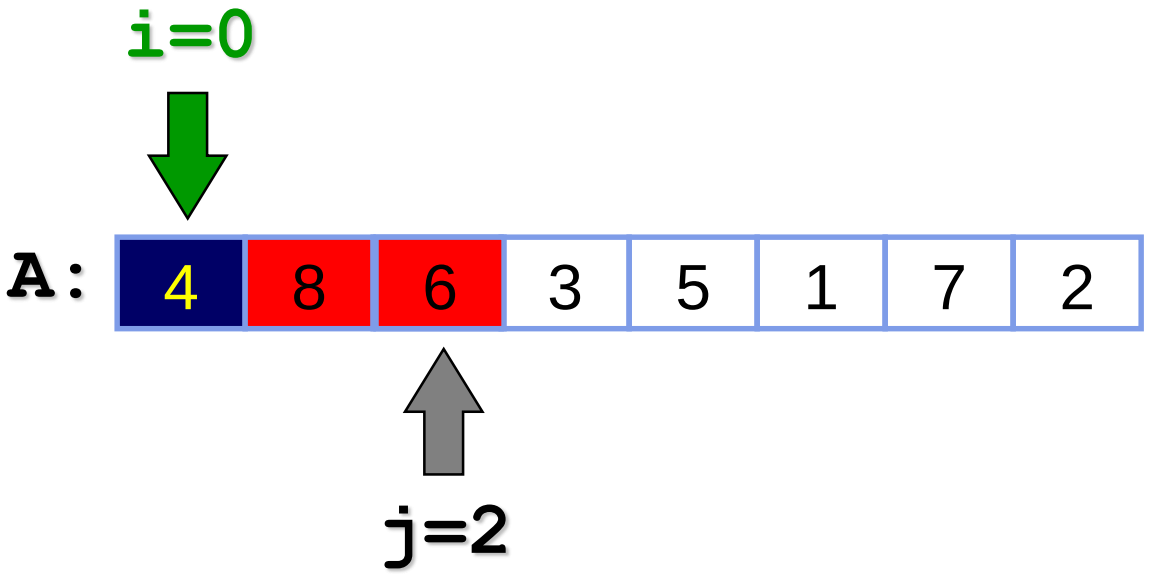


Partition Example

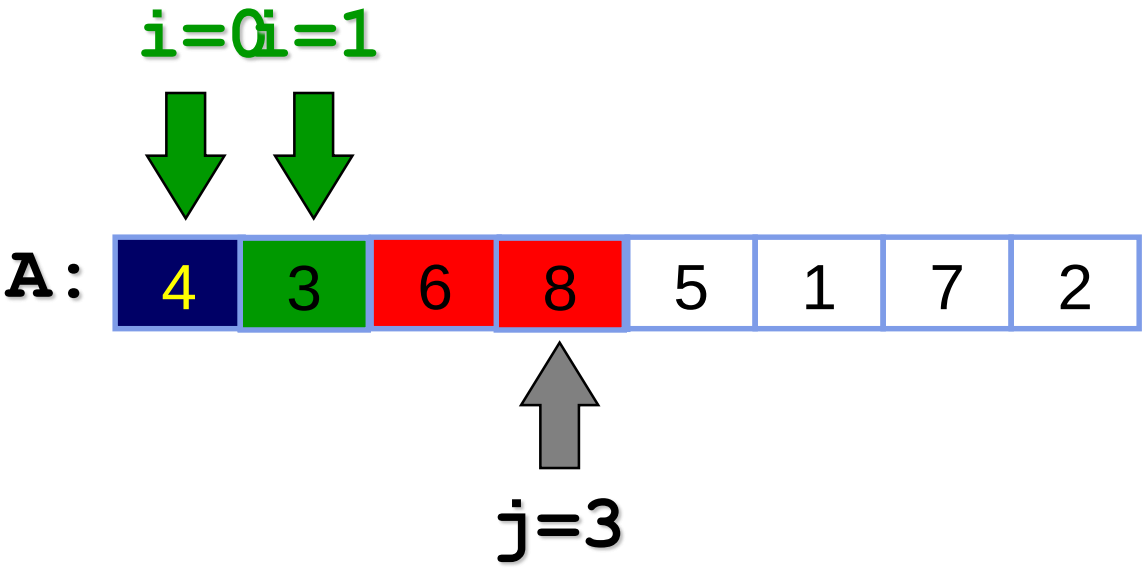




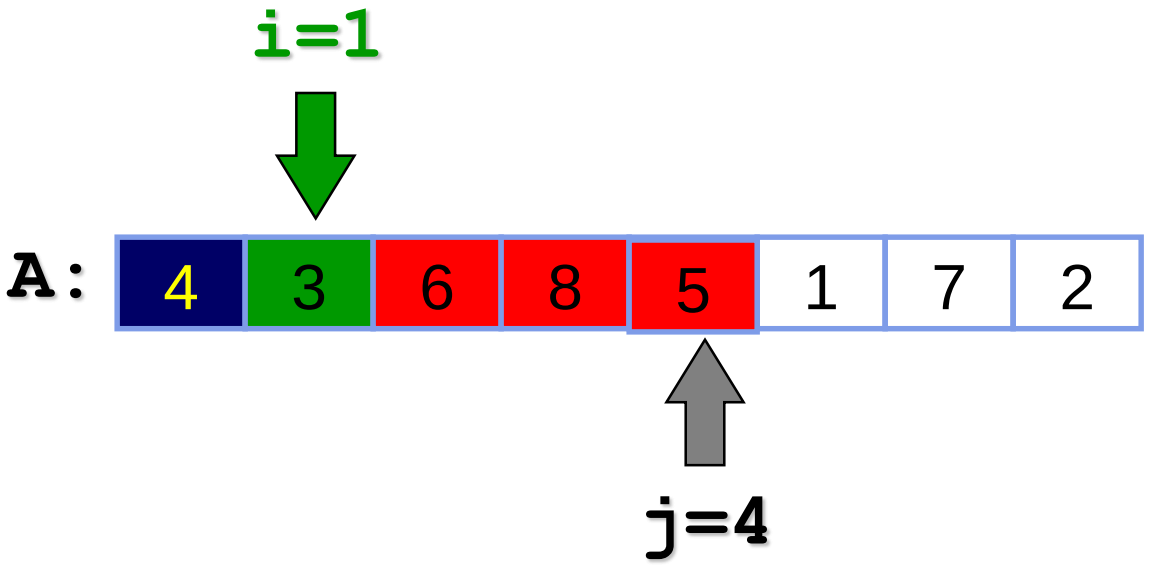
Partition Example



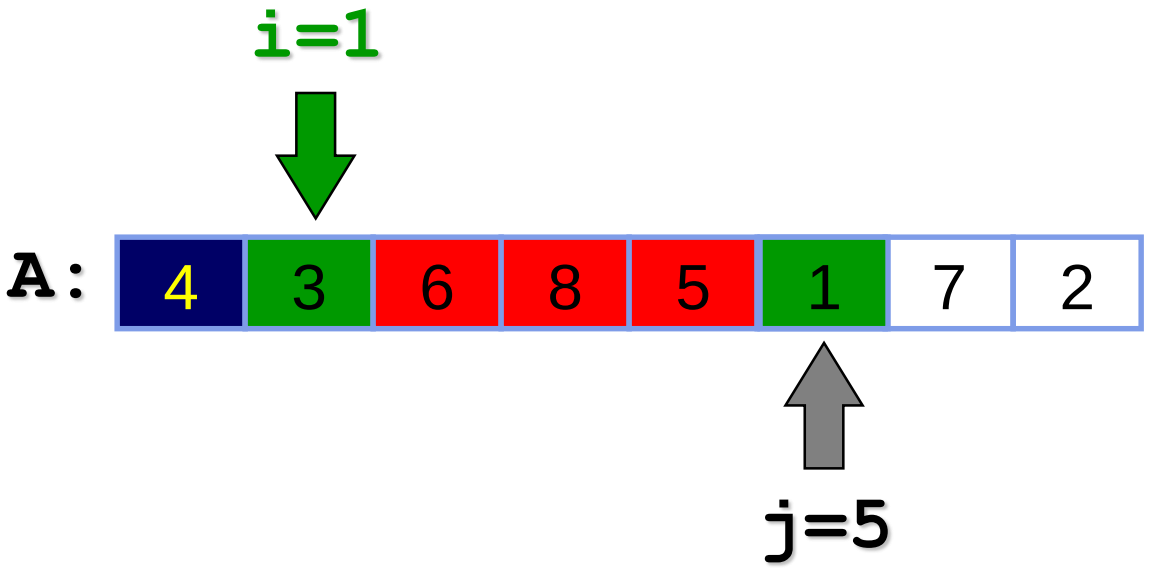
Partition Example



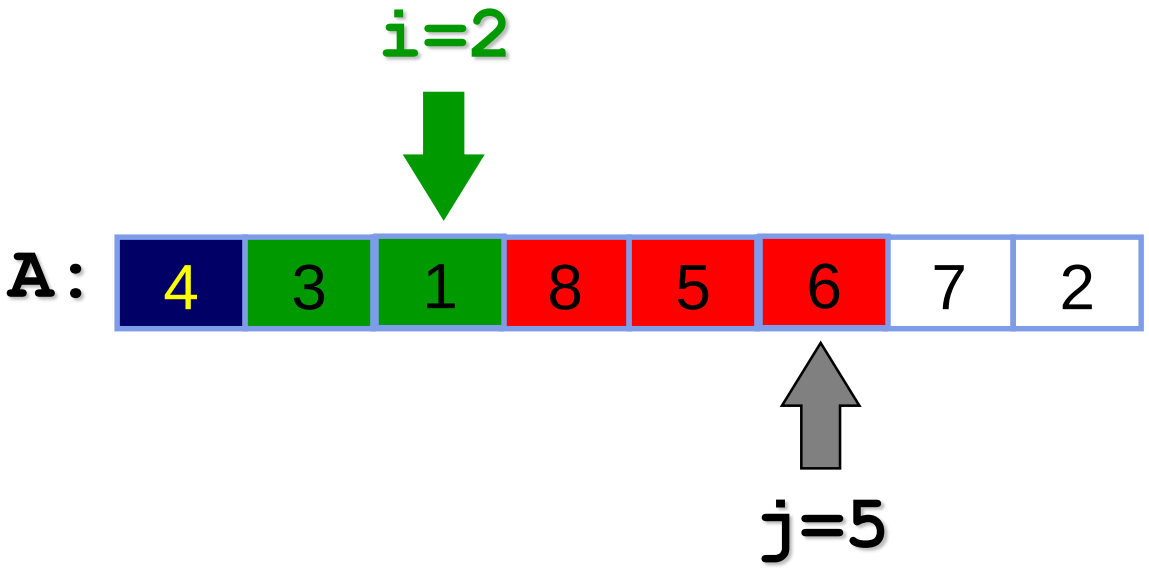
Partition Example



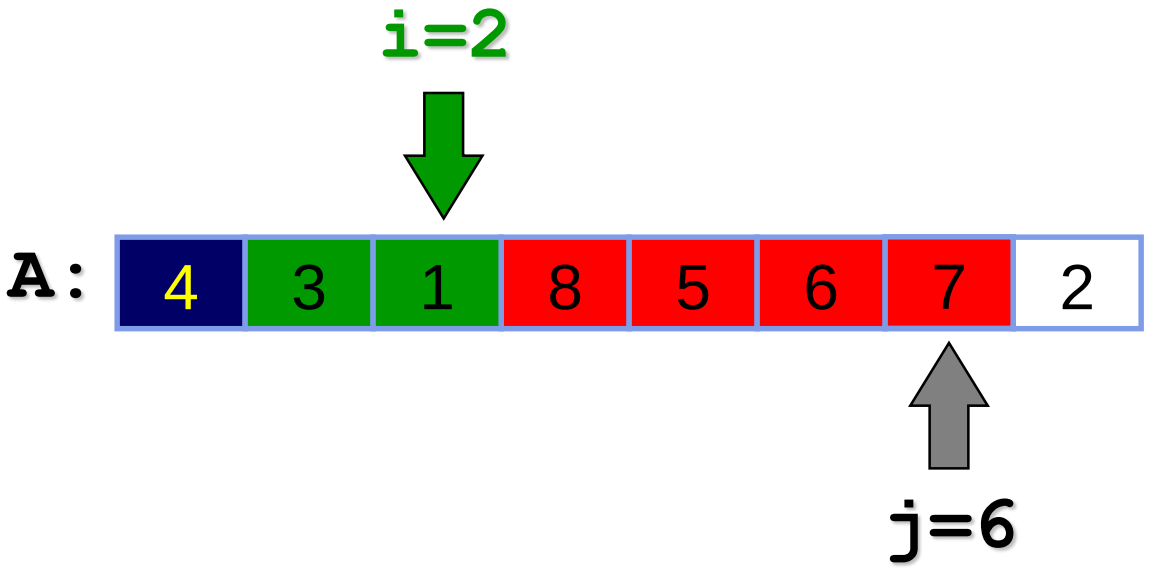
Partition Example



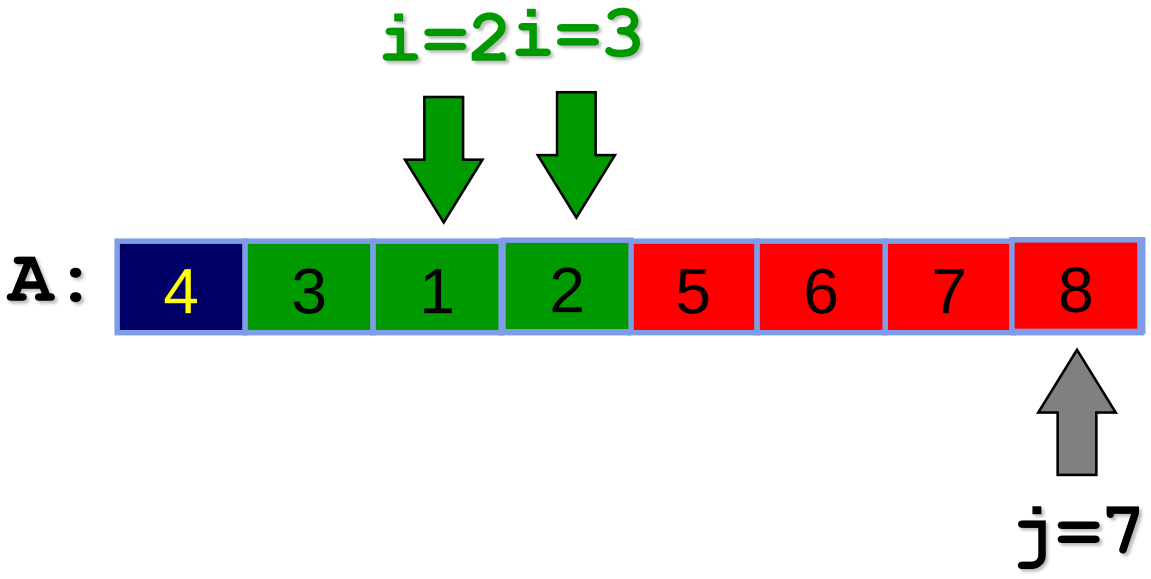
Partition Example



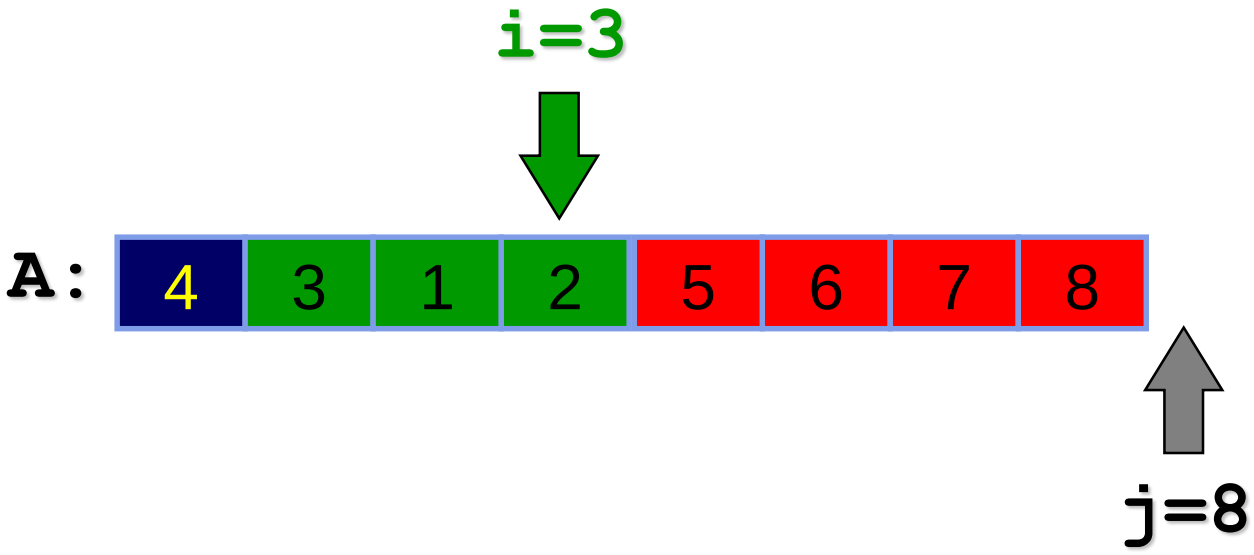
Partition Example



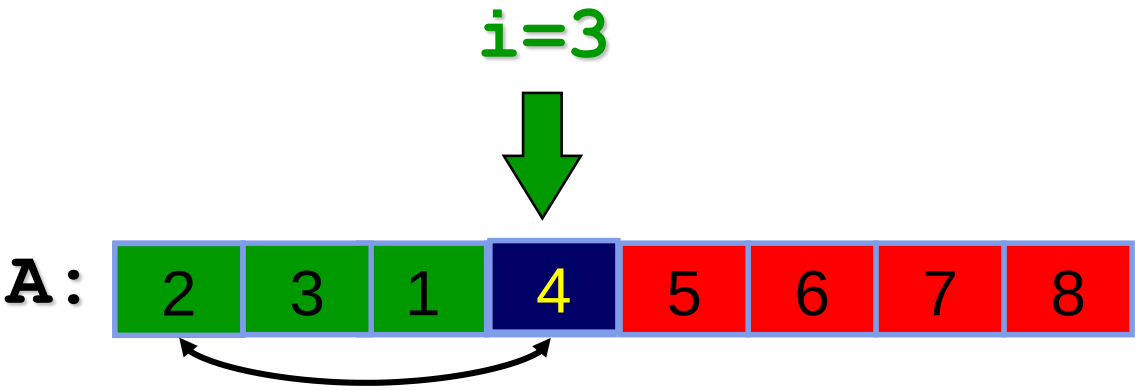
Partition Example



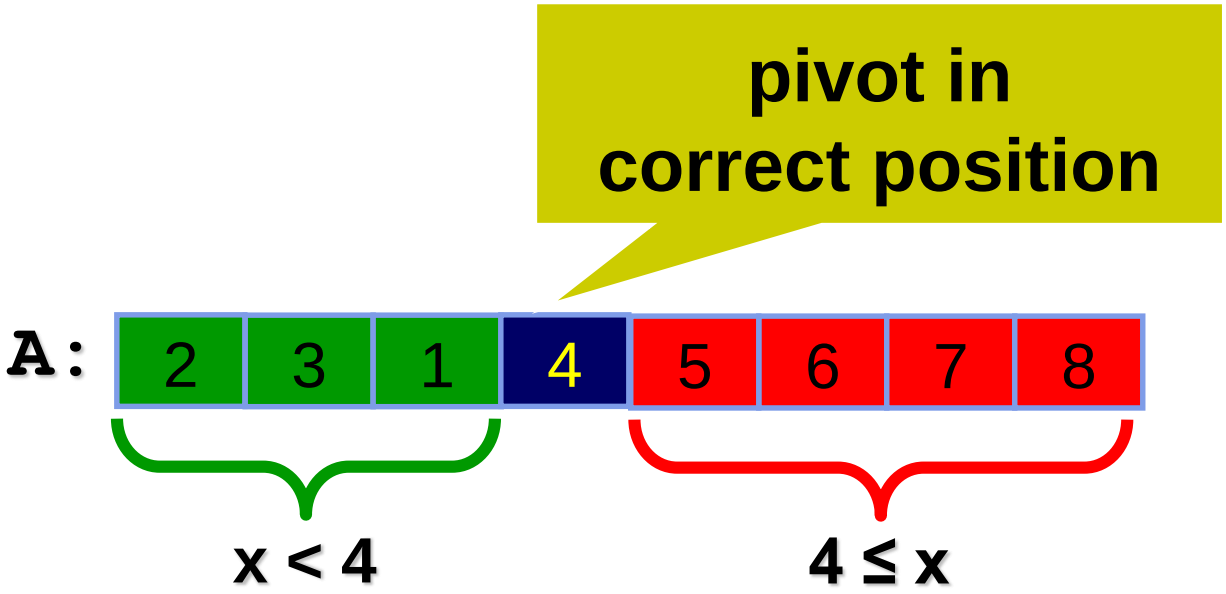
Partition Example

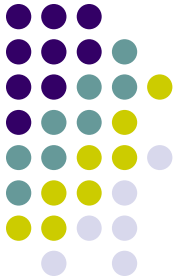


Partition Example



Partition Example



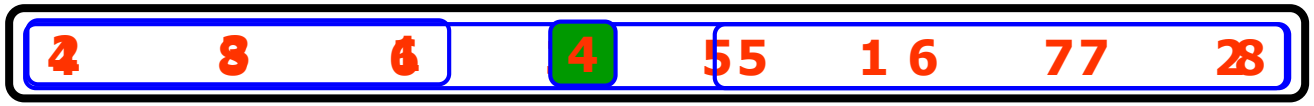


```
Partition(A, left, right)
1.    x ← A[left]
2.    i ← left
3.    for j ← left+1 to right
4.         if A[j] < x then
5.              i ← i + 1
6.              swap(A[i], A[j])
7.         end if
8.    end for j
9.    swap(A[i], A[left])
10.   return i
```

n = right – left +1
Time: cn for some constant c
Space: constant

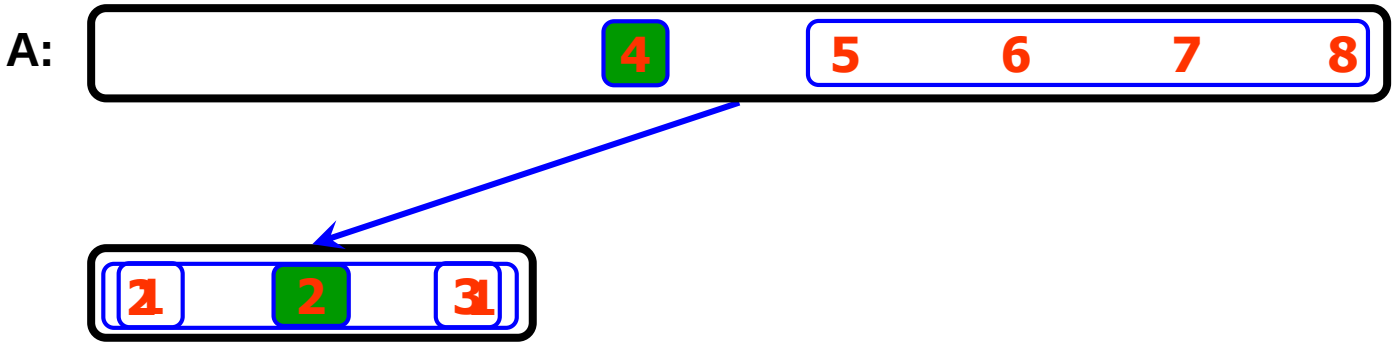
Quick-Sort(A, 0, 7)

Partition

A: 

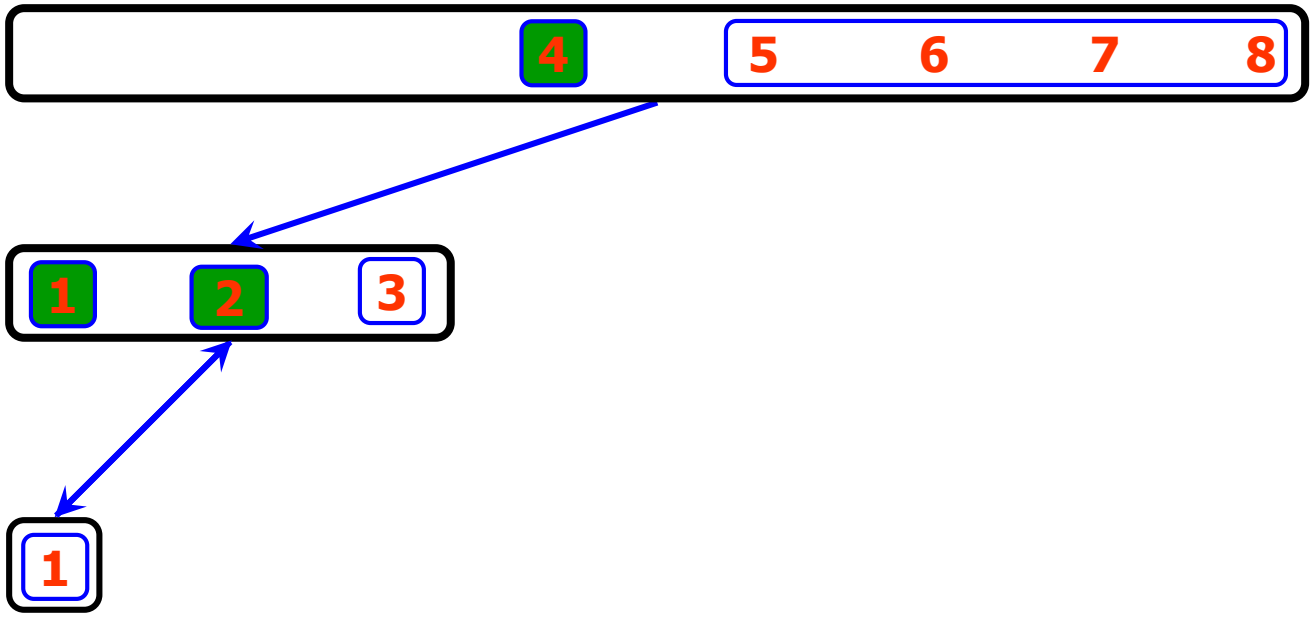
Quick-Sort(A, 0, 7)

Quick-Sort(A, 0, 2), partition



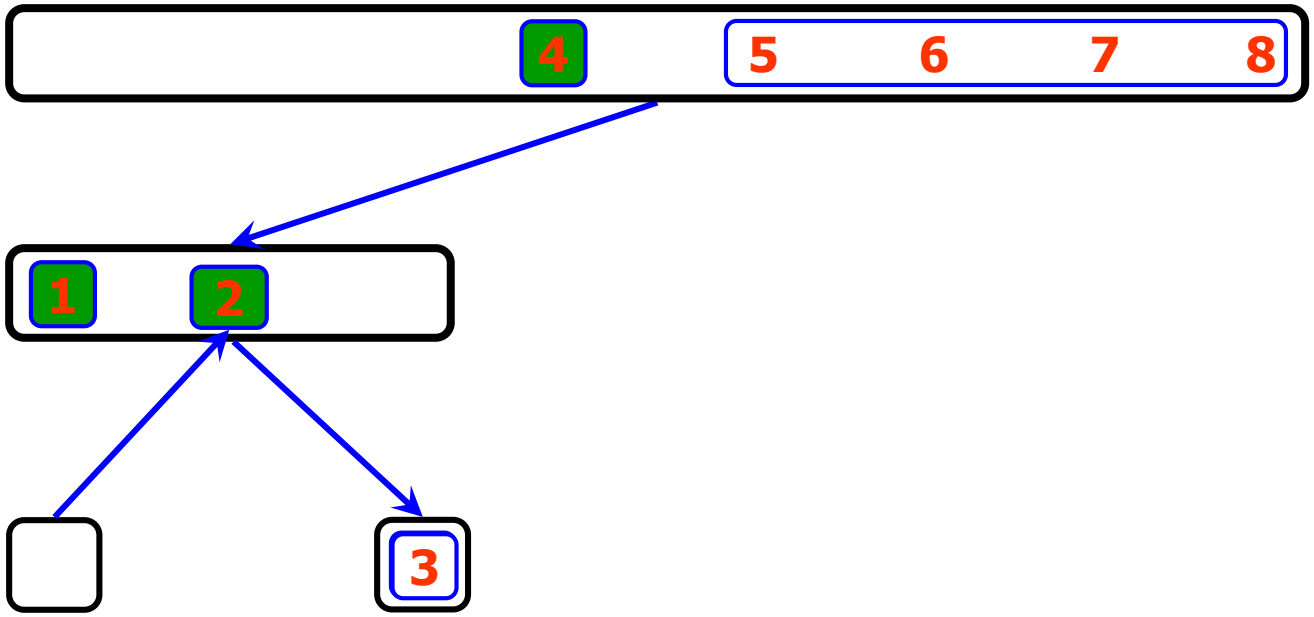
Quick-Sort(A, 0, 7)

Quick-Sort(A, 0, 0), base case



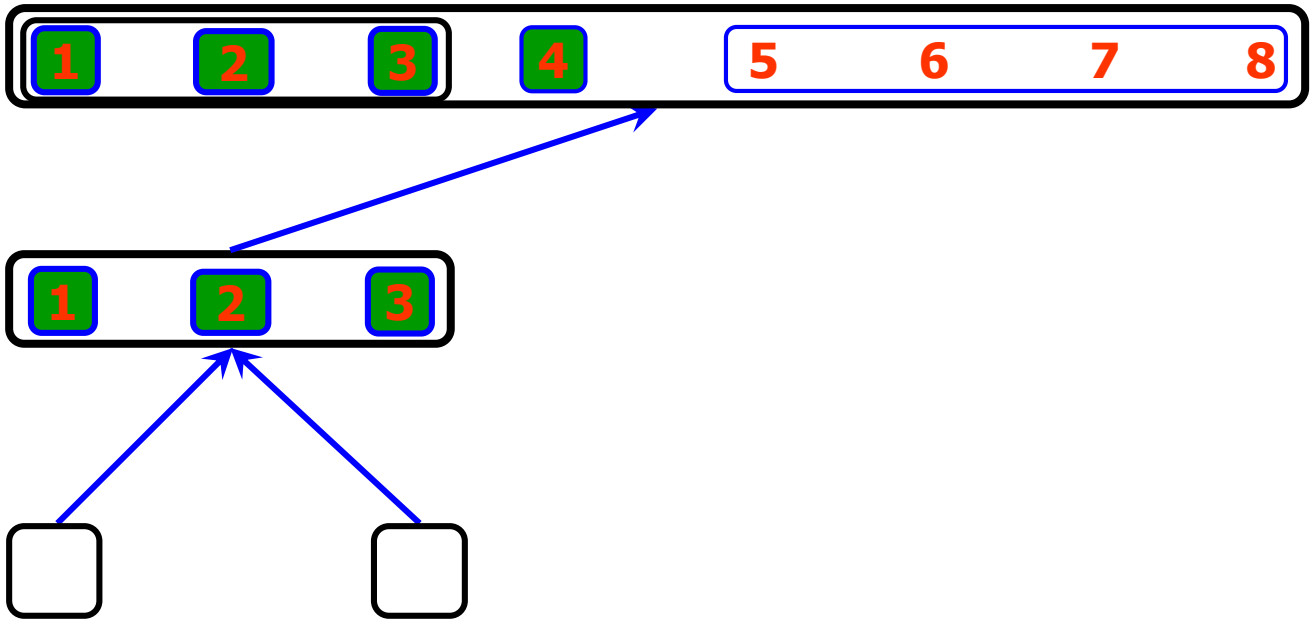
Quick-Sort(A, 0, 7)

Quick-Sort(A, 1, 1), base case



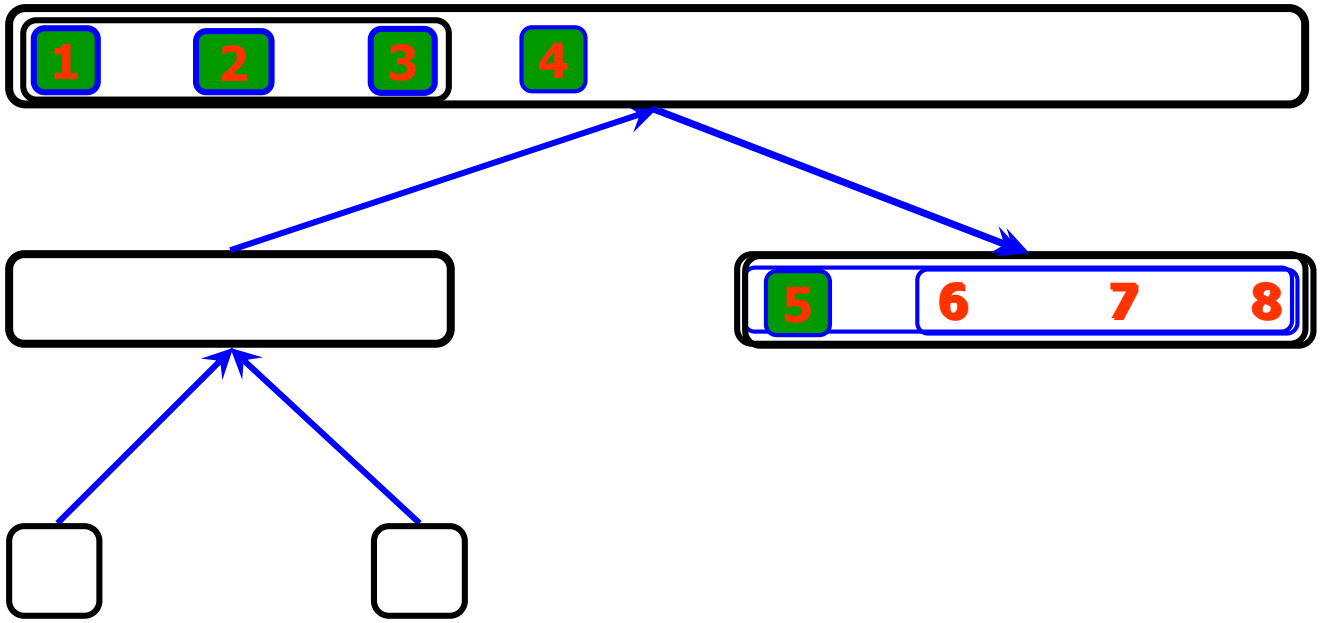
Quick-Sort(A, 0, 7)

Quick-Sort(A, 0, 2), return



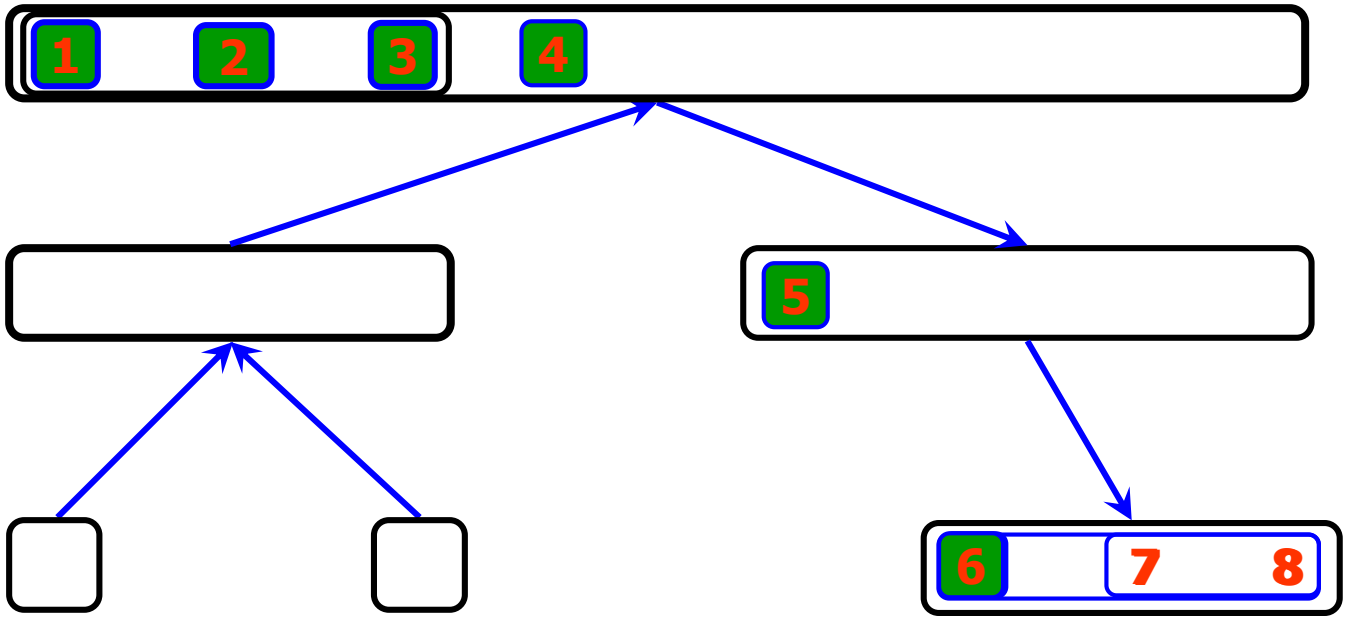
Quick-Sort(A, 0, 7)

Quick-Sort(A, 4, 7) , partition



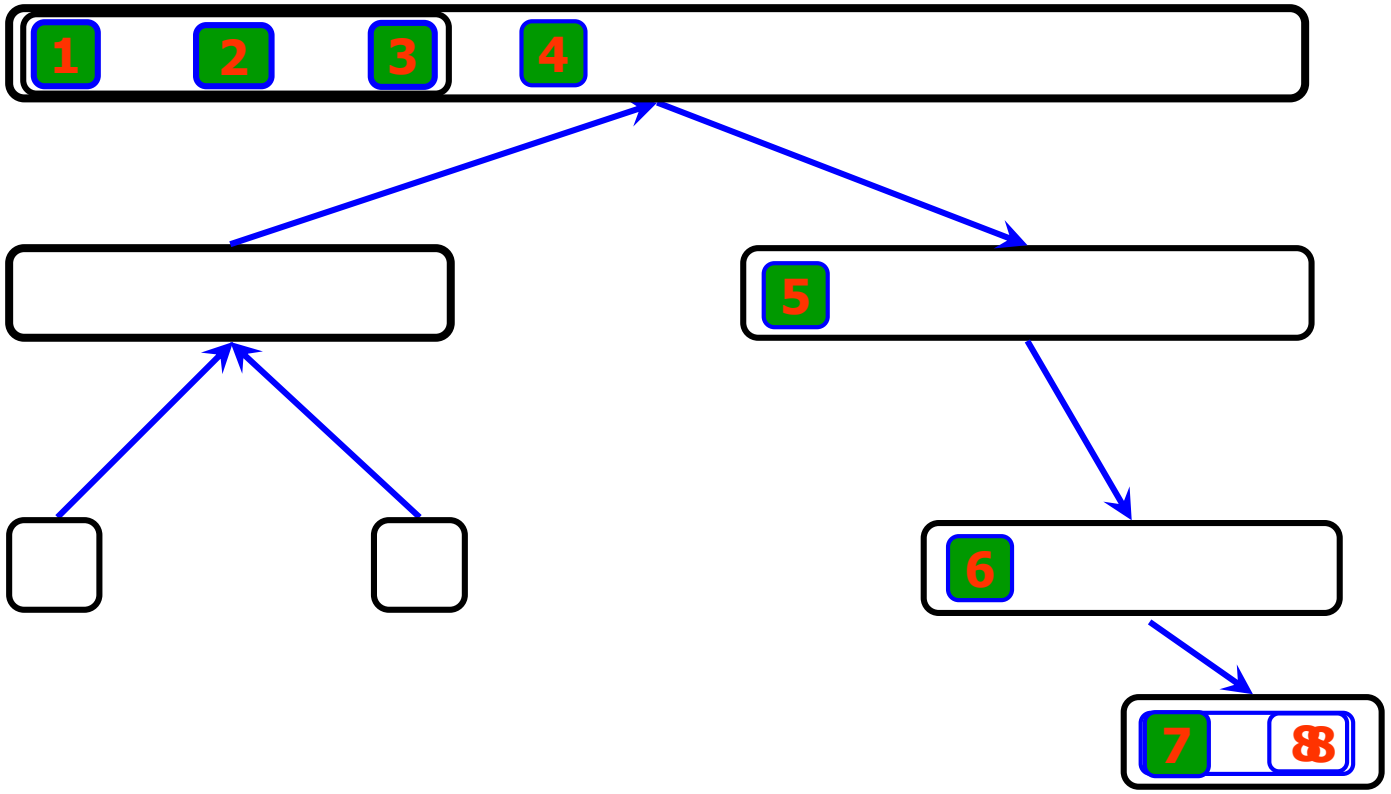
Quick-Sort(A, 0, 7)

Quick-Sort(A, 5, 7) , partition



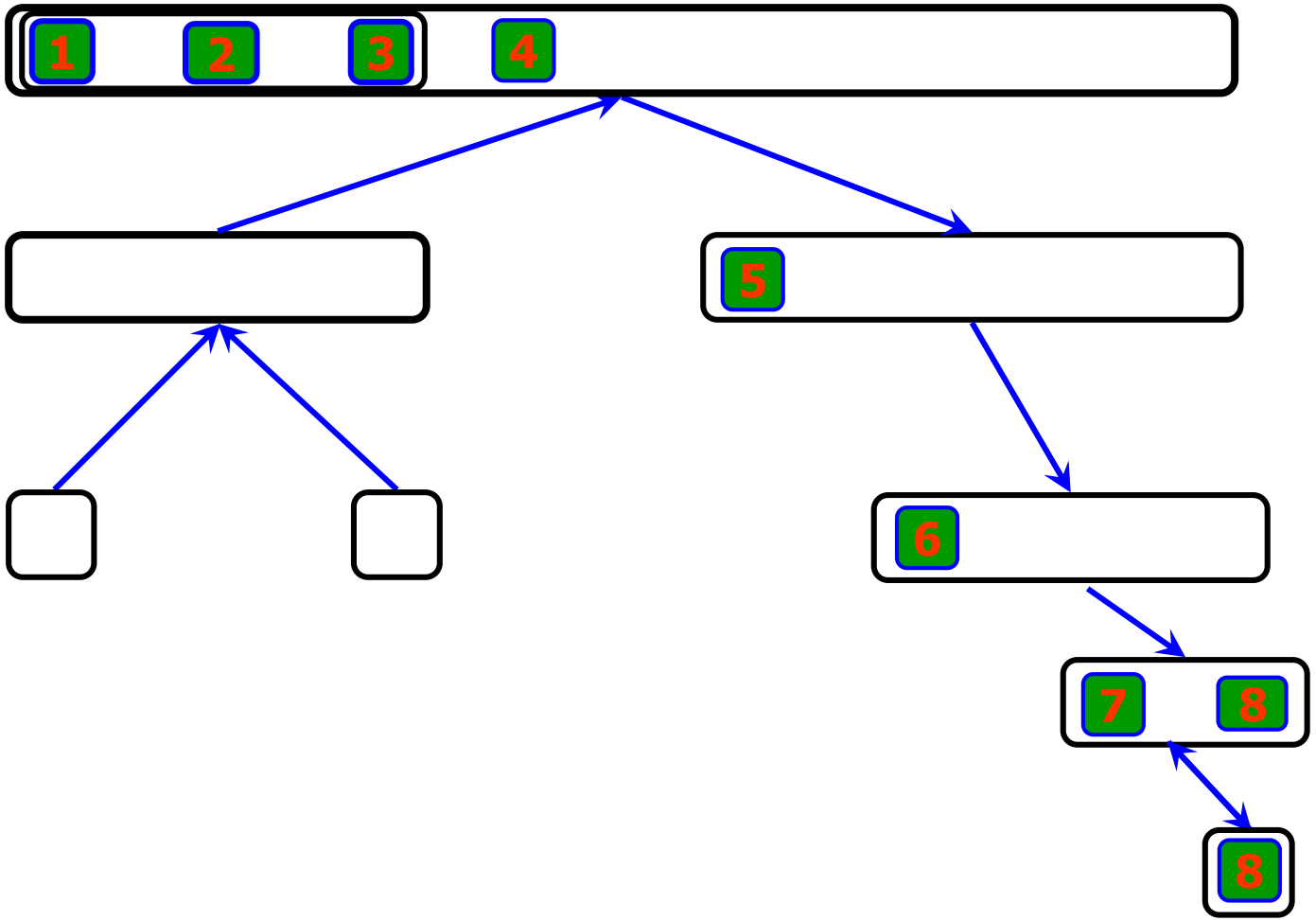
Quick-Sort(A, 0, 7)

Quick-Sort(A, 6, 7) , partition



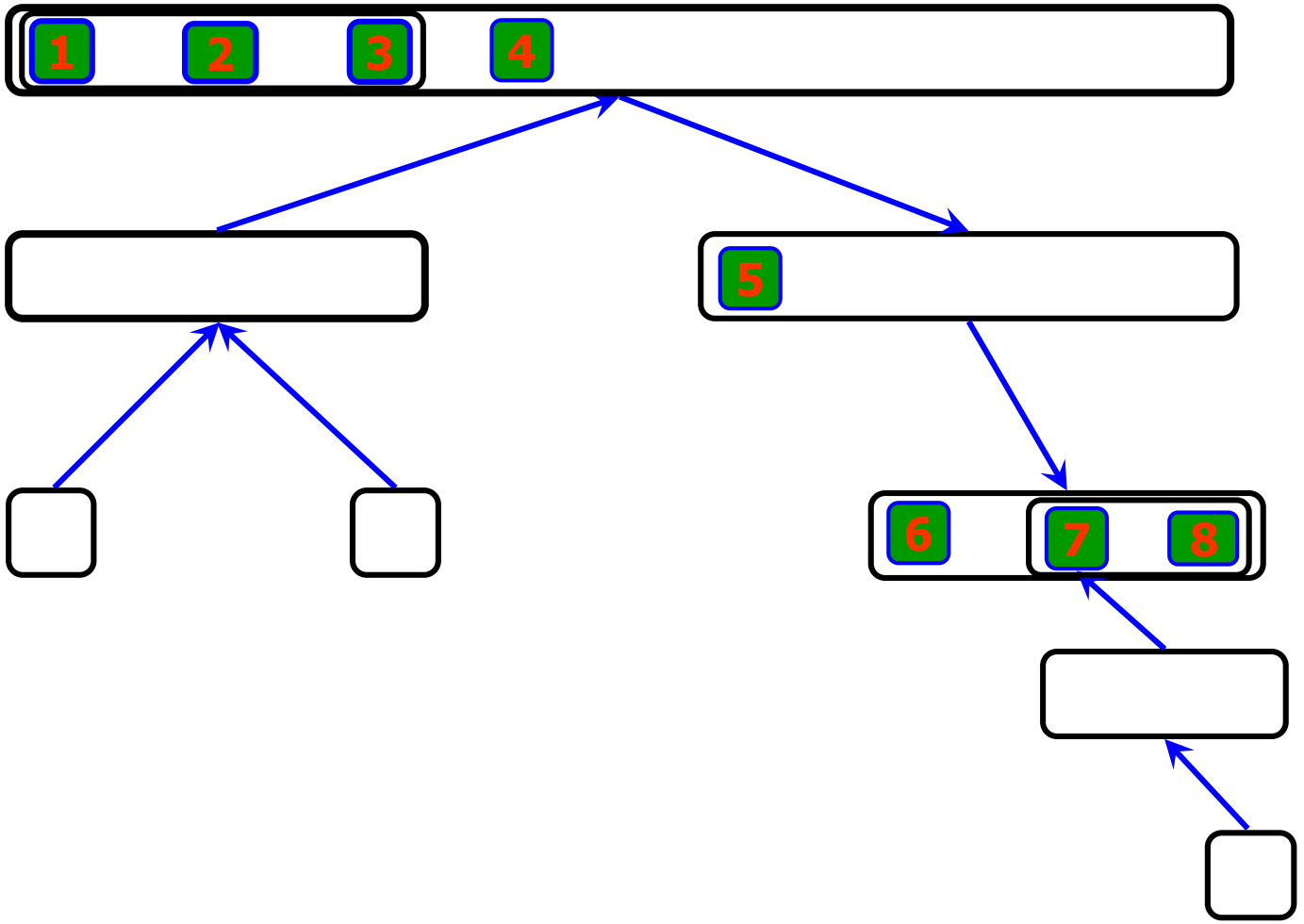
Quick-Sort(A, 0, 7)

Quick-Sort(A, 7, 7) , base case



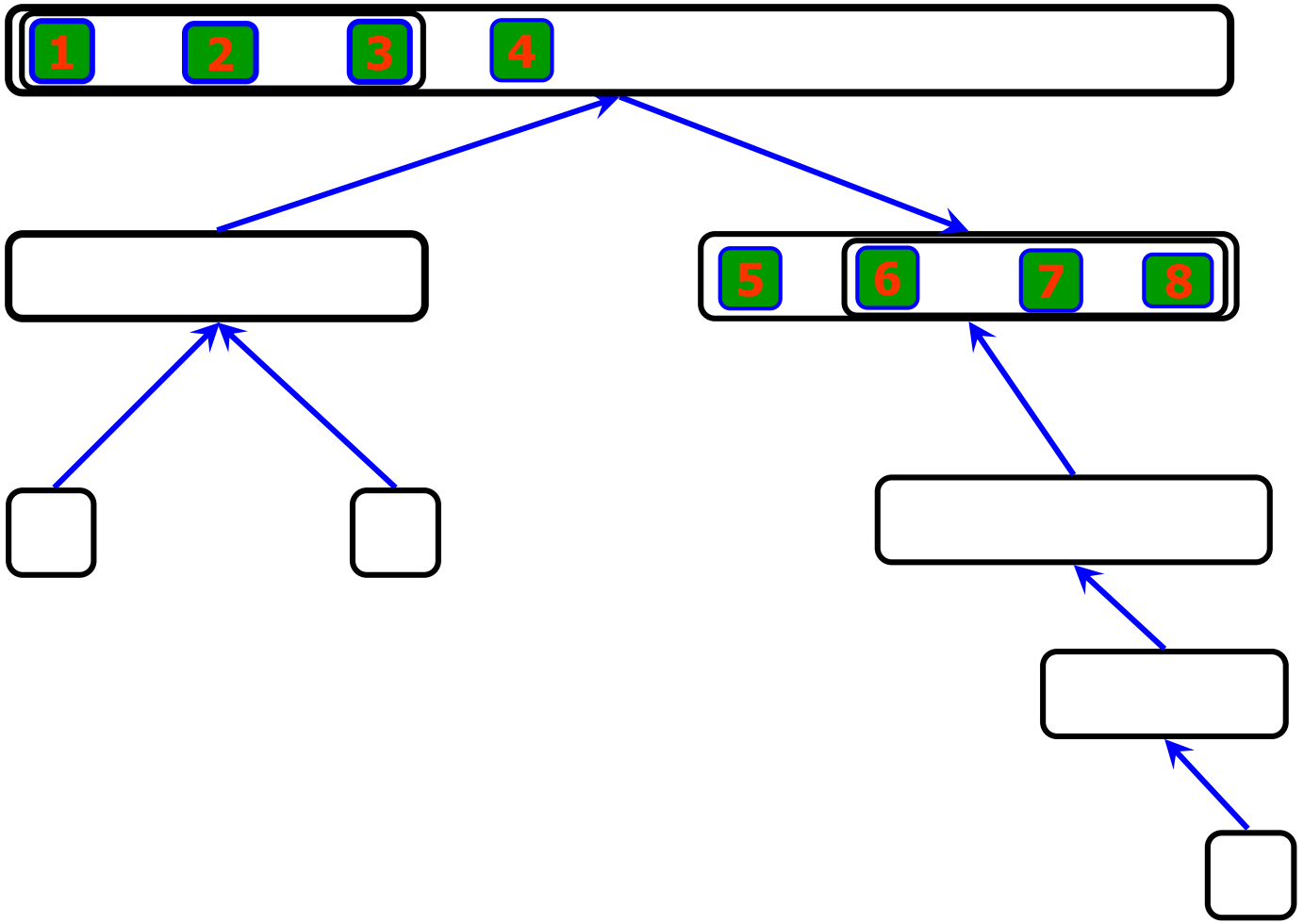
Quick-Sort(A, 0, 7)

Quick-Sort(A, 6, 7) , return



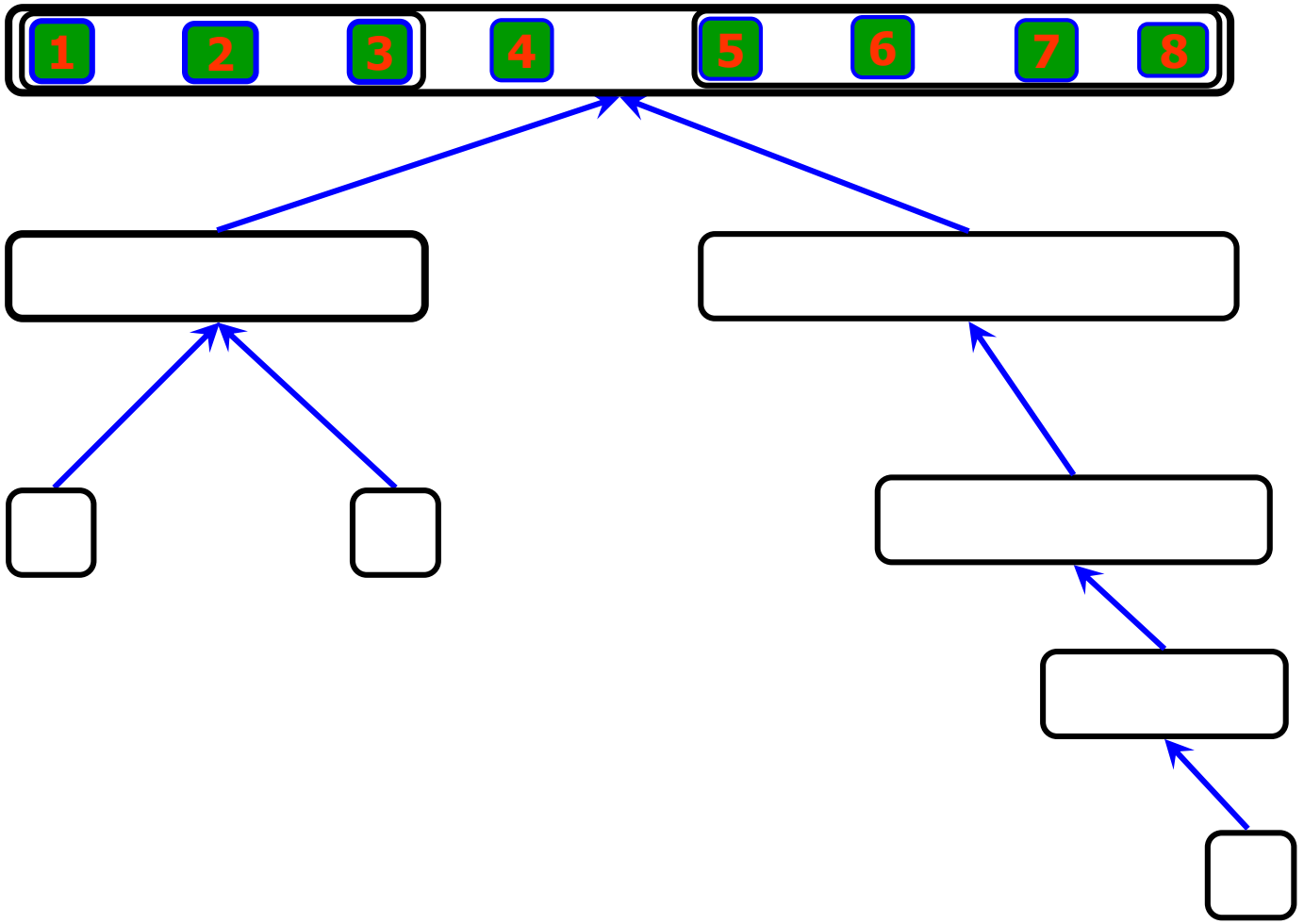
Quick-Sort(A, 0, 7)

Quick-Sort(A, 5, 7) , return



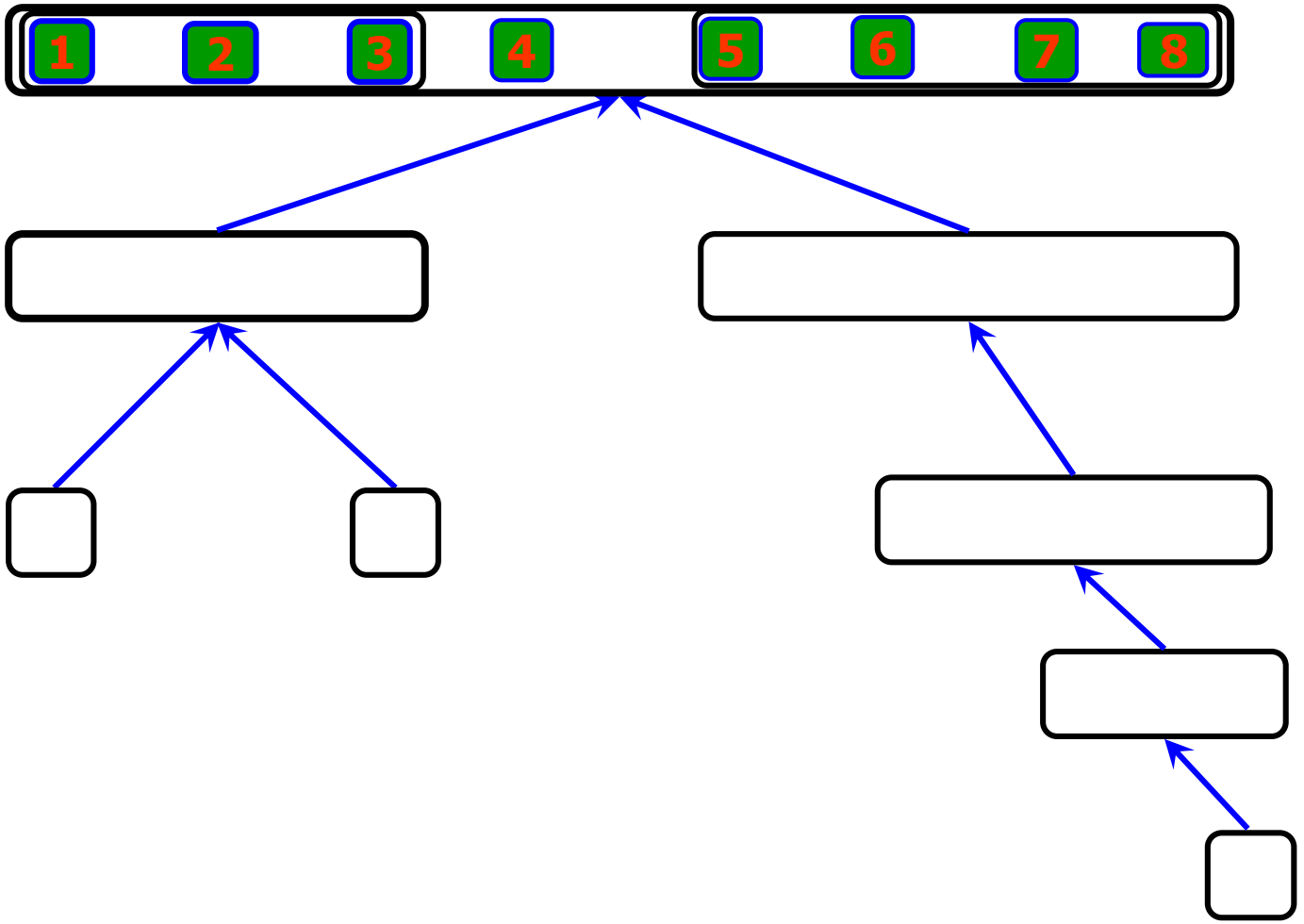
Quick-Sort(A, 0, 7)

Quick-Sort(A, 4, 7) , return



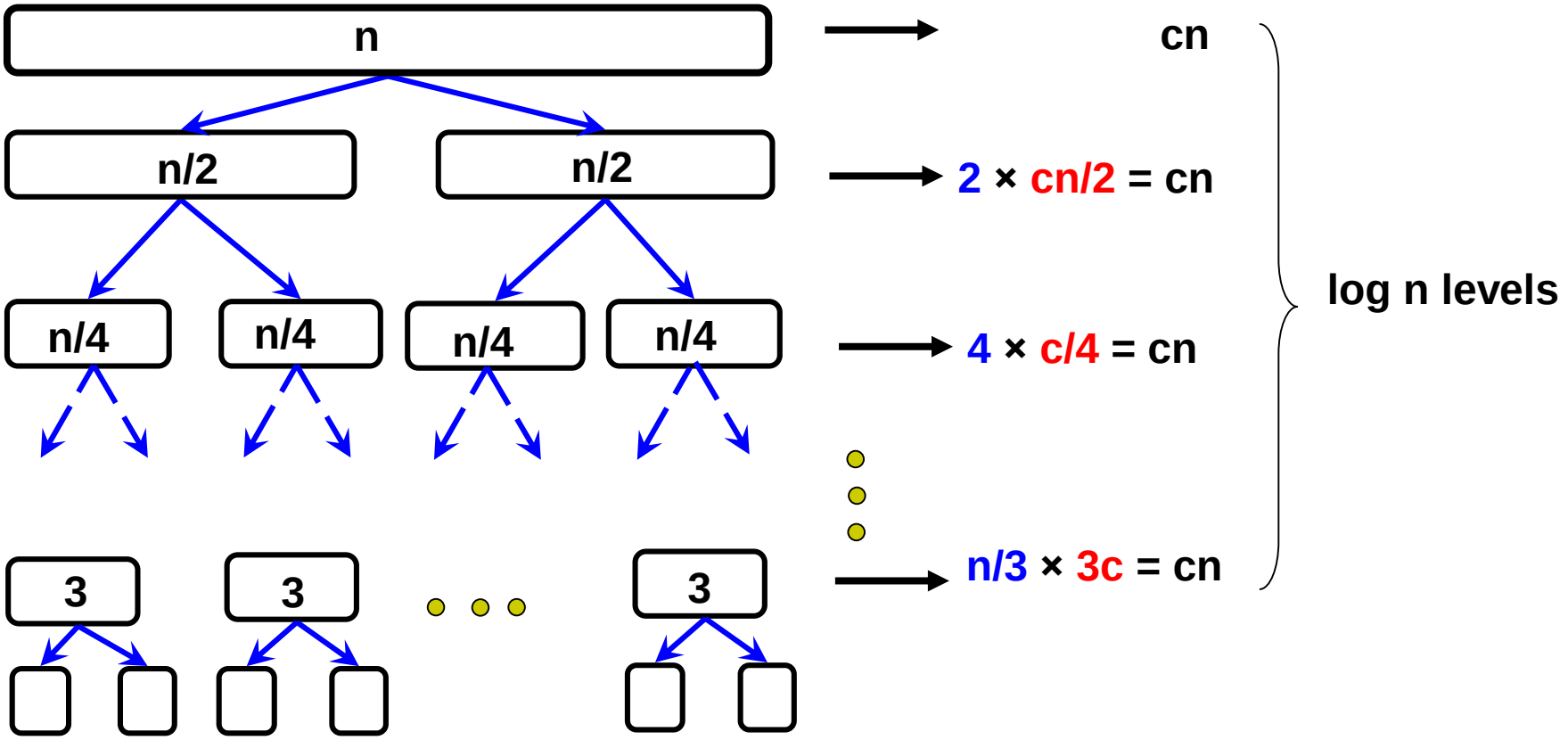
Quick-Sort(A, 0, 7)

Quick-Sort(A, 0, 7) , done!



Quick-Sort: Best Case

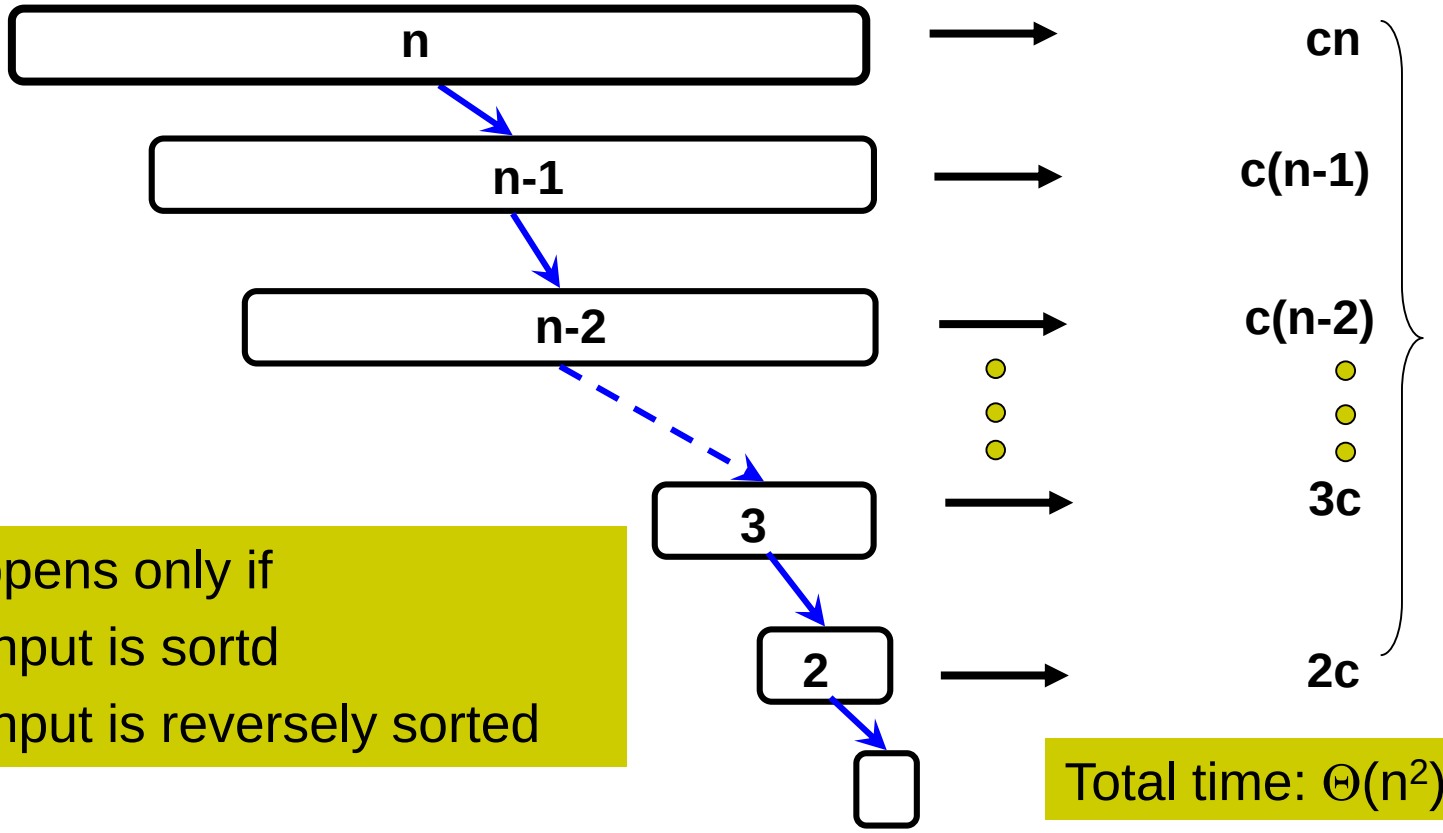
- Even Partition



Total time: Θ(nlogn)

Quick-Sort: Worst Case

- Unbalanced Partition



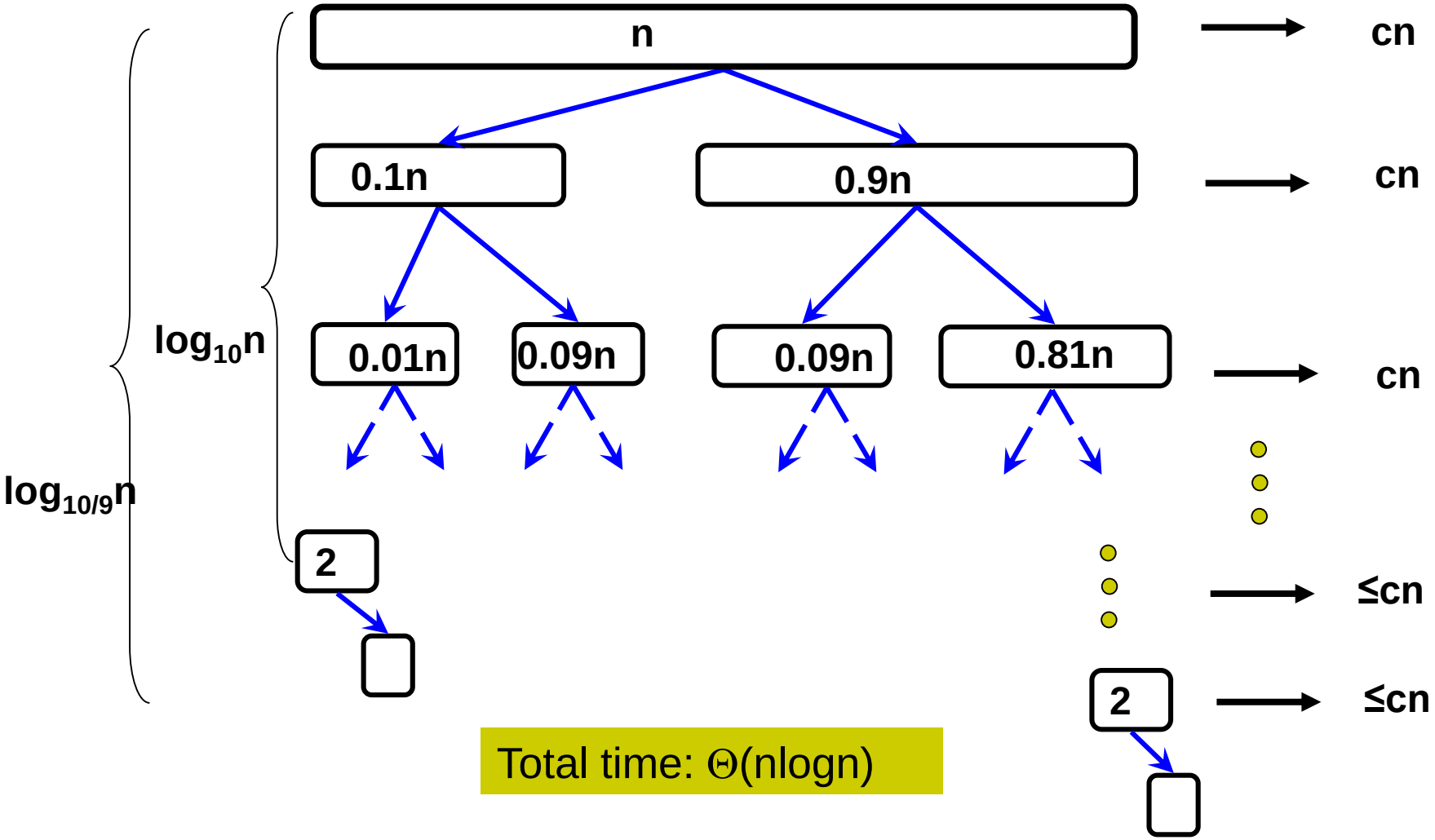
Happens only if

- input is sortd
- input is reversely sorted

Total time: $\Theta(n^2)$

Quick-Sort: an Average Case

- Suppose the split is 1/10 : 9/10



Total time: Θ(nlogn)

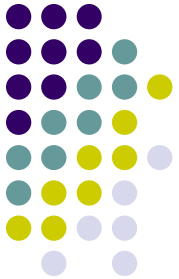
Quick-Sort Summary

- Time
  - Most of the work done in partitioning.
  - Average case takes **$\Theta(n \log(n))$** time.
  - Worst case takes **$\Theta(n^2)$** time

Space

  - Sorts in-place, i.e., does not require additional space

Summary



- Quick-Sort
  - Most of the work done in partitioning
  - Average case takes **$\Theta(n \log(n))$** time
  - Worst case takes **$\Theta(n^2)$** time
  - **$\Theta(1)$ space**