Analysis of Sorting Algorithms

CMSC 142 Lecture 2, Part 2

- https://www.youtube.com/watch?v=ywWBy6J5gz8

Quick Sort

- One of the most elegant sorting algorithms; prevalent in practice
- Uses **divide-and-conquer approach**, but without merge step
- **Idea:** Pick an element of the array (pivot), and re-arrange the array so that values to the left of the pivot are less than the pivot, while values to the right of the pivot are greater than pivot
- Rearrangement puts the pivot into its correct / rightful position
- Recursively solving smaller subproblems takes care of the rest

Quick Sort

Follow three steps recursively:

- Find the pivot that divides the array into two halves
- Quick sort the left half
- Quick sort the right half

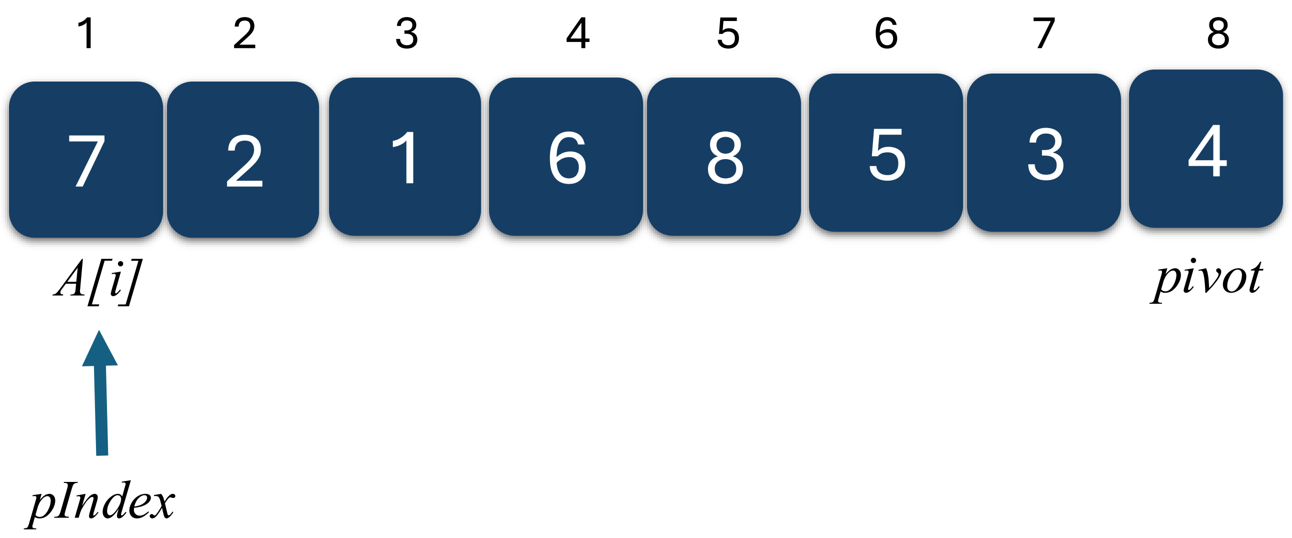
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]        4
    pIndex = start        1
    for i=start to end-1:     i=1
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? ***NO***



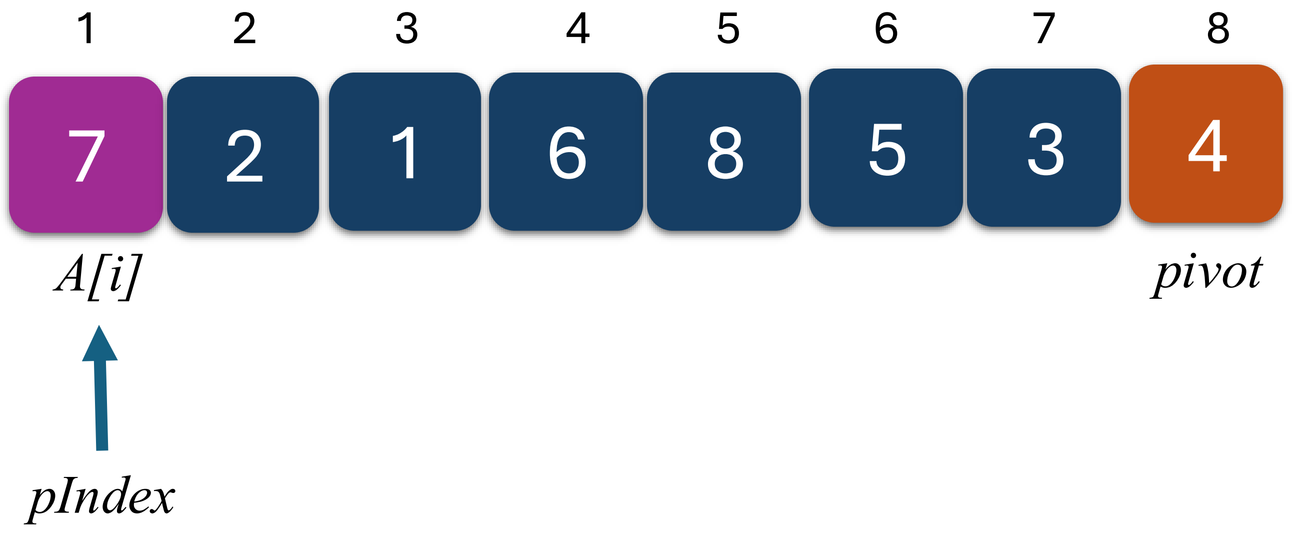
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]            4
    pIndex = start            1
    for i=start to end-1:     i=1
        if A[i] <= pivot:     F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **NO**



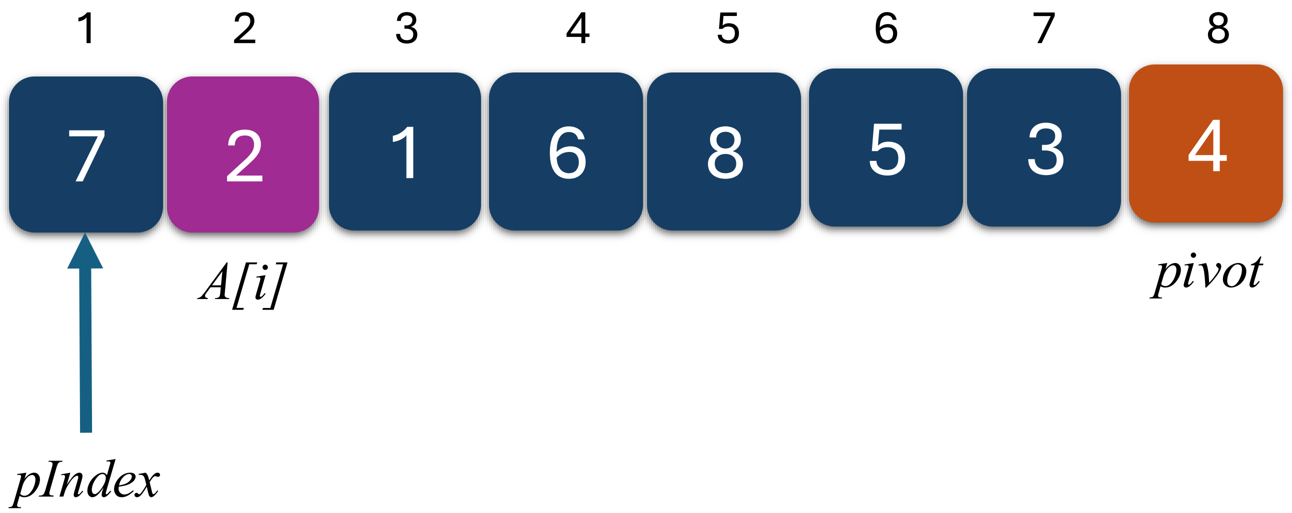
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]              4
    pIndex = start
    for i=start to end-1:       i=2
        if A[i] <= pivot:       T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



Quicksort

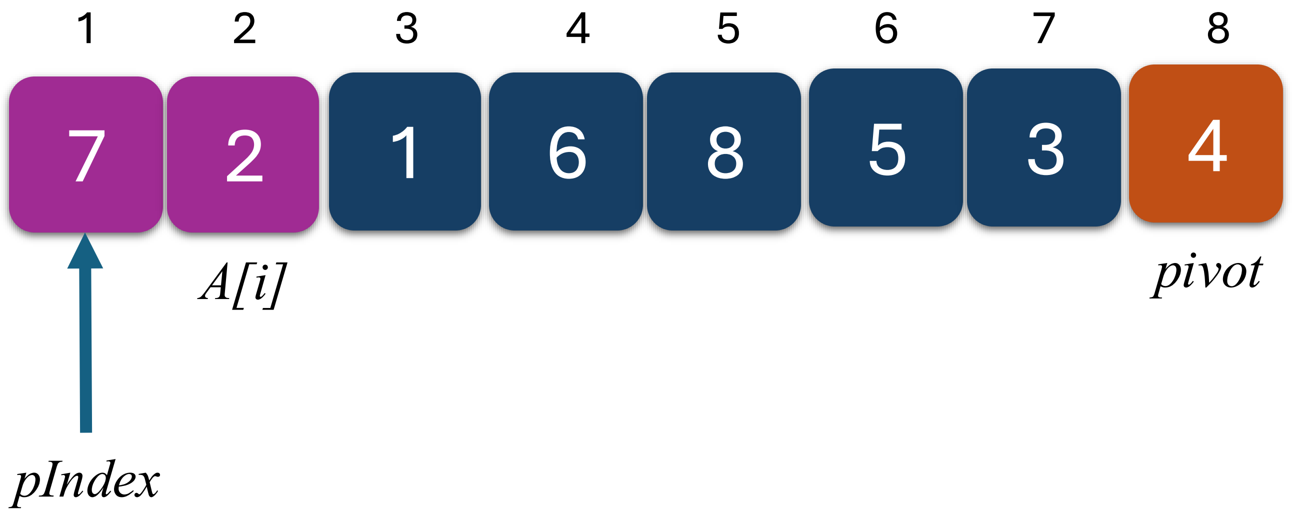
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]   4
    pIndex = start
    for i=start to end-1:   i=2
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? ***YES***



Quicksort

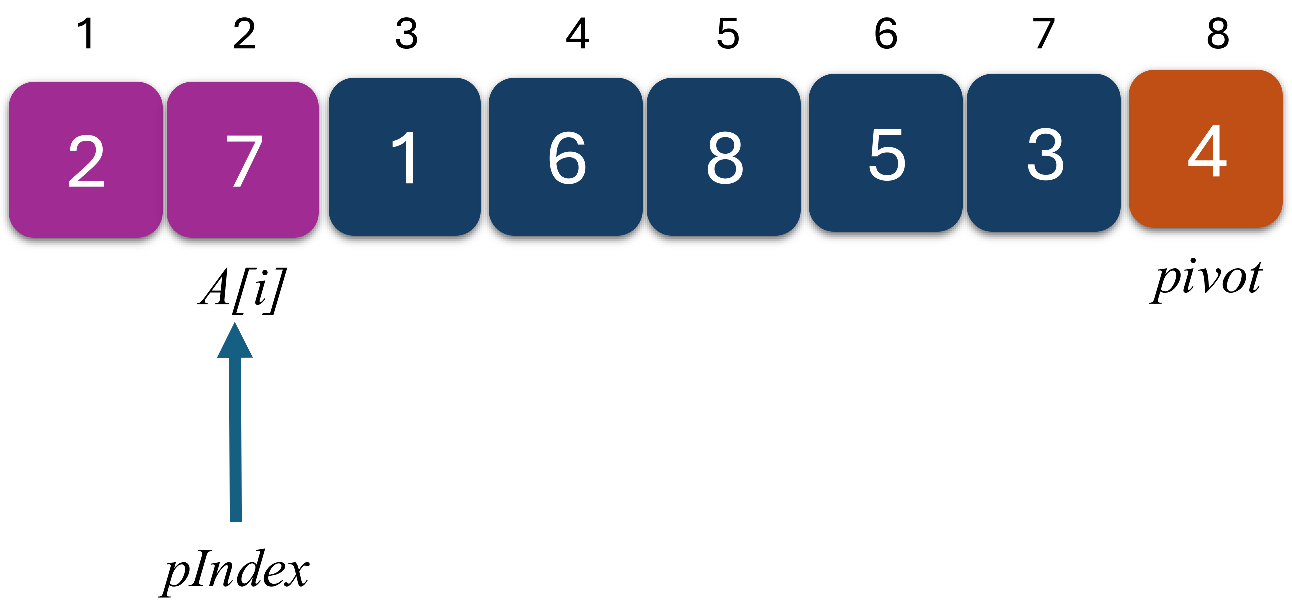
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]   4
    pIndex = start
    for i=start to end-1:   i=2
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   2
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



Quicksort

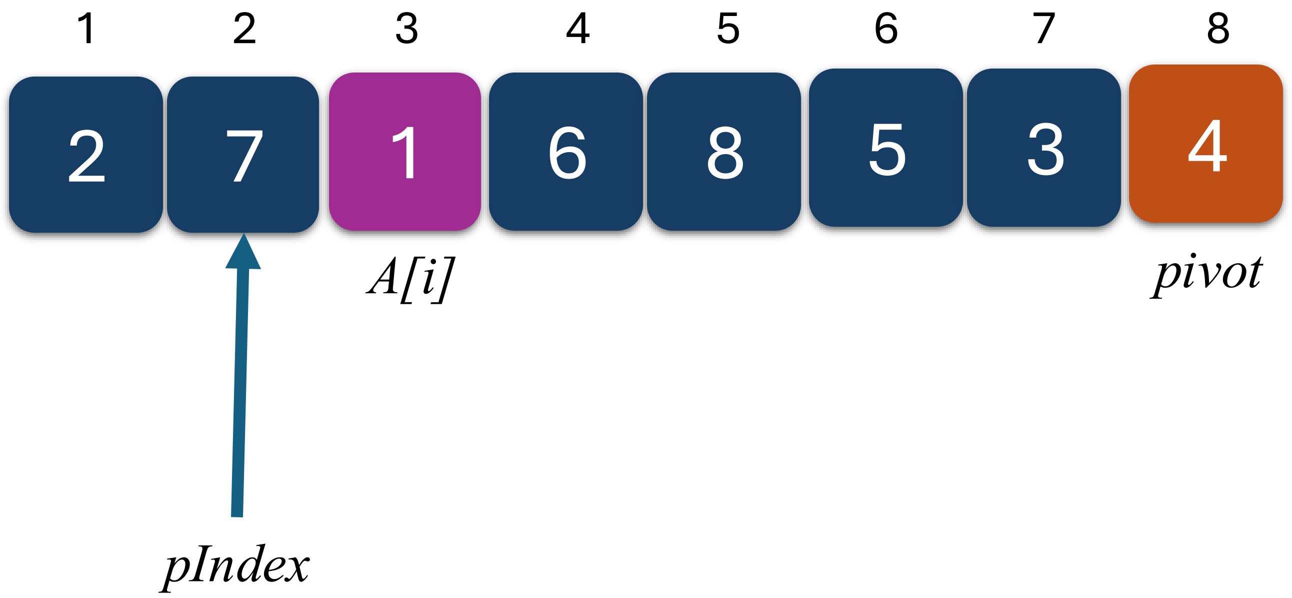
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]            4
    pIndex = start
    for i=start to end-1:     i=3
        if A[i] <= pivot:     T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     2
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? ***YES***



Quicksort

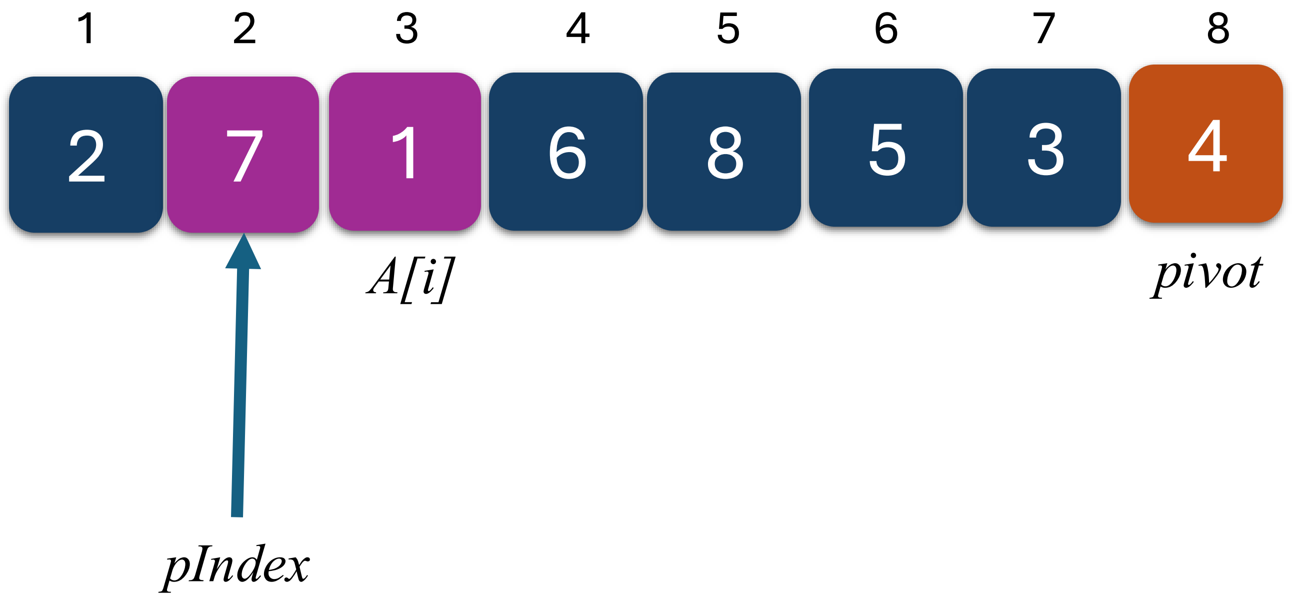
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]            4
    pIndex = start
    for i=start to end-1:     i=3
        if A[i] <= pivot:     T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     2
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



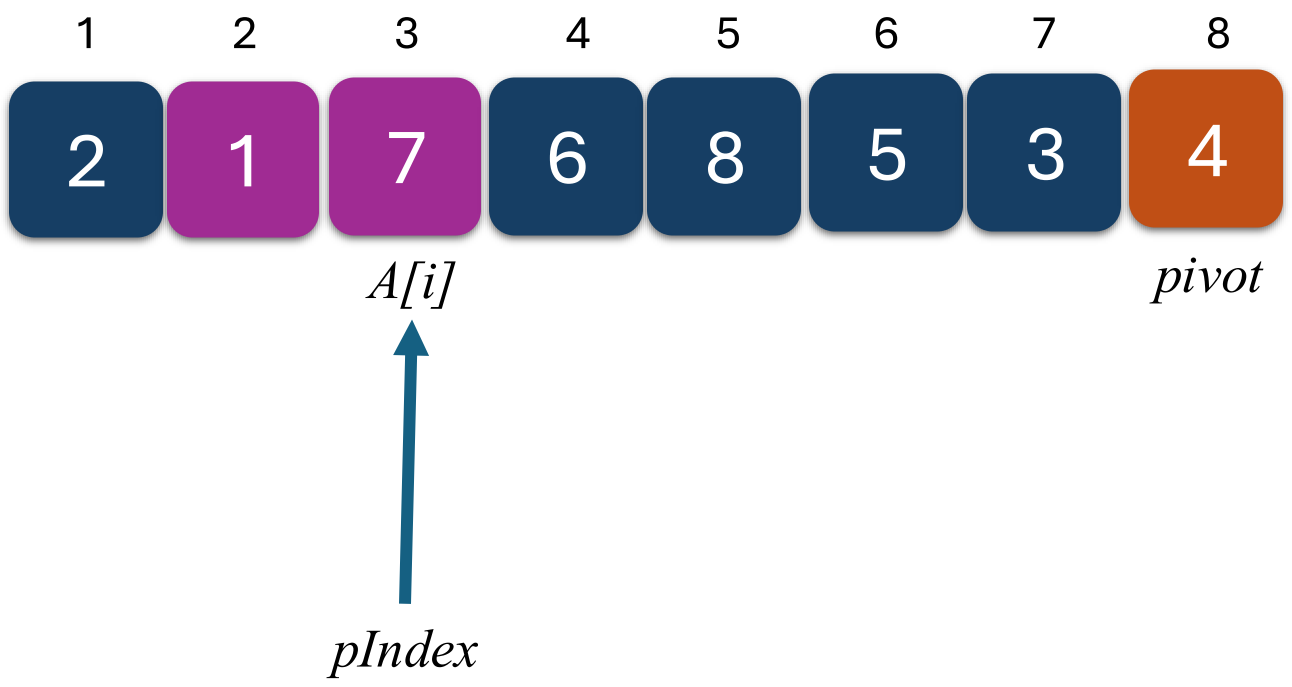
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]                4
    pIndex = start
    for i=start to end-1:         i=3
        if A[i] <= pivot:         T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     3
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



Quicksort

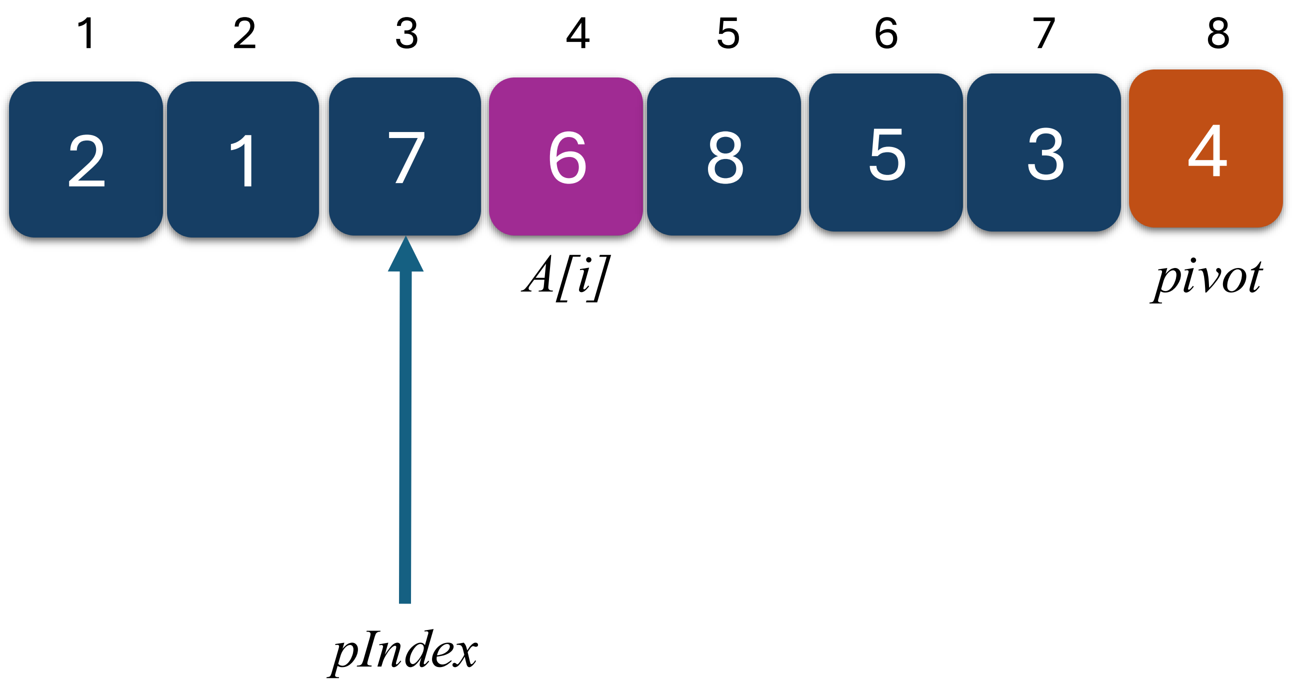
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]              4
    pIndex = start
    for i=start to end-1:       i=4
        if A[i] <= pivot:       F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   3
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **NO**



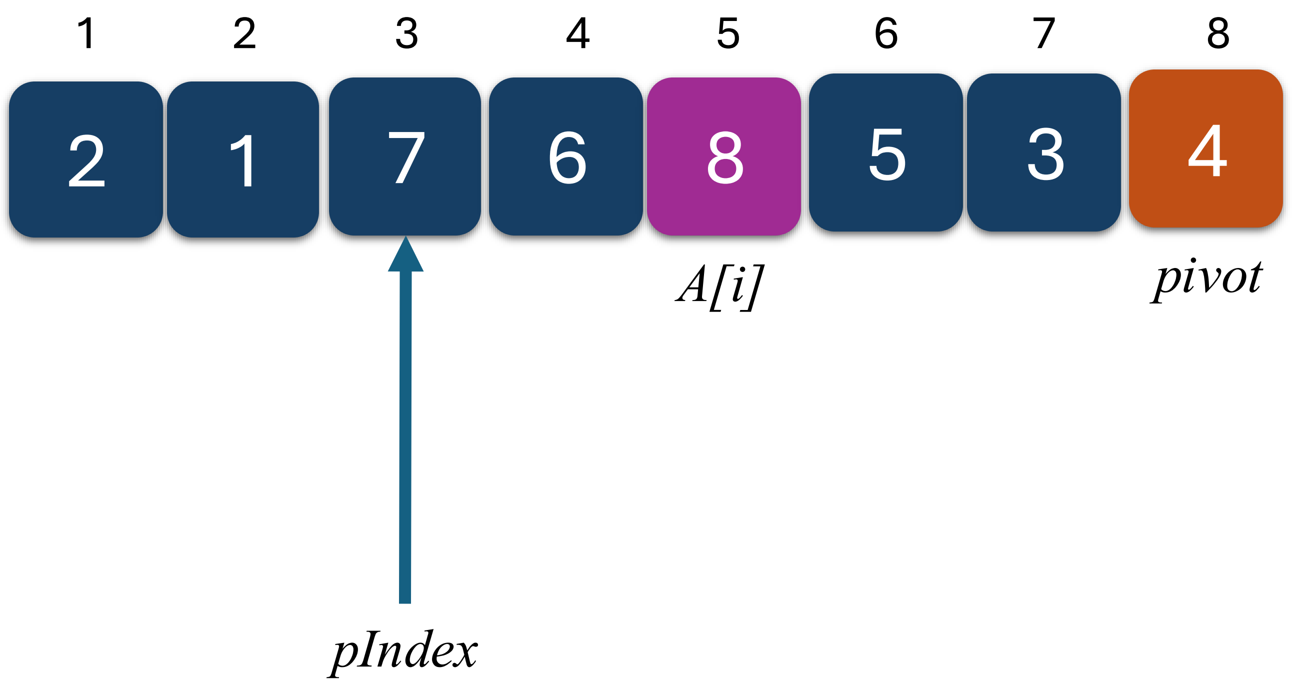
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]              4
    pIndex = start
    for i=start to end-1:       i=5
        if A[i] <= pivot:       F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   3
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **NO**



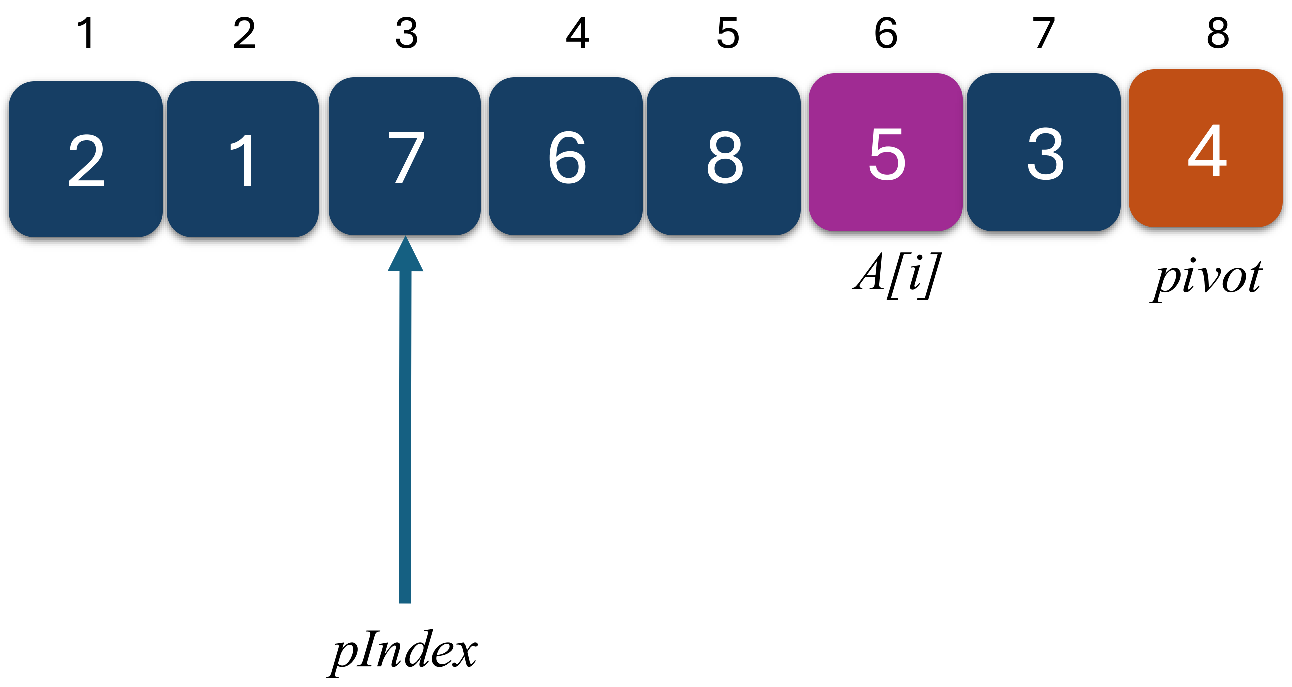
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]                 4
    pIndex = start
    for i=start to end-1:          i=6
        if A[i] <= pivot:          F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1      3
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **NO**



Quicksort

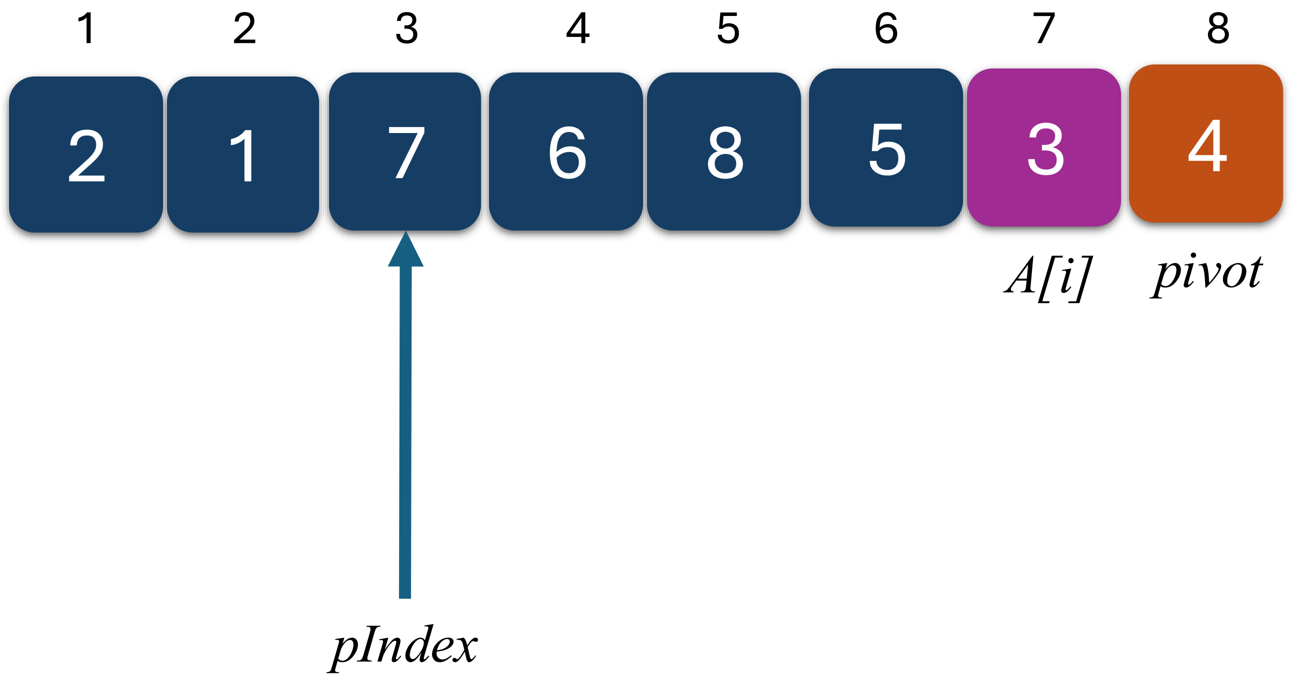
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]  4
    pIndex = start
    for i=start to end-1:  i=7
        if A[i] <= pivot:  T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1  3
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



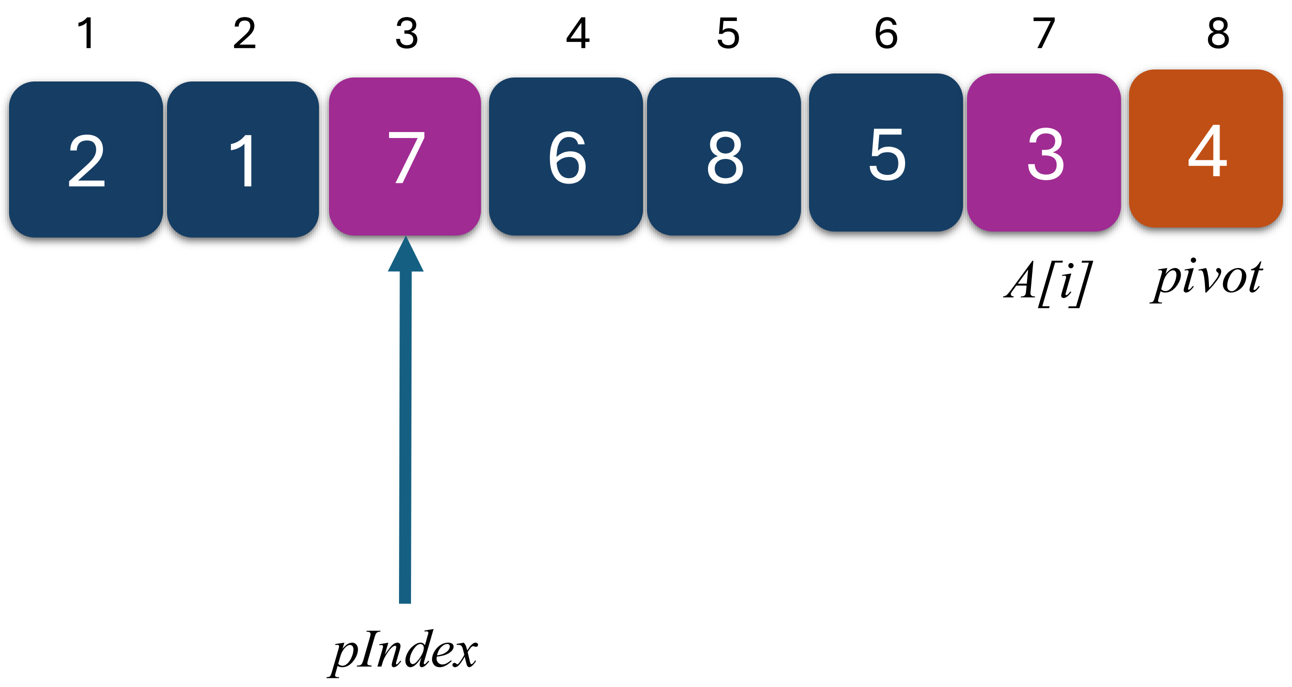
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]                4
    pIndex = start
    for i=start to end-1:         i=7
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1         4
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? **YES**



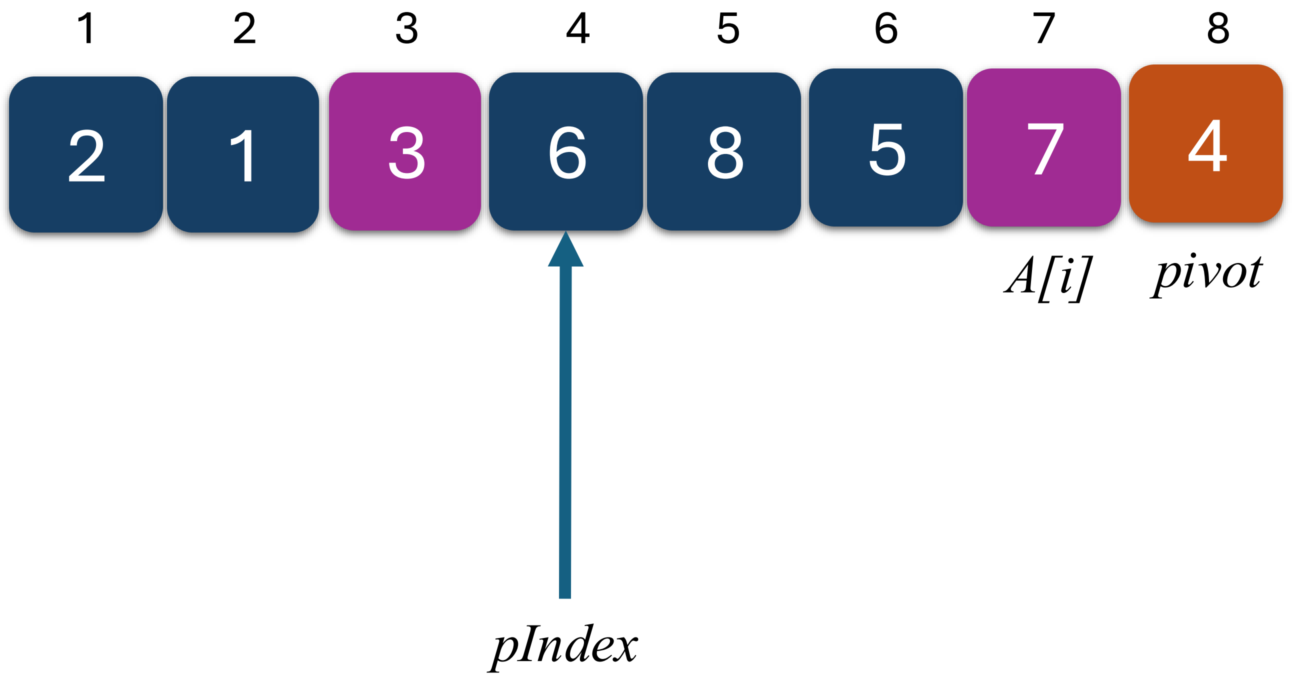
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]   4
    pIndex = start
    for i=start to end-1:   i=7
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   4
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? ***YES***



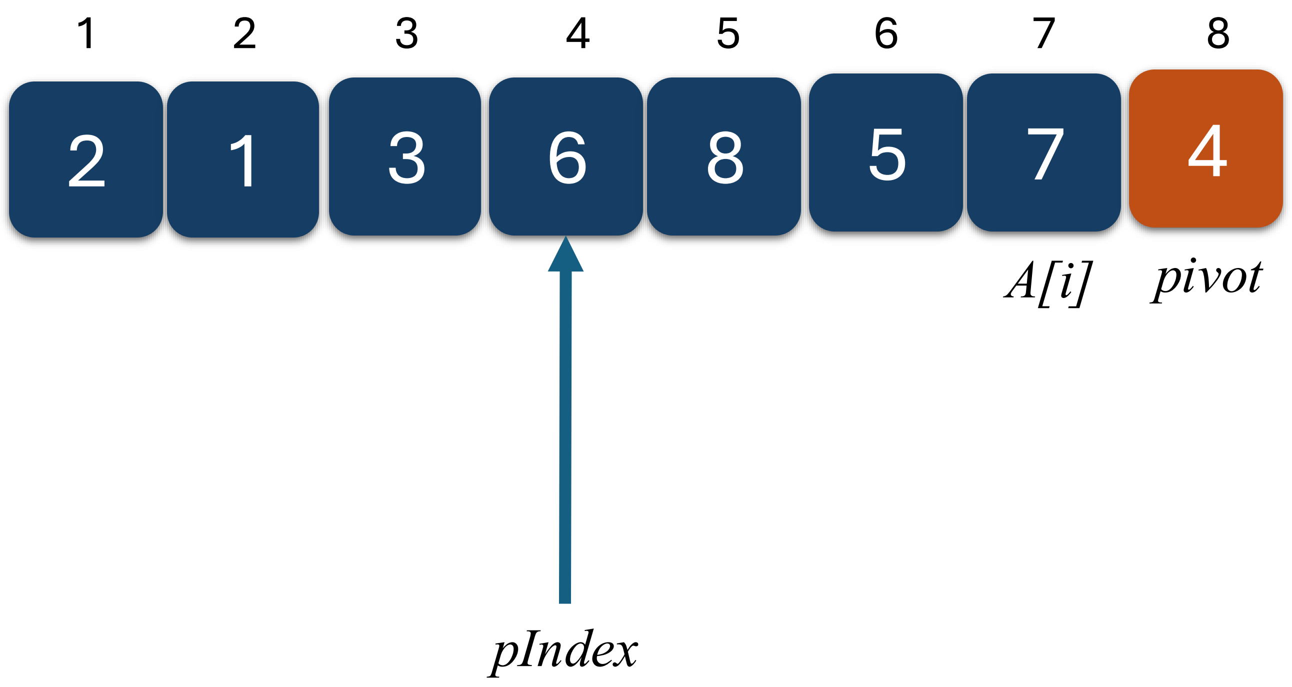
Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]  4
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1  4
    swap(A[pIndex], pivot)
    return pIndex
```

A[i] <= pivot? ***YES***

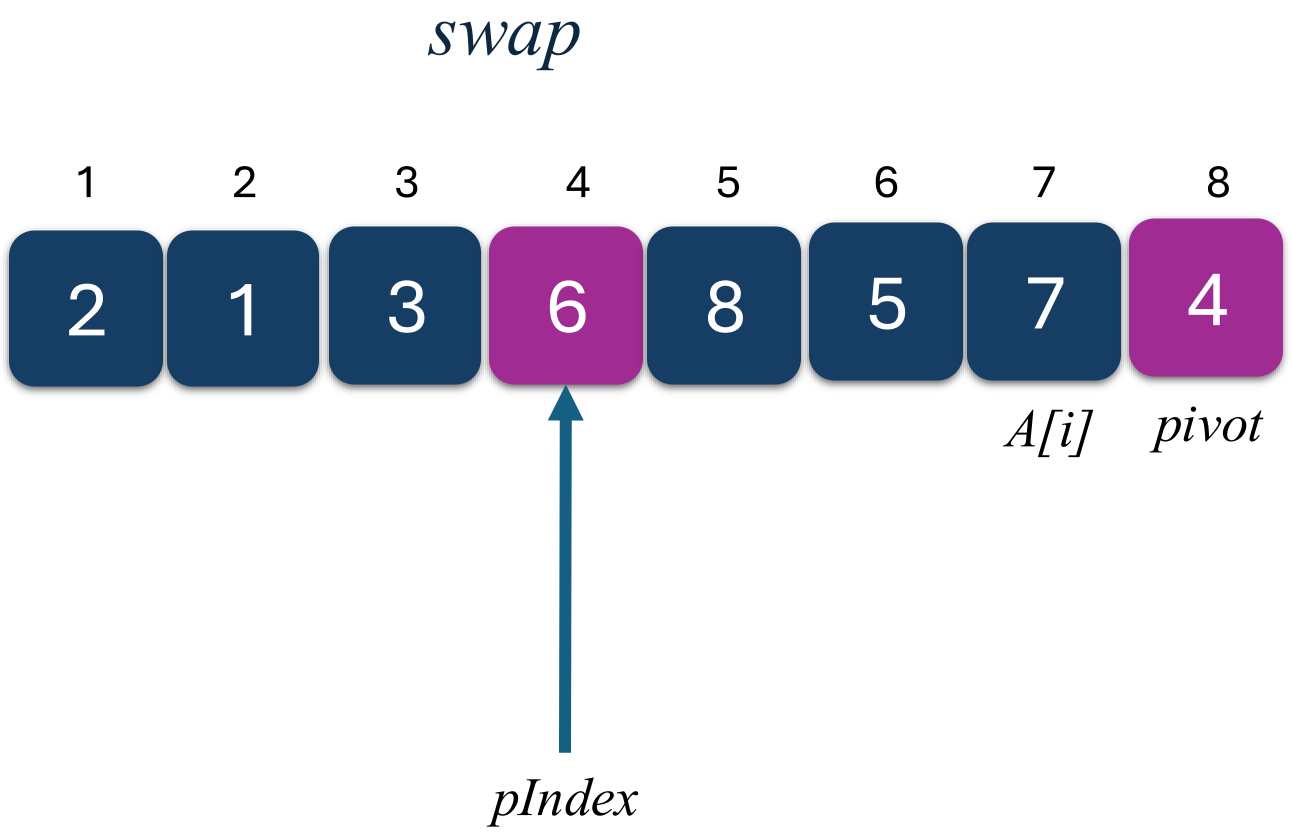


Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex = Partition(A, 1, N)

```
Partition(A, start, end):
    pivot = A[end]                4
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     4
    swap(A[pIndex], pivot)        Swap(A[4], 4)
    return pIndex
```



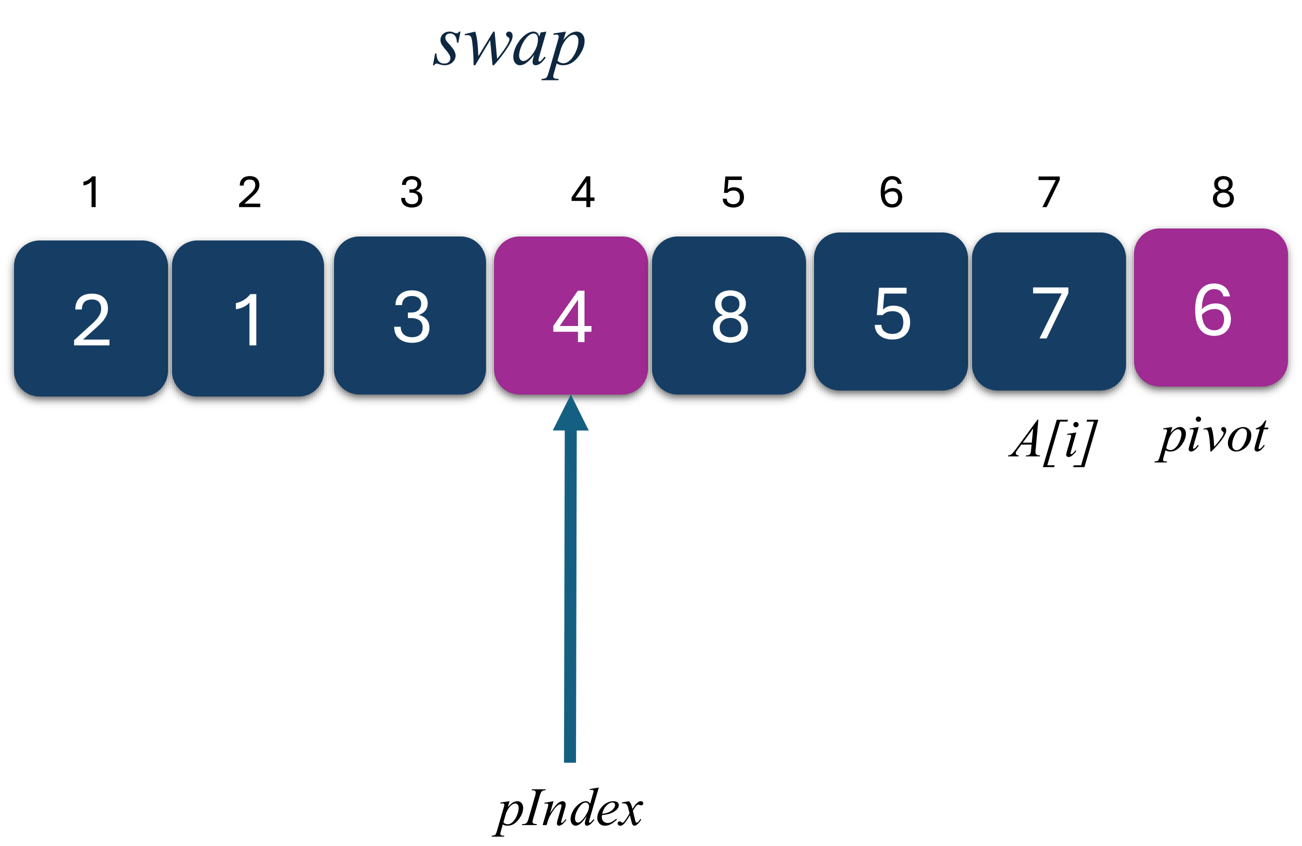
Quicksort

Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex=4

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



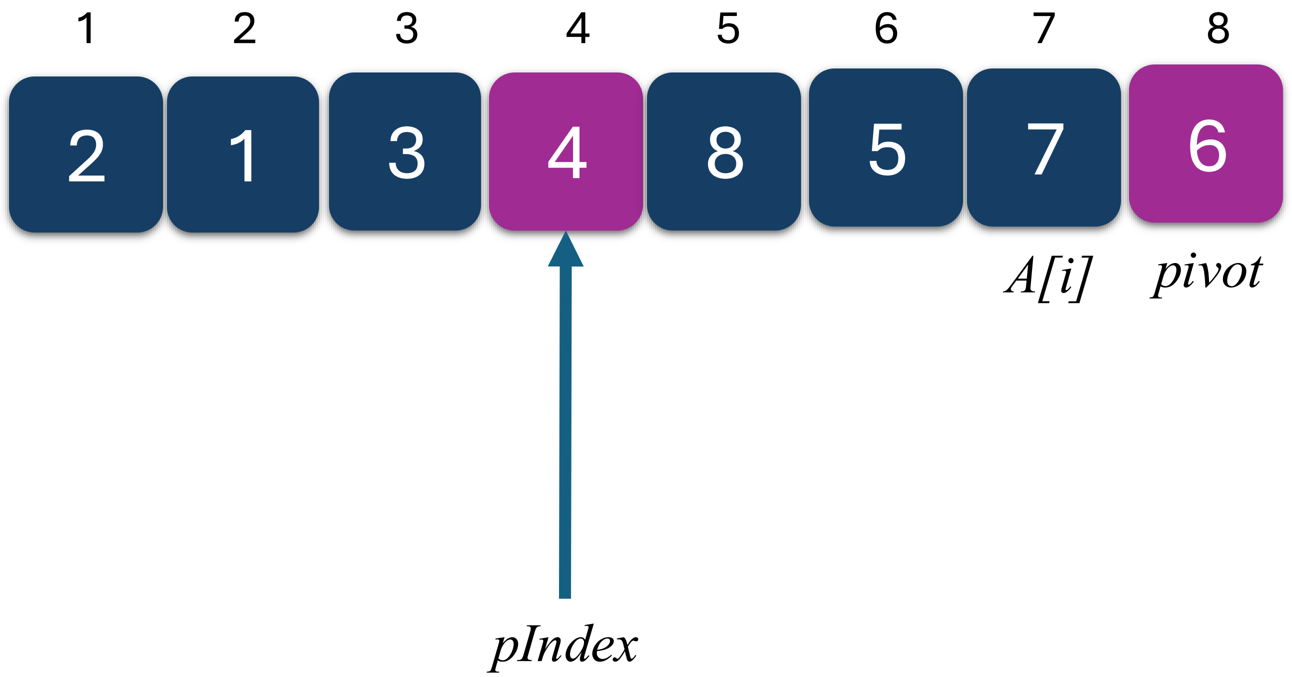
Quicksort

Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex=4

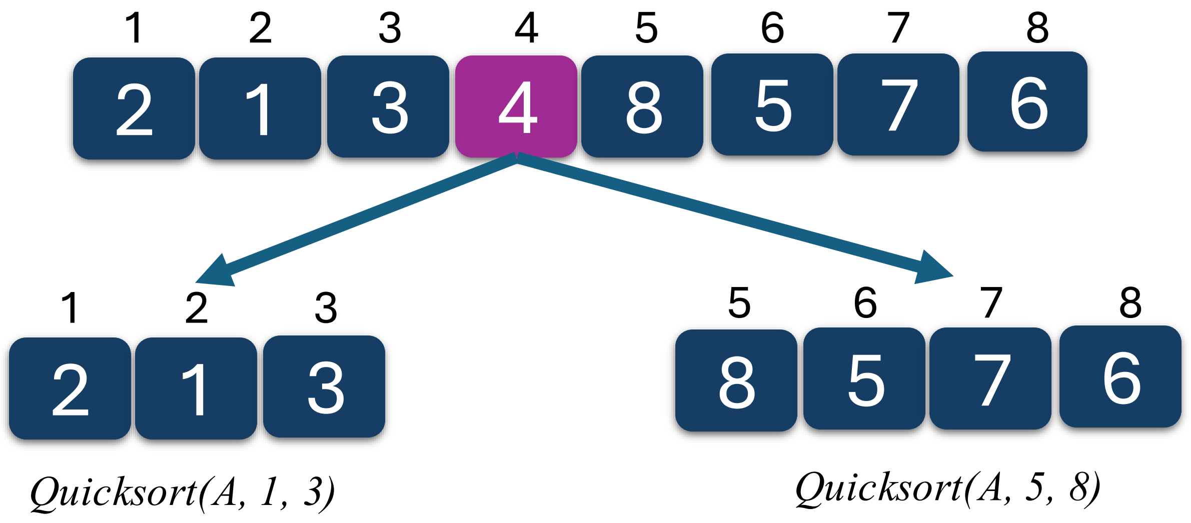
```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



Quicksort

Initial call: quick_sort(A, 1, N)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)      pIndex=4
        Quicksort(A, start, pIndex-1)          Quicksort(A, 1,3)
        Quicksort(A, pIndex+1, end)            Quicksort(A, 5,8)

Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

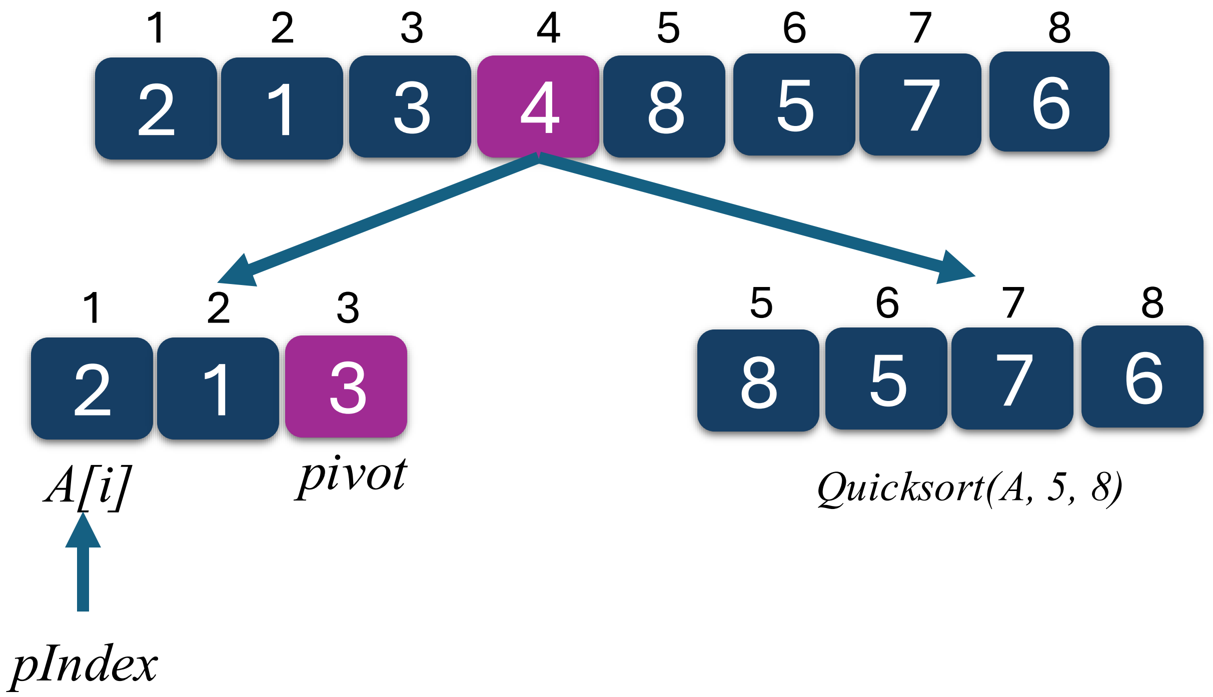


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)      Quicksort(A, 1,3)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 5,8)
```

```
Partition(A, start, end):
    pivot = A[end]                3
    pIndex = start                1
    for i=start to end-1:         i=1
        if A[i] <= pivot:         T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     T
    swap(A[pIndex], pivot)
    return pIndex
```

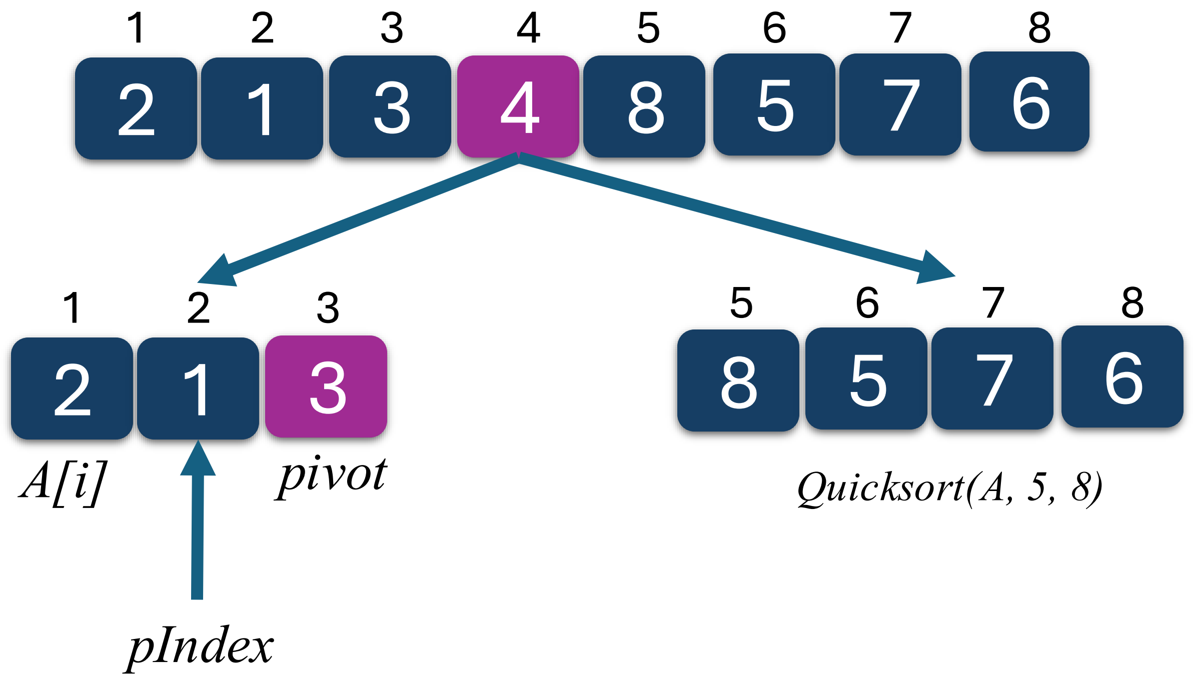


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)      Quicksort(A, 1,3)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 5,8)
```

```
Partition(A, start, end):
    pivot = A[end]            3
    pIndex = start
    for i=start to end-1:     i=1
        if A[i] <= pivot:     T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1     2
    swap(A[pIndex], pivot)
    return pIndex
```

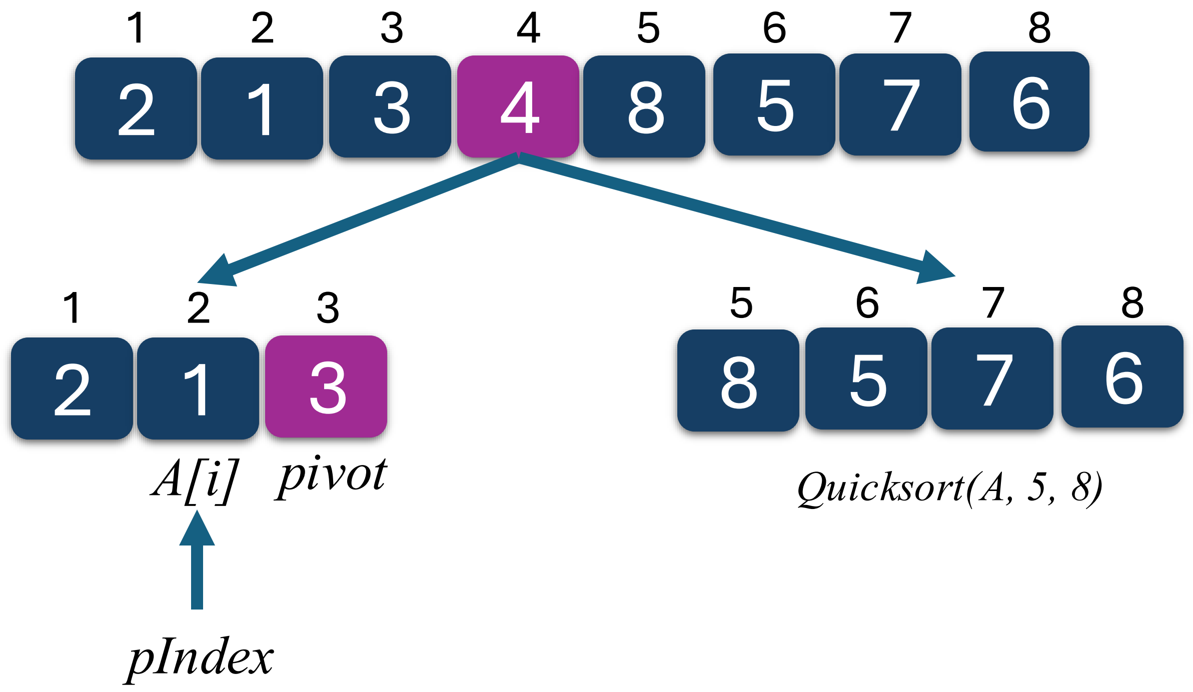


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)    Quicksort(A, 1,3)
        Quicksort(A, pIndex+1, end)      Quicksort(A, 5,8)
```

```
Partition(A, start, end):
    pivot = A[end]               3
    pIndex = start
    for i=start to end-1:        i=2
        if A[i] <= pivot:        T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1    2
    swap(A[pIndex], pivot)
    return pIndex
```

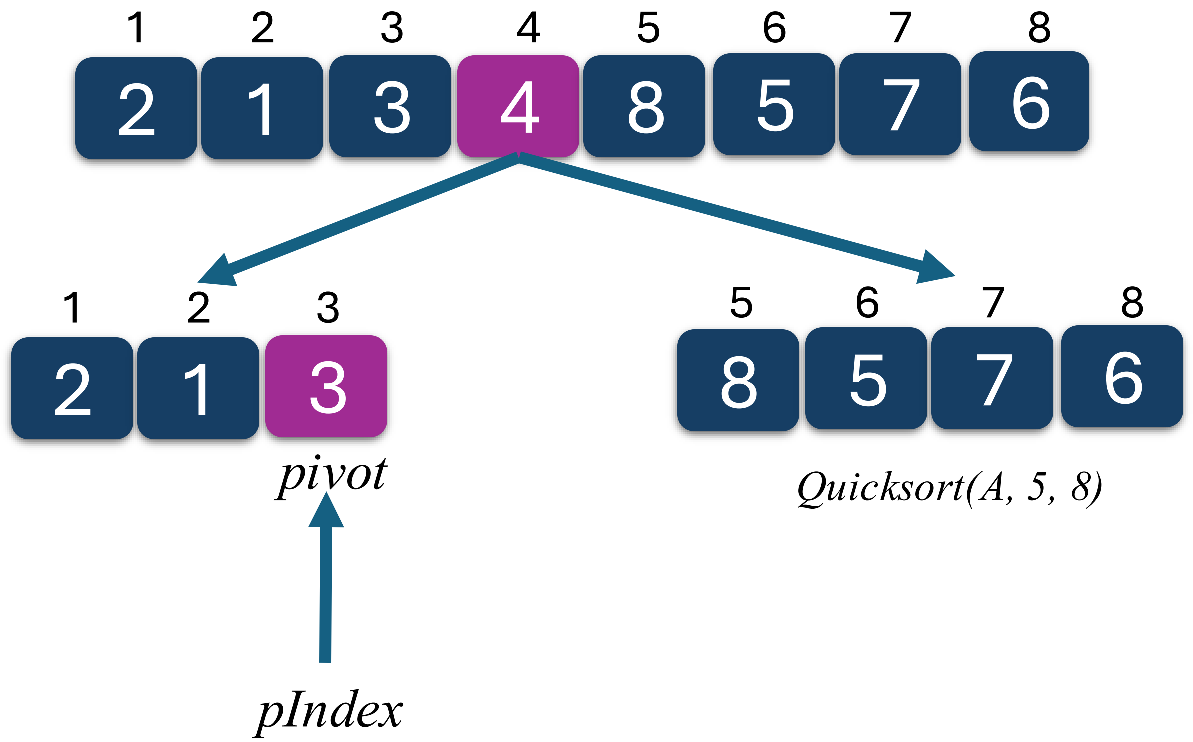


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)     Quicksort(A, 1,3)
        Quicksort(A, pIndex+1, end)       Quicksort(A, 5,8)
```

```
Partition(A, start, end):
    pivot = A[end]    3
    pIndex = start
    for i=start to end-1:    i=2
        if A[i] <= pivot:    T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1    3
    swap(A[pIndex], pivot)
    return pIndex
```

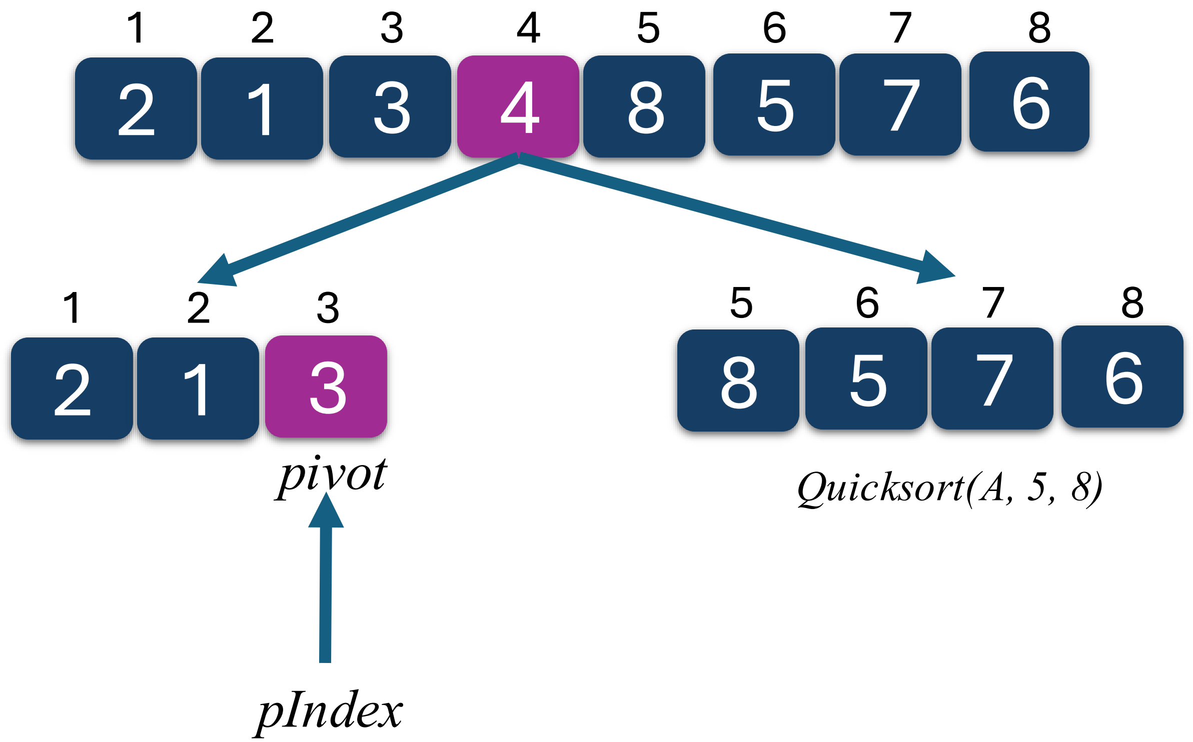


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
pIndex=3    Quicksort(A, start, pIndex-1)     Quicksort(A, 1,2)
        Quicksort(A, pIndex+1, end)       Quicksort(A, 2,2)
```

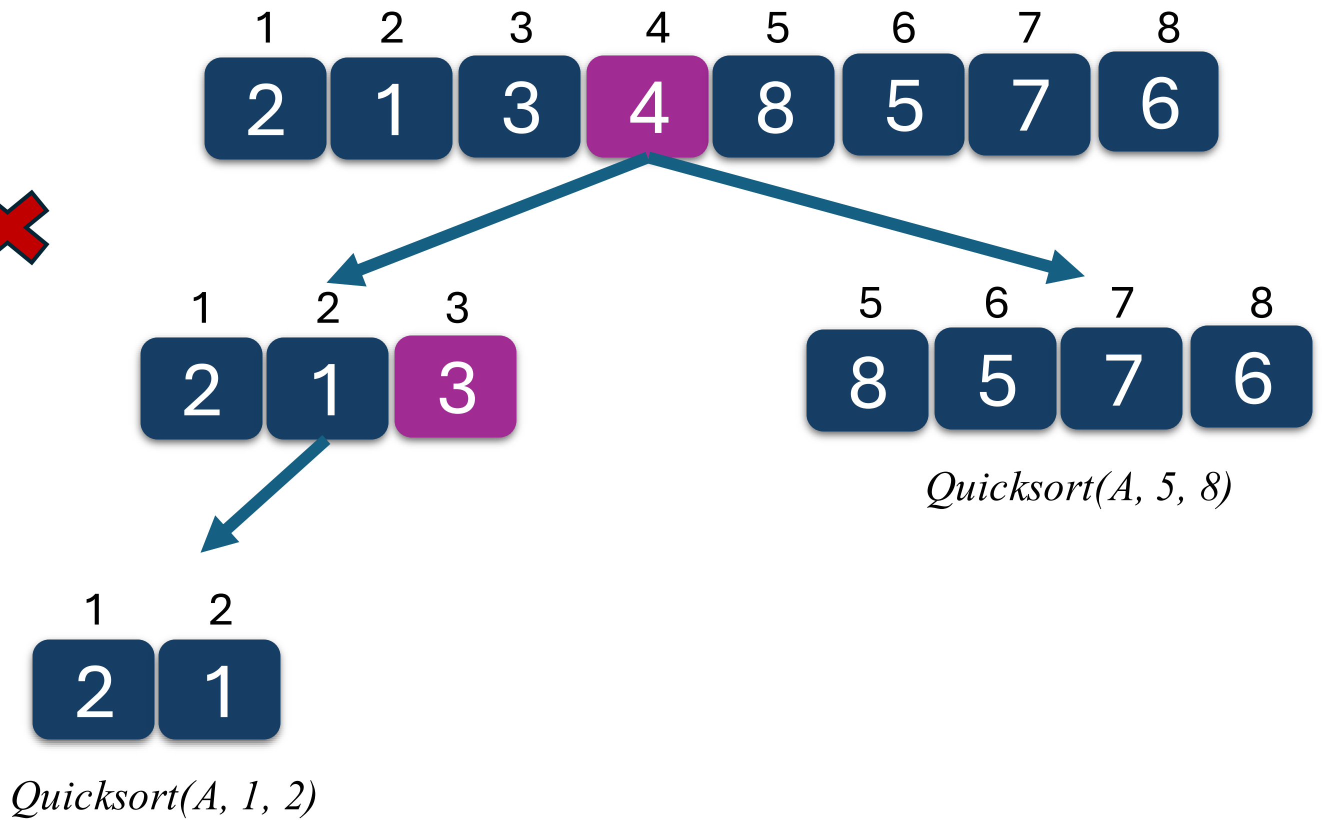
```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)     Swap(3, 3)
    return pIndex
```



Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
pIndex=3    Quicksort(A, start, pIndex-1)    Quicksort(A, 1,2)
        Quicksort(A, pIndex+1, end)    Quicksort(A, 2,2) ✗
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



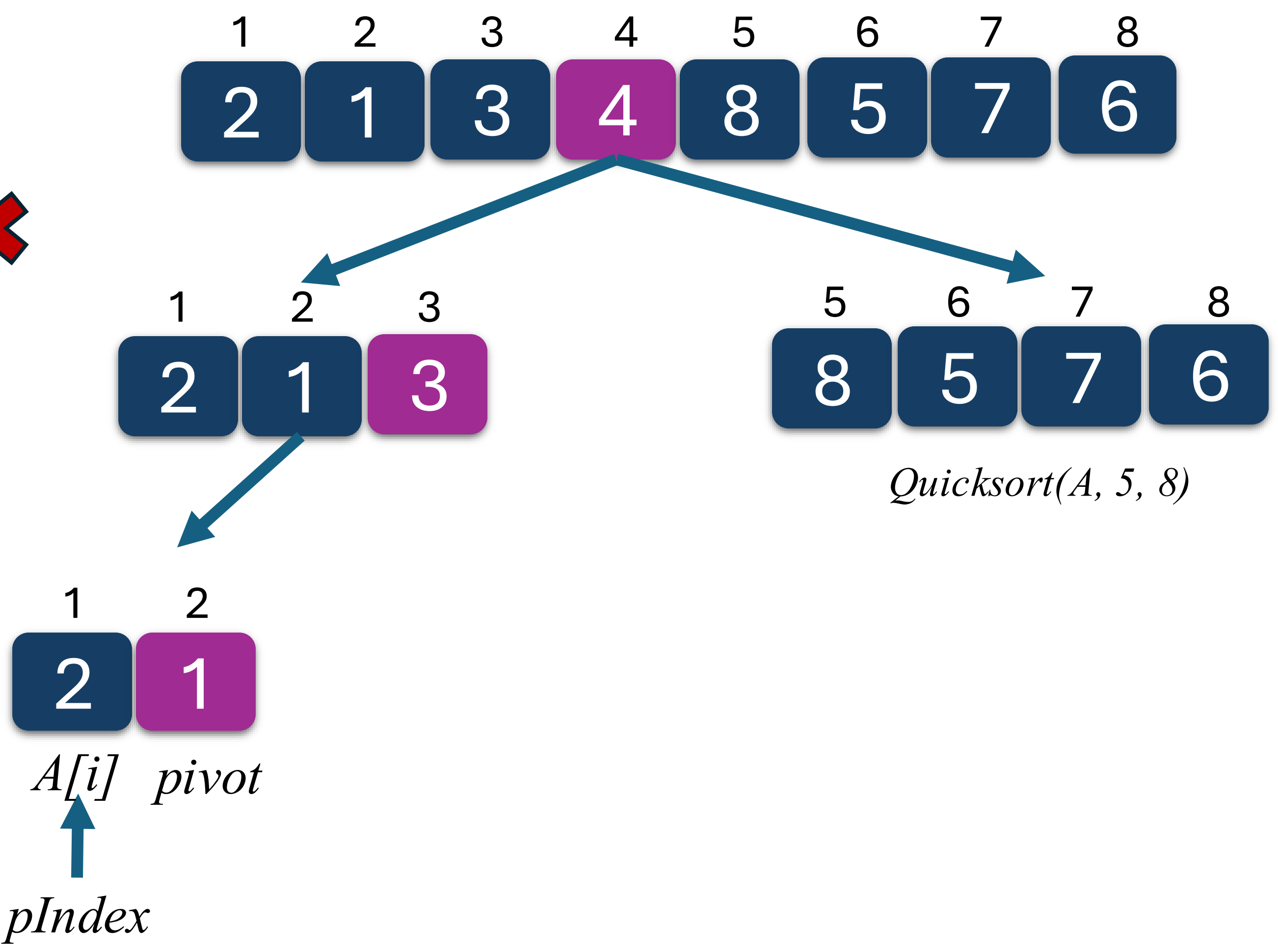
Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)      Quicksort(A, 1,2)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 2,2)
```

pIndex=3

```
Partition(A, start, end):
    pivot = A[end]                1
    pIndex = start                1
    for i=start to end-1:         i=1
        if A[i] <= pivot:         F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

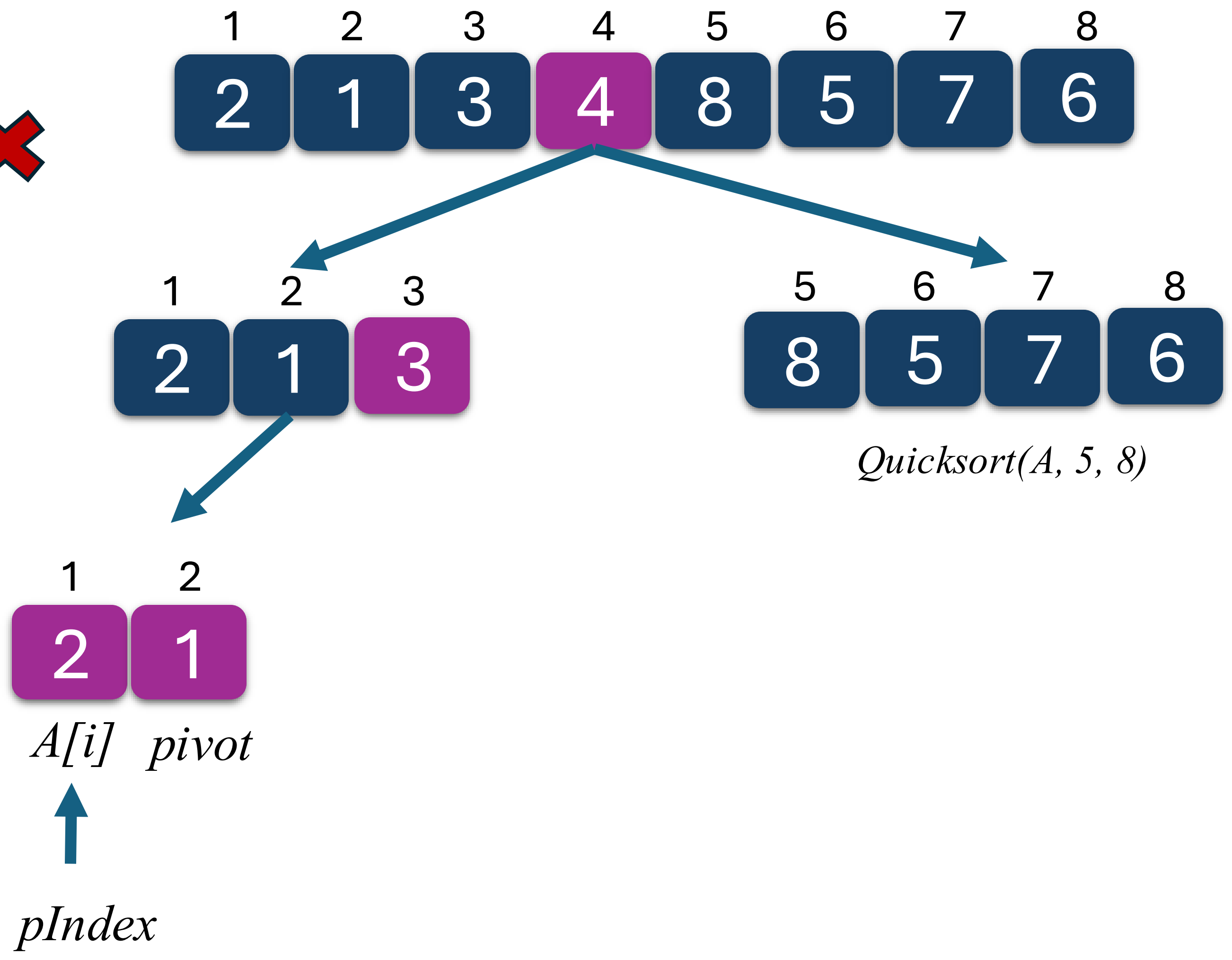


Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
pIndex=1  Quicksort(A, start, pIndex-1)   Quicksort(A, 1,0)  ✗
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



Quicksort

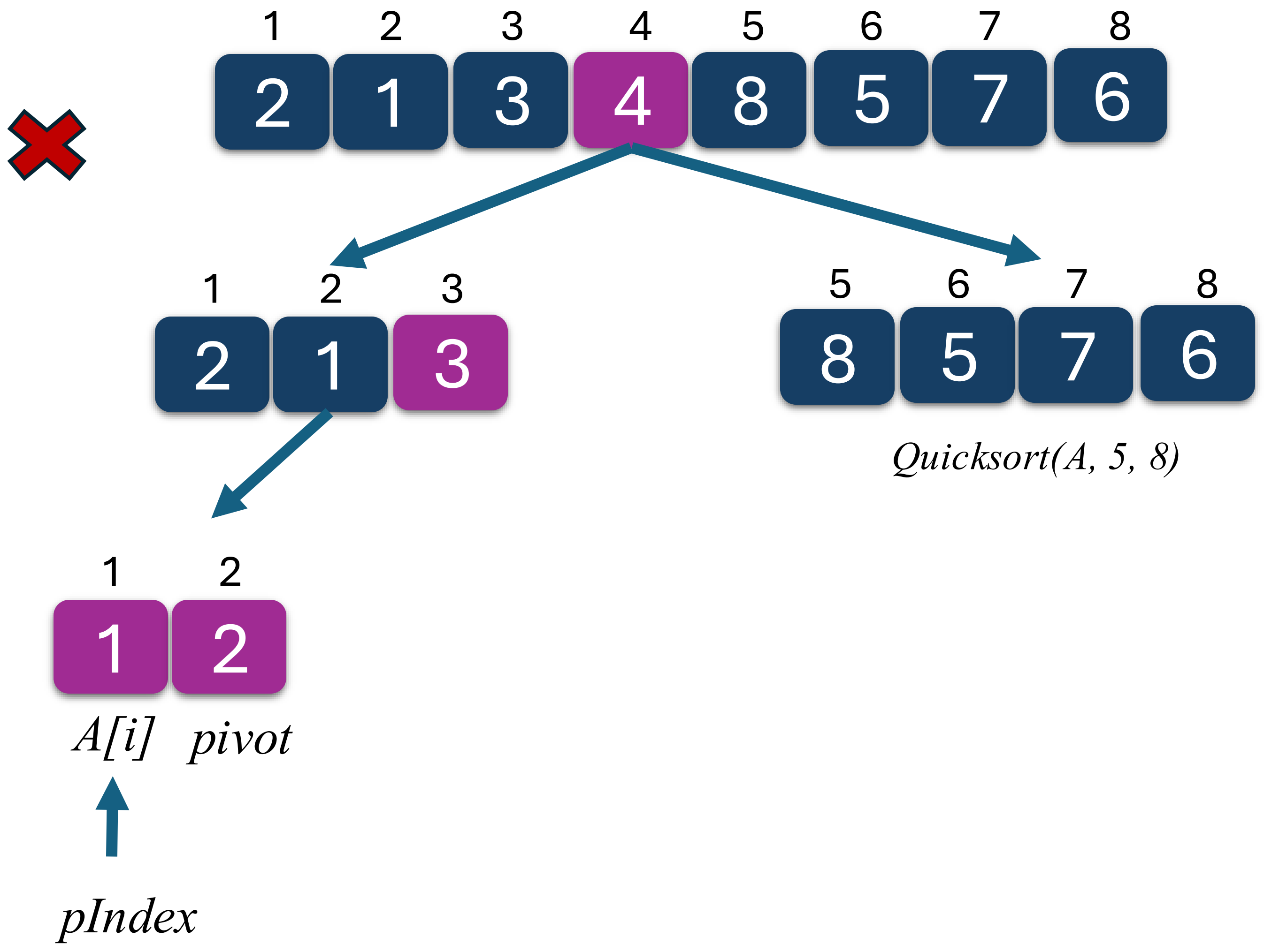
Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

pIndex=1

Quicksort(A, 1,0)

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



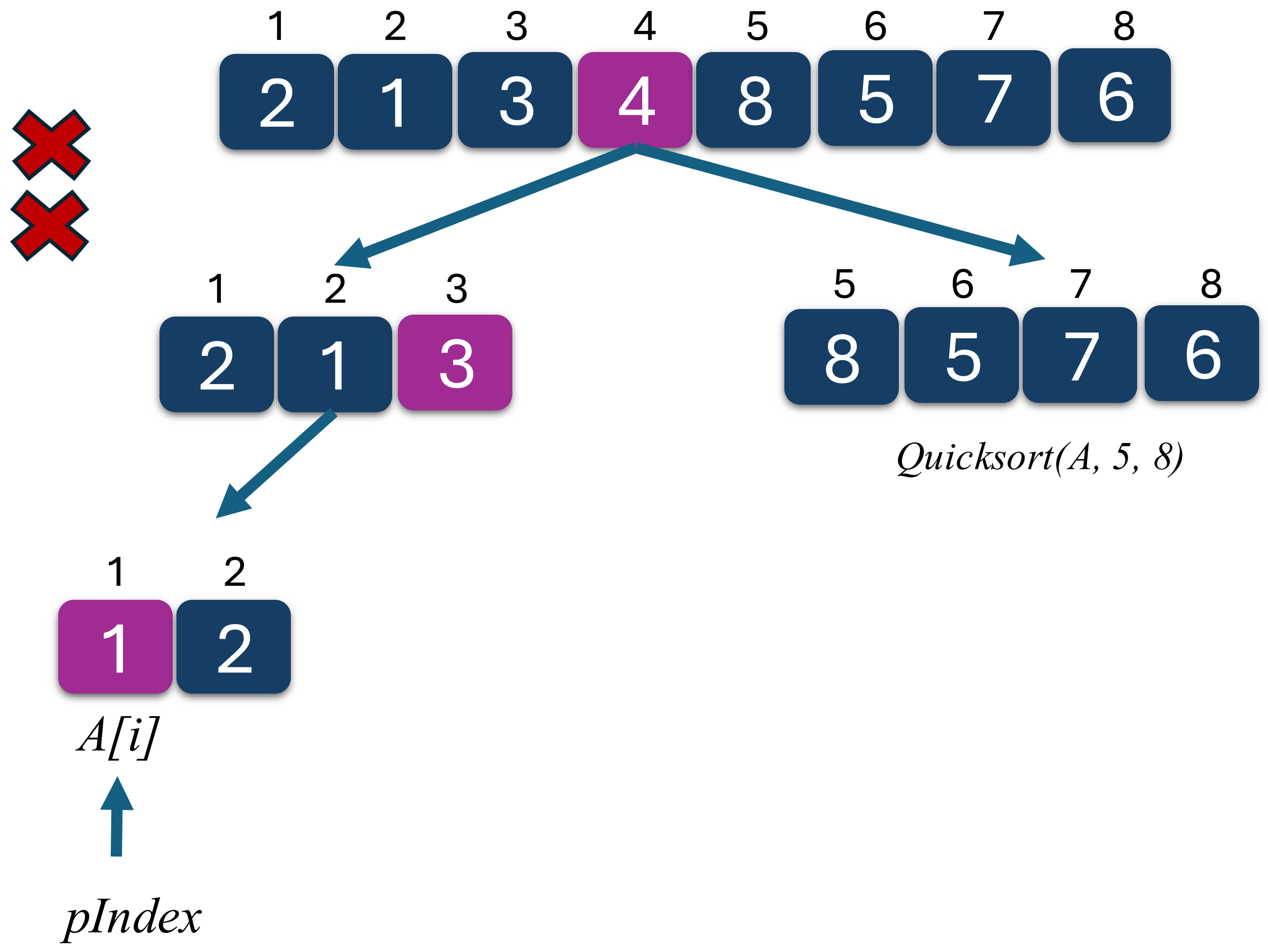
Quicksort

Call on Left Subarray: quick_sort(A, 1, 3)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)      Quicksort(A, 1,0)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 2,2)
```

pIndex=1

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

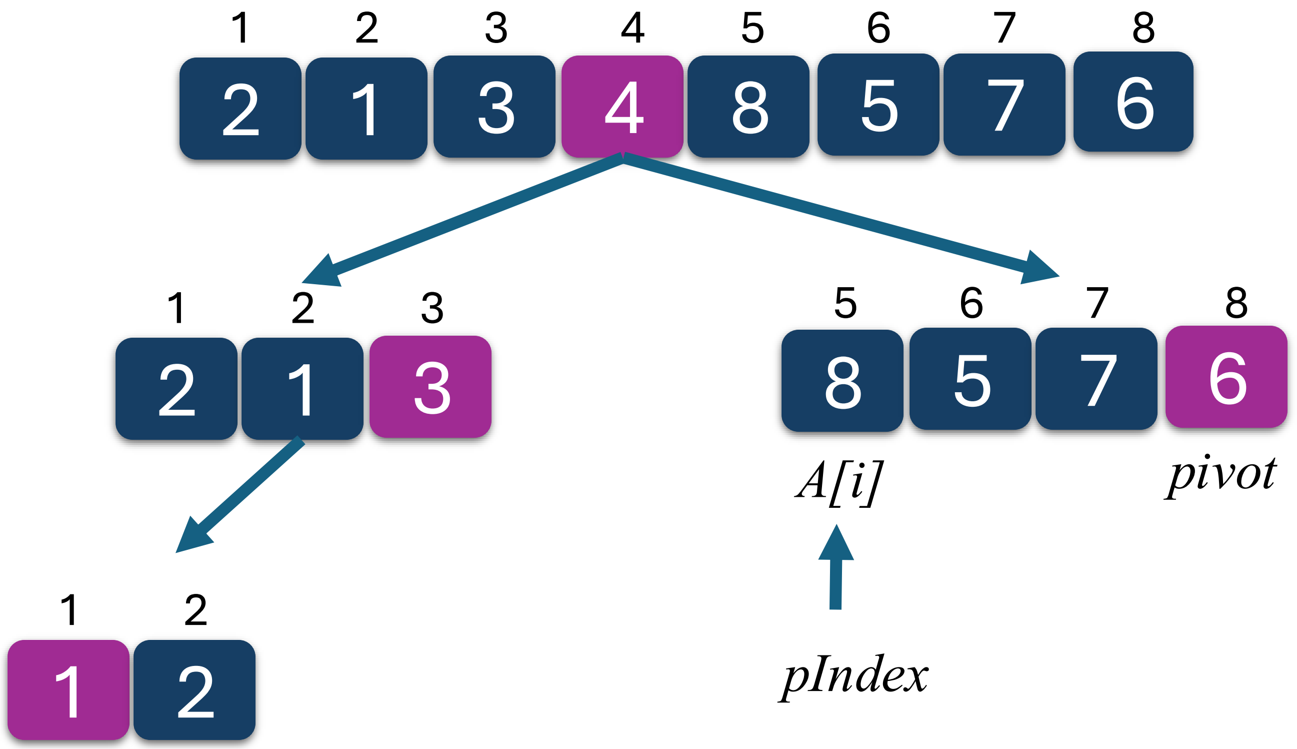


Quicksort

Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)    Quicksort(A, 5,8)
```

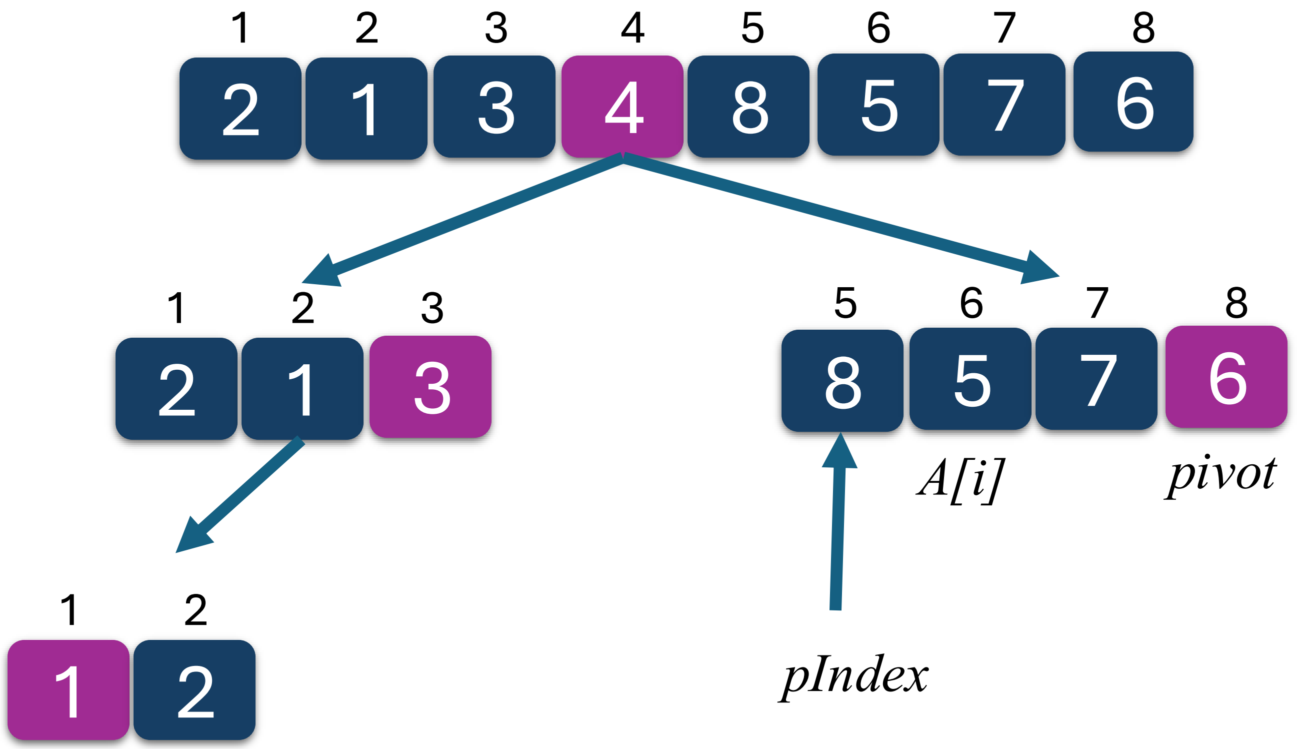
```
Partition(A, start, end):
    pivot = A[end]    8
    pIndex = start    5
    for i=start to end-1:    i=5
        if A[i] <= pivot:    F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)    Quicksort(A, 5,8)
```

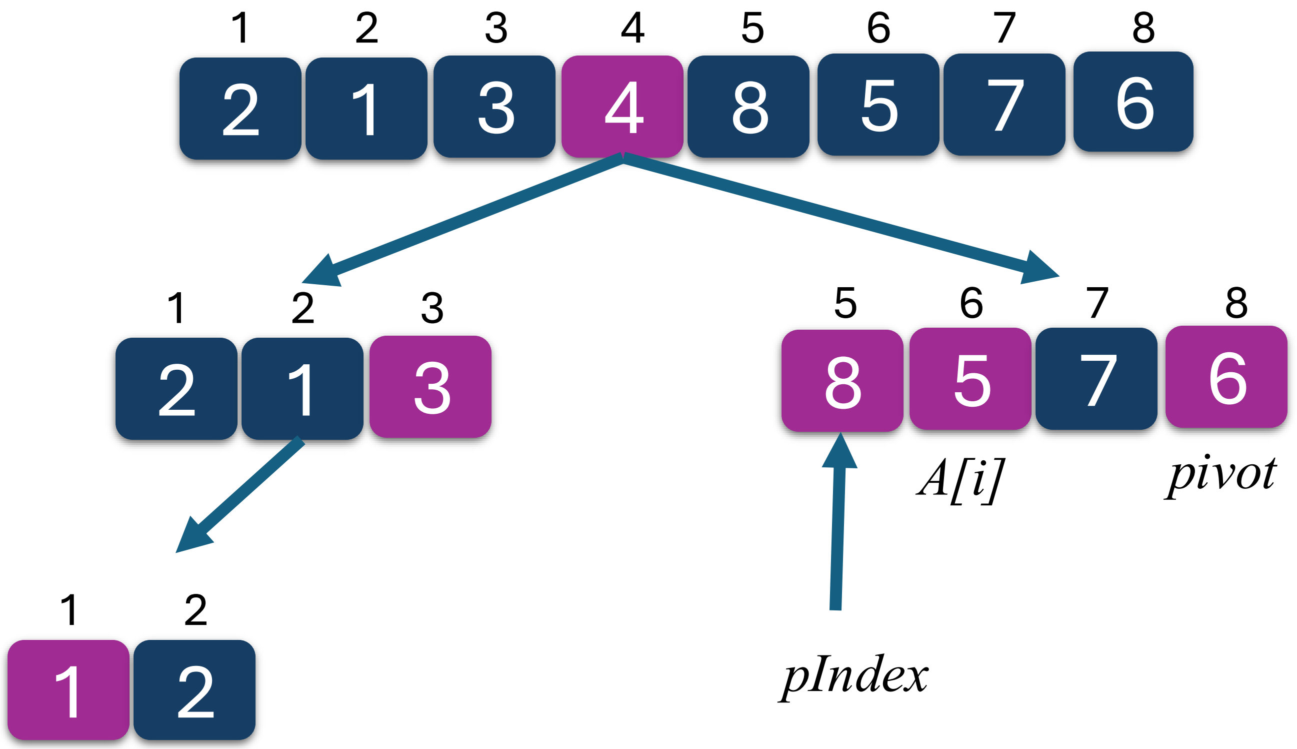
```
Partition(A, start, end):
    pivot = A[end]    8
    pIndex = start
    for i=start to end-1:    i=6
        if A[i] <= pivot:    T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

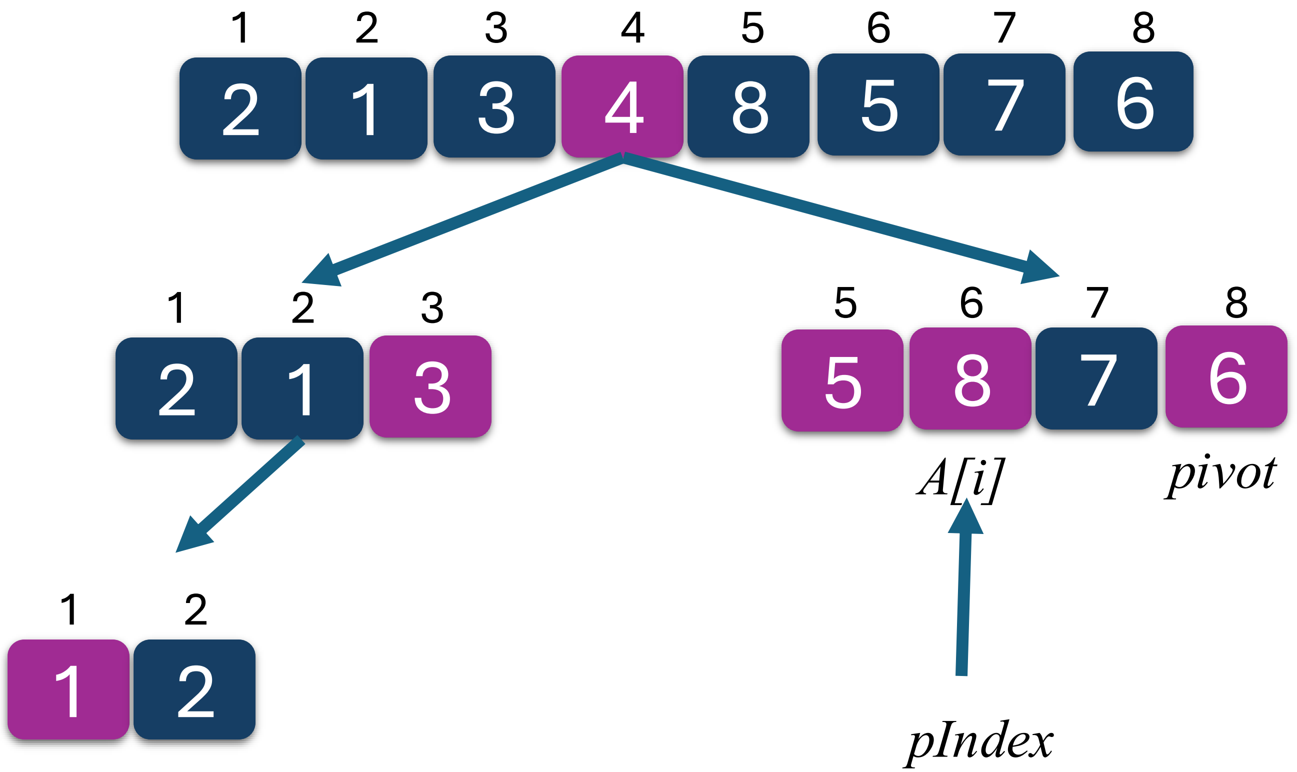
```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)    Quicksort(A, 5,8)
```

```
Partition(A, start, end):
    pivot = A[end]    8
    pIndex = start
    for i=start to end-1:    i=6
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



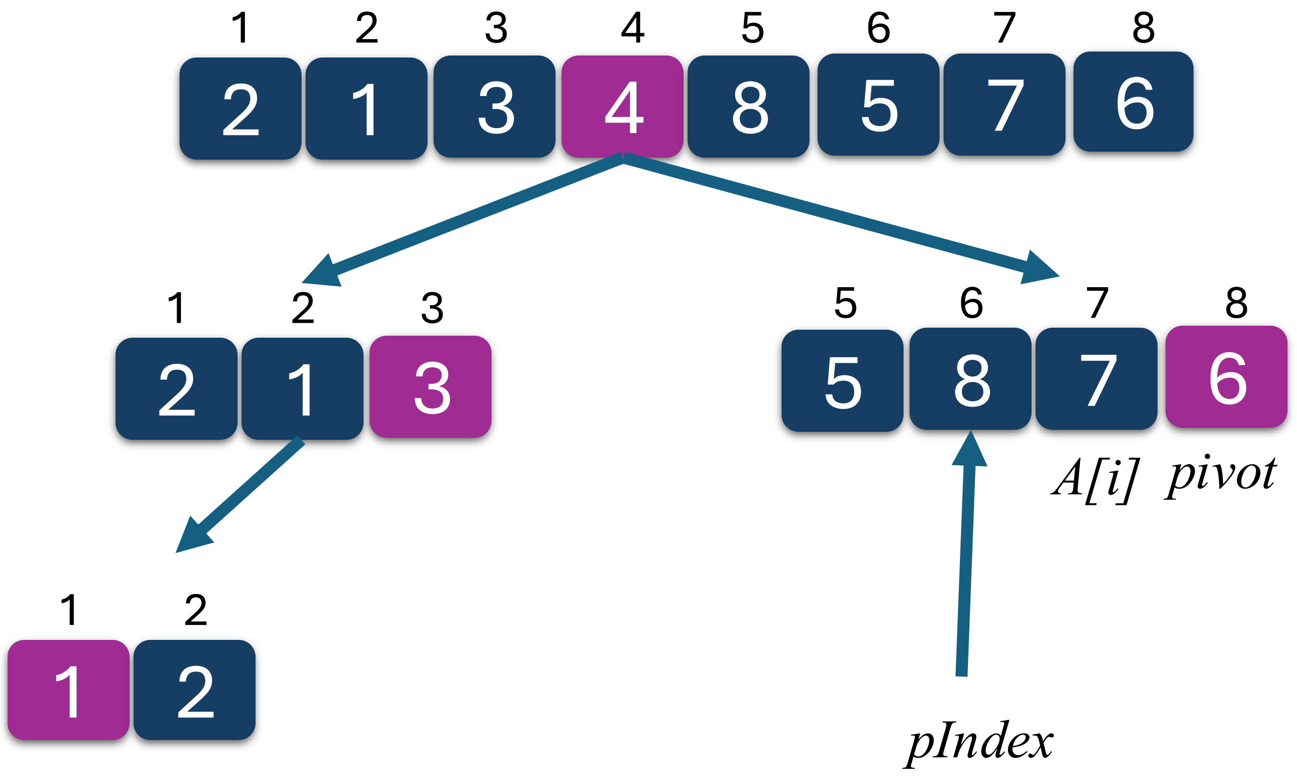
Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)    Quicksort(A, 5,8)


Partition(A, start, end):
    pivot = A[end]  8
    pIndex = start
    for i=start to end-1:  i=6
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   6
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

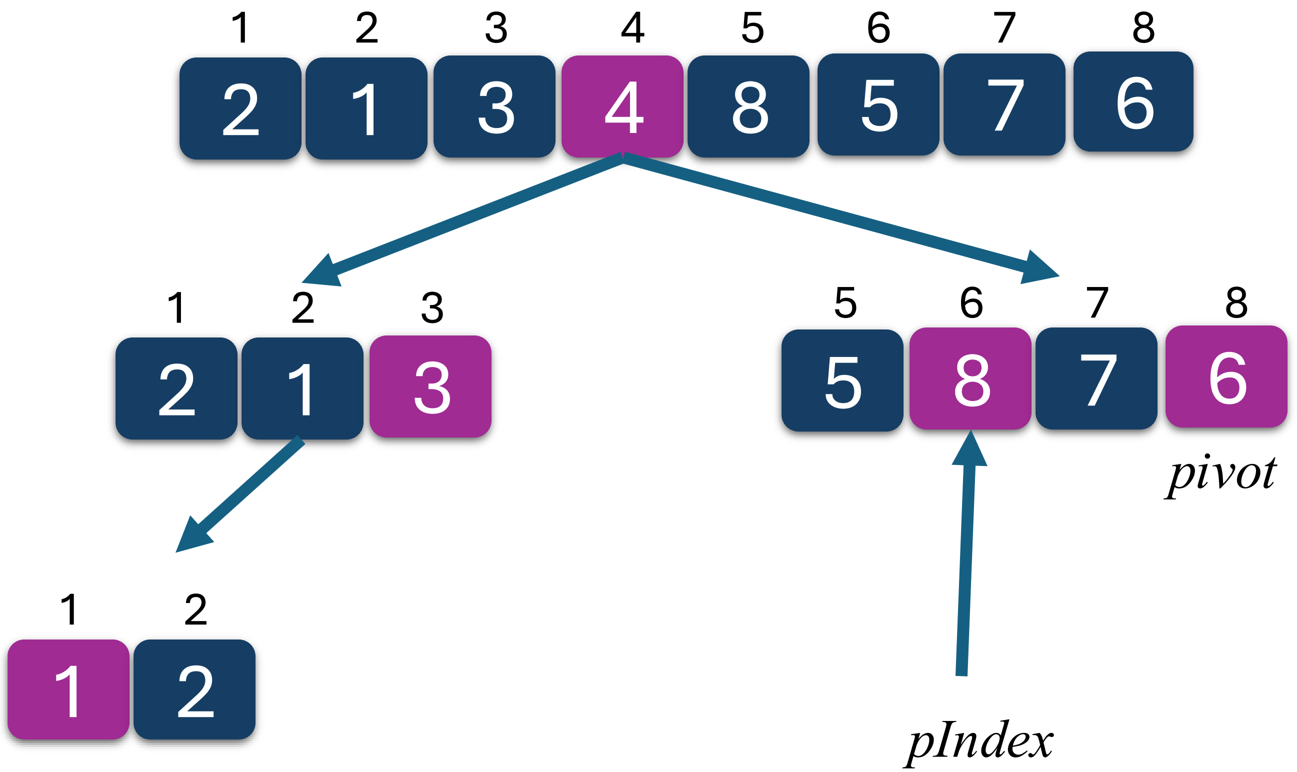
```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 5,8)

Partition(A, start, end):
    pivot = A[end]                 8
    pIndex = start
    for i=start to end-1:          i=7
        if A[i] <= pivot:          F
            swap(A[i], A[pIndex])
            pIndex = pIndex+1      6
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)        Quicksort(A, 5,8)
```

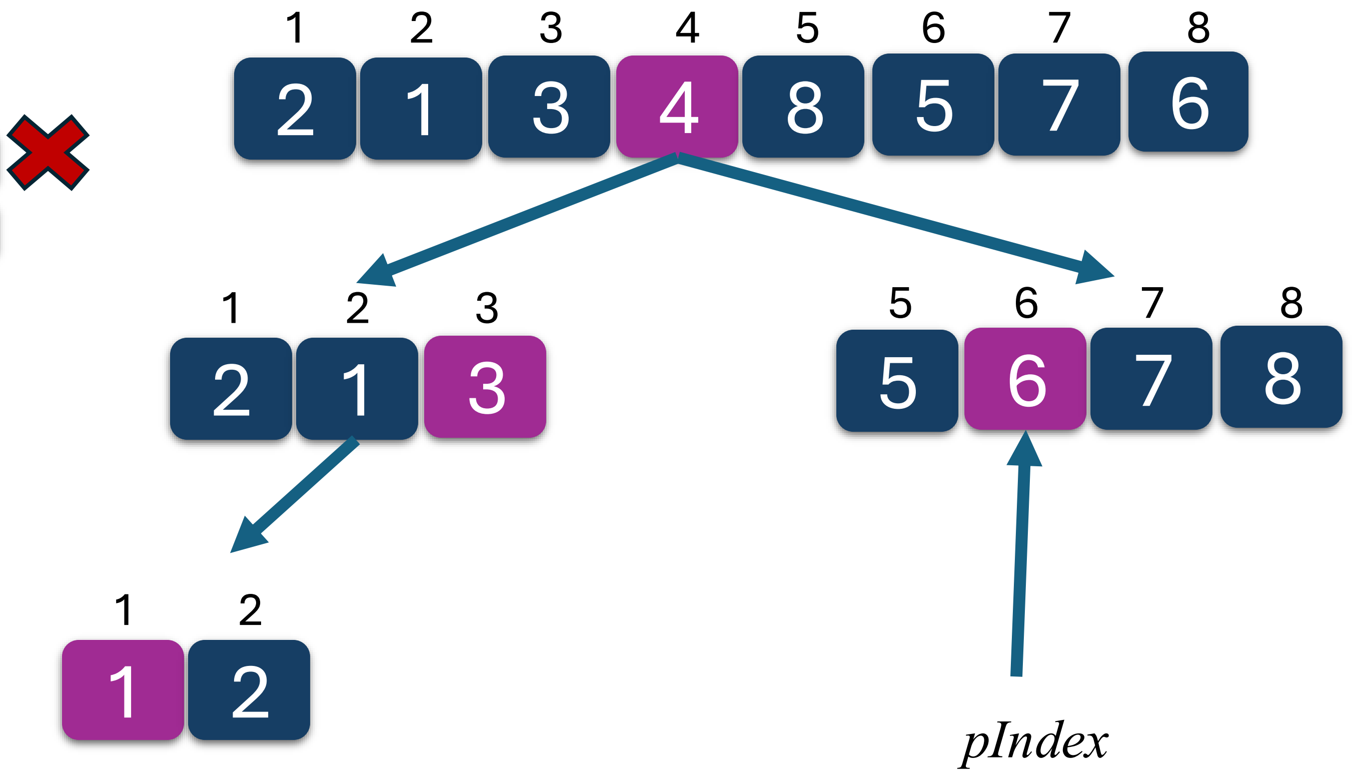
```
Partition(A, start, end):
    pivot = A[end]        8
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1        6
    swap(A[pIndex], pivot)        swap(8, 6)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
pIndex=6      pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)   Quicksort(A, 5,5)
        Quicksort(A, pIndex+1, end)     Quicksort(A, 7,8)
```

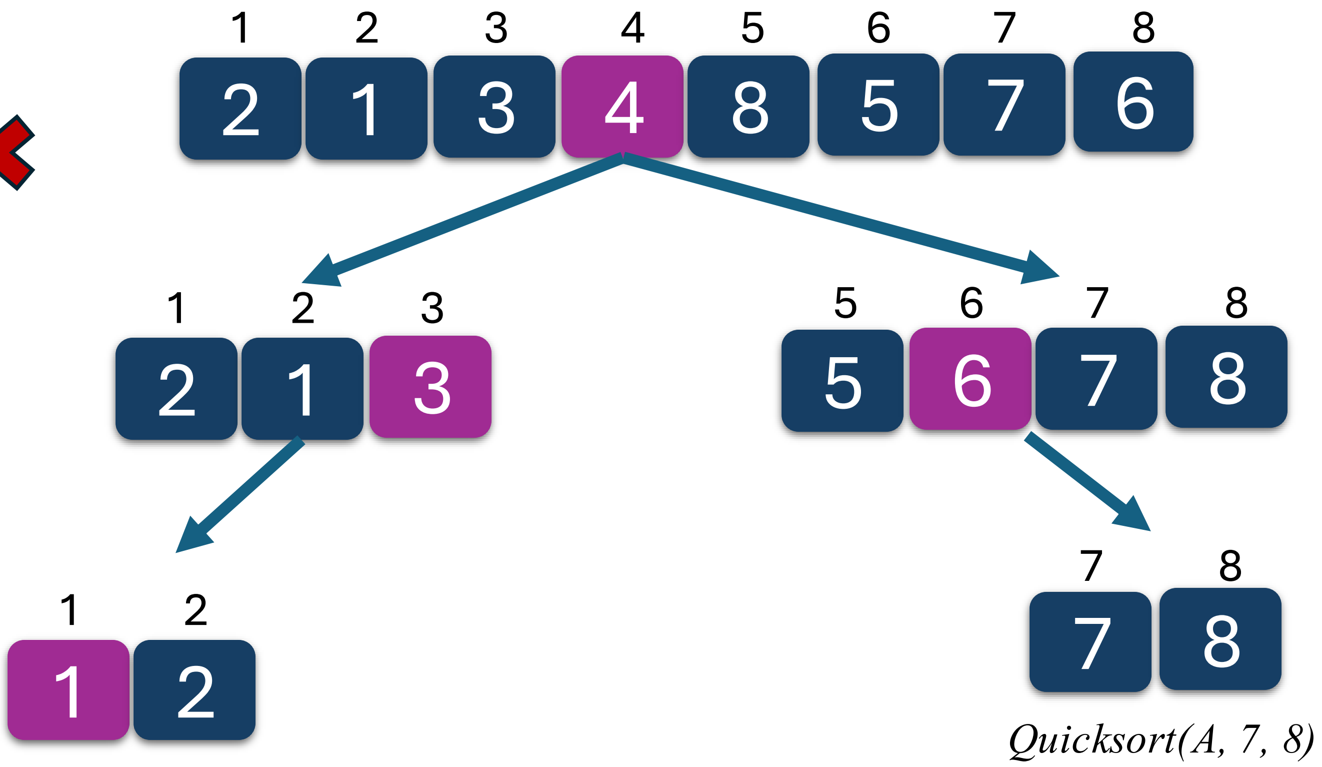
```
Partition(A, start, end):
    pivot = A[end]   8
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   6
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
pIndex=6    pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)    Quicksort(A, 5,5) ✗
        Quicksort(A, pIndex+1, end)      Quicksort(A, 7,8)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

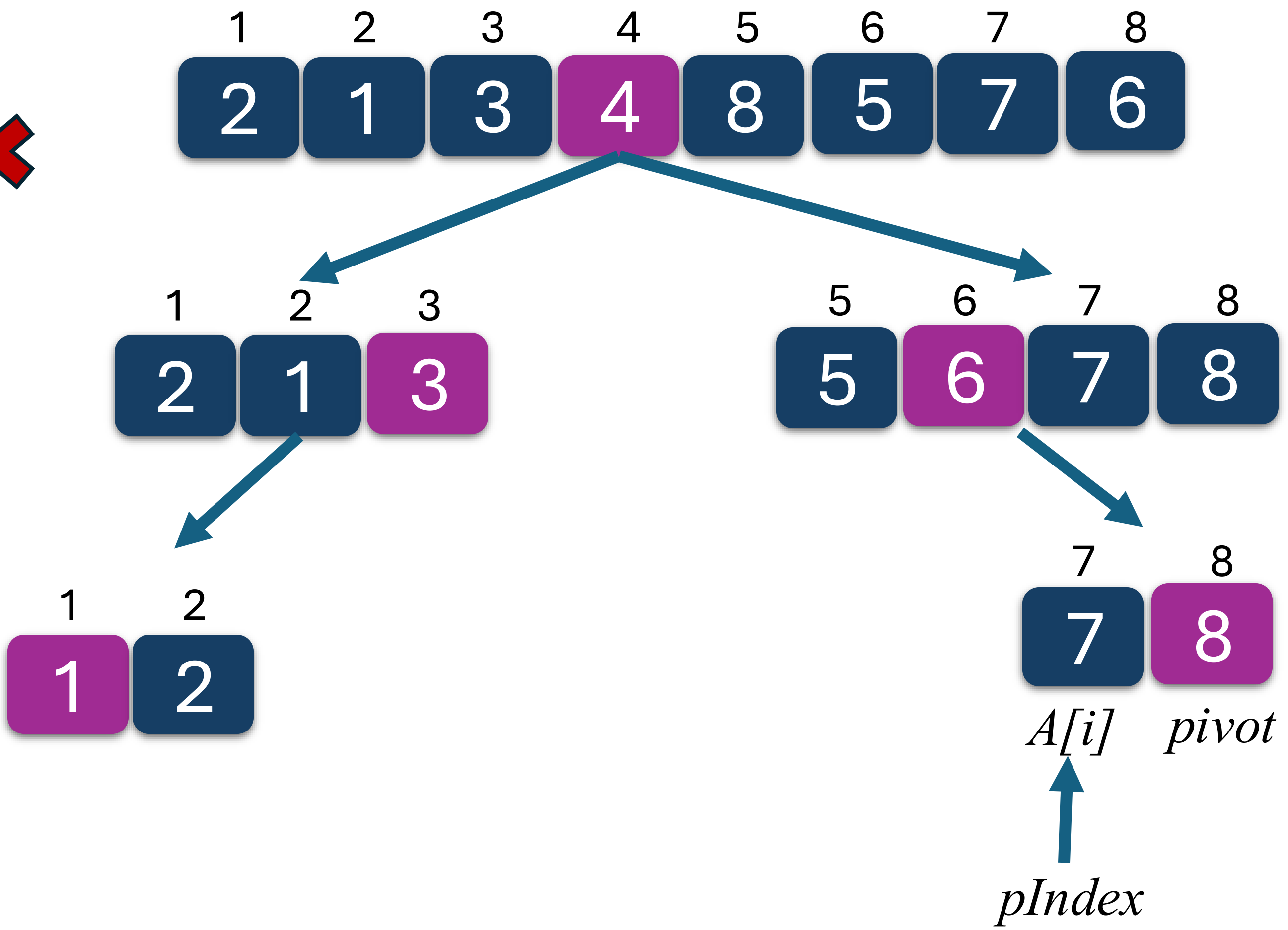


Quicksort

Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):                                   
    if(start<end):
        pIndex = Partition(A, start, end)          pIndex=6
        Quicksort(A, start, pIndex-1)              Quicksort(A, 5,5)
        Quicksort(A, pIndex+1, end)                Quicksort(A, 7,8)
```

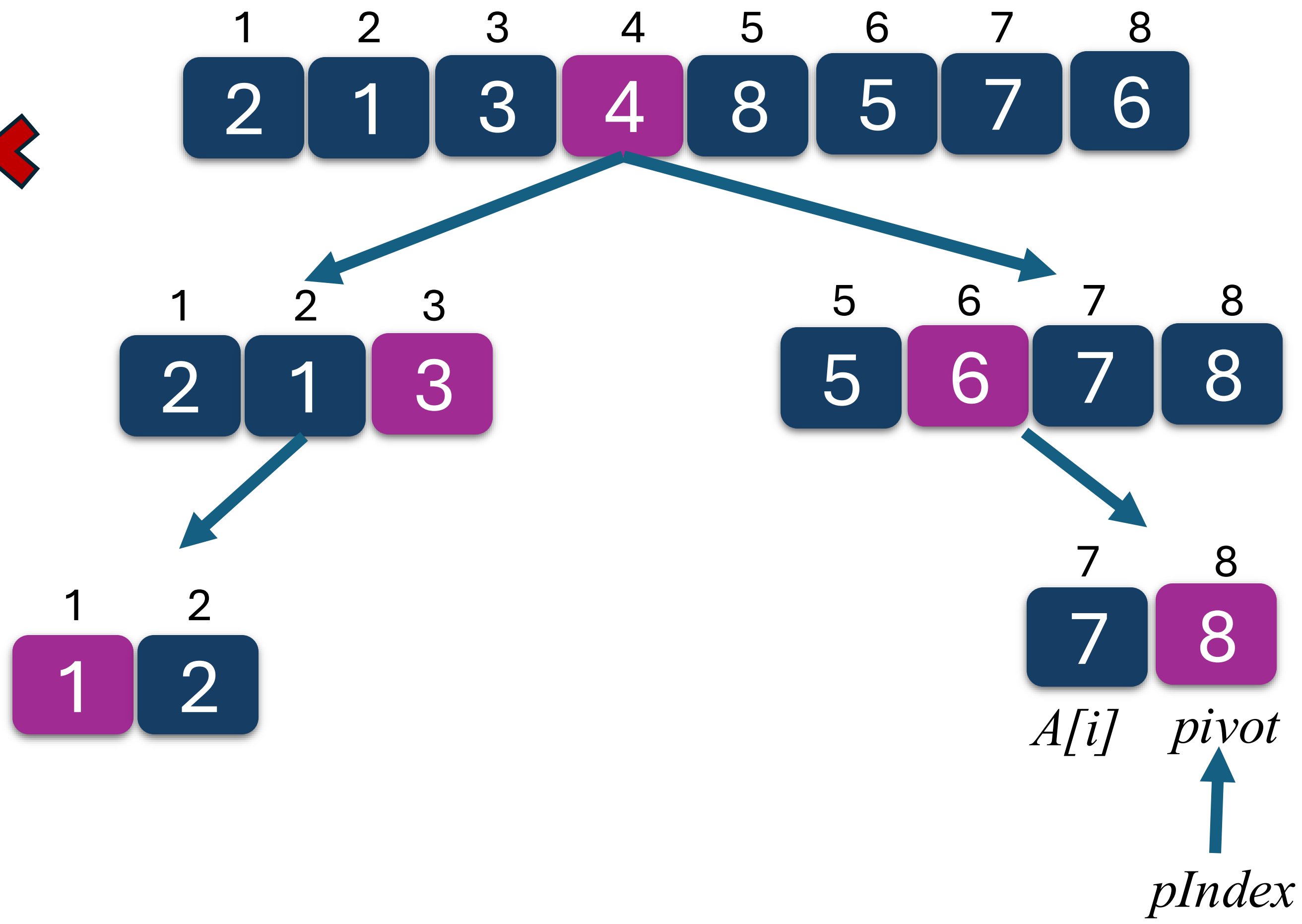
```
Partition(A, start, end):
    pivot = A[end]                  8
    pIndex = start                  7
    for i=start to end-1:           i=7
        if A[i] <= pivot:           T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
pIndex=6    pIndex = Partition(A, start, end)
            Quicksort(A, start, pIndex-1)    Quicksort(A, 5,5)
            Quicksort(A, pIndex+1, end)      Quicksort(A, 7,8)
```

```
Partition(A, start, end):
    pivot = A[end]    8
    pIndex = start
    for i=start to end-1:    i=7
        if A[i] <= pivot:    T
            swap(A[i], A[pIndex])
            pIndex = pIndex+1    8
    swap(A[pIndex], pivot)
    return pIndex
```

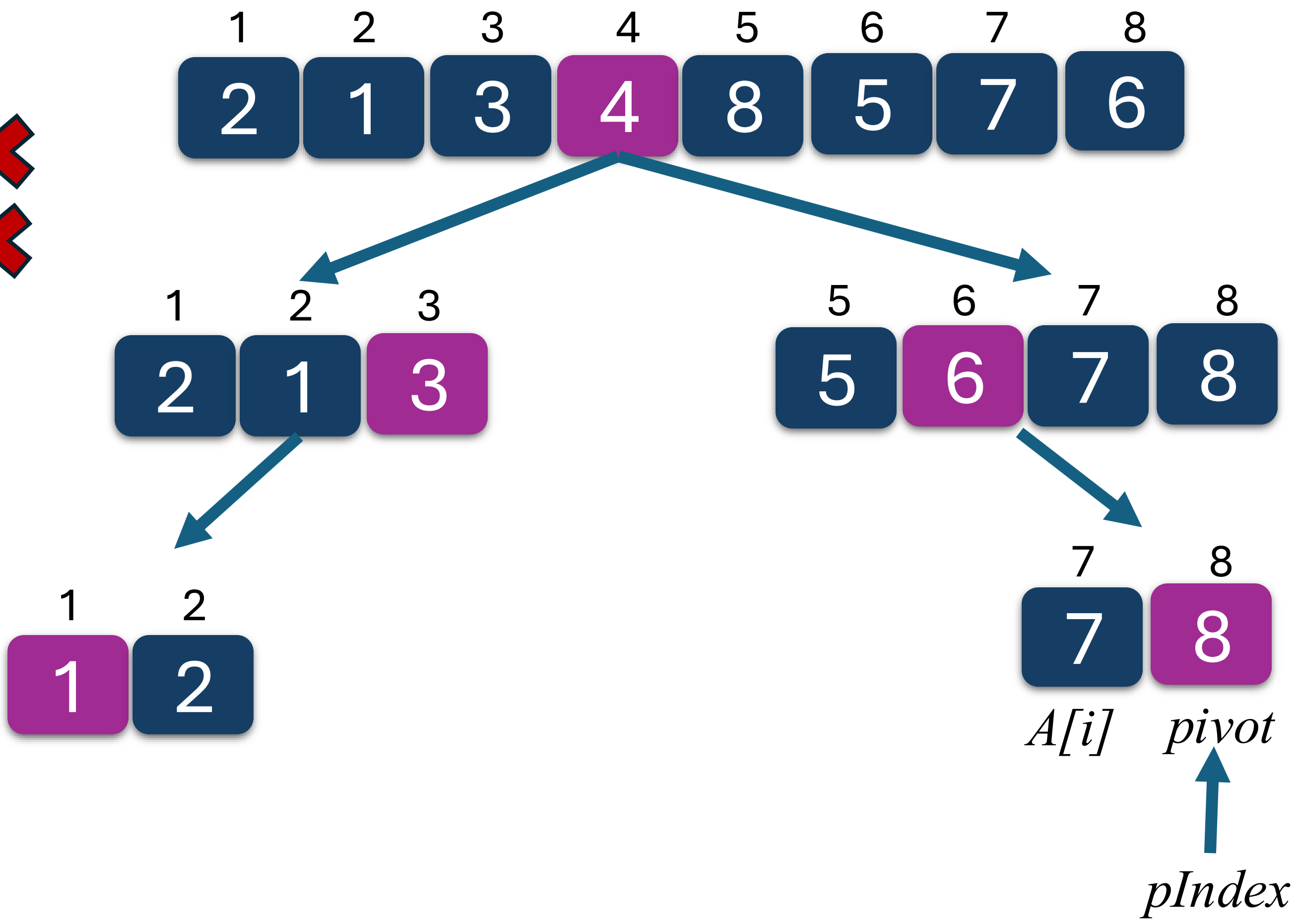


Quicksort

Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)          pIndex=8
        Quicksort(A, start, pIndex-1)              Quicksort(A, 7,7) ✗
        Quicksort(A, pIndex+1, end)                Quicksort(A, 9,8) ✗
```

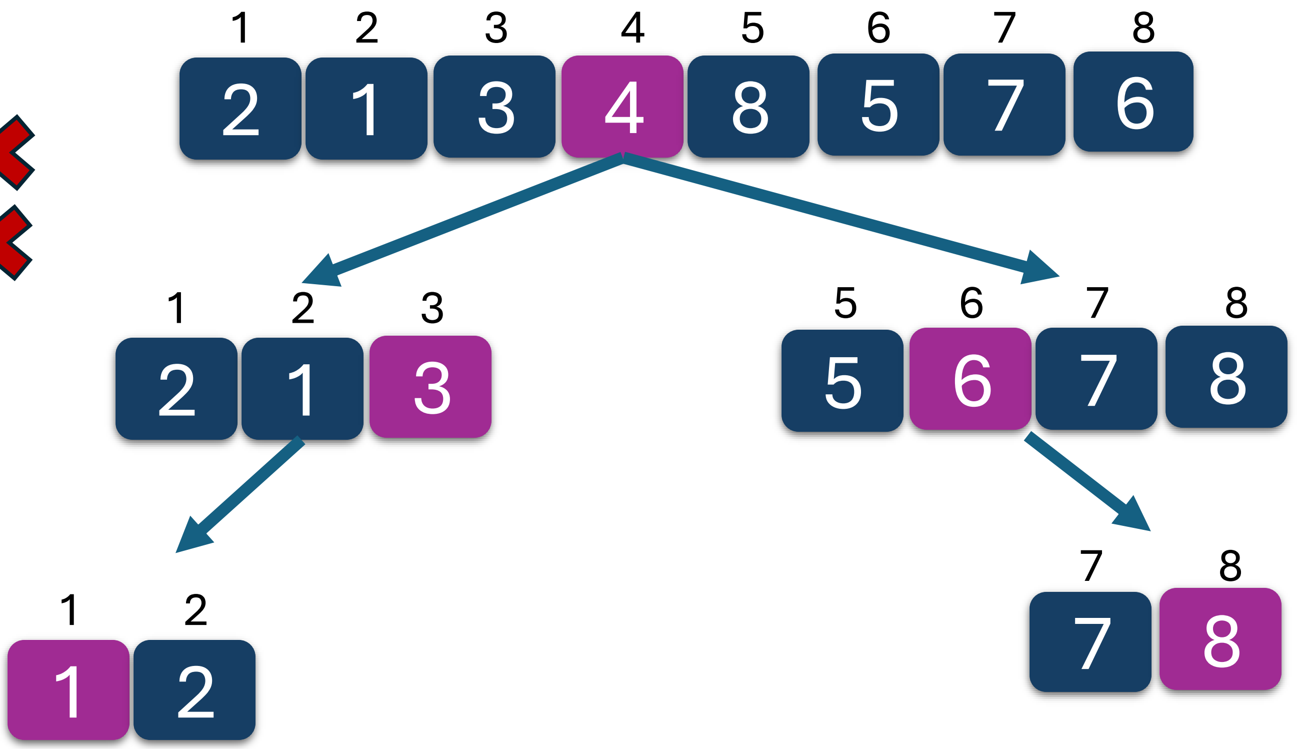
```
Partition(A, start, end):
    pivot = A[end]                  8
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1       8
    swap(A[pIndex], pivot)          swap(8, 8)
    return pIndex
```



Call on Right Subarray: QuickSort(A, 5, 8)

```
Quicksort(A, start, end):
    if(start<end):
pIndex=8    pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)   Quicksort(A, 7,7) ✗
        Quicksort(A, pIndex+1, end)     Quicksort(A, 9,8) ✗
```

```
Partition(A, start, end):
    pivot = A[end]   8
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1   8
    swap(A[pIndex], pivot)   swap(8, 8)
    return pIndex
```



Quick Sort Complexity

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

Best Case?

Worst Case?

Quick Sort Complexity

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

Best Case?

Best-Case / Average-Case RT:

- The pivot **divides the array into two equal halves** at every step.
- The depth of the recursion tree is **O(logn)**
- At each level, all n elements are processed → O(n) per level.
- **Total Complexity: O(nlogn)**

Quick Sort Complexity

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

Best Case?

Best-Case / Average-Case RT:

- The pivot **divides the array into two equal halves** at every step.
- The depth of the recursion tree is **O(logn)**
- At each level, all n elements are processed → O(n) per level.
- **Total Complexity: O(nlogn)**

Quick Sort Complexity

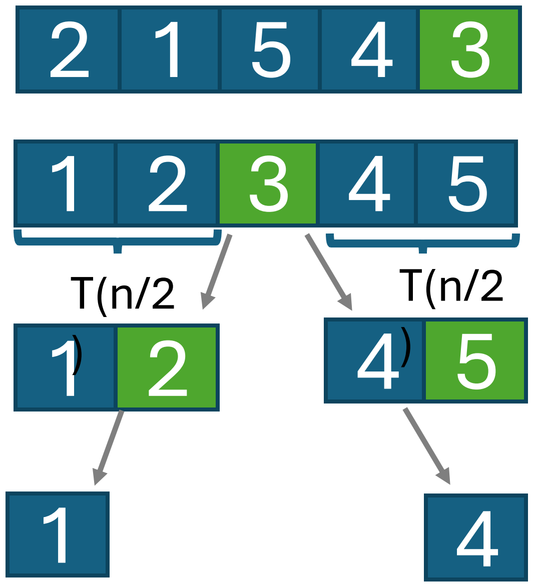
```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

Best Case?

Best-Case / Average-Case RT:

•The pivot **divides the array into two equal halves** at every step.



```
Quicksort(A, start, end):
    if(start<end):                              $O(1)$
        pIndex = Partition(A, start, end)       $O(n)$
        Quicksort(A, start, pIndex-1)           $T\left(\frac{N}{2}\right)+n$
        Quicksort(A, pIndex+1, end)             $T\left(\frac{N}{2}\right)+n$
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])      }
            pIndex = pIndex+1          } runs O(n) times
    swap(A[pIndex], pivot)
    return pIndex
```

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + n + n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$

$$T(n) = \left(8T\left(\frac{n}{8}\right) + n\right) + 2n$$

$$T(n) = 2^kT\left(\frac{n}{2^k}\right) + kn$$

$$\dots$$

$$T(n) = 2^kT\left(\frac{2^k}{2^k}\right) + kn$$

$$T(n) = 2^kT(1) + \boxed{k}n$$

Stops until T(1) or if the array size=1 or if $\frac{N}{2^k} = 1$

Now, if $n = 2^k$

$n = 2^k$

$k = \log_2 n$

```
Quicksort(A, start, end):
    if(start<end):                                   $O(1)$
        pIndex = Partition(A, start, end)            $O(n)$
        Quicksort(A, start, pIndex-1)                $T\left(\frac{N}{2}\right) + n$
        Quicksort(A, pIndex+1, end)                  $T\left(\frac{N}{2}\right) + n$
```

$$Quick\ Sort = 2^k T\left(\frac{N}{2^k}\right) + k * O(n)$$

Stops until T(1) or if the array size=1 or if $\frac{N}{2^k} = 1$

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log_2 N = \log_2 2^k$$

$$\log_2 N = k$$

...

$$Quick\ Sort = 2^k T\left(\frac{N}{2^k}\right) + k * O(n)$$

$$Quick\ Sort = n * T(1) + logn * O(n)$$

$$Quick\ Sort = n + logn * O(n) = O(nlogn)$$

Quick Sort Complexity

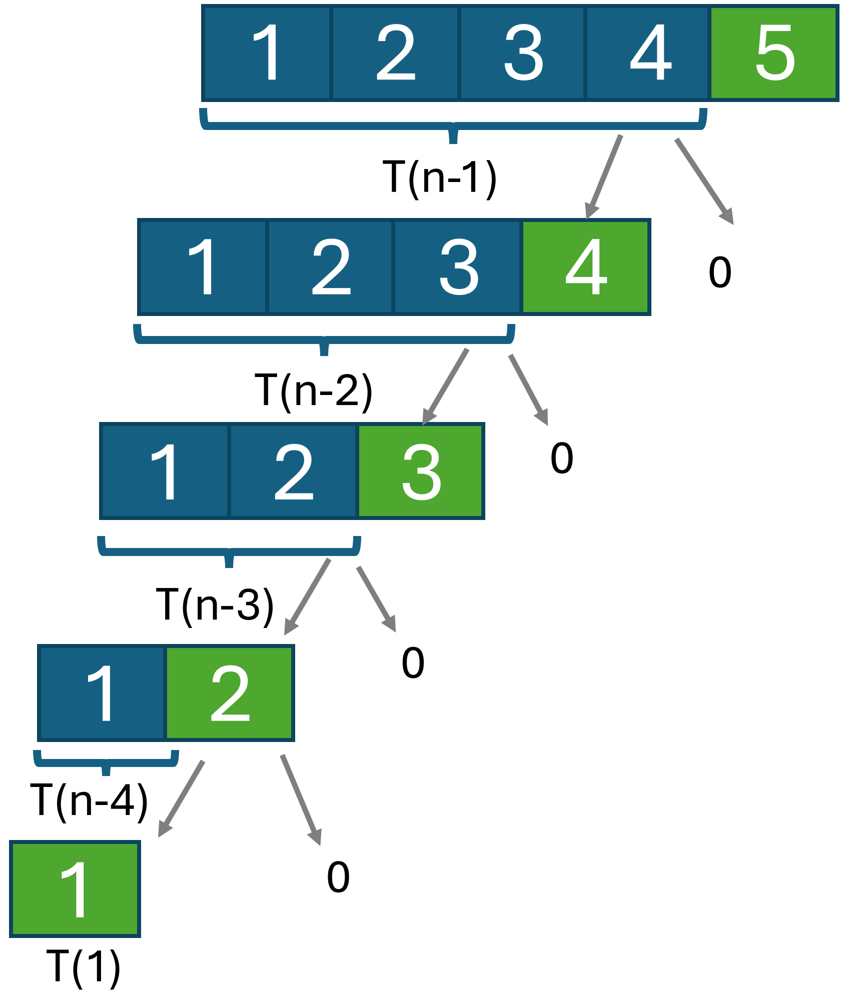
```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)


Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```

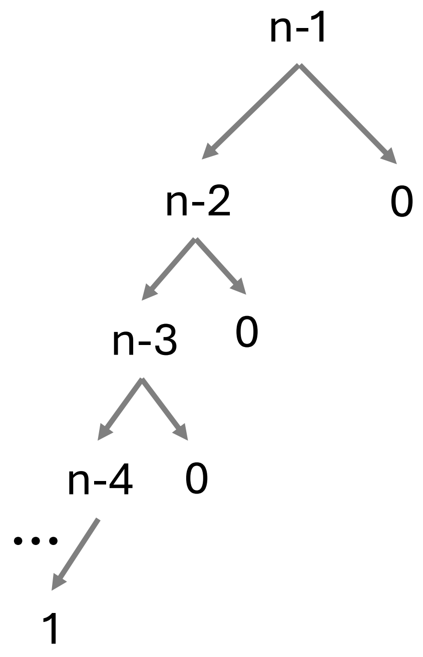
Worst Case?

- If the pivot **always picks the smallest or largest element** (bad choice).
- The array is partitioned **very unbalanced** (one subarray has n-1 elements, the other has 0).
- The recursion tree has a depth of **O(n)**.
- Each level processes O(n) elements.
- **Total Complexity: $O(n^2)$**

Quick Sort Complexity

```
Quicksort(A, start, end):
    if(start<end):                                  $O(1)$
        pIndex = Partition(A, start, end)           $O(n)$
        Quicksort(A, start, pIndex-1)               $T(n-1)+n$
        Quicksort(A, pIndex+1, end)
```





Quick Sort Complexity

```
Quicksort(A, start, end):
    if(start<end):                                   $O(1)$
        pIndex = Partition(A, start, end)            $O(n)$
        Quicksort(A, start, pIndex-1)                $T(n-1)+n$
        Quicksort(A, pIndex+1, end)
```

$$T(n) = T(n-1) + n$$

$$T(n) = [T(n-2) + (n-1)] + n$$

$$T(n) = [T(n-3) + n - 2] + (n-1) + n$$

$$T(n) = [T(n-4) + n - 3] + (n-2) + (n-1) + n$$

$$T(n) = T(n-k) + [(n-k) \ldots + (n-3) + (n-2) + (n-1) + n]$$

Ends if n-k = 1

Looks familiar right? It's a series!

$$T(n) = T(1) + [1 \ldots + (n-3) + (n-2) + (n-1) + n]$$

$$T(n) = O(1) + \frac{n(n+1)}{2}$$

$$T(n) = O(1) + O(n^2)$$

$$T(n) = \boldsymbol{O(n^2)}$$

Quick Sort Complexity: Choosing a Pivot

- Choosing the pivot is important for the performance of quicksort
- **Example:** If we always choose the first element of the array as pivot, given an already sorted array, quicksort will run in $O(N^2)$ → sum of arithmetic series 1 + 2 +..+ N = N(N+1) / 2 ∊ O(N2)
- **Example**: If we somehow always select the median element of array as pivot (best case), quicksort will run in O(N log N) → similar to merge sort, divide in half; height of recursion tree is O(log2 N), while partition runs in O(N) per level
- **Goal**: Choose pivot that makes two nearly balanced partitions
- **Solution**: Choose pivot randomly; each item has equal chance to be chosen
- Random pivots are pretty good, often enough
- Getting a 25-75 split is usually good enough; half of the array elements give a 25-75 split or better, if used as a pivot
- Running time depends on the quality of pivot

Quick Sort Complexity:

- Memory: O(1) → using in-place partition

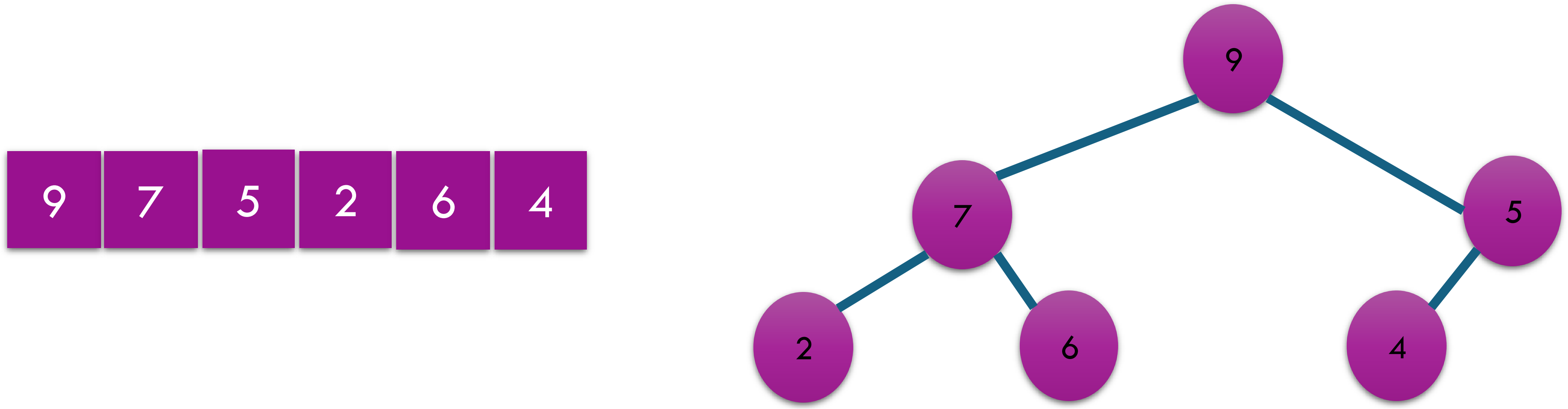
Heap Sort

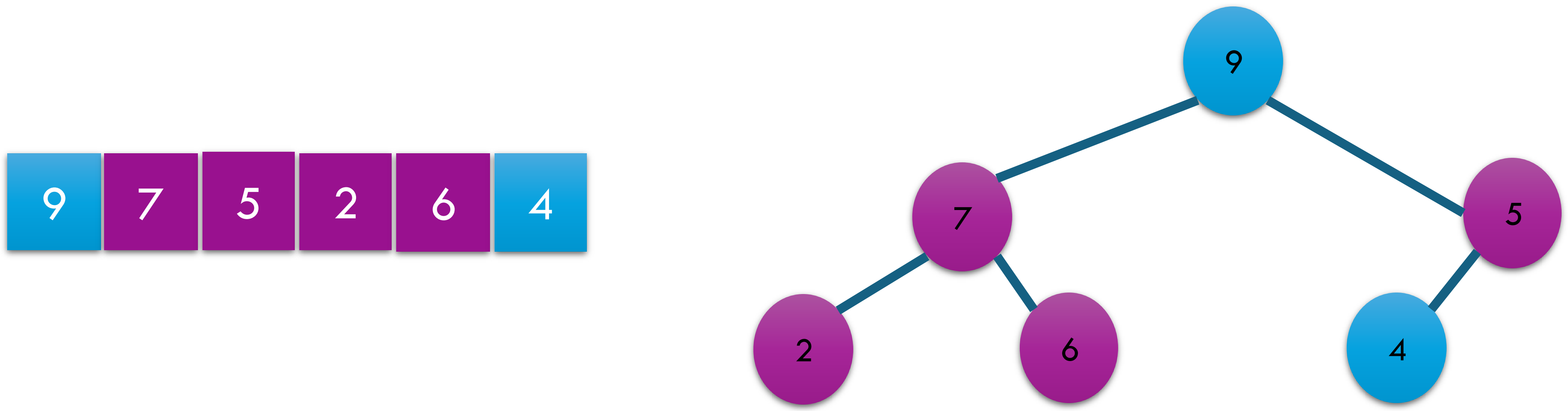
Heap sort

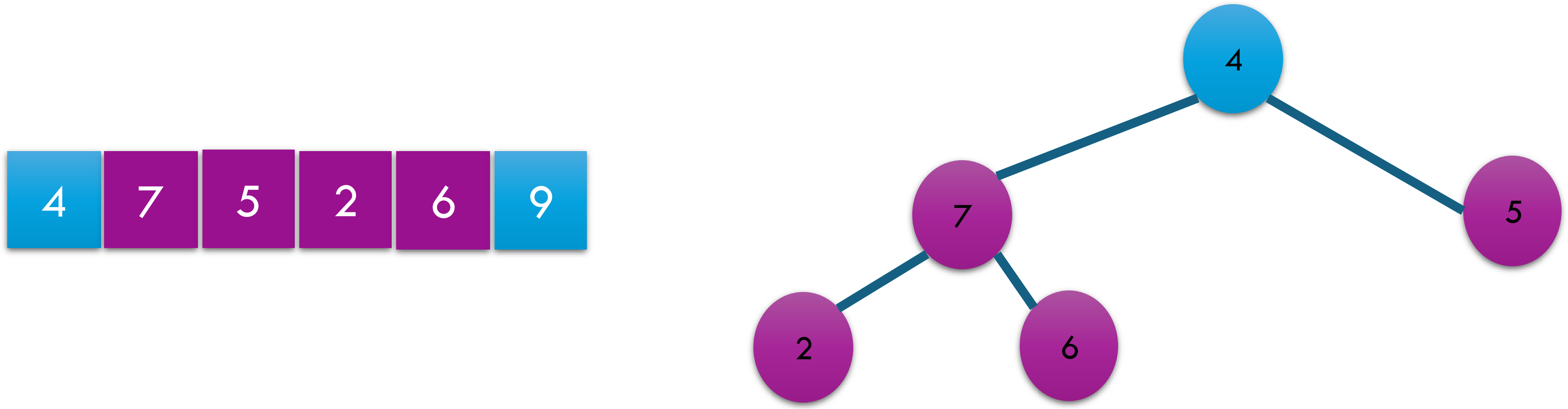
- **heap sort**: an algorithm to sort an array of N elements by turning the array into a heap, then doing a `remove` N times
  - the elements will come out in sorted order!
  - we can put them into a new sorted array
  - what is the runtime?

A max-heap

- the heaps shown have been minimum heaps because the elements come out in ascending order
- a *max-heap* is the same thing, but with the elements in descending order

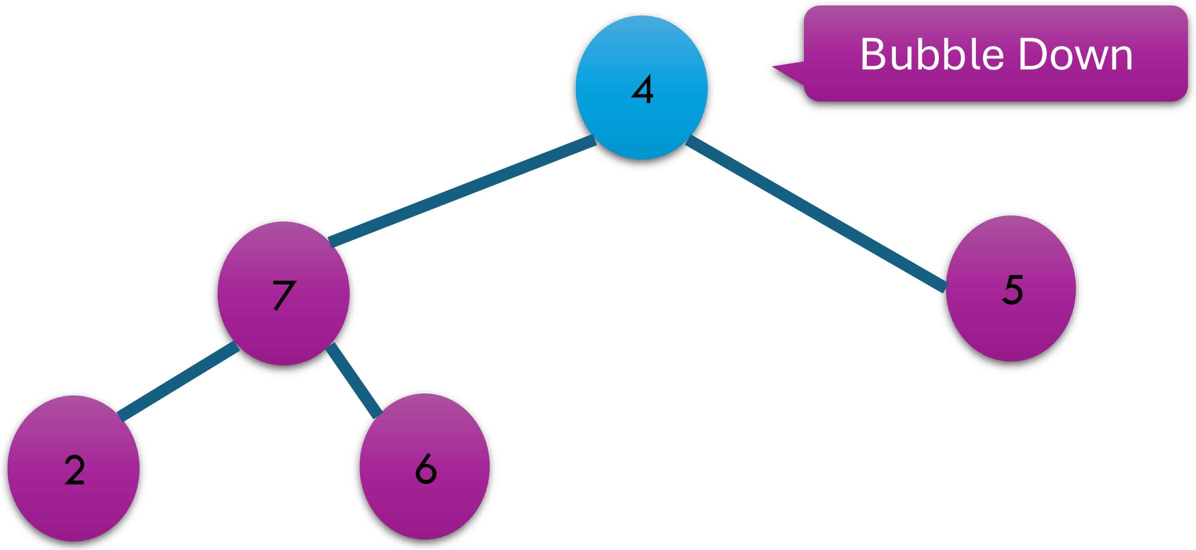

9
7
5
2
6
4
9
7
5
2
6
4


9
7
5
2
6
4
9
7
5
2
6
4


4
7
5
2
6
9
4
7
5
2
6

4	7	5	2	6

9

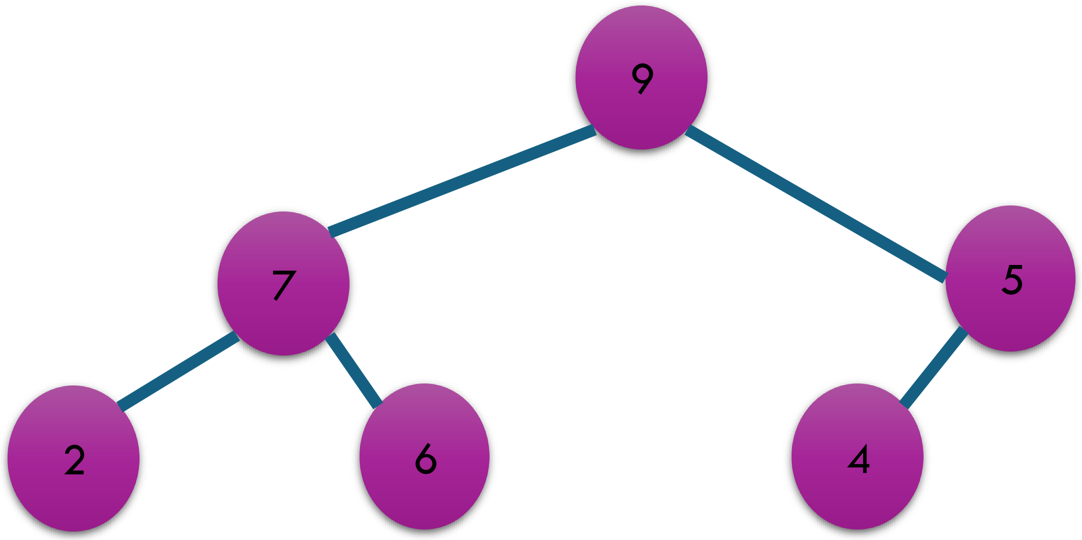


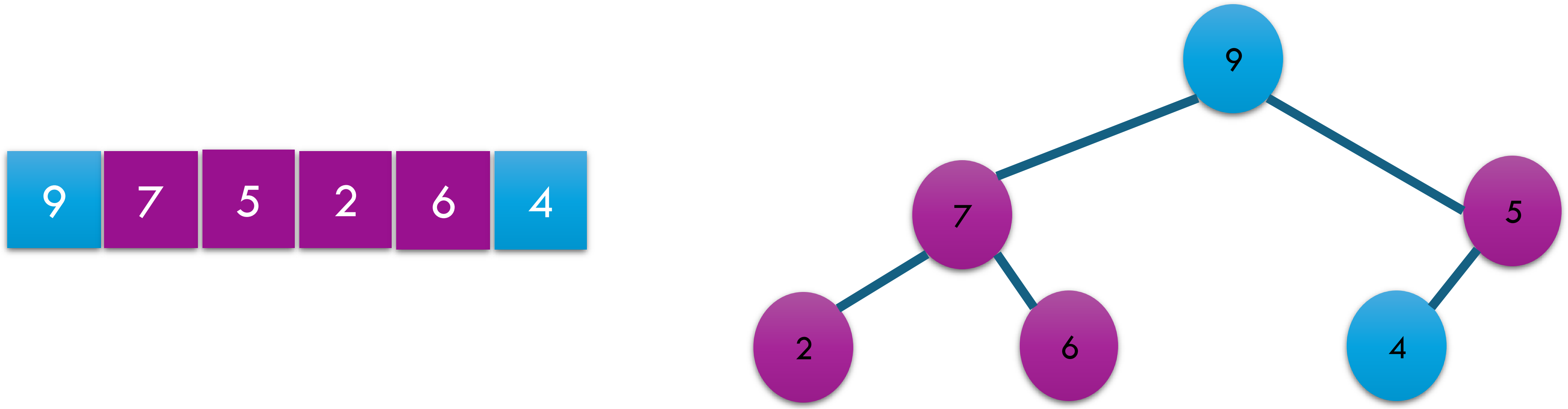
Improved heap sort

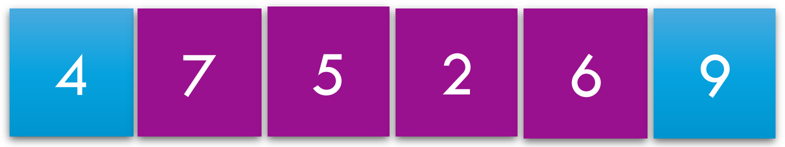
- the heap sort shown requires a second array

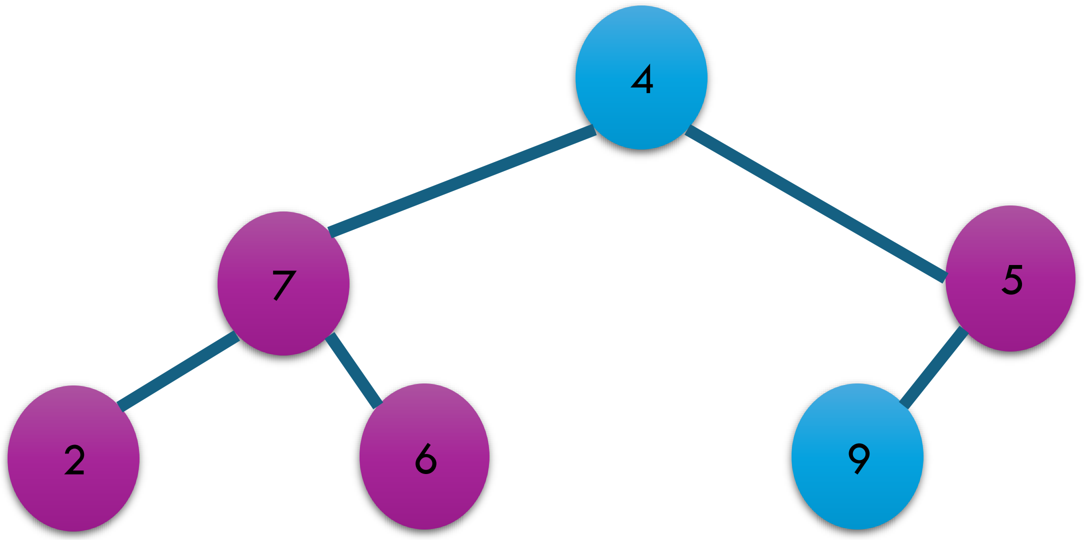
- we can use a max-heap to implement an improved version of heap sort that needs no extra storage
    - O(*n* log *n*) runtime
    - no external storage required!
    - useful on low-memory devices
    - elegant

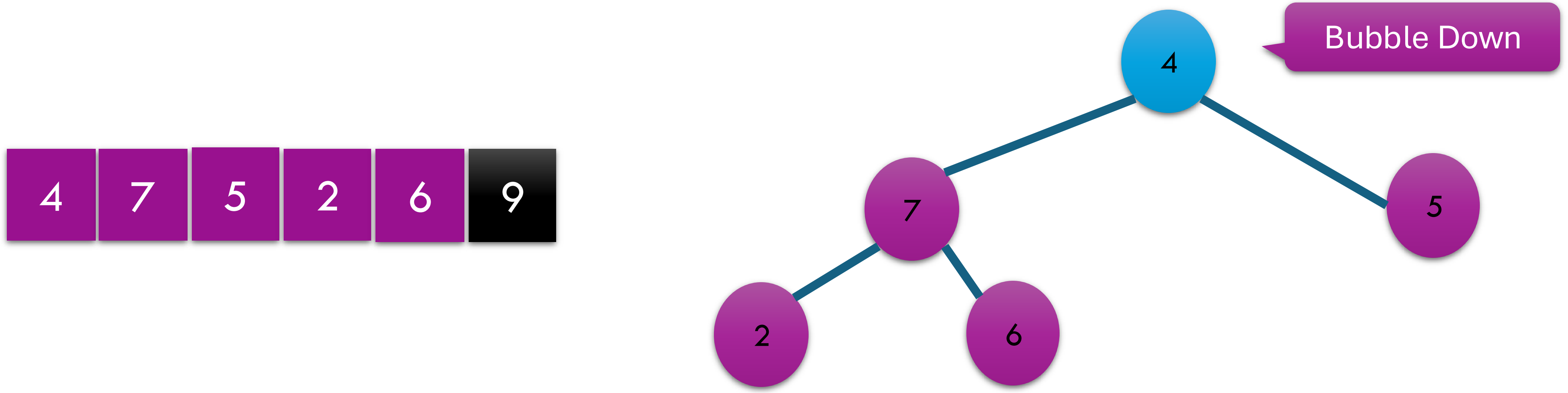
9	7	5	2	6	4

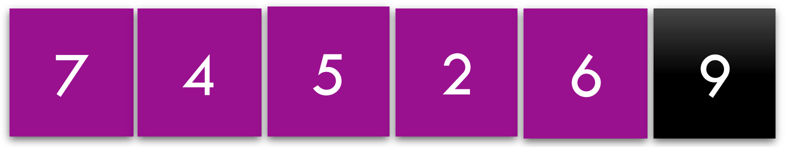


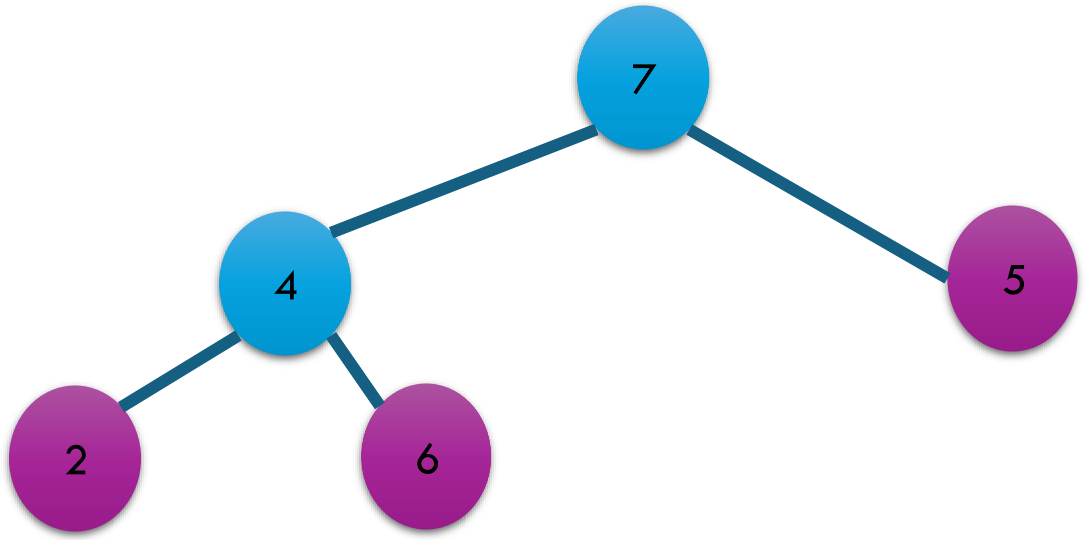

9
7
5
2
6
4
9
7
5
2
6
4

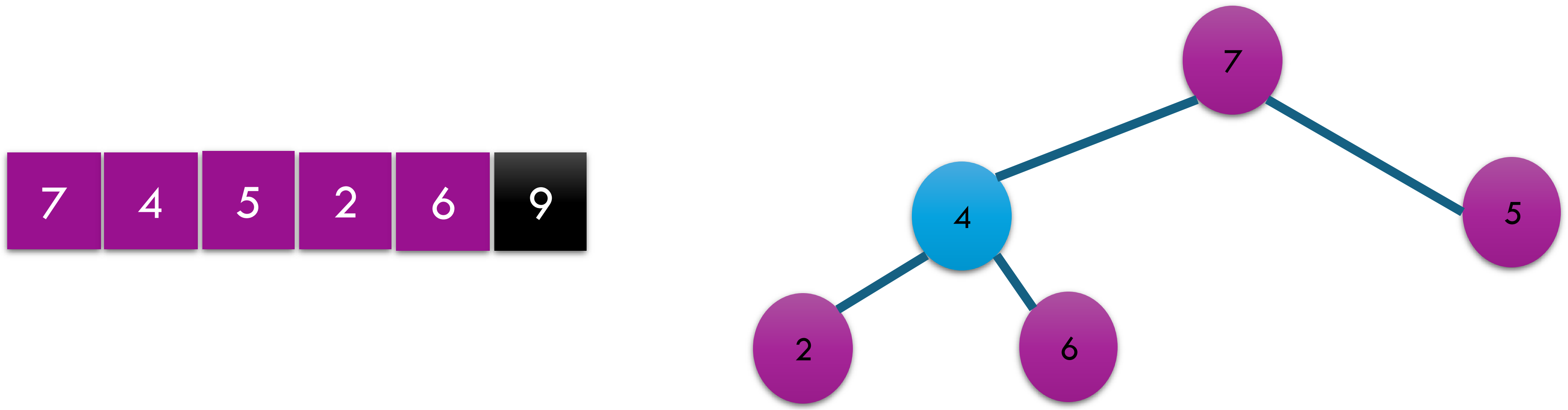

4
7
5
2
6
9

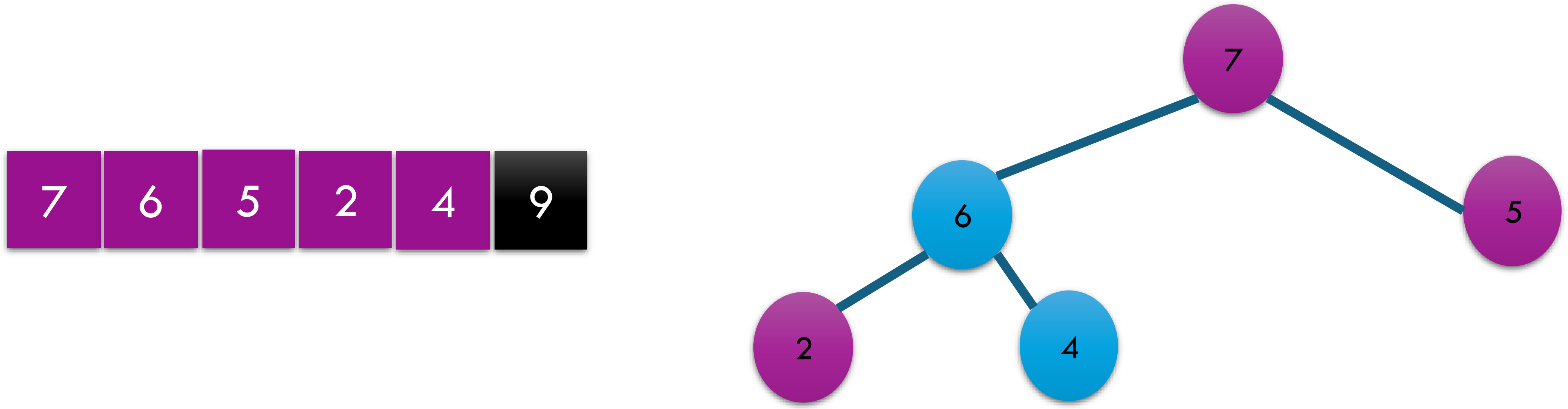

4
7
5
2
6
9

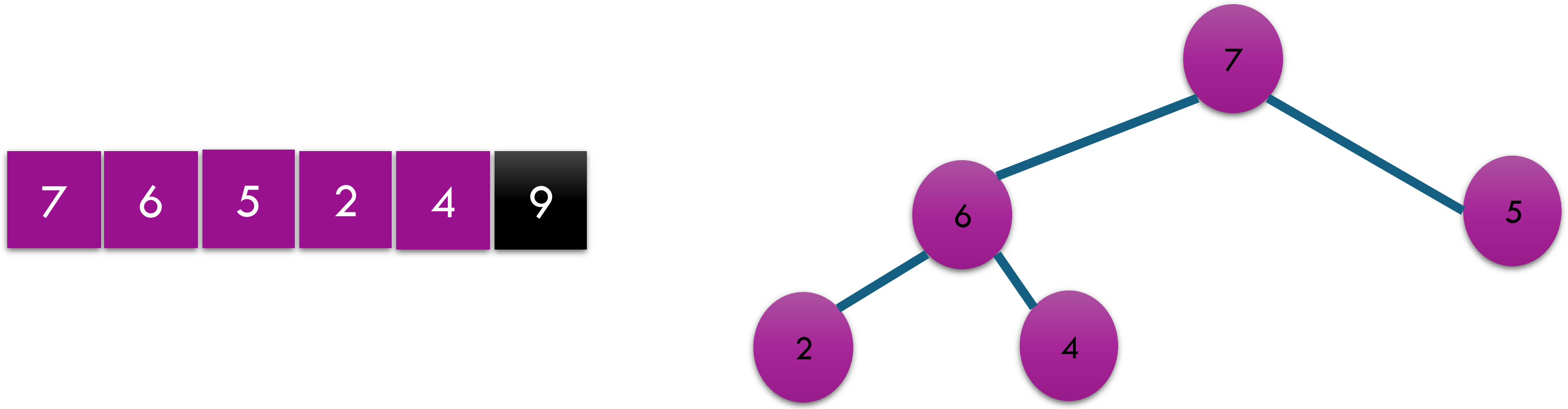

4
7
5
2
6
9
Bubble Down
4
7
5
2
6

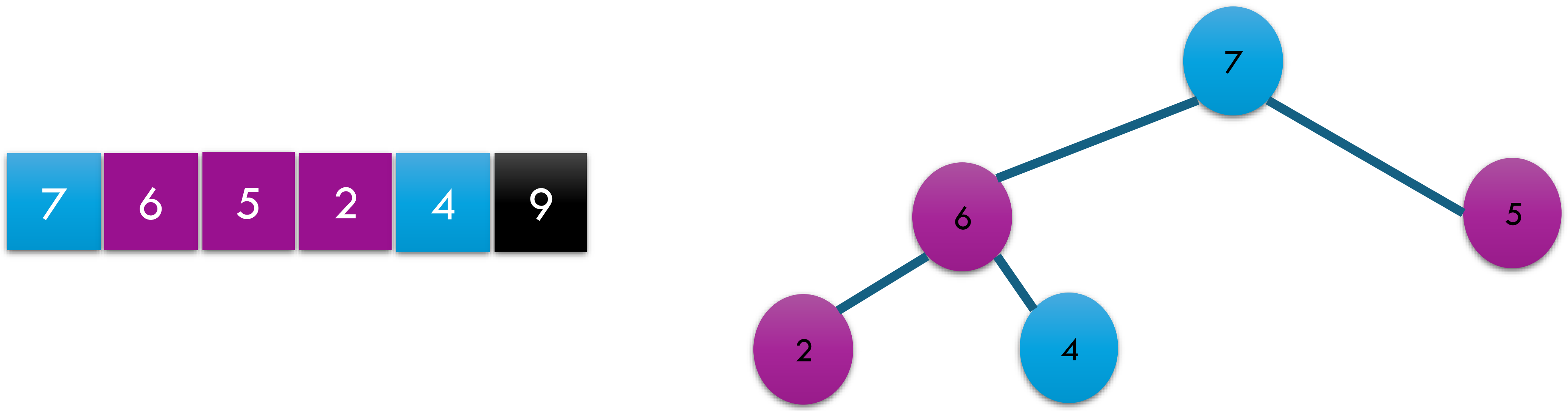

7
4
5
2
6
9

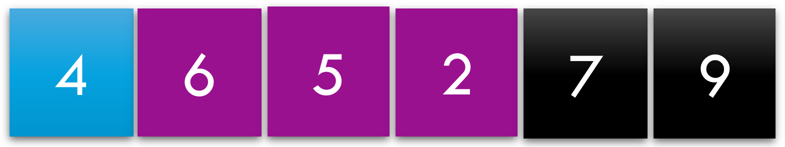

7
4
5
2
6

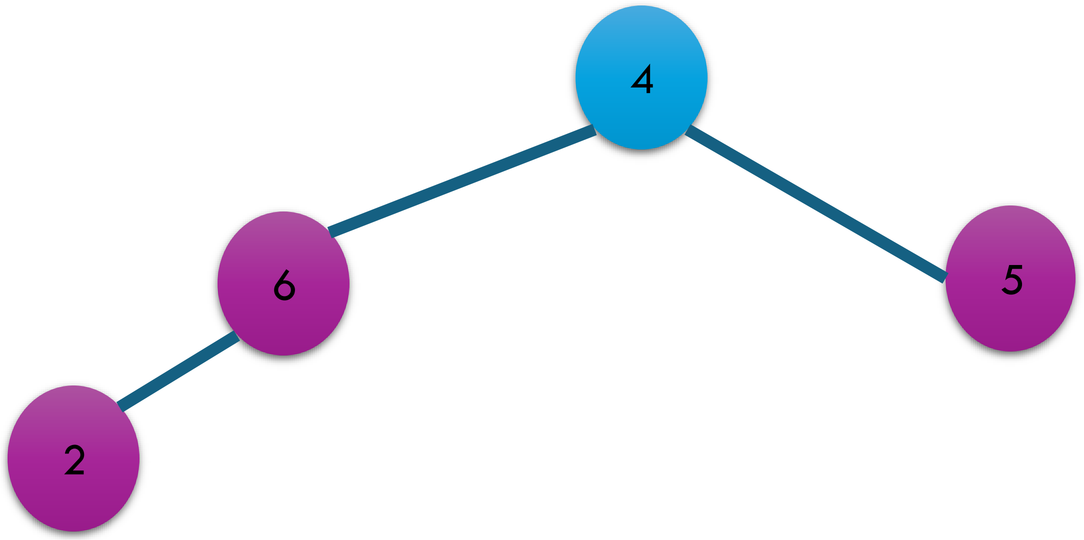

7
4
5
2
6
9
7
4
5
2
6

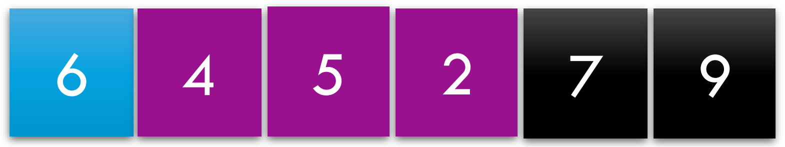

7
6
5
2
4
9
7
6
5
2
4

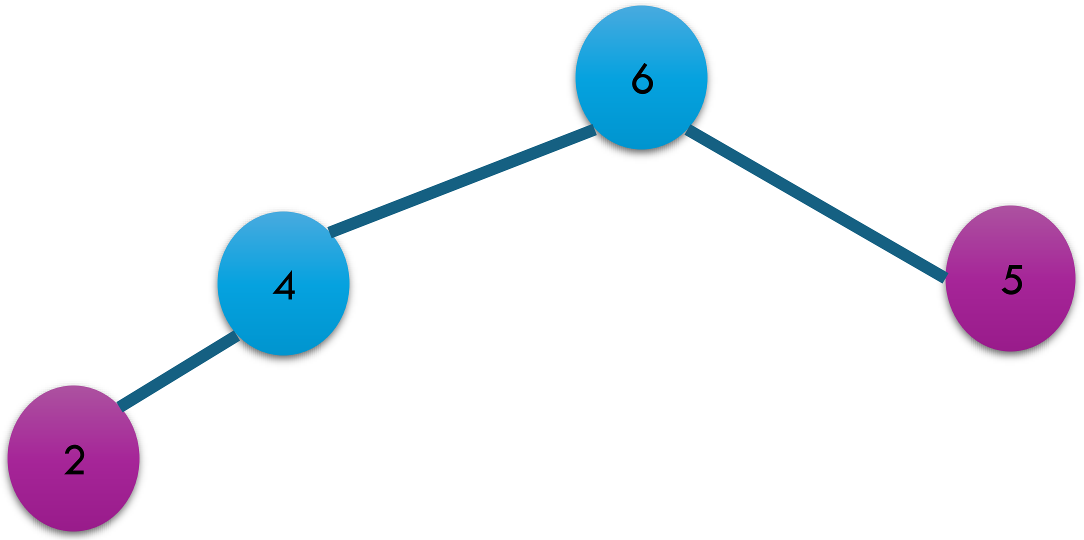

7
6
5
2
4
9
7
6
5
2
4

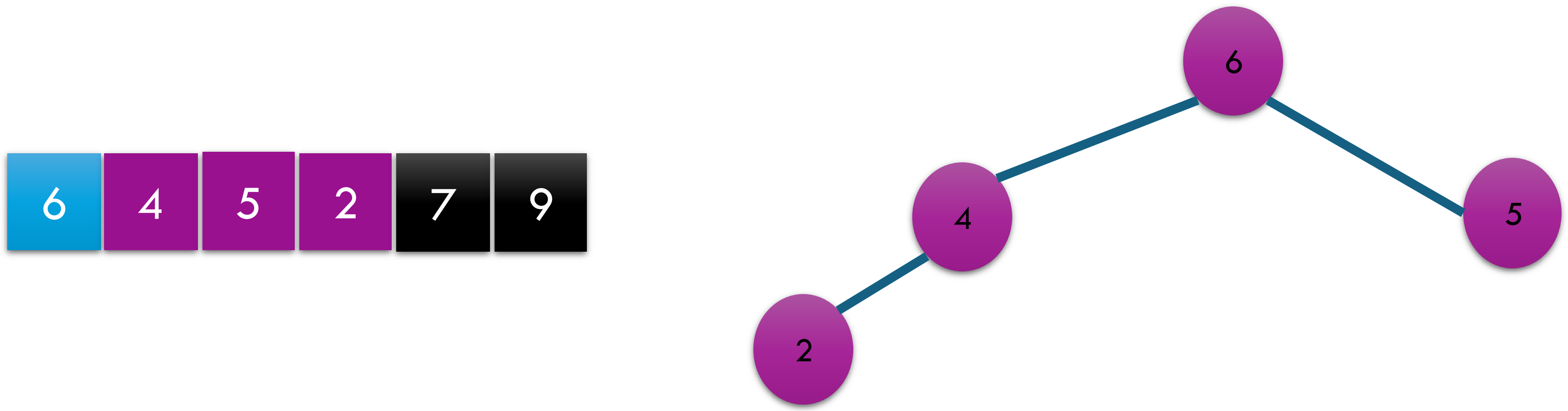

7
6
5
2
4
9
7
6
5
2
4

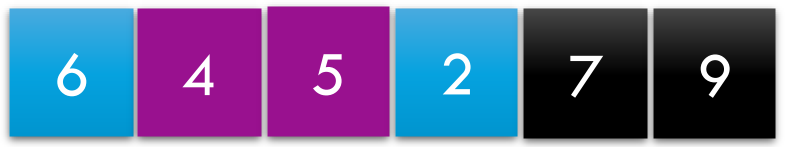

4
6
5
2
7
9

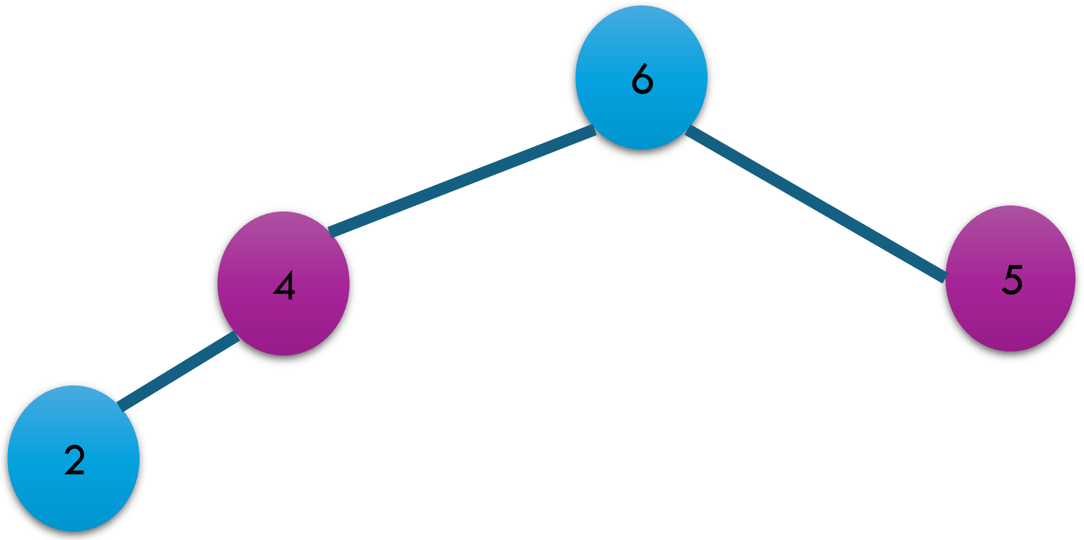

4
6
5
2

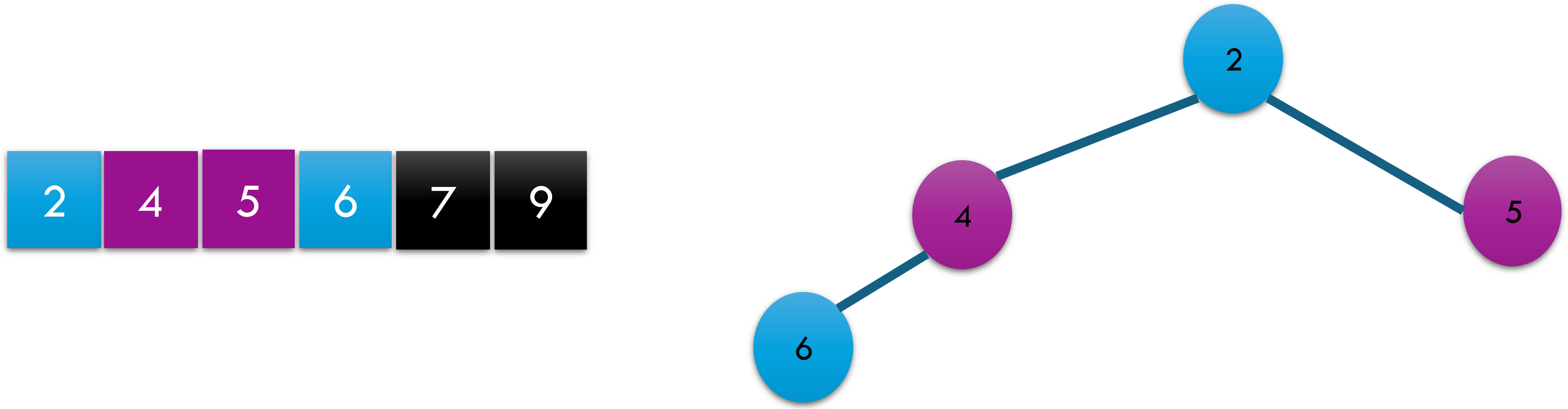

6
4
5
2
7
9


6
4
5
2

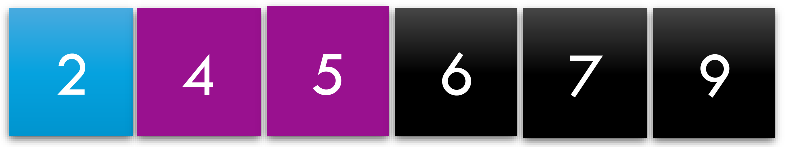

6
4
5
2
7
9
6
4
5
2

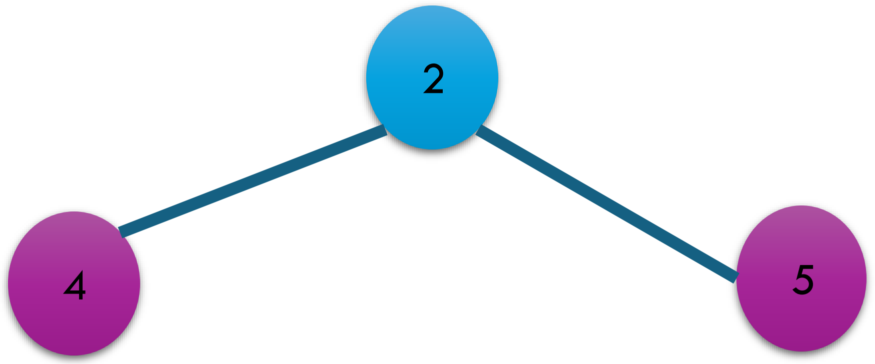

6
4
5
2
7
9

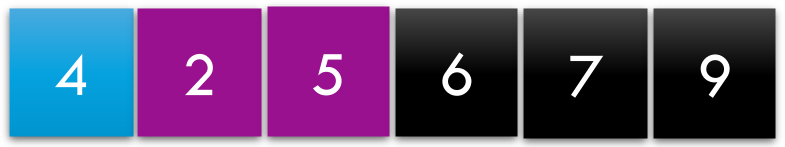

6
4
5
2

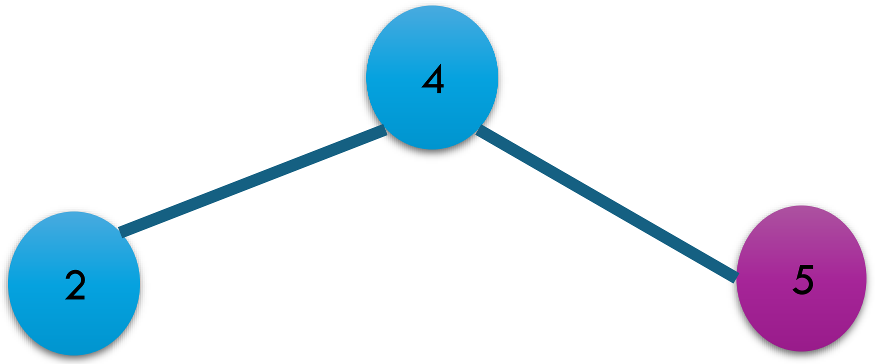


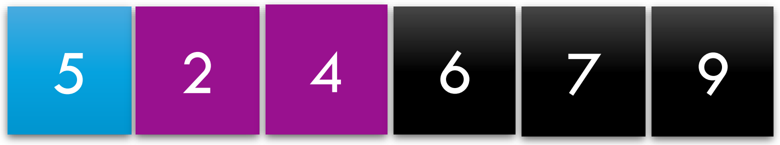
2
4
5
6
7
9
2
4
5
6

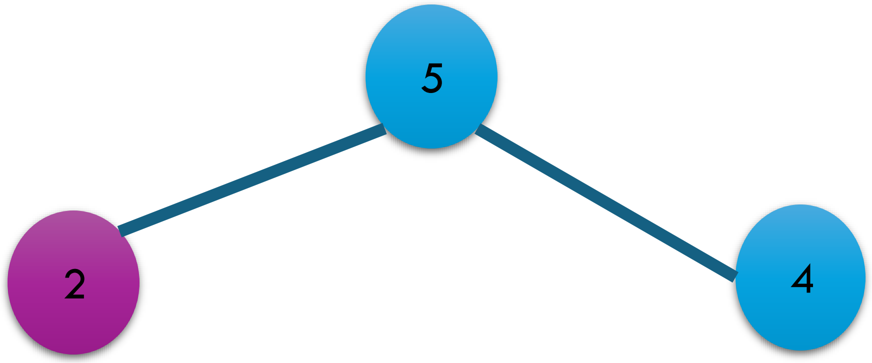

2
4
5
6
7
9

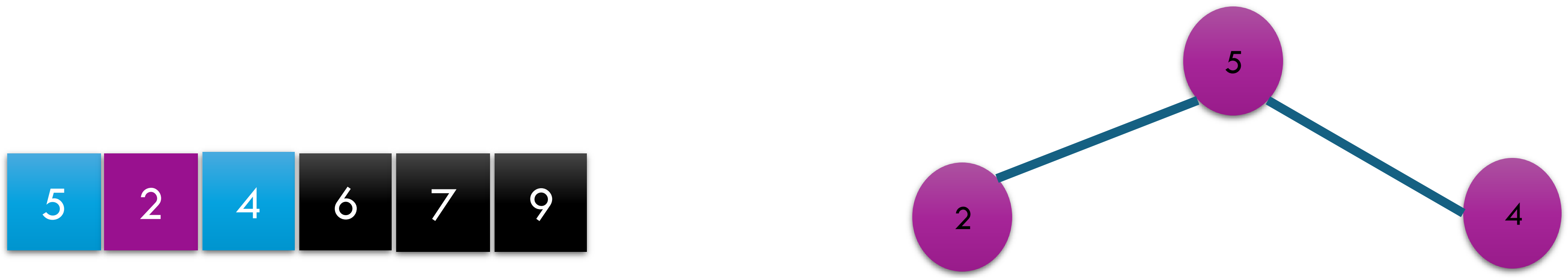

2
4
5

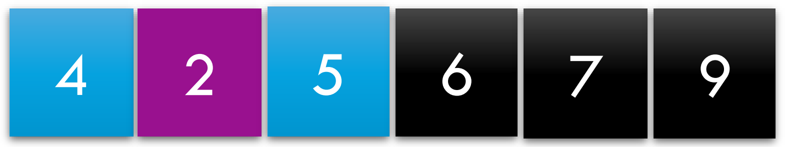

4
2
5
6
7
9

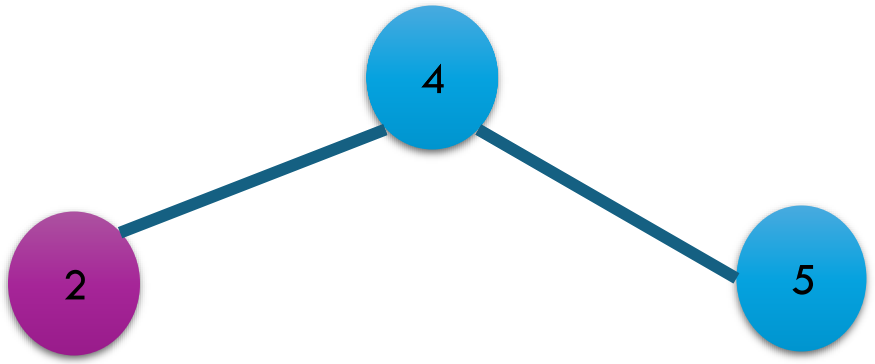

4
2
5

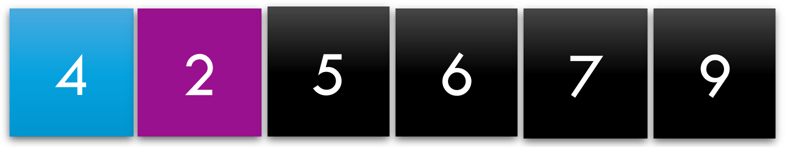

5
2
4
6
7
9

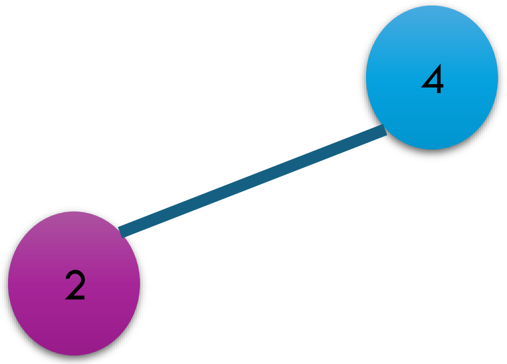

5
2
4

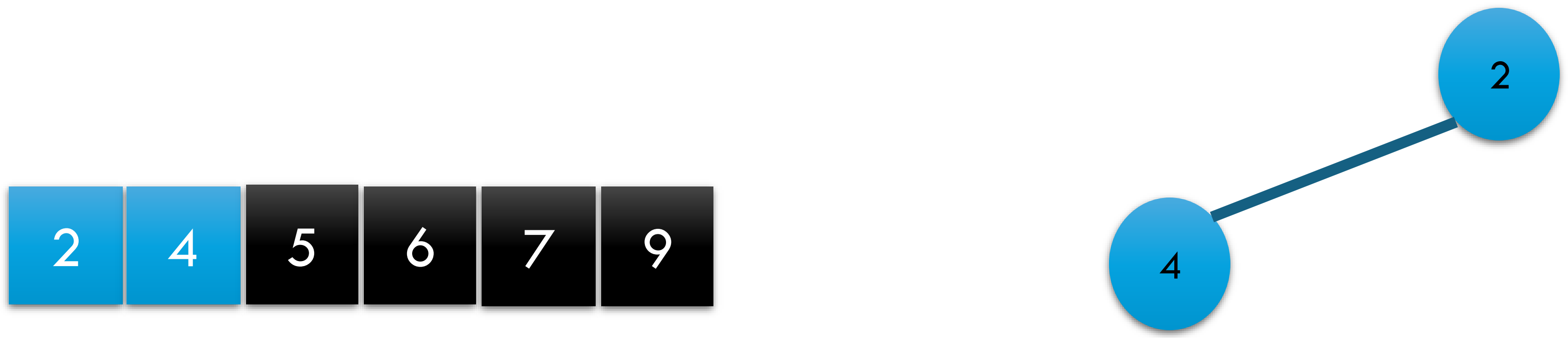

5
2
4
6
7
9
5
2
4

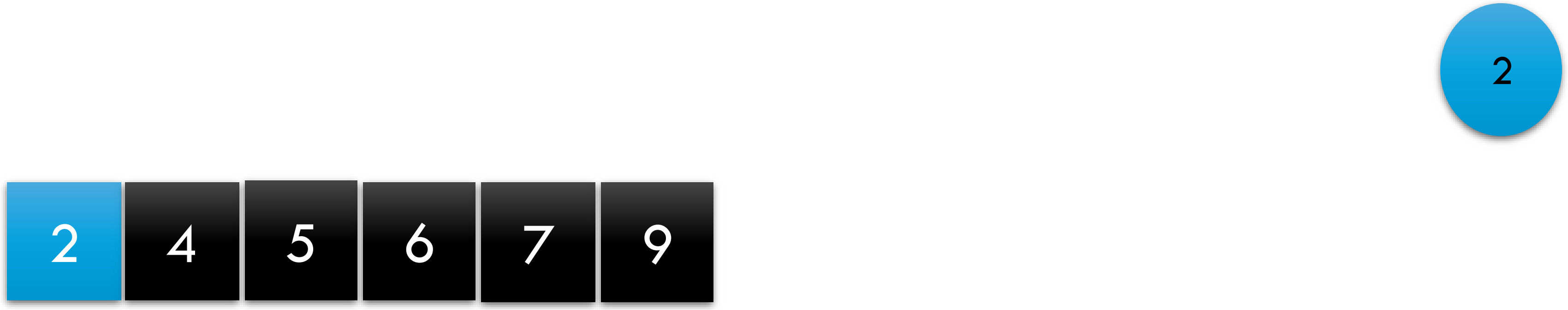

4
2
5
6
7
9


4
2
5


4
2
5
6
7
9


4
2


2
4
5
6
7
9
2
4


2
4
5
6
7
9
2

2	4	5	6	7	9

What’s the running time of heapsort?

What’s the running time of heapsort?

- Presorting, or building the initial heap would require O(n) time
- Then for the Heapsort, it would require O(nlogn)
- Everytime we remove an element (the max element), it performs the downheap operation to maintain the heap ordering, which has a complexity of O(logn)
- For every node n in the heap, removeMin of logn, so n*logn

Quiz

Sort the following using quicksort. Show each step. Count the total number of timesteps to complete the task.

57	64	70	85	13	22	39	48

```
Quicksort(A, start, end):
    if(start<end):
        pIndex = Partition(A, start, end)
        Quicksort(A, start, pIndex-1)
        Quicksort(A, pIndex+1, end)
```

```
Partition(A, start, end):
    pivot = A[end]
    pIndex = start
    for i=start to end-1:
        if A[i] <= pivot:
            swap(A[i], A[pIndex])
            pIndex = pIndex+1
    swap(A[pIndex], pivot)
    return pIndex
```