Quick Sort

Instructor: Krishna Venkatasubramanian

CSC 212

Partly based on slides from Prof. Carolina Ruiz's CS2223 at WPI

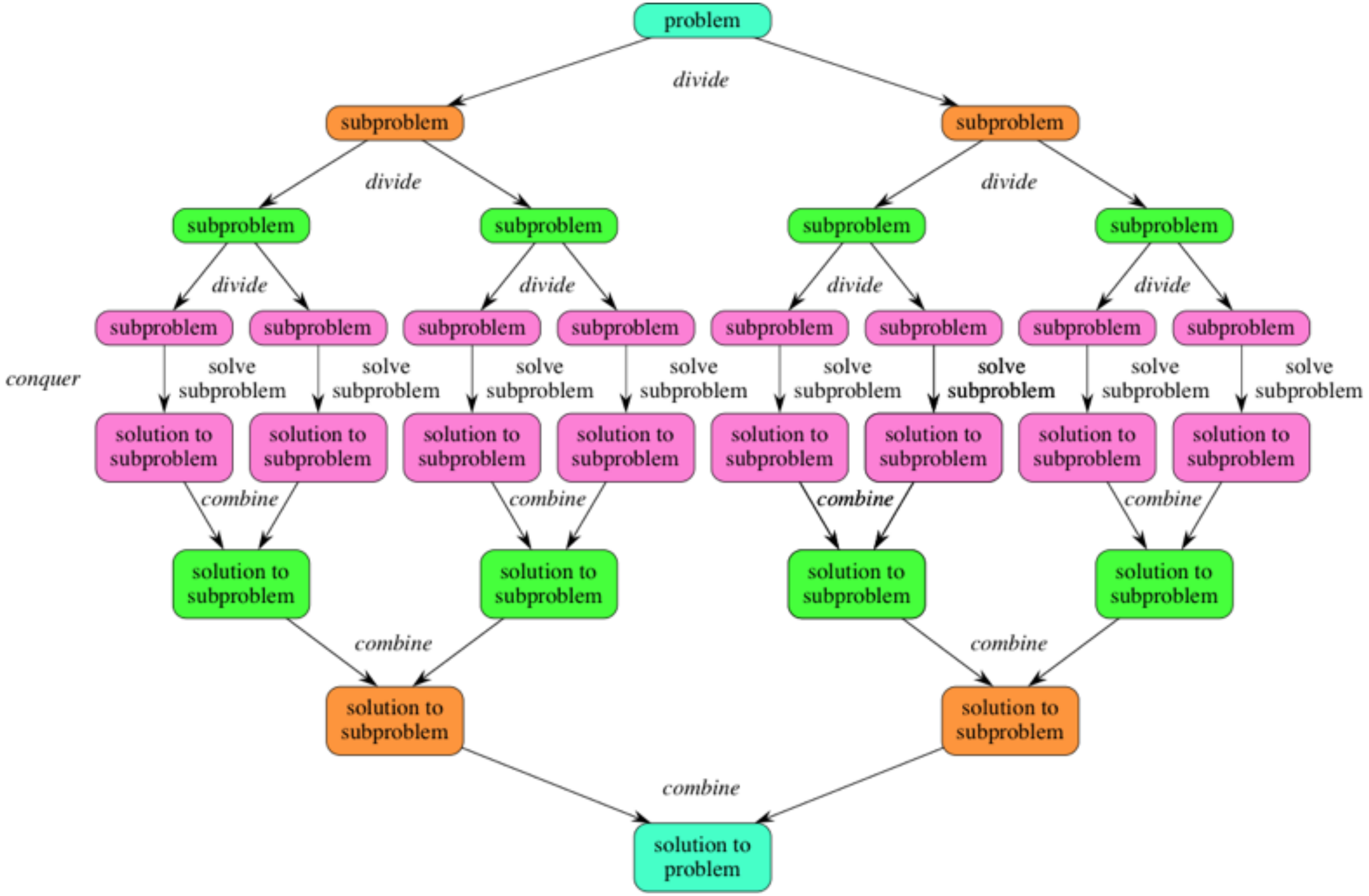
Divide and Conquer Algorithms

- **Divide:** Break the larger problem into sub-problems that are smaller instances of the same problem

- **Conquer:** the sub-problems are solved ***recursively***
  - If the sub-problem is really small, then solve in a straight-forward manner

- **Combine:** combine the solutions of the sub-problems to find the solution of the original problem!

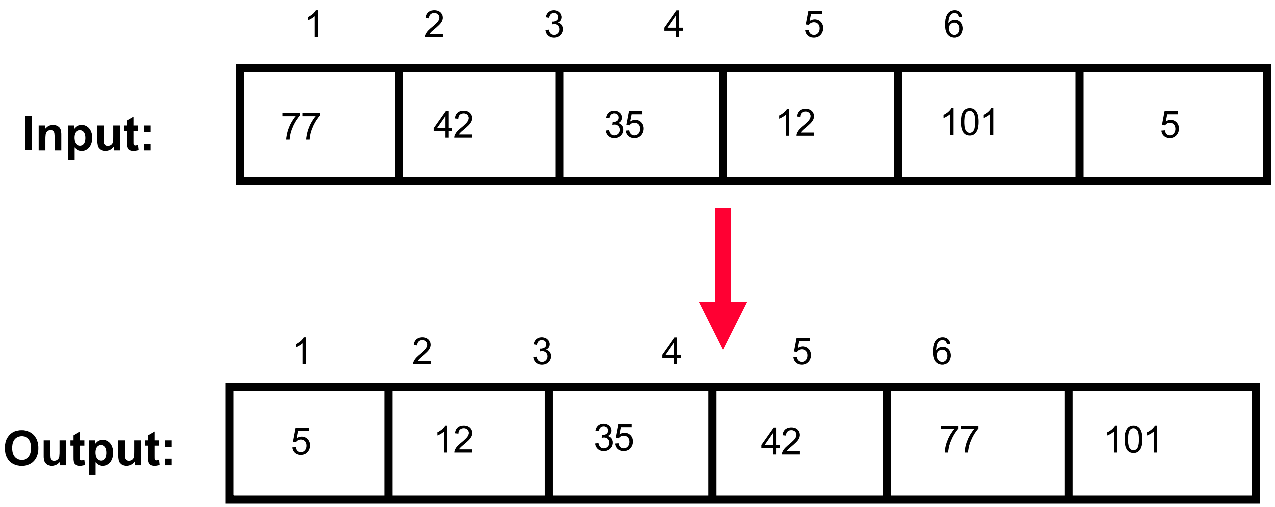
Visually Speaking



https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms

Sorting: Problem Definition

- **Sorting takes an unordered collection and makes it an ordered one.**



Input:

1	2	3	4	5	6
77	42	35	12	101	5

Output:

1	2	3	4	5	6
5	12	35	42	77	101

How can we sort an array using divide and conquer approach?

Sorting Algorithms

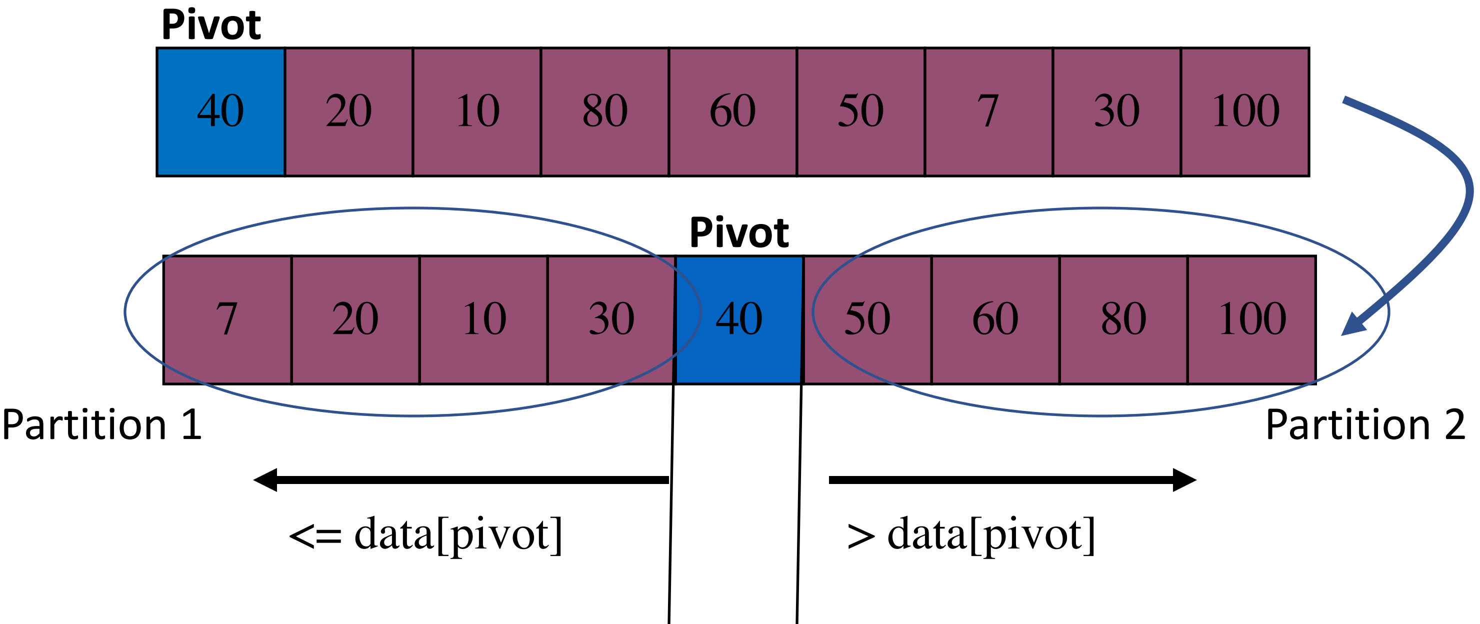
- *Insertion Sort --- covered already*
- *Bubble Sort --- covered already*
- *Selection Sort --- covered already*
- *Merge Sort --- covered already*
- **Quick Sort**
- Linear-Time Sort
- Heap Sort
- …

Quicksort Algorithm

```
QuickSort(A,left,right)
  if right-left +1 == 1
    return
  else
    pivot = Partition(A,left,right)
    QuickSort(A,left, pivot)
    QuickSort(A,pivot+1,right)
```

How to Partition

- Given an array A
  - Pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements **less than or equal to pivot**
    - Elements **greater than pivot**



Pivot

40	20	10	80	60	50	7	30	100

Pivot

7	20	10	30	40	50	60	80	100

Partition 1

Partition 2

<= data[pivot]

> data[pivot]

Partition Example

We are given array of n integers to sort:

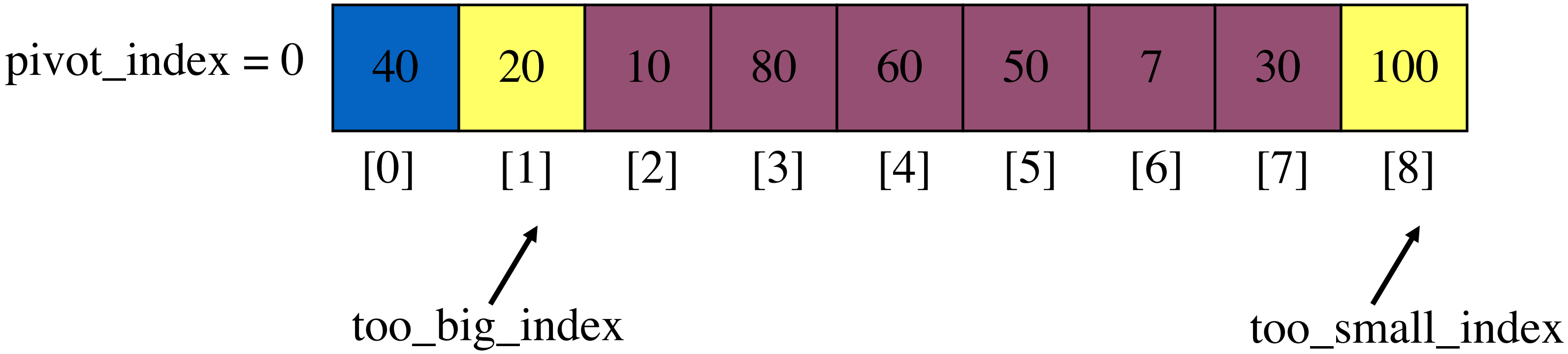
40	20	10	80	60	50	7	30	100

Partition Example

There are a **number of ways to pick the pivot element**. In this example, we **will use the first element** in the array:

40	20	10	80	60	50	7	30	100

Partition Example (Pseudocode)



pivot_index = 0

40	20	10	80	60	50	7	30	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

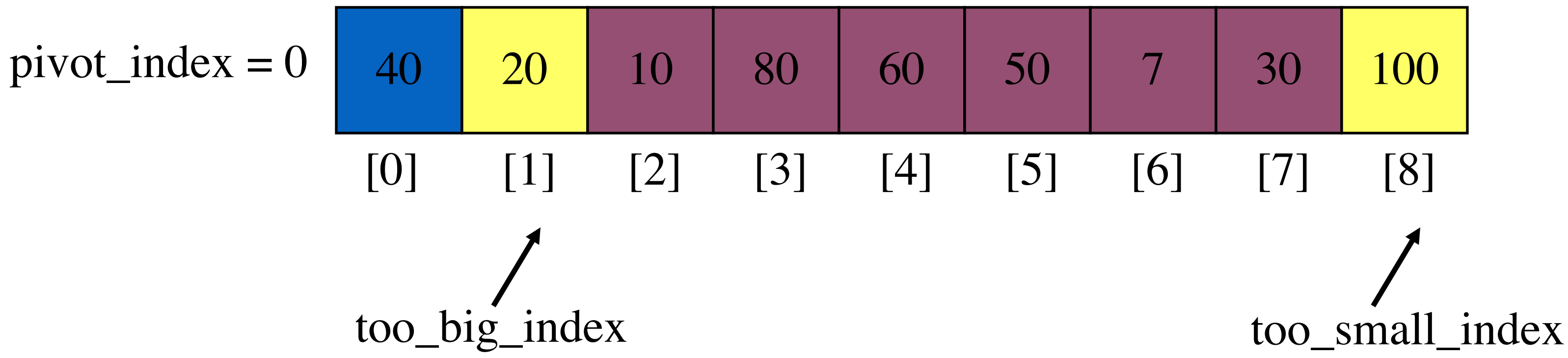
too_big_index

too_small_index

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
```

Note the pre-increment



pivot_index = 0

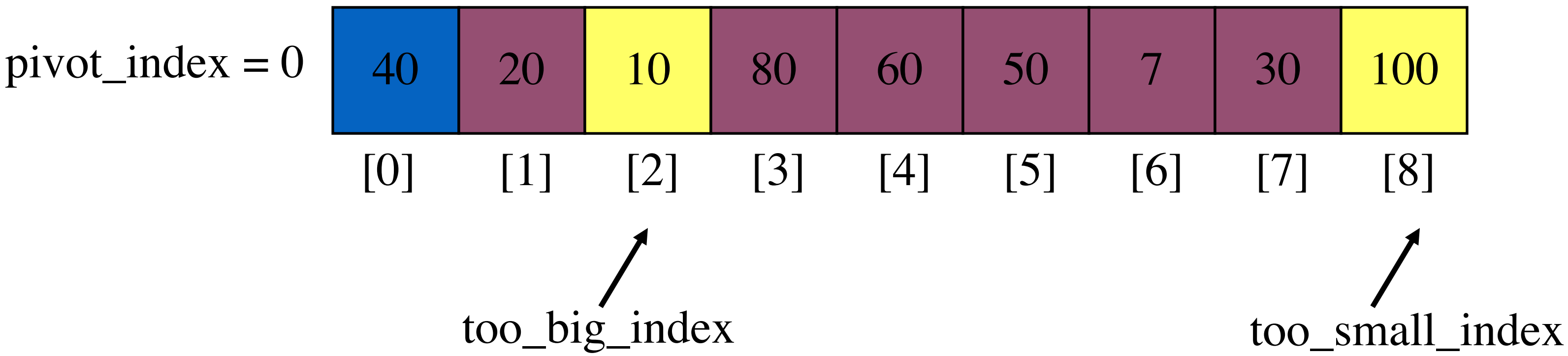
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	80	60	50	7	30	100

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1.  while data[too_big_index] <= data[pivot]
            too_big_index = too_big_index+1
```



pivot_index = 0

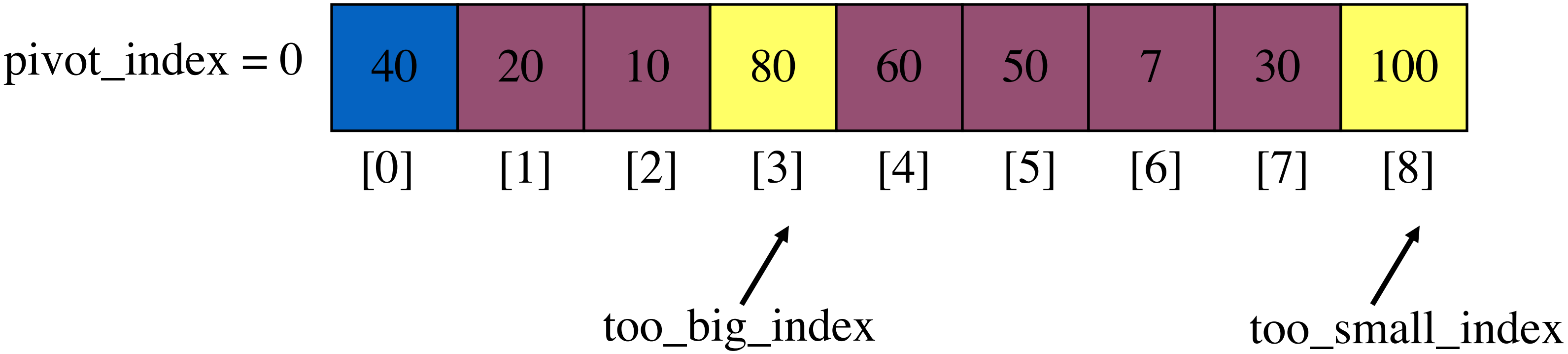
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	80	60	50	7	30	100

too_big_index

too_small_index

Partition Example (Pseudocode)

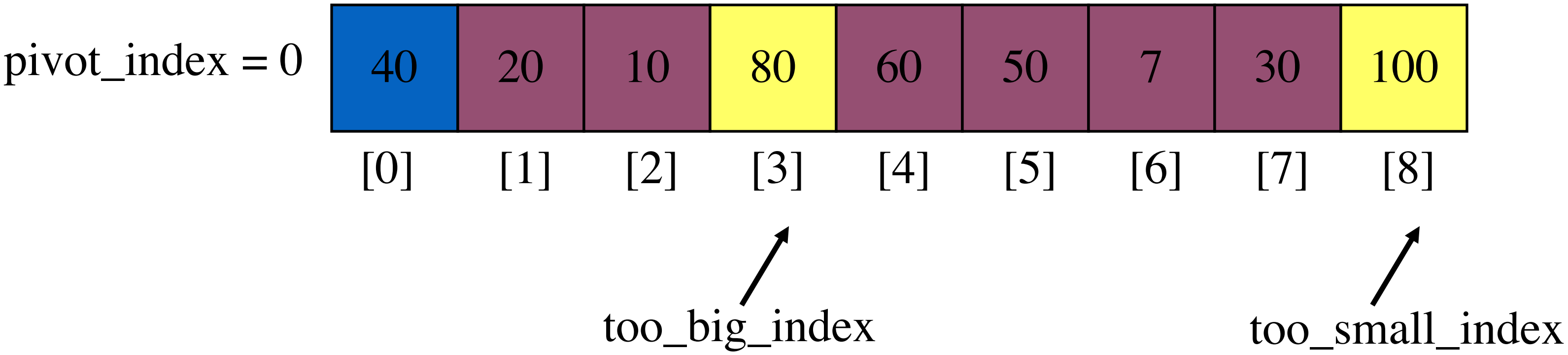
```
→ 1. while data[too_big_index] <= data[pivot]
         ++too_big_index
```



Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
       ++too_big_index
2. while data[too_small_index] > data[pivot]
       --too_small_index
```

Note the pre-decrement



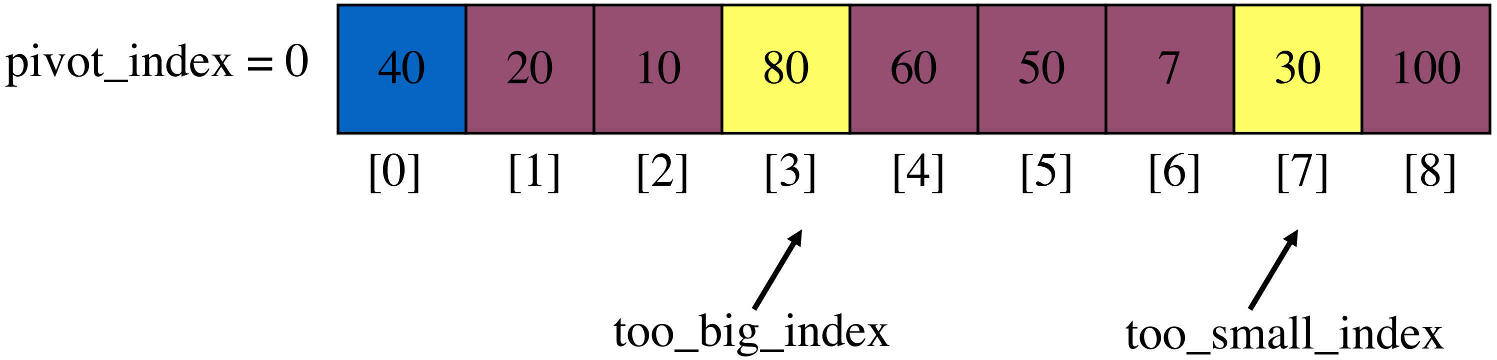
pivot_index = 0	40	20	10	80	60	50	7	30	100
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
2. while data[too_small_index] > data[pivot]
        --too_small_index
```



pivot_index = 0

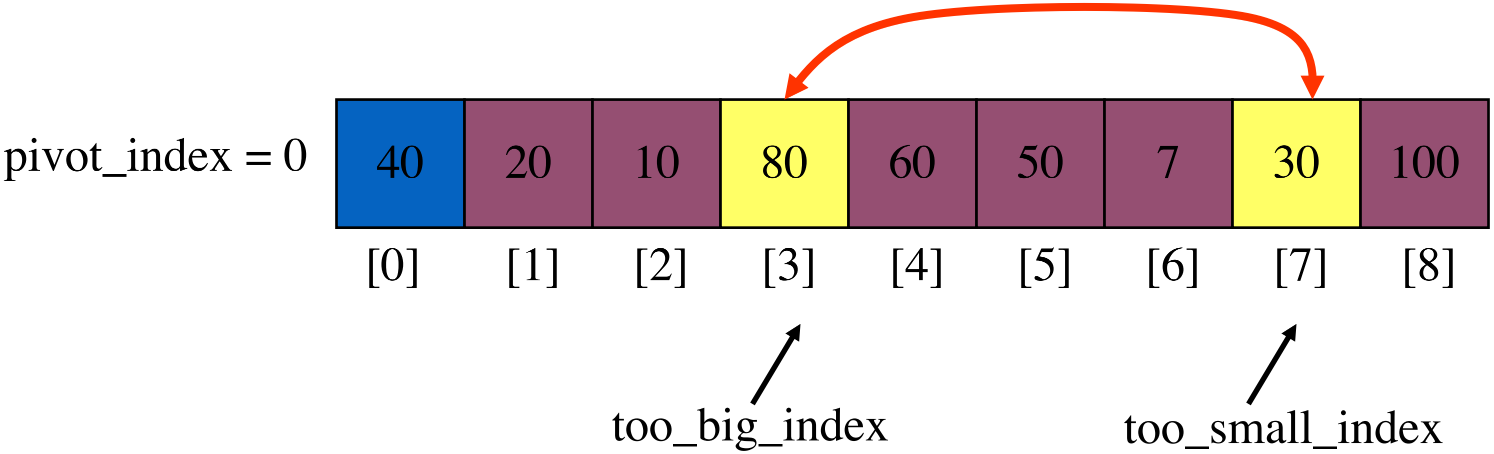
40	20	10	80	60	50	7	30	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

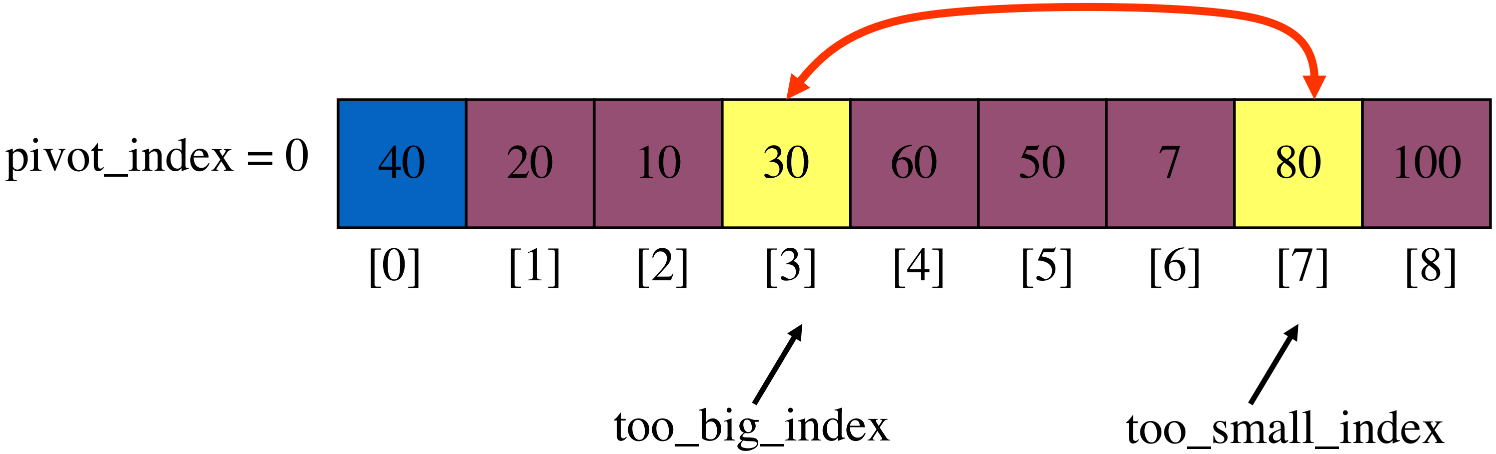
Partition Example (Pseudocode)

```
1.  while data[too_big_index] <= data[pivot]
        ++too_big_index
2.  while data[too_small_index] > data[pivot]
        --too_small_index
3.  if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
```



Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
2. while data[too_small_index] > data[pivot]
        --too_small_index
3. if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
```



pivot_index = 0

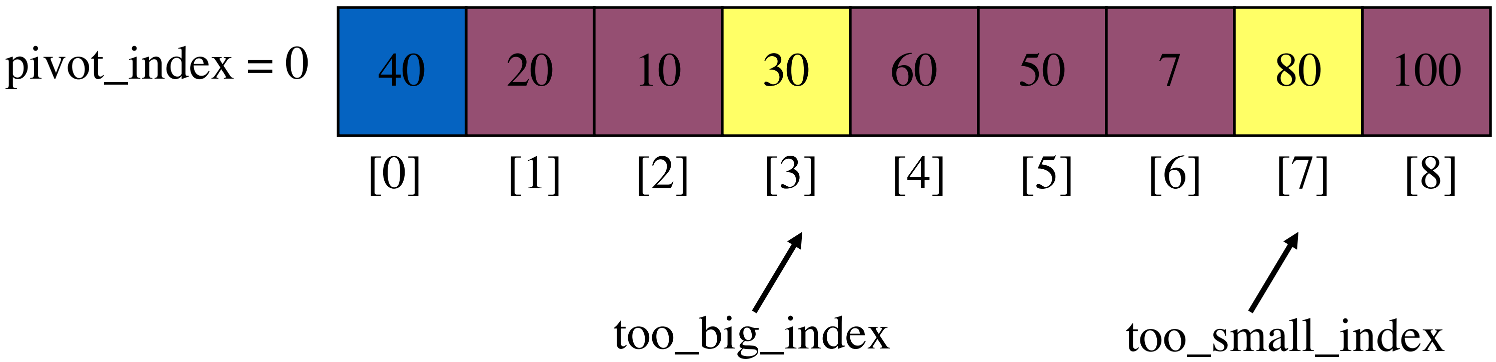
40	20	10	30	60	50	7	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
2. while data[too_small_index] > data[pivot]
        --too_small_index
3. if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
4. while too_small_index >= too_big_index, go to Step 1.
```

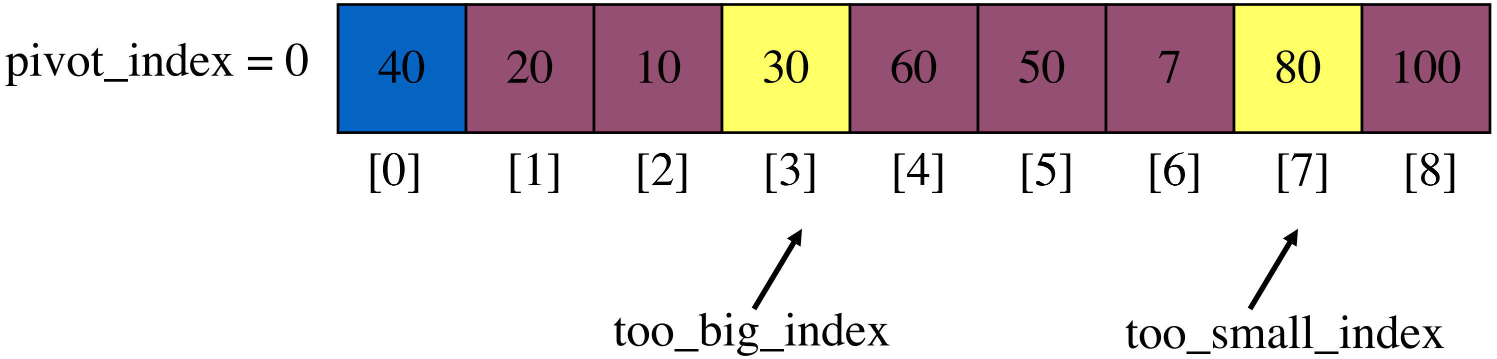


pivot_index = 0

40	20	10	30	60	50	7	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

Partition Example (Pseudocode)

```
→ 1. while data[too_big_index] <= data[pivot]
          ++too_big_index
  2. while data[too_small_index] > data[pivot]
          --too_small_index
  3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to Step 1.
```

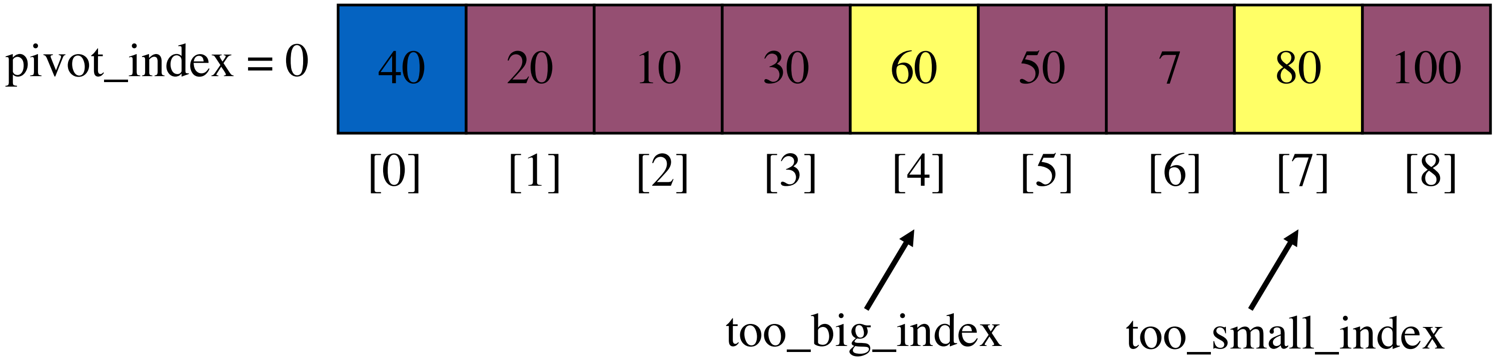


pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	60	50	7	80	100

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
          ++too_big_index
2. while data[too_small_index] > data[pivot]
          --too_small_index
3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
4. while too_small_index >= too_big_index, go to Step 1.
```

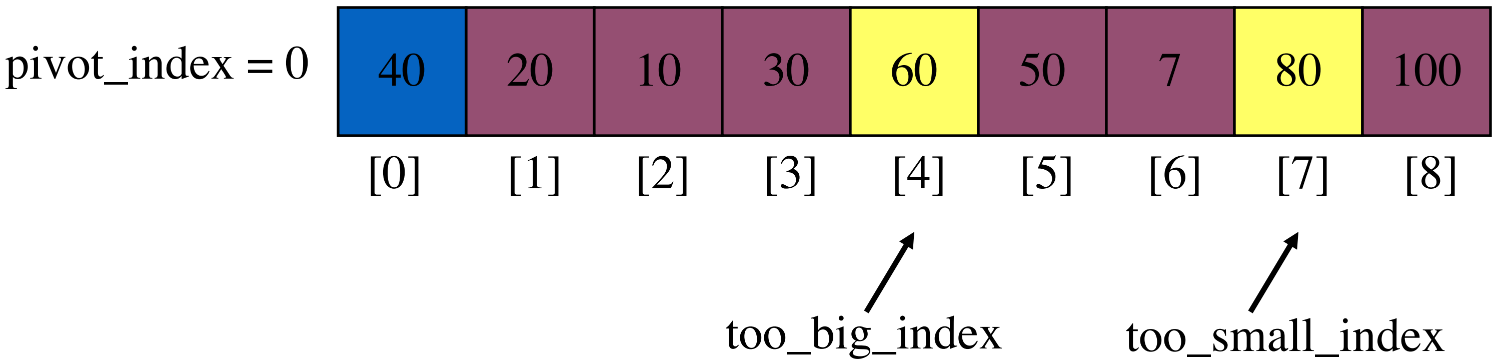


pivot_index = 0

40	20	10	30	60	50	7	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

Partition Example (Pseudocode)

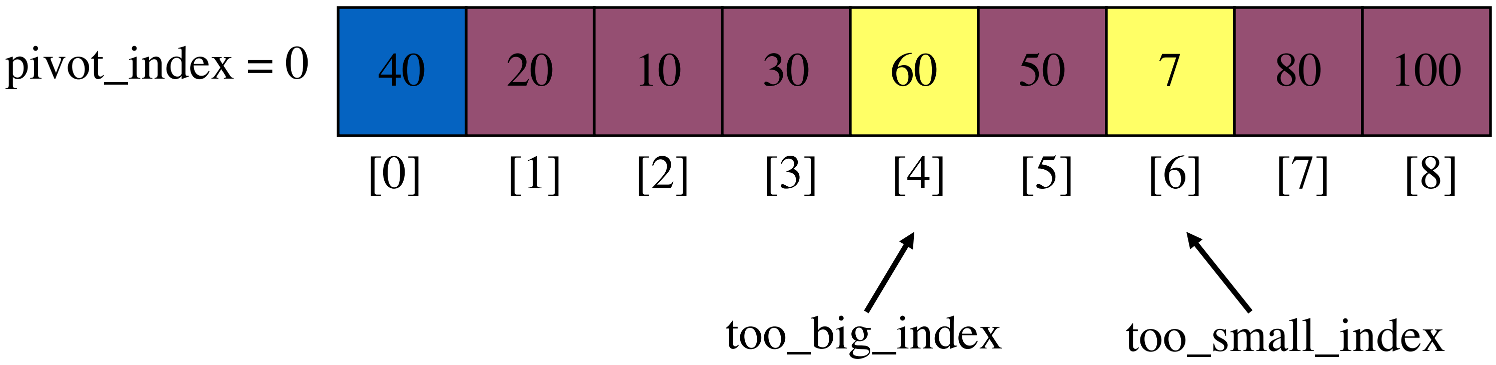
```
  1. while data[too_big_index] <= data[pivot]
         ++too_big_index
→ 2. while data[too_small_index] > data[pivot]
         --too_small_index
  3. if too_big_index < too_small_index
         swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to Step 1.
```



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	60	50	7	80	100

Partition Example

```
  1. while data[too_big_index] <= data[pivot]
          ++too_big_index
→ 2. while data[too_small_index] > data[pivot]
          --too_small_index
  3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to Step 1.
```

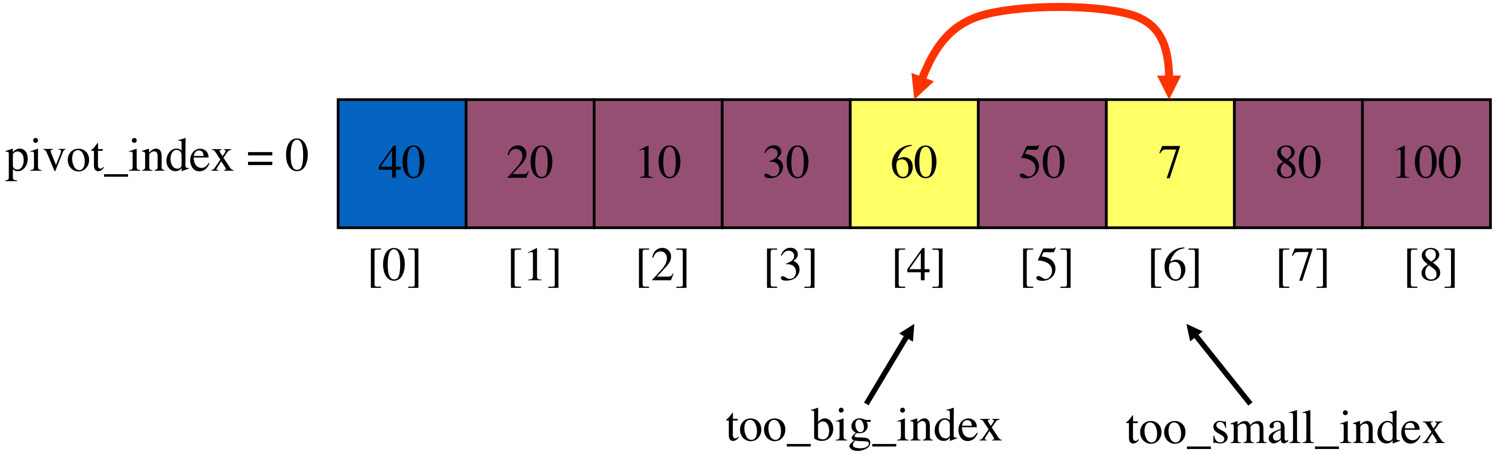


pivot_index = 0

40	20	10	30	60	50	7	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
2. while data[too_small_index] > data[pivot]
        --too_small_index
3. if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
4. while too_small_index >= too_big_index, go to Step 1.
```

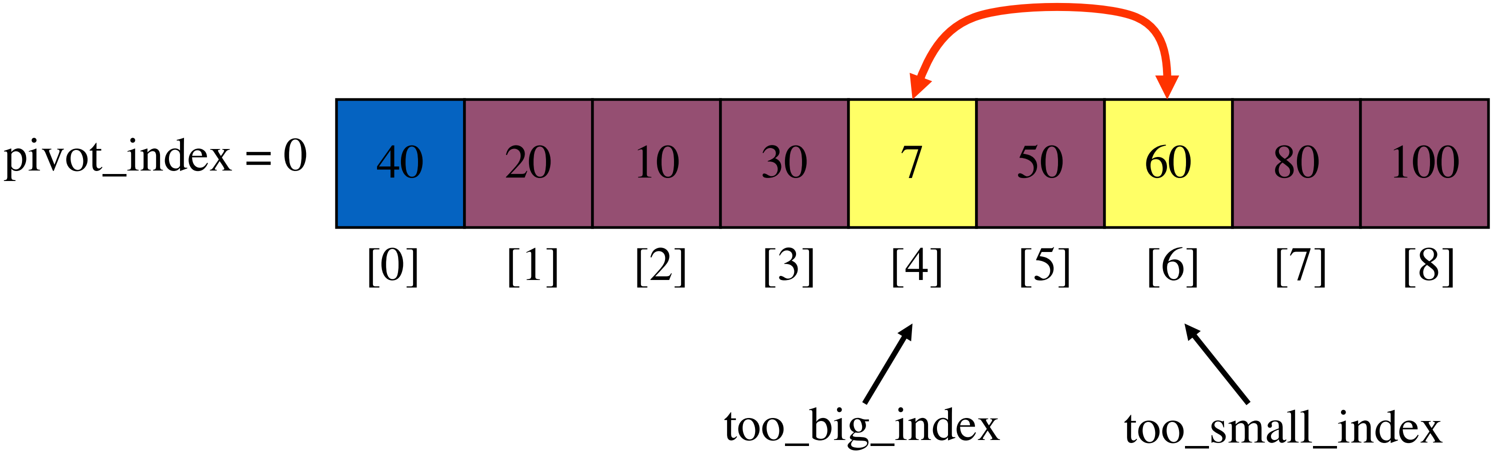


pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	60	50	7	80	100

Partition Example

```
   1. while data[too_big_index] <= data[pivot]
          ++too_big_index
   2. while data[too_small_index] > data[pivot]
          --too_small_index
→  3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
   4. while too_small_index >= too_big_index, go to Step 1.
```



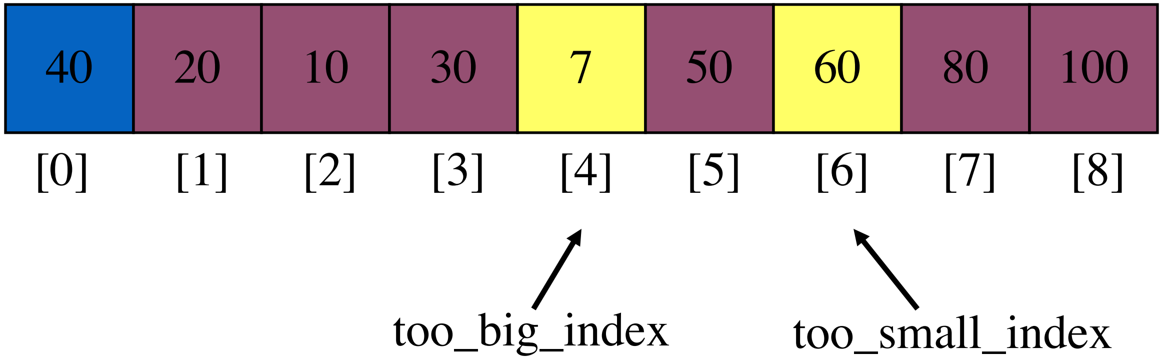
pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

Partition Example (Pseudocode)

```
   1. while data[too_big_index] <= data[pivot]
          ++too_big_index
   2. while data[too_small_index] > data[pivot]
          --too_small_index
   3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
→  4. while too_small_index >= too_big_index, go to Step 1.
```

pivot_index = 0



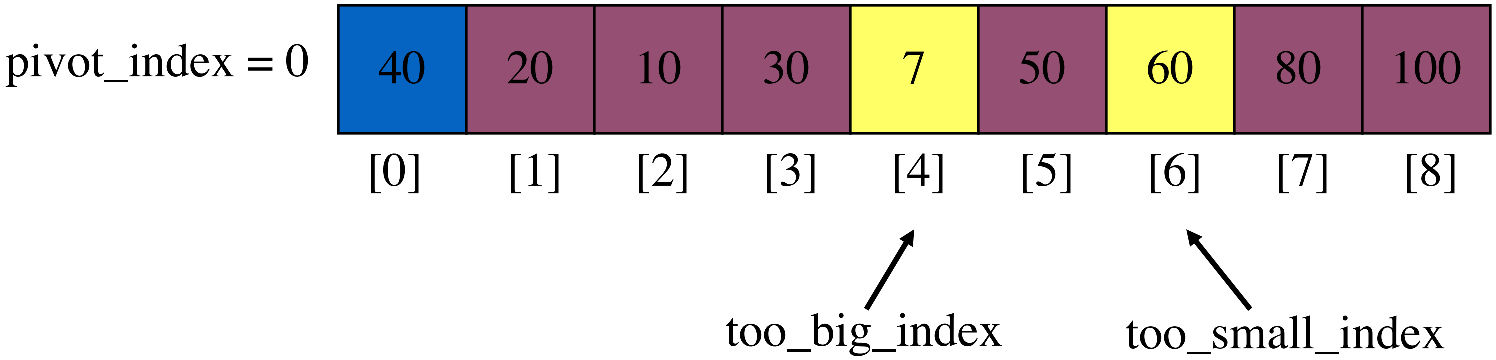
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
          ++too_big_index
2. while data[too_small_index] > data[pivot]
          --too_small_index
3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
4. while too_small_index >= too_big_index, go to Step 1.
```



pivot_index = 0

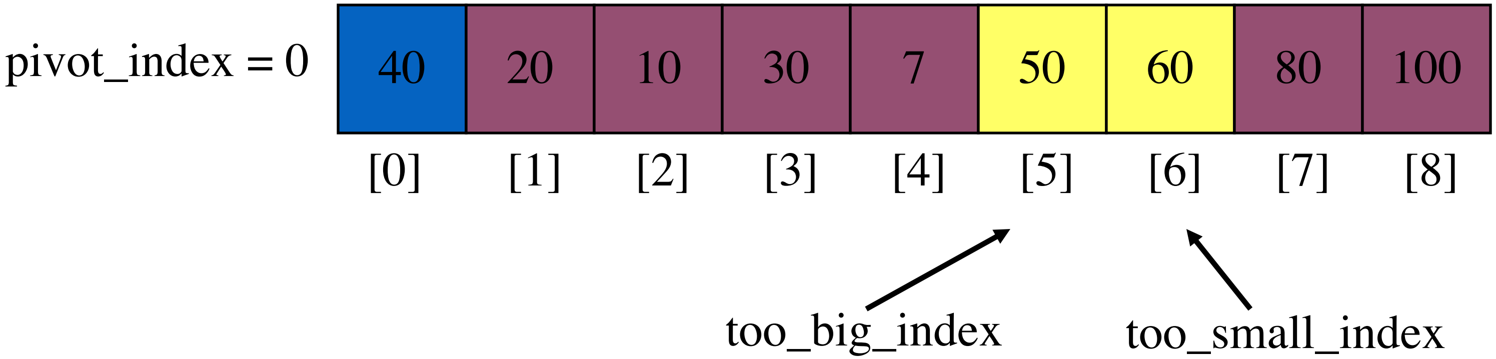
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1.  while data[too_big_index] <= data[pivot]
        ++too_big_index
2.  while data[too_small_index] > data[pivot]
        --too_small_index
3.  if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
4.  while too_small_index >= too_big_index, go to Step 1.
```



pivot_index = 0

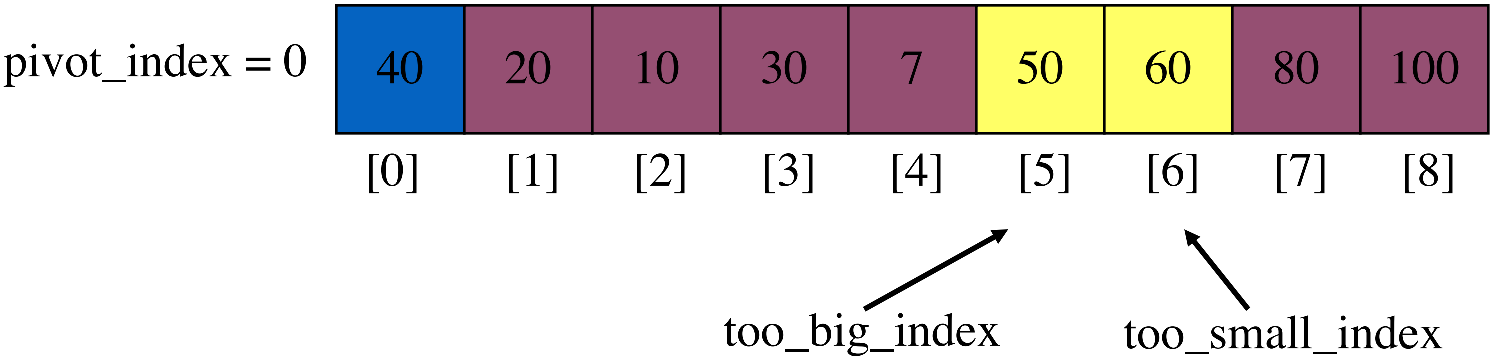
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

too_big_index

too_small_index

Partition Example (Pseudocode)

```
1.  while data[too_big_index] <= data[pivot]
            ++too_big_index
2.  while data[too_small_index] > data[pivot]
            --too_small_index
3.  if too_big_index < too_small_index
            swap data[too_big_index] and data[too_small_index]
4.  while too_small_index >= too_big_index, go to Step 1.
```



pivot_index = 0

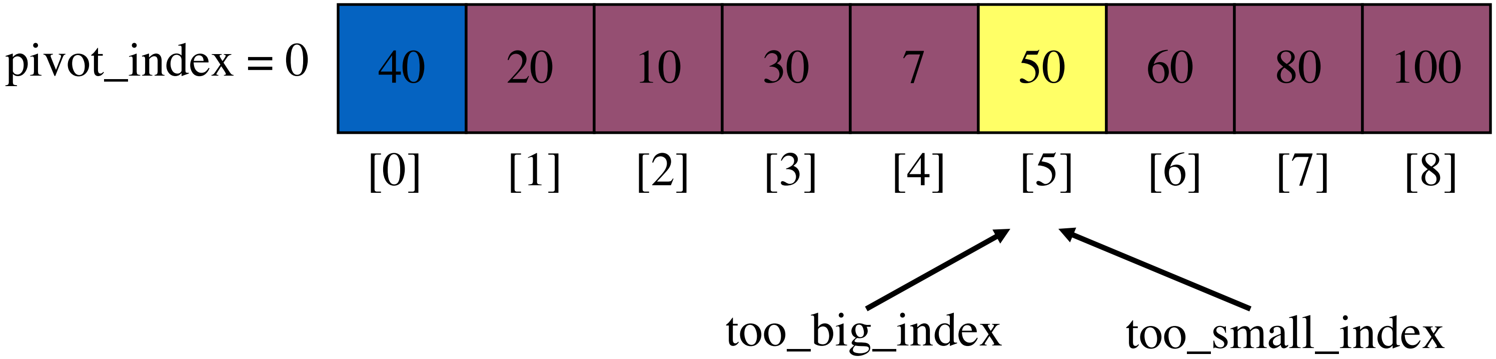
40	20	10	30	7	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Example (Pseudocode)

```
   1. while data[too_big_index] <= data[pivot]
          ++too_big_index
→  2. while data[too_small_index] > data[pivot]
          --too_small_index
   3. if too_big_index < too_small_index
          swap data[too_big_index] and data[too_small_index]
   4. while too_small_index >= too_big_index, go to Step 1.
```



pivot_index = 0

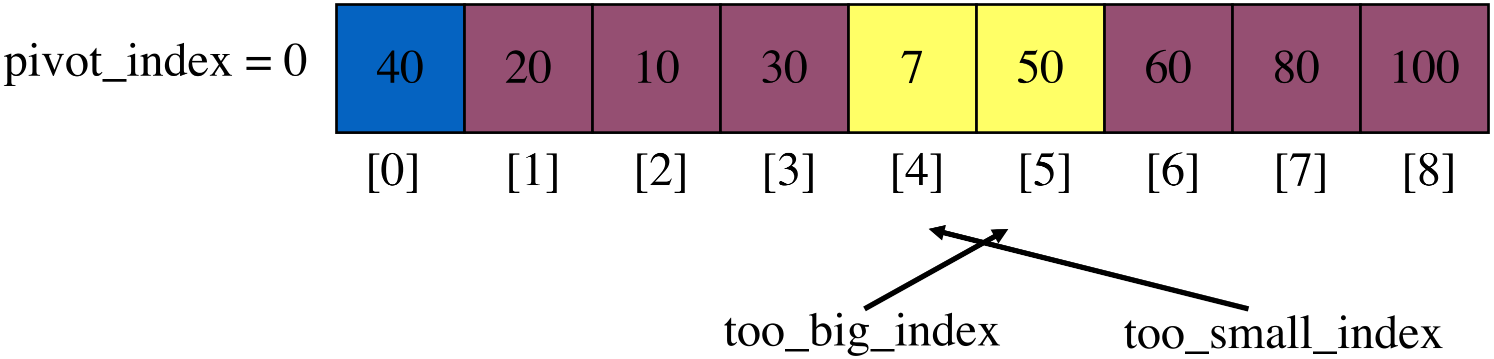
40	20	10	30	7	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Example

```
  1. while data[too_big_index] <= data[pivot]
         ++too_big_index
→ 2. while data[too_small_index] > data[pivot]
         --too_small_index
  3. if too_big_index < too_small_index
         swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to Step 1.
```

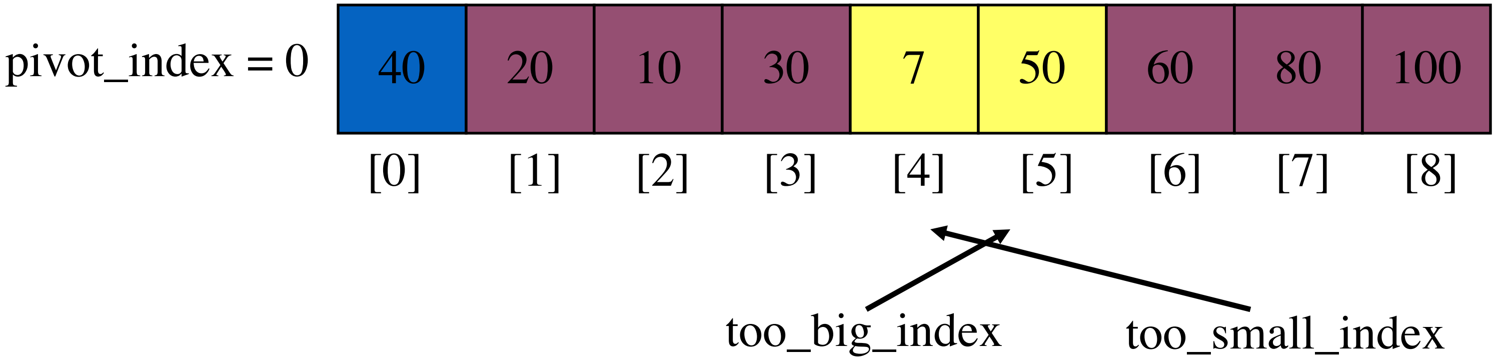


pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

Partition Example (Pseudocode)

```
  1. while data[too_big_index] <= data[pivot]
         ++too_big_index
  2. while data[too_small_index] > data[pivot]
         --too_small_index
→ 3. if too_big_index < too_small_index
         swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to Step 1.
```

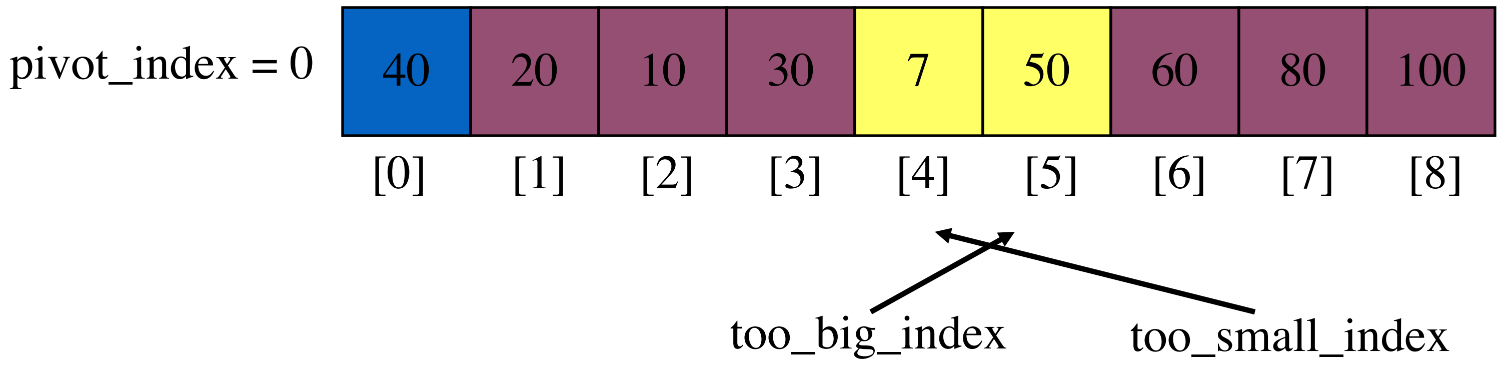


pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

Partition Example (Pseudocode)

```
  1. while data[too_big_index] <= data[pivot]
         ++too_big_index
  2. while data[too_small_index] > data[pivot]
         --too_small_index
  3. if too_big_index < too_small_index
         swap data[too_big_index] and data[too_small_index]
→ 4. while too_small_index >= too_big_index, go to Step 1.
```



pivot_index = 0

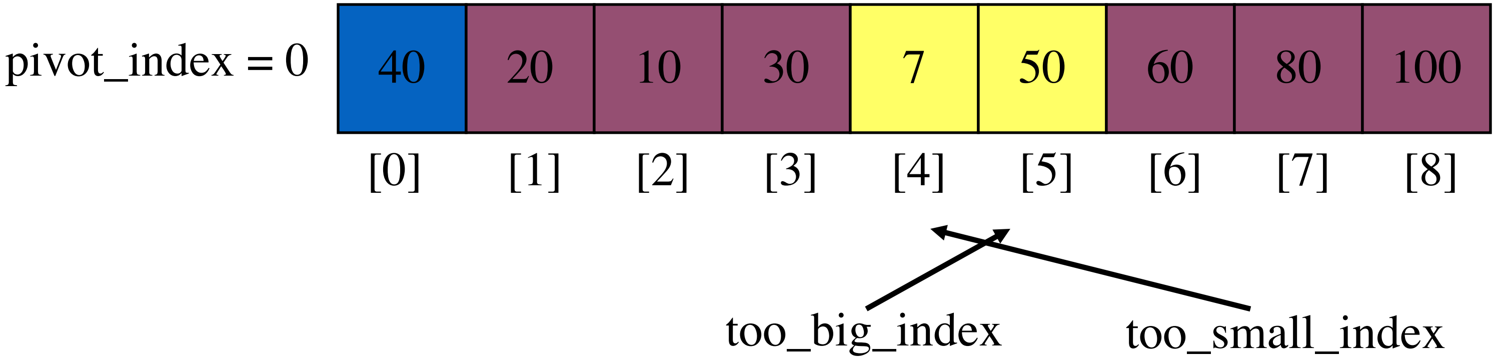
40	20	10	30	7	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Example (Pseudocode)

```
  1. while data[too_big_index] <= data[pivot]
         ++too_big_index
  2. while data[too_small_index] > data[pivot]
         --too_small_index
  3. if too_big_index < too_small_index
         swap data[too_big_index] and data[too_small_index]
  4. while too_small_index >= too_big_index, go to 1.
→ 5. swap data[too_small_index] and data[pivot]
```



pivot_index = 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
40	20	10	30	7	50	60	80	100

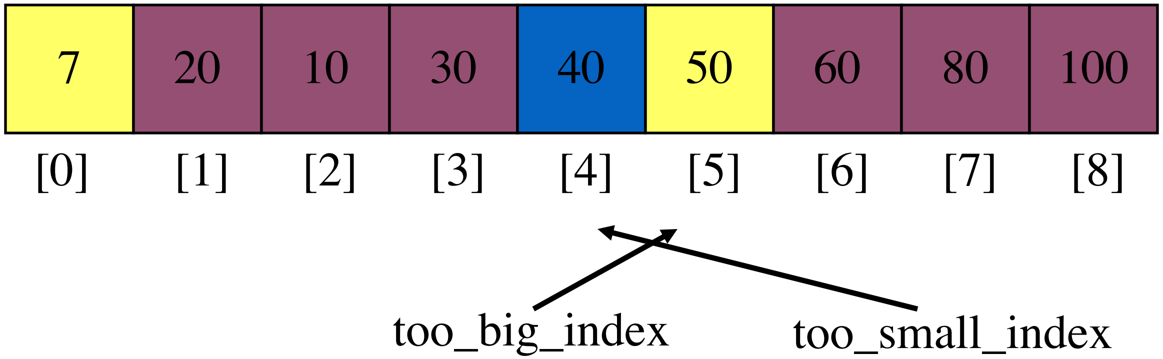
too_big_index

too_small_index

Partition Example (Pseudocode)

```
1. while data[too_big_index] <= data[pivot]
        ++too_big_index
2. while data[too_small_index] > data[pivot]
        --too_small_index
3. if too_big_index < too_small_index
        swap data[too_big_index] and data[too_small_index]
4. while too_small_index >= too_big_index, go to 1.
5. swap data[too_small_index] and data[pivot]
```

pivot_index = 4

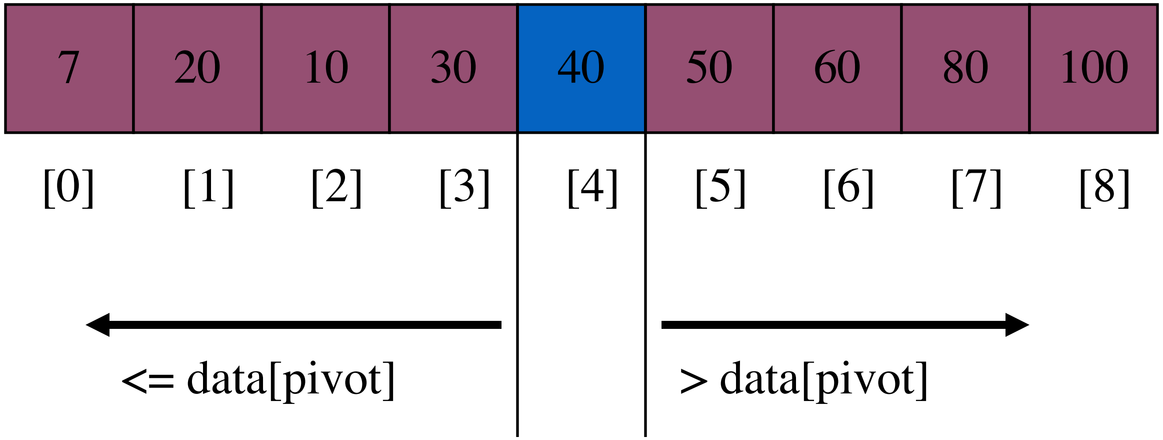


7	20	10	30	40	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

Partition Result



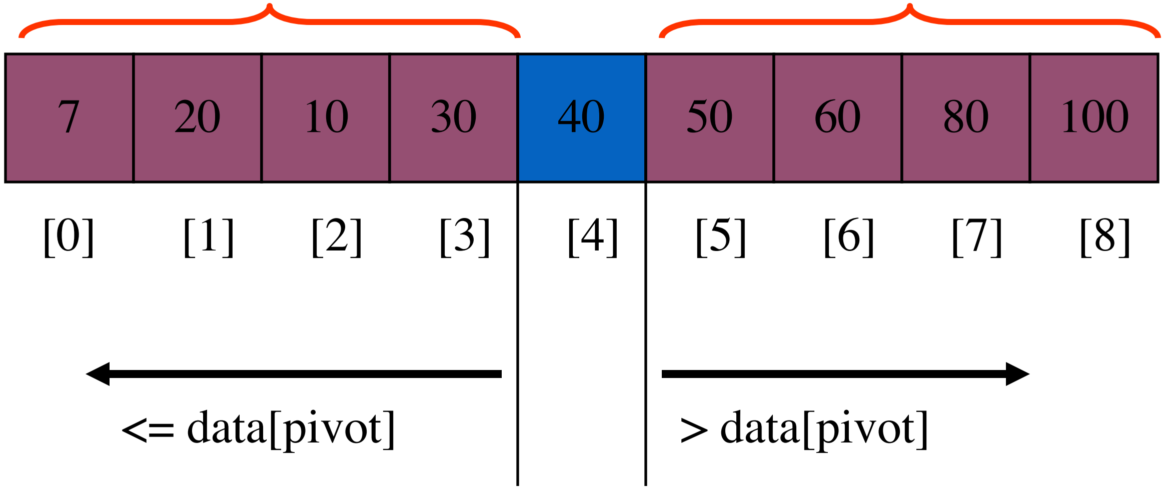
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
7	20	10	30	40	50	60	80	100

<= data[pivot]

> data[pivot]

All this is done **in-place**, and does not require extra memory

Recursion: Quicksort Sub-arrays

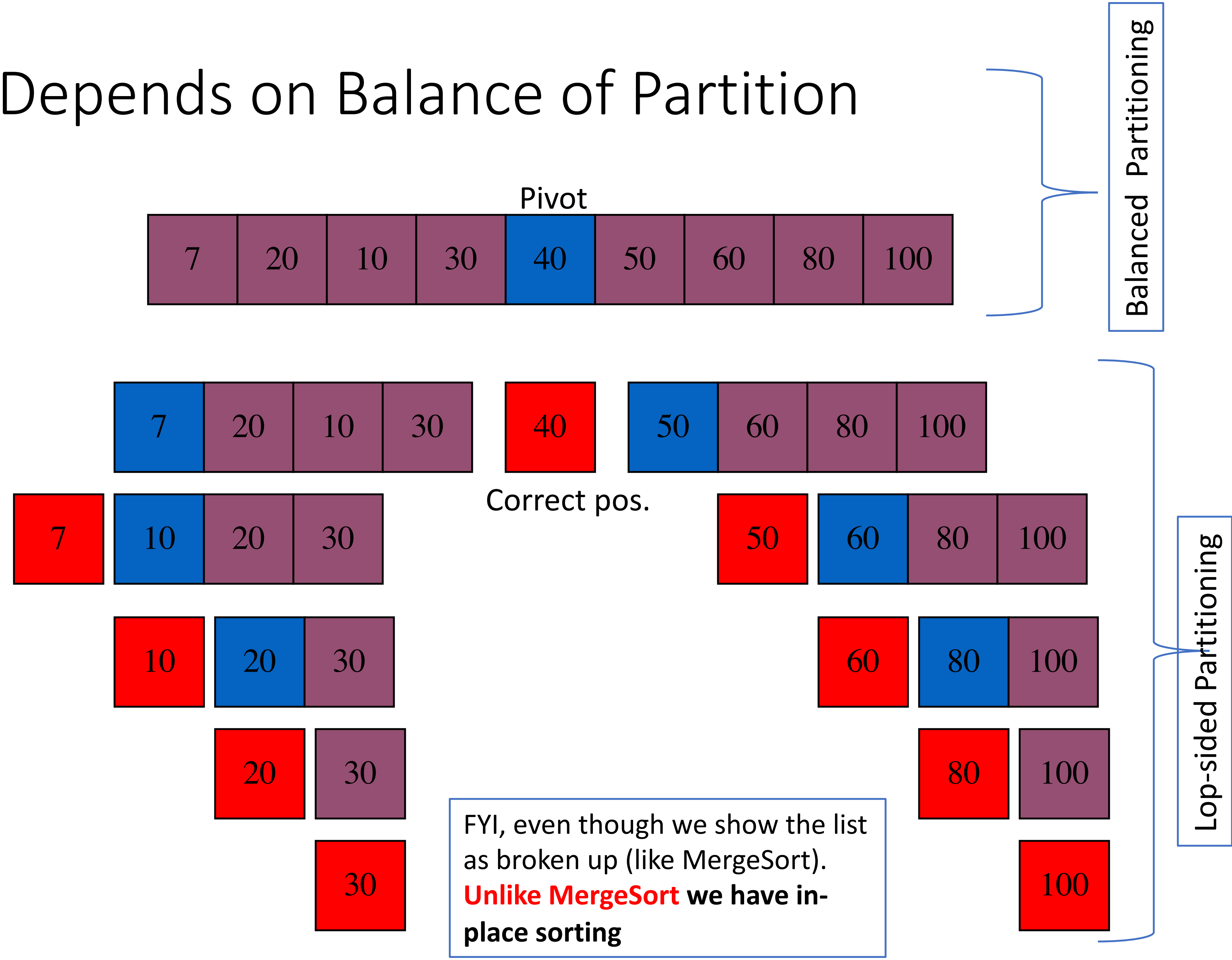


Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- What is **best case running time?**
  - Recursion:
    - Partition splits array in two sub-arrays of size n/2
    - Quicksort each sub-array
- Depth of recursion tree?
  - **O($\log_2 n$)**
- Number of accesses in partition?
  - **O(n)**
- Best case running time: **O($n \log_2 n$)**

Worst case running time?

Depends on Balance of Partition



FYI, even though we show the list as broken up (like MergeSort). **Unlike MergeSort we have in-place sorting**

That's all Folks!
Any Question ?