BCSE202L Data structures and Algorithms

Quicksort

Quicksort

- A

7	6	10	5	9	2	1	15	7
0	1	2	3	4	5	6	7	8

```
1   def partition(a, l, h):
2       pivot = a[l]
3       i = l
4       j = h
5       while i < j:
6           while a[i] <= pivot:
7               i = i + 1
8           while a[j] > pivot:
9               j = j - 1
10          if i < j:
11              swap(a[i], a[j])
12      swap(a[l], a[j])
13      return j
```

```
1    def quicksort(a, l, h):
2        if l < h:
3            pivot_index = prtition(a, l, h)
4            quicksort(a, l, pivot_index-1)
5            quicksort(a, pivot_index, u)
```

Example

We are given array of n integers to sort:

40	20	10	80	60	50	7	30	100

Pick Pivot Element

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

Partitioning Array

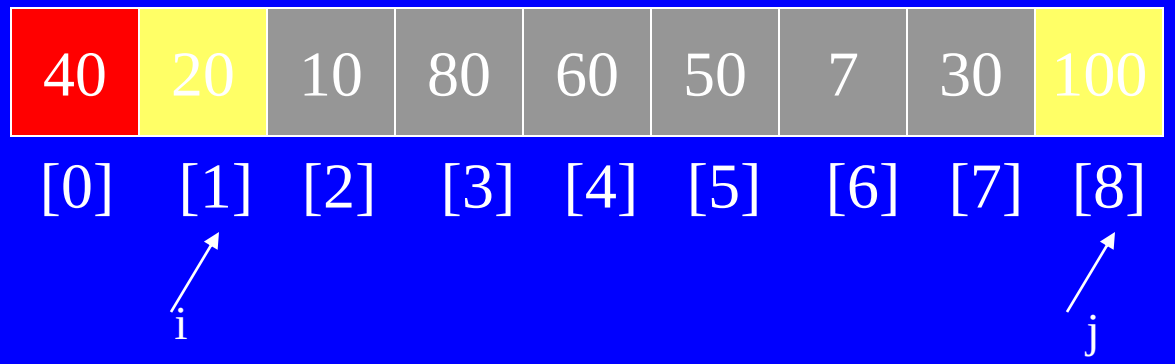
Given a pivot, partition the elements of the array such that the resulting array consists of:

1. One sub-array that contains elements >= pivot
2. Another sub-array that contains elements < pivot

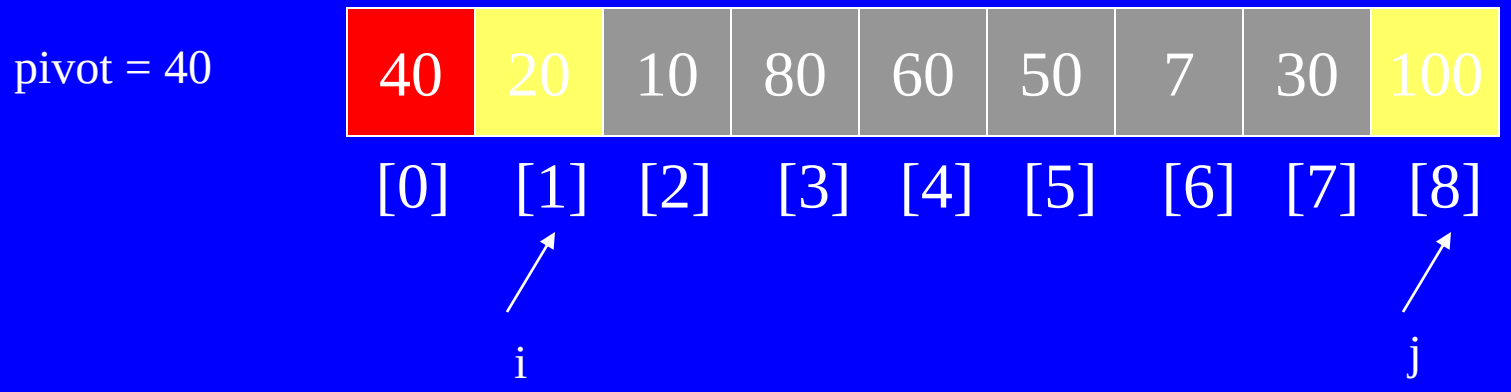
The sub-arrays are stored in the original data array.

Partitioning loops through, swapping elements below/above pivot.

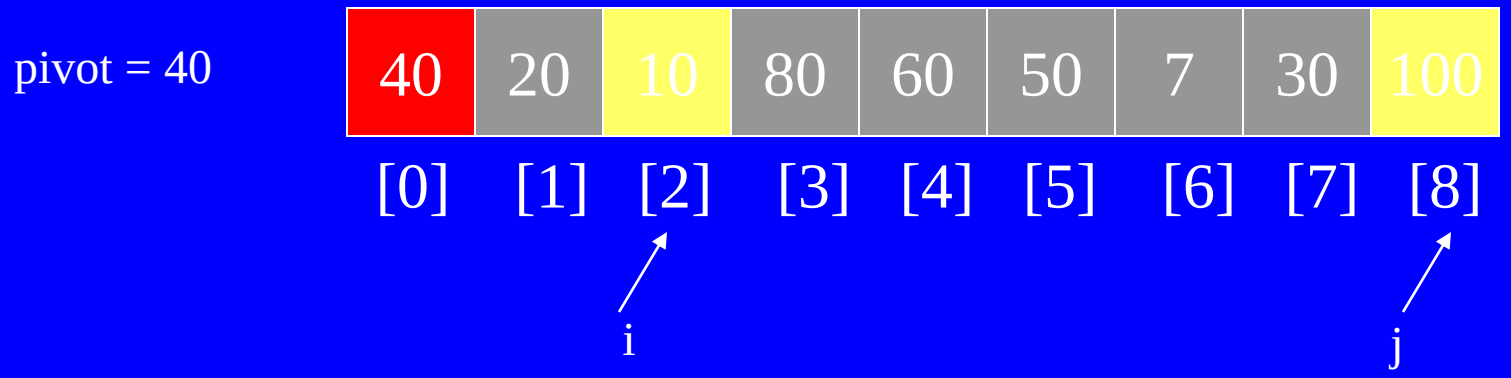
pivot = 40



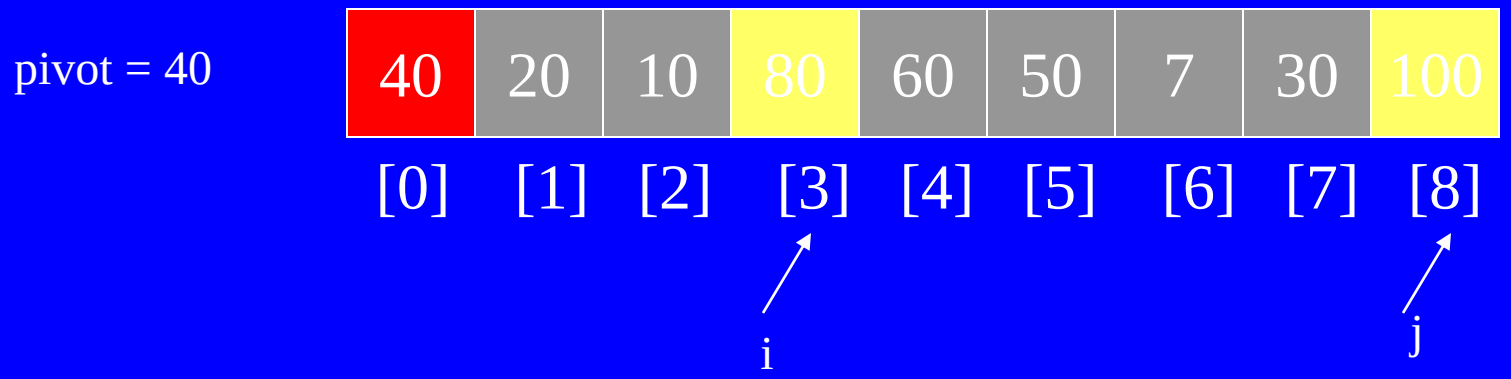
1. While A[i] <= pivot
   i++



1. While A[i] <= pivot
   i++

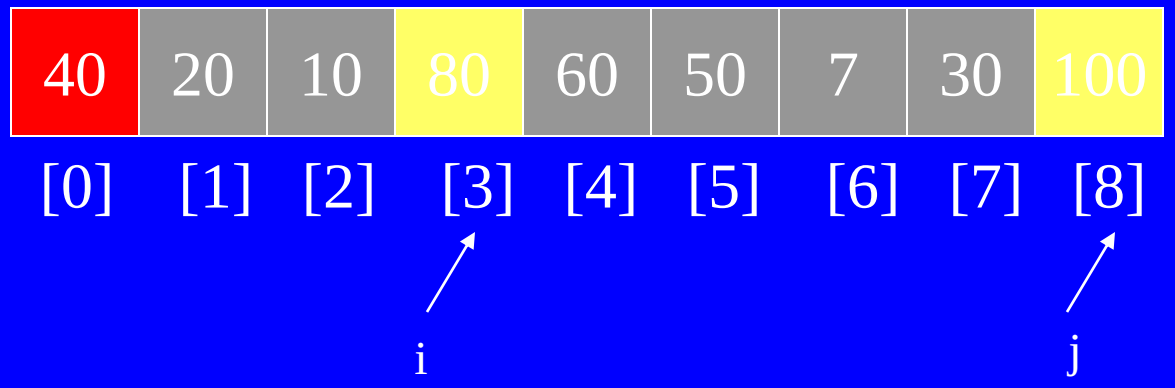


1. While A[i] <= pivot
   i++



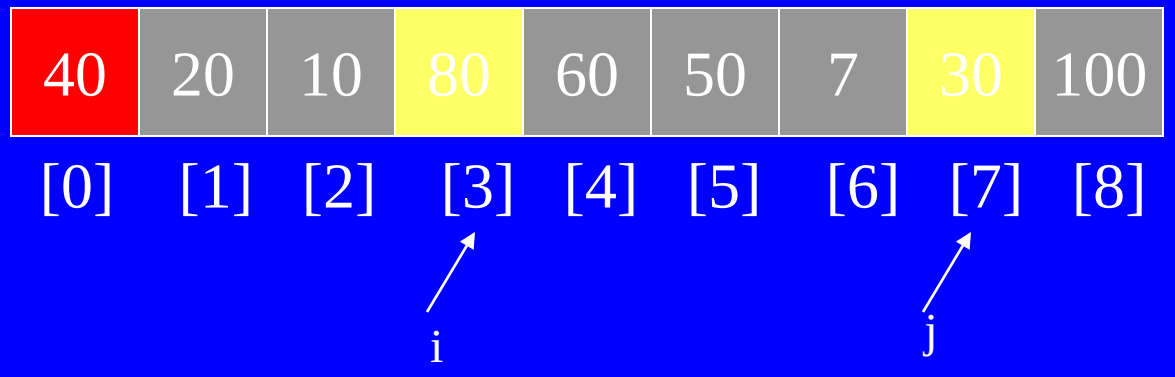
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--

pivot = 40

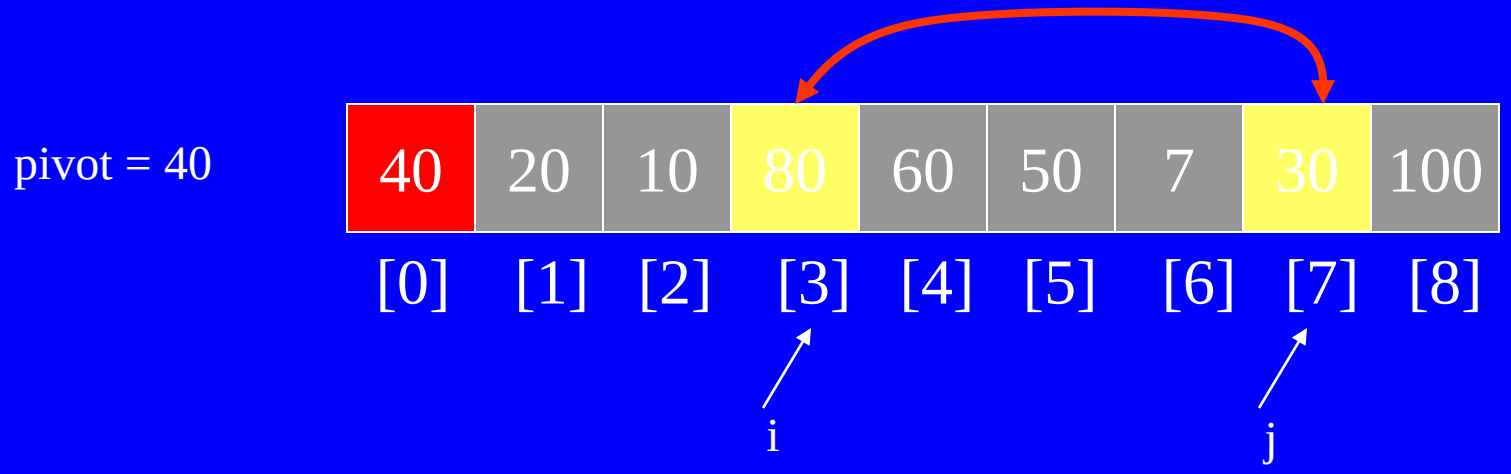


1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--

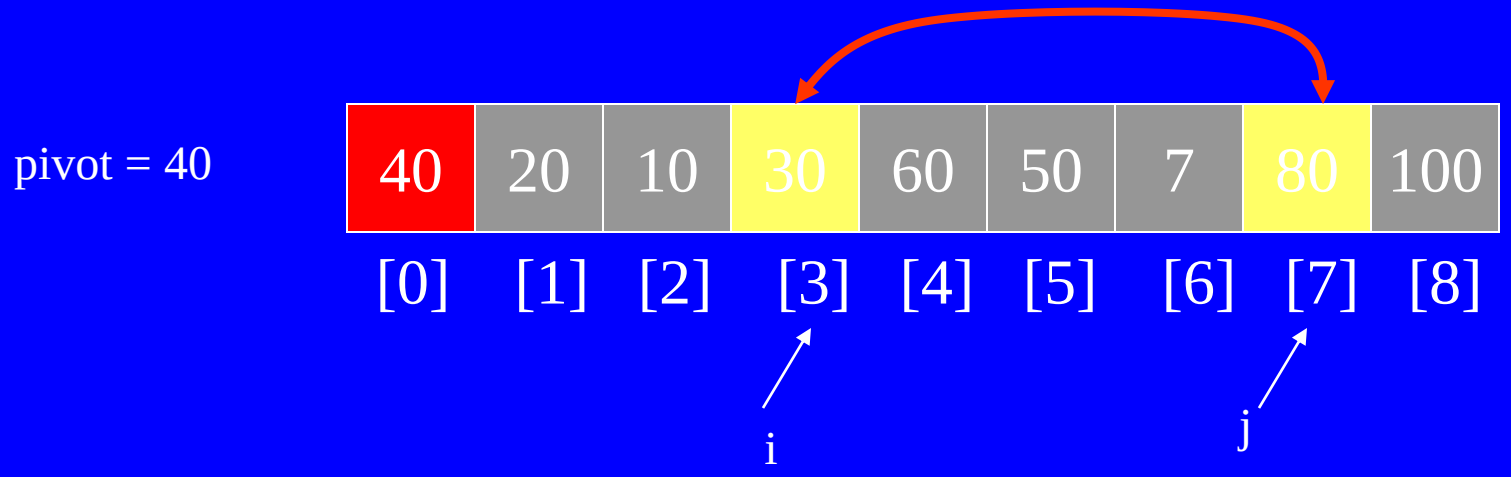
pivot = 40



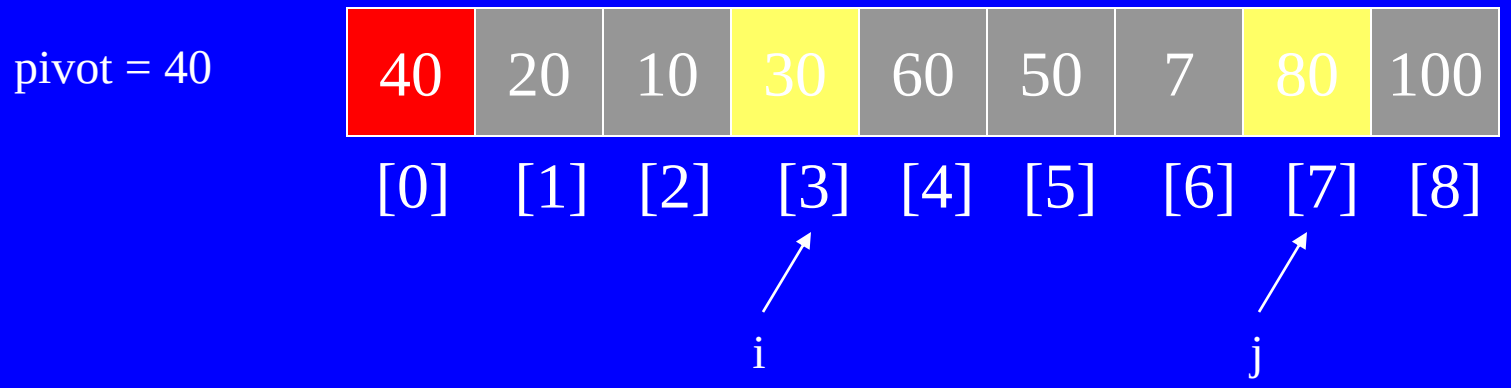
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]



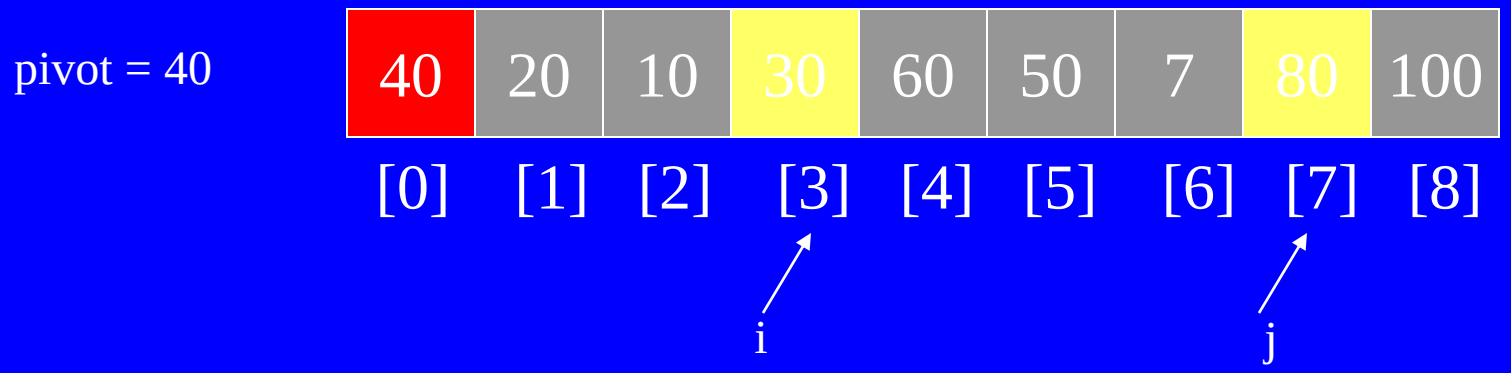
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]



1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

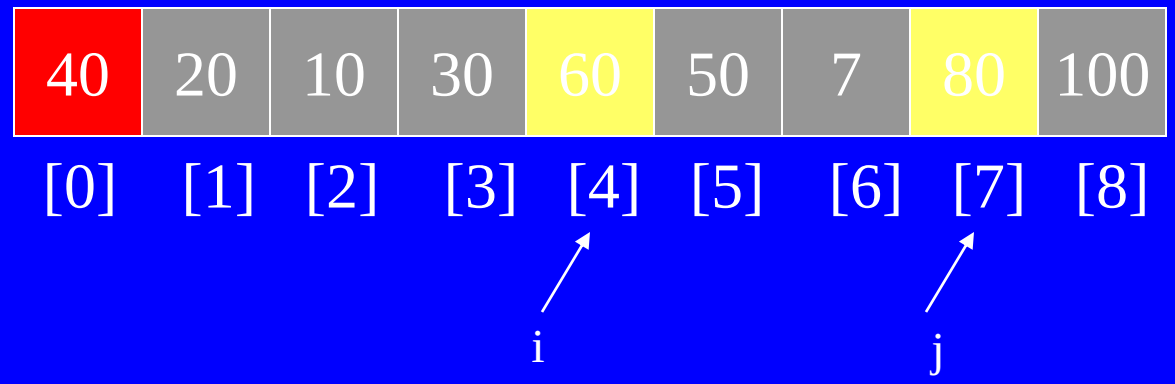


1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.



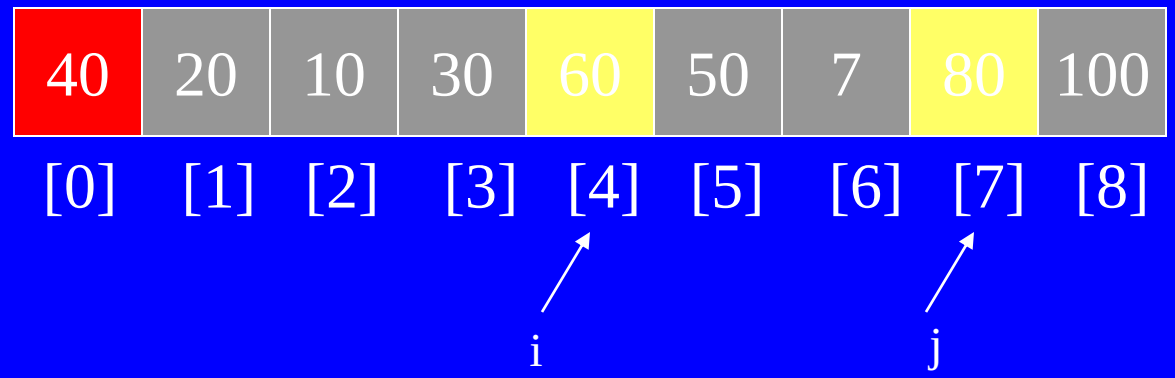
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



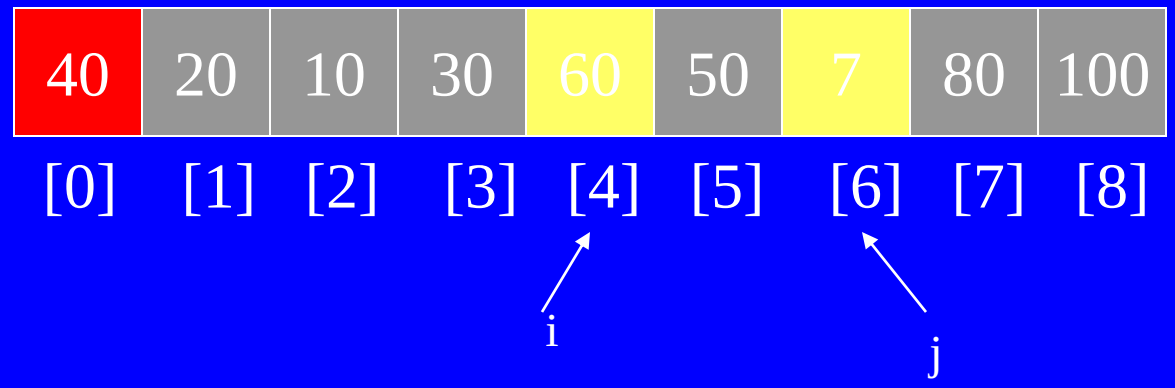
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40

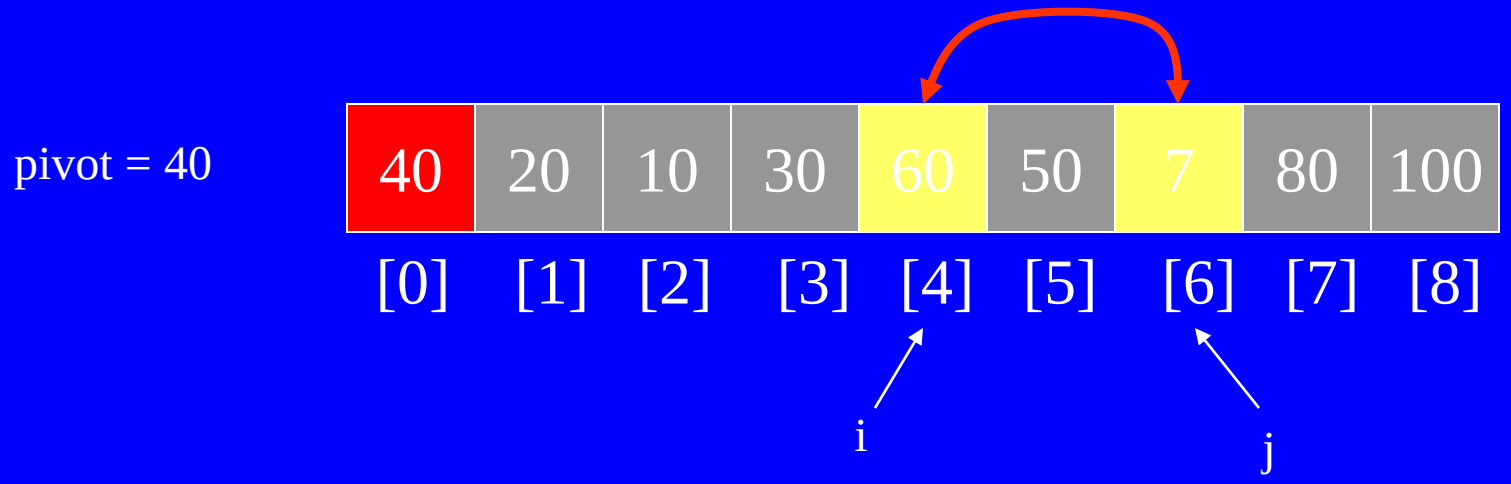


1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

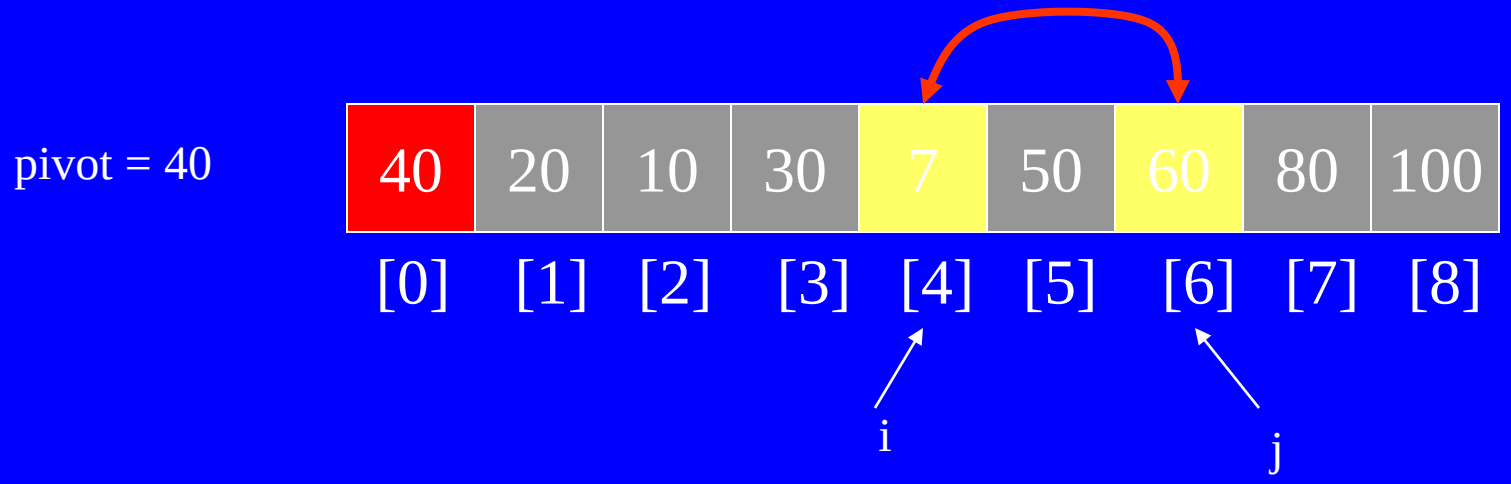
pivot = 40



1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

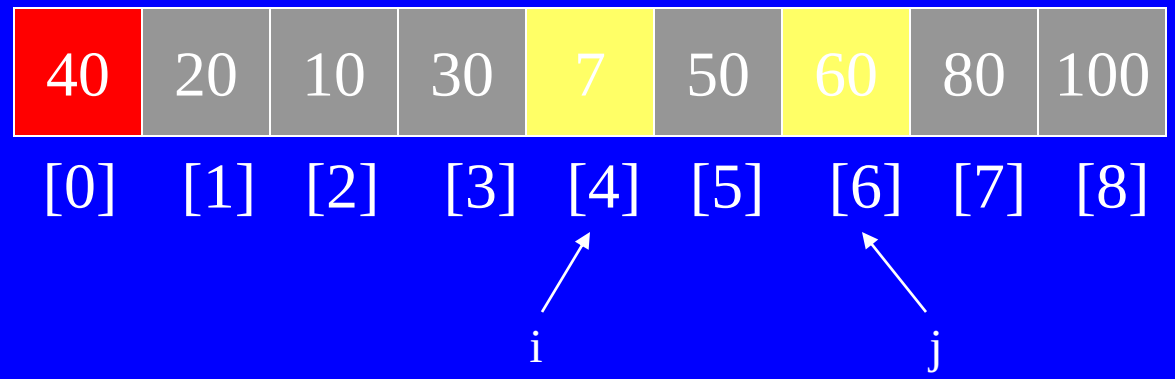


1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.



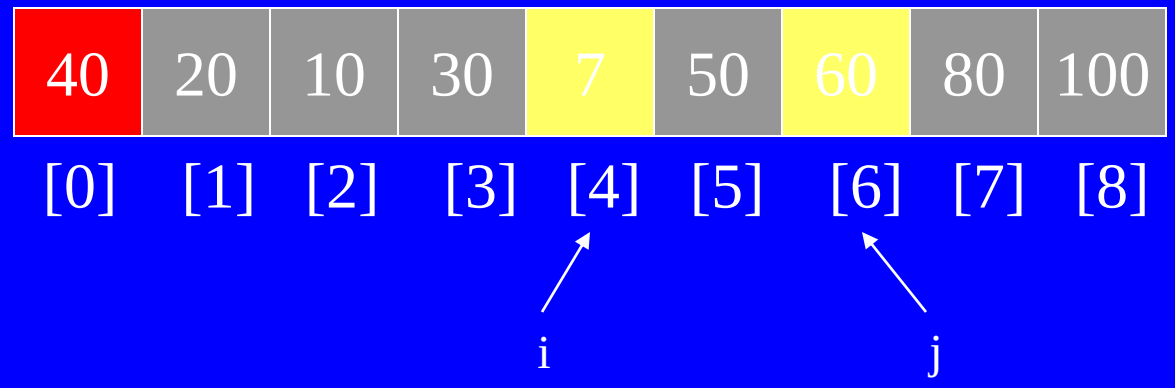
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



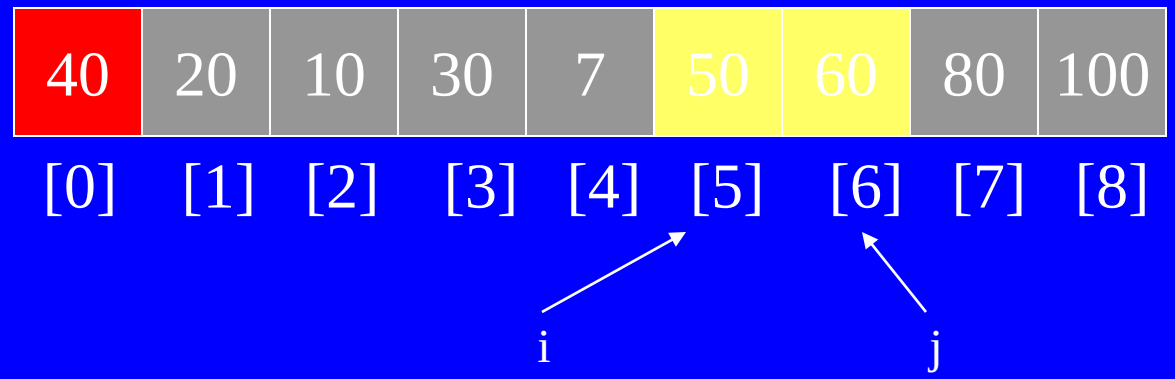
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



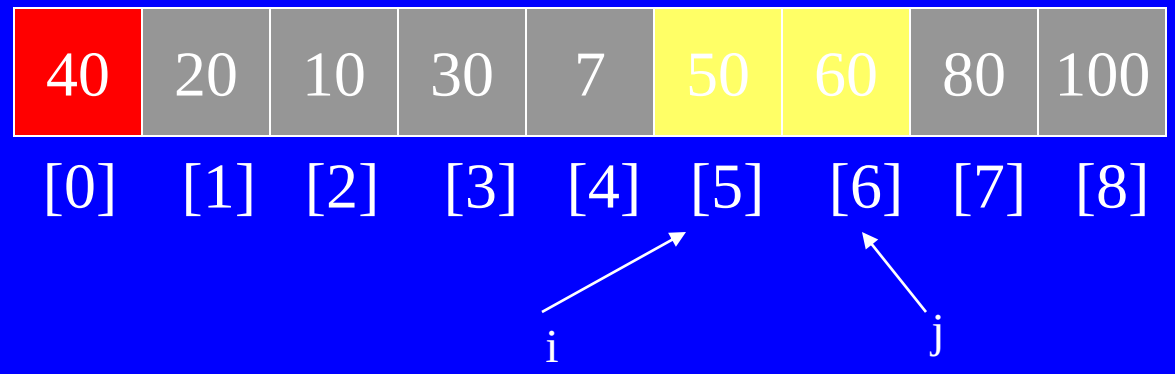
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



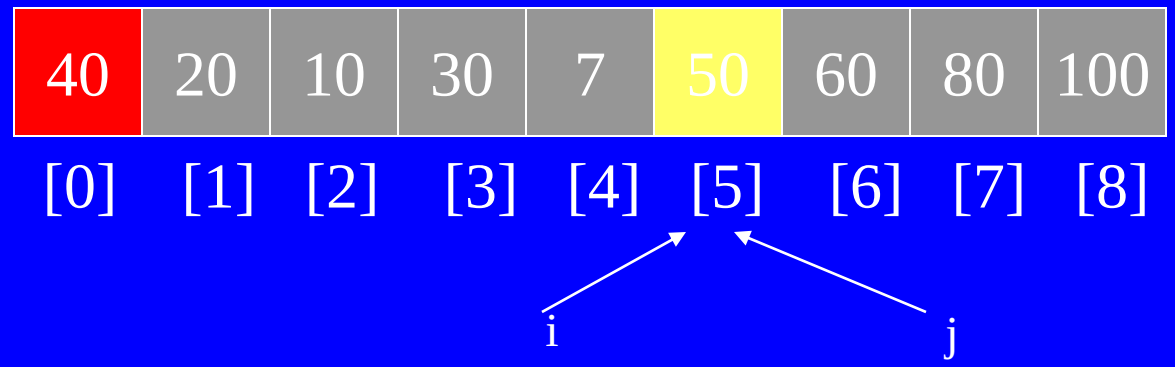
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



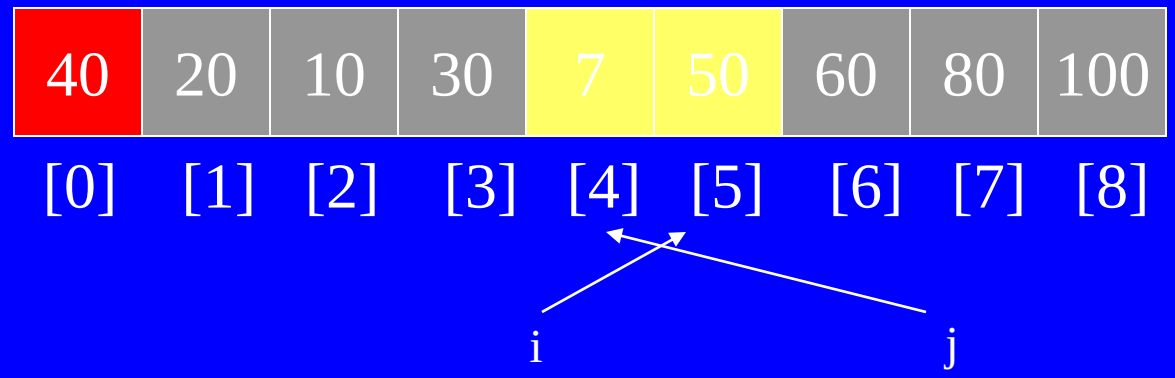
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



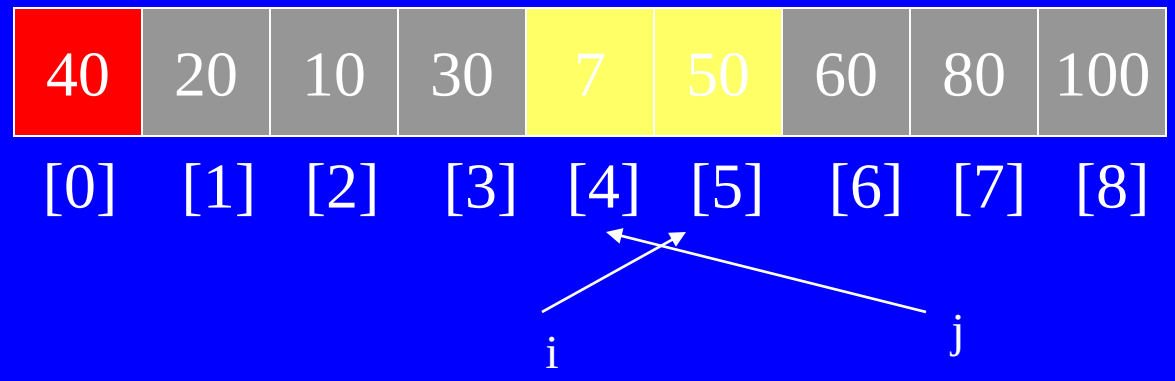
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



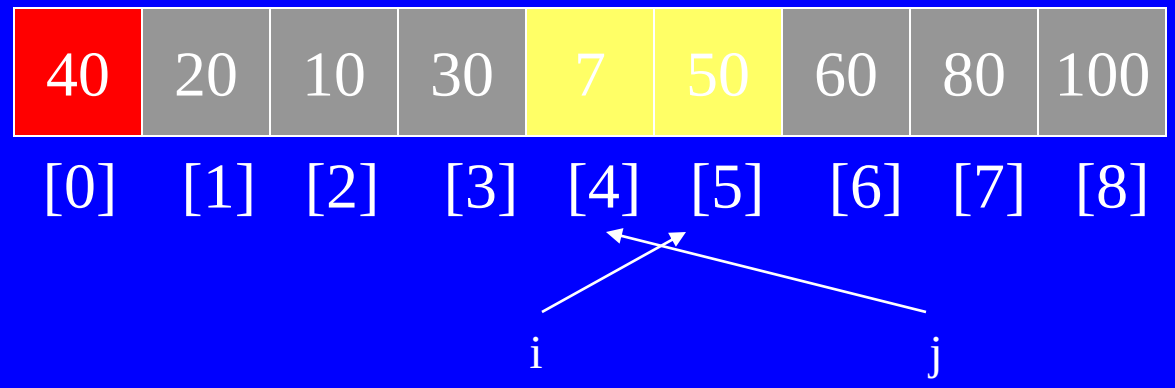
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



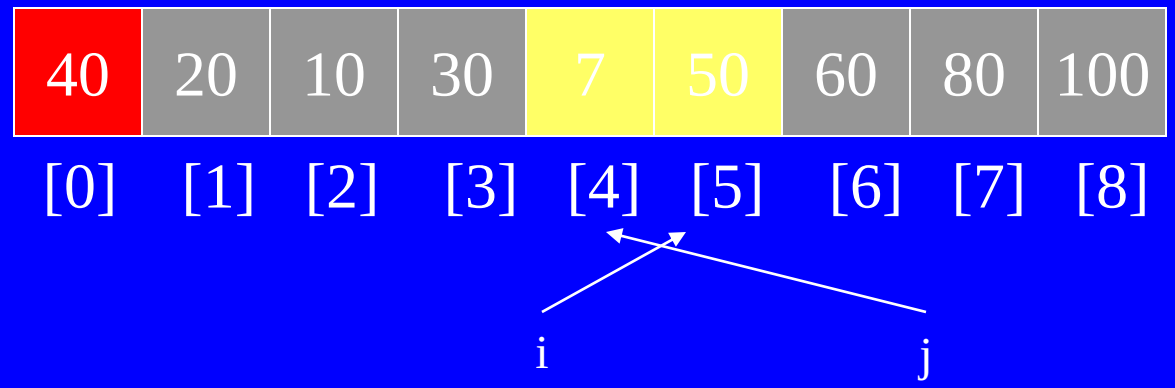
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.

pivot = 40



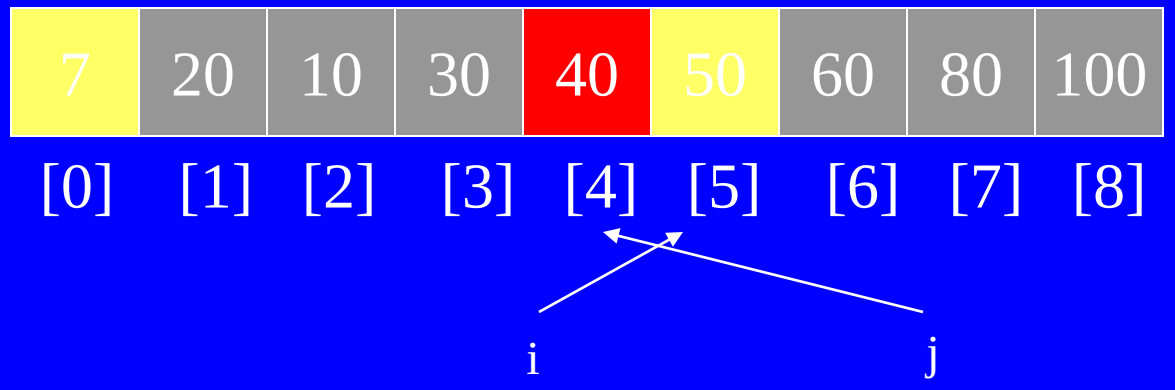
1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.
5. Swap A[j] and pivot

pivot = 40

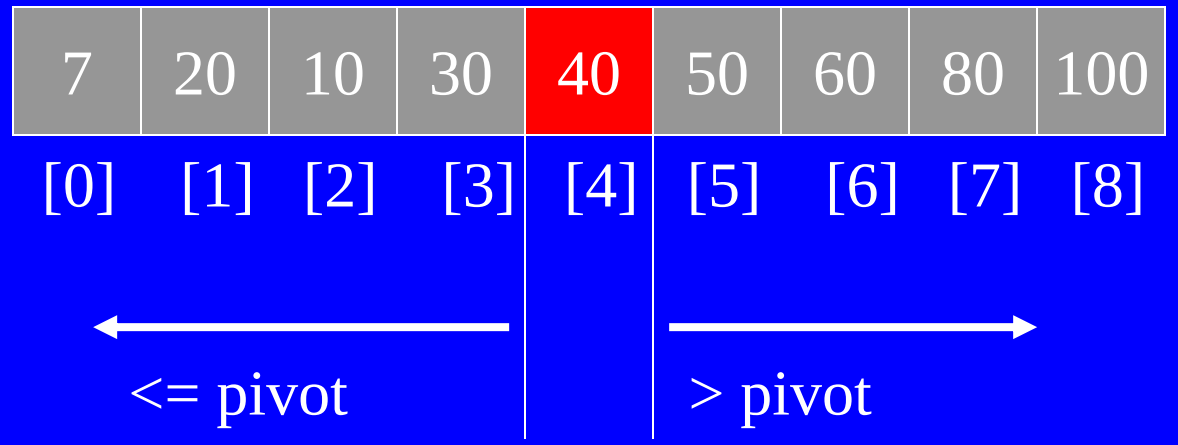


1. While A[i] <= pivot
   i++
2. While A[j] > pivot
   j--
3. If i < j
   swap A[i] and A[j]
4. While j > i, go to 1.
5. Swap A[j] and pivot

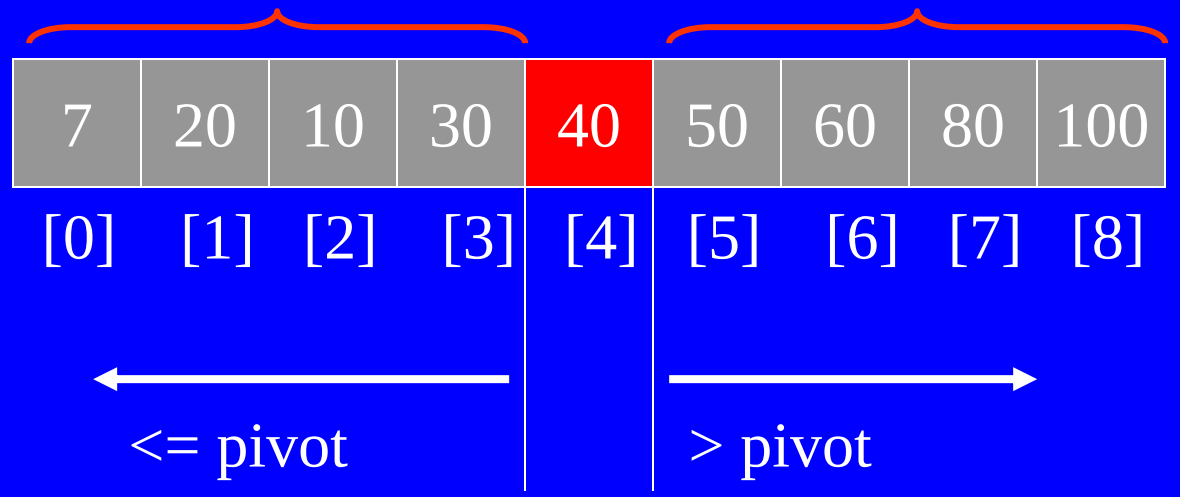
pivot = 40



Partition Result



Recursion: Quicksort Sub-arrays



Quicksort Analysis

- Worst case: pivot is either maximum or minimum
- One partition is empty, other has size n-1
- $T(n) = T(n-1) + n = O(n^2)$
- This will happen for already sorted array
- But mathematically, worst case is very rare
- So most of the time, $T(n) = 2T(n/2)+n = O(nlogn)$
- If at all happens, its because of the fixed choice of the pivot
- Randomized quicksort
- Ideal value of pivot → median, divide array into equal halves
- In practice, quicksort is very fast
- In spreadsheet/Python
- No additional list needed, in-place, unlike merge sort

Thank you!!