



Advanced Data Structures and Algorithms

Associate Professor Dr. Raed Ibraheem Hamed
University of Human Development,
College of Science and Technology
Computer Science Department

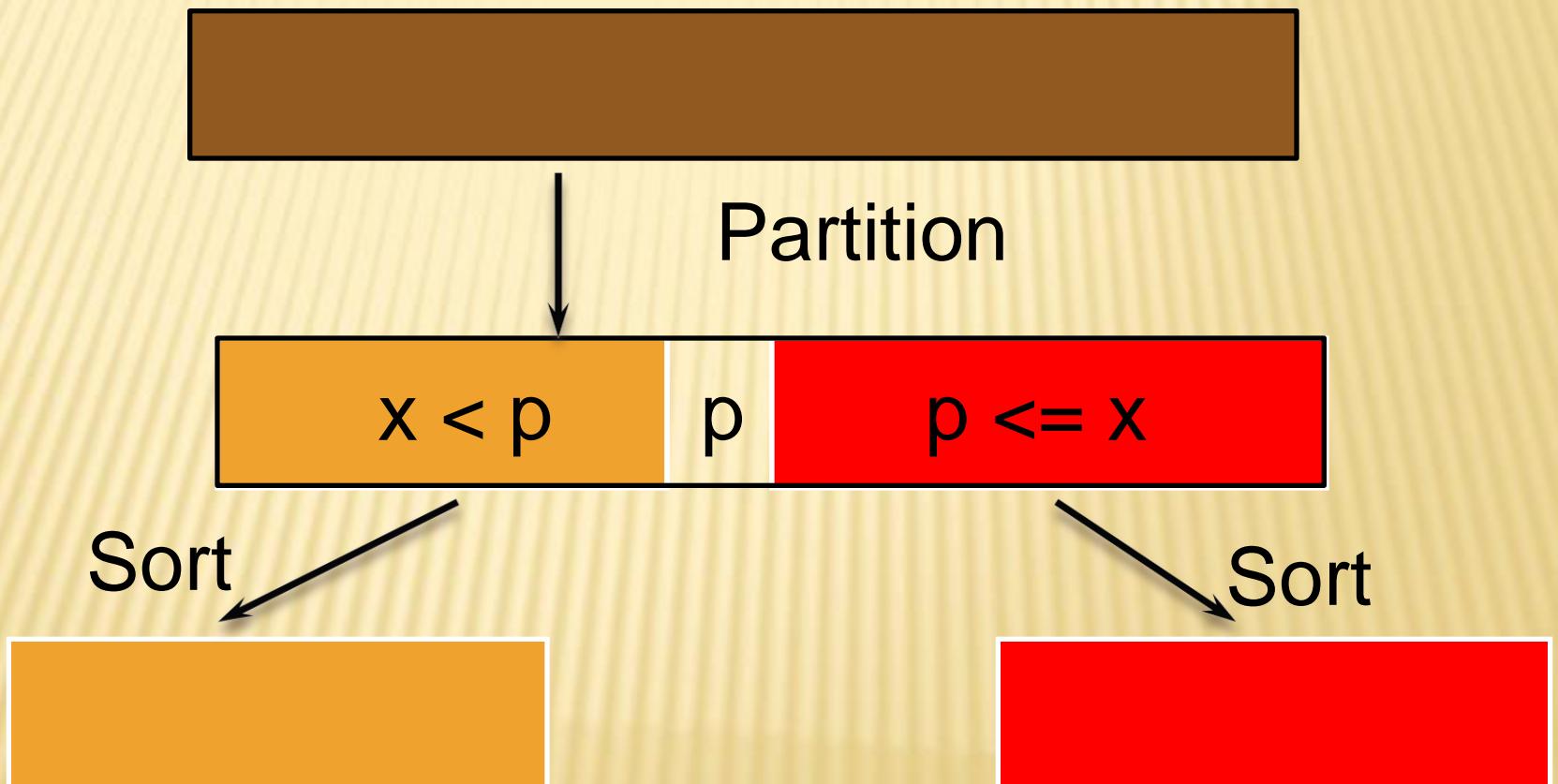
2015 – 2016

Quick-sort

Quick-sort is a randomized sorting algorithm based on the divide-and-conquer paradigm:

- + Divide: pick a random element x (called pivot) and partition S into
 - ✗ L elements less than x
 - ✗ E elements equal x
 - ✗ G elements greater than x
- + Recur: sort L and G
- + Conquer: join L , E and G

Quicksort: Partitioning an array



Quicksort: Partitioning an array

array:

7	2	1	8	6	3	5	4
---	---	---	---	---	---	---	---



- 1) Pivot [4] size: 8
 - 2) We have two counters i, j
 - 3) Set $i = \text{position } (p)-1$
 - 4) Set j goes from p to r (run on the first element)
 - 5) Choose the last element $a(r)$ as pivot
-

Compare $a(j)$ and $a(r)$



$$a(j) = j + 1$$

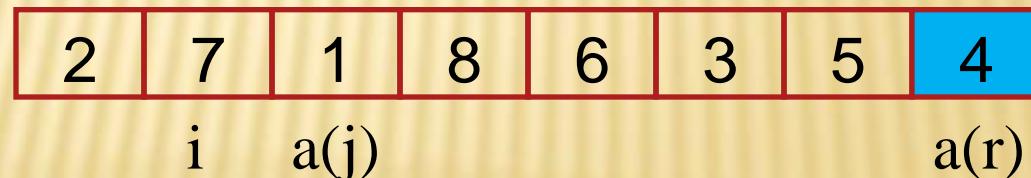
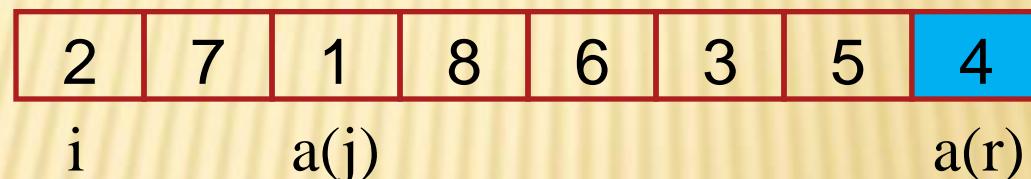
Quicksort: Partitioning an array

Compare $a(j)$ and $a(r)$



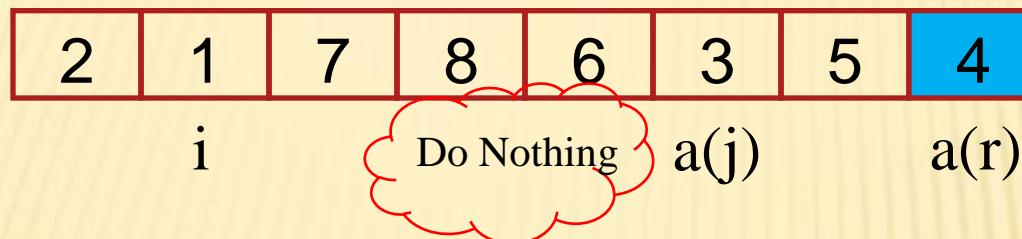
If $a(j) < a(r)$ then

$i = i + 1$, and exchange $a(i)$ and $a(j)$



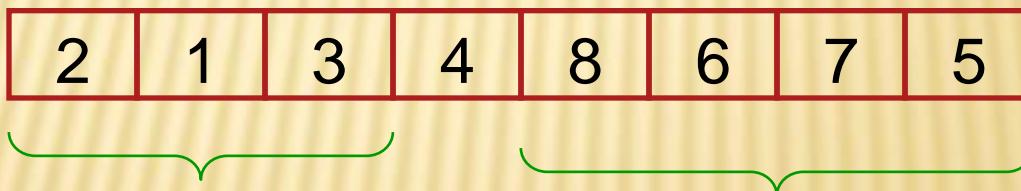
$i = i + 1$, and exchange $a(i)$ and $a(j)$

Quicksort: Partitioning an array



When j reach the end **Pivot should be at $i + 1$**

The Result



Recursively

Recursively

Example:

Quick Sort

“pivot
element”

array:

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

Example:

Partition

mid: 4



array:

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

index: 0



array:

15	89	35	14	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----

k=1



Example: Partition

index : 0



array :

15	89	35	14	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k : 1

Example: Partition

index: 0



array:

15	89	35	14	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k: 2

Example:

Partition

index : 0



array :

15	89	35	14	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----

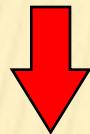


k : 3

Example:

Partition

index : 1



array :

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----

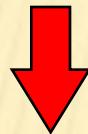
k : 3



Example:

Partition

index : 1



array :

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----

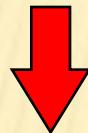


k : 4

Example:

Partition

index : 1



array :

15	14	35	89	24	5	37	13	20	7	70
----	----	----	----	----	---	----	----	----	---	----



k : 5

Example:

Partition

index: 2



array:



k: 5



Example:

Partition

index: 2



array:



k: 6

Example:

Partition

index: 2



array:



k: 7 etc...

Example:

Partition

index : 4



array :

15	14	5	13	7	35	37	89	20	24	70
----	----	---	----	---	----	----	----	----	----	----



k : 11

Example:

Partition

index : 4

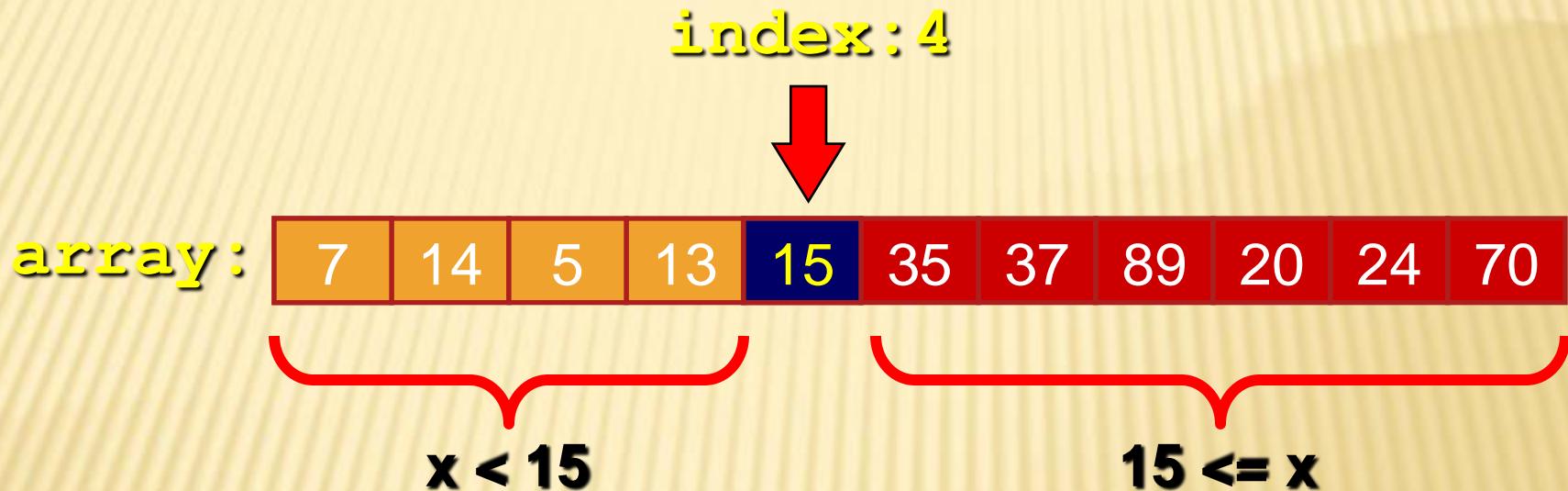


array :

15	14	5	13	7	35	37	89	20	24	70
----	----	---	----	---	----	----	----	----	----	----

Example:

Partition



Example:

pivot now in
correct position

array:

7	14	5	13	15	35	37	89	20	24	70
---	----	---	----	----	----	----	----	----	----	----



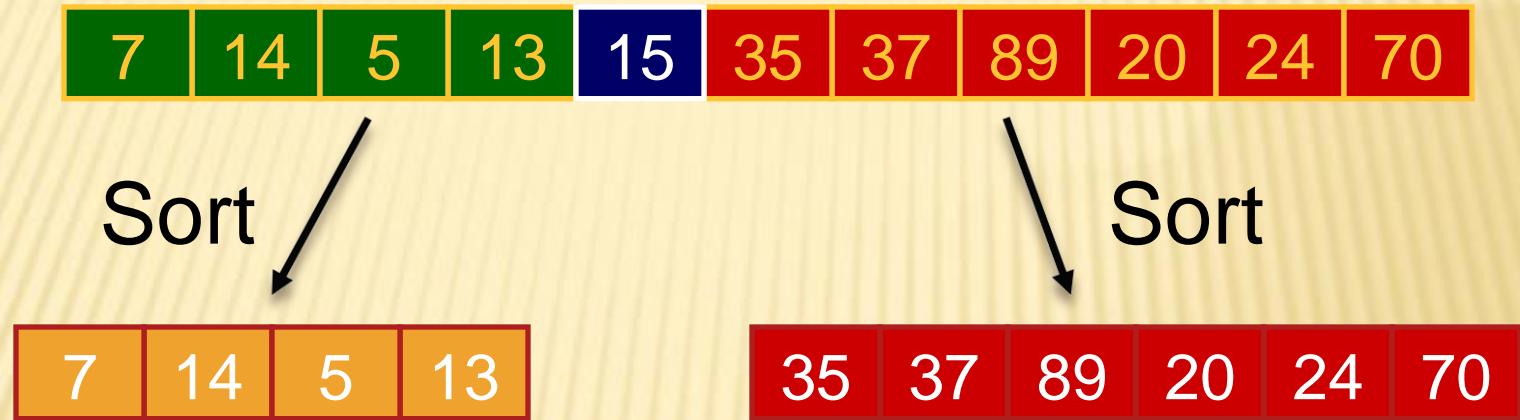
$x < 15$



$15 \leq x$

Example:

Partition



That is, the array will not be sorted, but all the elements less than the pivot will be to the left of the pivot, and all the elements greater than the pivot will be to the right.

```
int partition(float array[], int size)
{
    int k;
    int mid = size/2;
    int index = 0;

    swap(array, array+mid);

    for (k = 1; k < size; k++)
    {
        if (list[k] < list[0])
        {
            index++;
            swap(array+k, array+index);
        }
    }

    swap(array, array+index);

    return index;
}
```

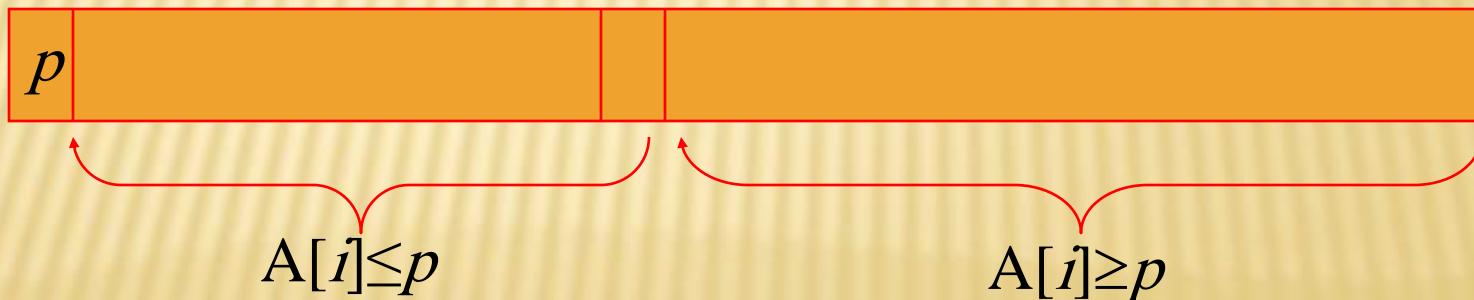
Quicksort

Quicksort uses the technique of divide-and-conquer in a different manner. We proceed as follows:

1. Pick an arbitrary element of the array (the pivot).
2. Divide the array into two segments, those that are smaller and those that are greater, with the pivot in between (the partition phase).
3. Recursively sort the segments to the left and right of the pivot.

Quicksort

- Select a *pivot* (partitioning element)
- Rearrange the list so that all the elements in the positions before the pivot are smaller than or equal to the pivot and those after the pivot are larger than or equal to the pivot
- Exchange the pivot with the last element in the first (i.e., \leq) sublist – the pivot is now in its final position
- Sort the two sublists recursively



Quicksort Algorithm

Given an array of n elements (e.g., integers):

- ✖ If array only contains one element, return
- ✖ Else
 - + pick one element to use as *pivot*.
 - + Partition elements into two sub-arrays:
 - ✖ Elements less than or equal to pivot
 - ✖ Elements greater than pivot
 - + Quicksort two sub-arrays
 - + Return results

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 \geq 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_index = 0`



`too_big_index`

`too_small_index`

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

A horizontal array of 10 colored boxes representing a list of integers. The first box at index 0 is red and contains the value 40. The other nine boxes are yellow and contain the values 20, 10, 80, 60, 50, 7, 30, and 100 respectively. Below each box is its corresponding index in brackets: [0], [1], [2], [3], [4], [5], [6], [7], and [8]. Two arrows point from labels at the bottom to specific indices: an arrow from the label "too_big_index" points to index [1], and another arrow from the label "too_small_index" points to index [8].

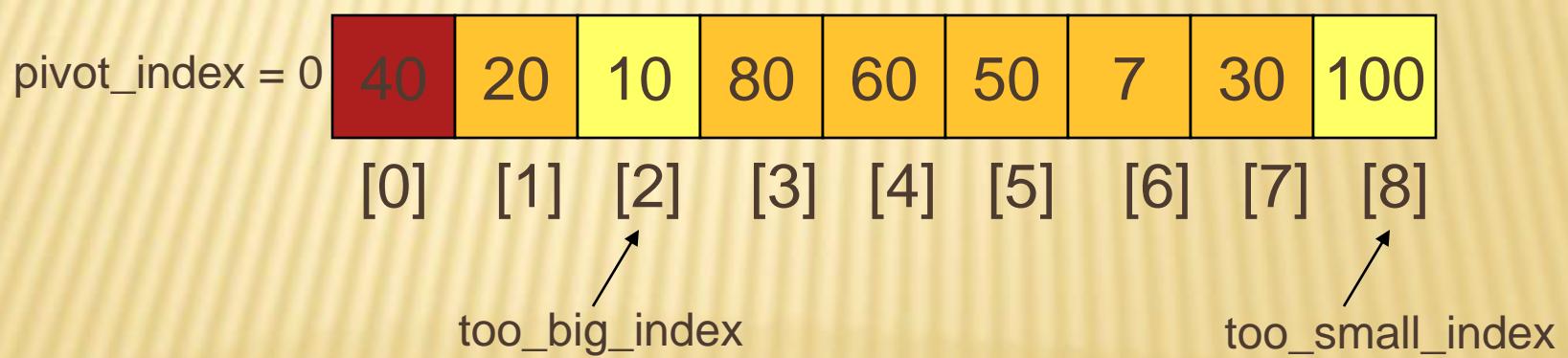
pivot_index = 0

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

too_big_index

too_small_index

1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 ++too_big_index



```
1. While data[too_big_index] <= data[pivot]
    ++too_big_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

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

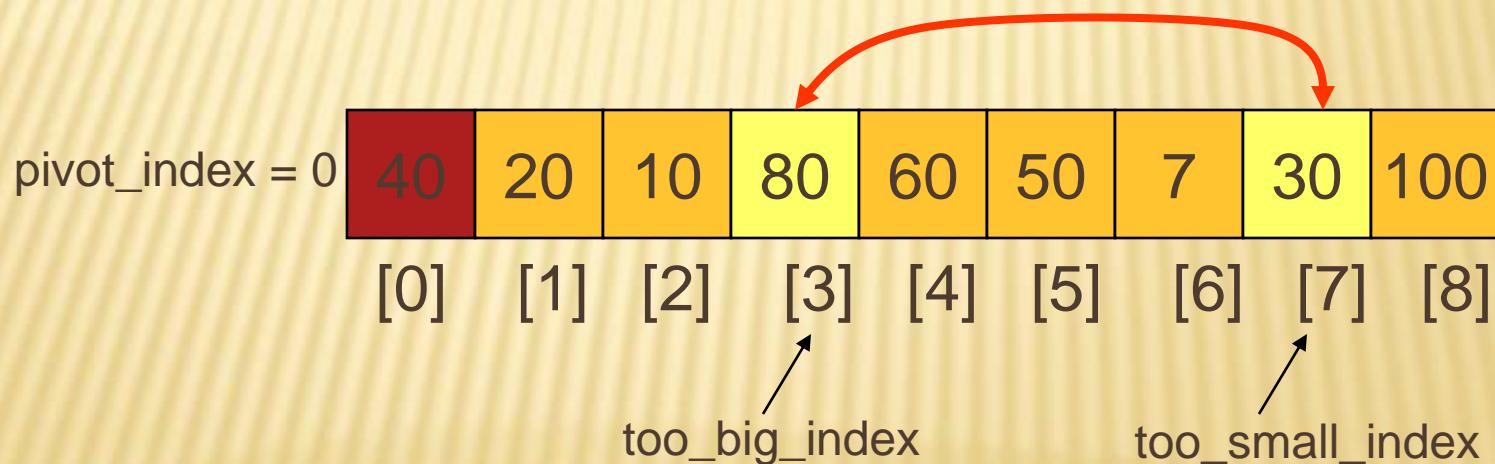
pivot_index = 0

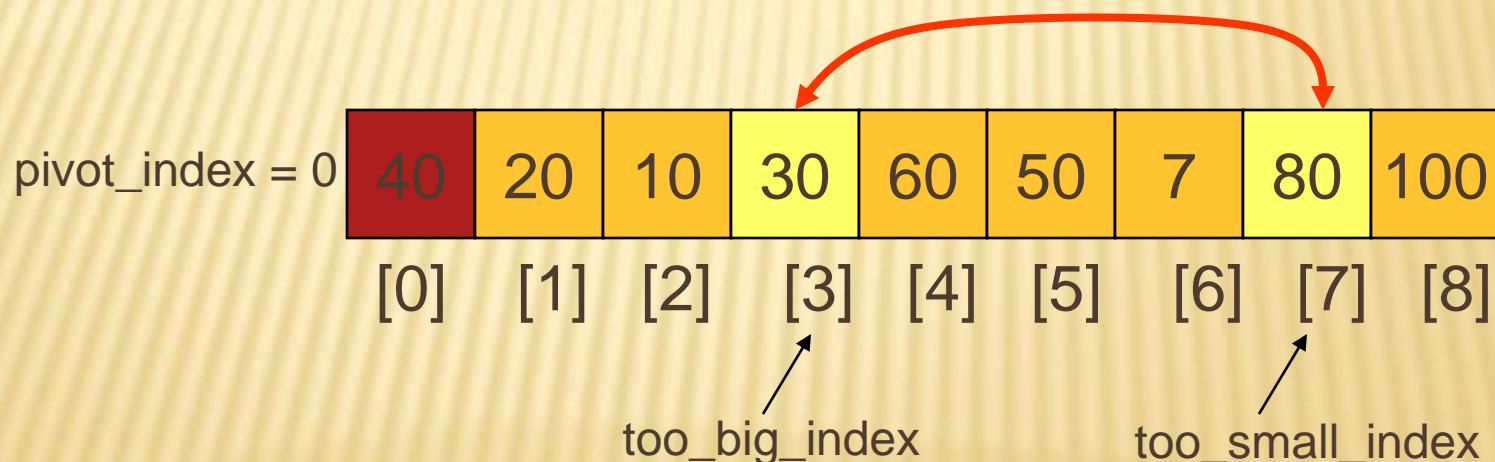
40	20	10	80	60	50	7	30	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index

too_small_index

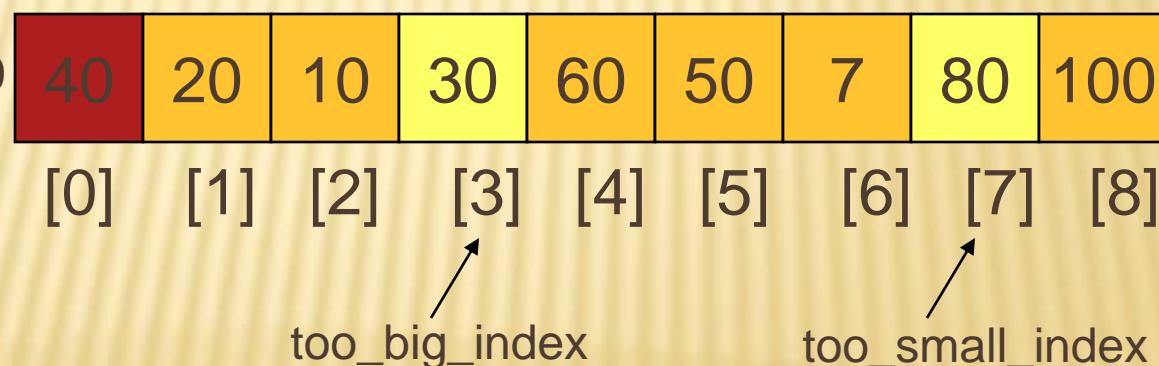
The diagram illustrates a step in the quicksort algorithm where the pivot element at index 0 (40) has been partitioned. Elements greater than 40 are moved to the right, and elements less than 40 are moved to the left. The element at index 3 (80) is identified as a 'too_big_index' because it is greater than the pivot. The element at index 7 (30) is identified as a 'too_small_index' because it is less than the pivot.





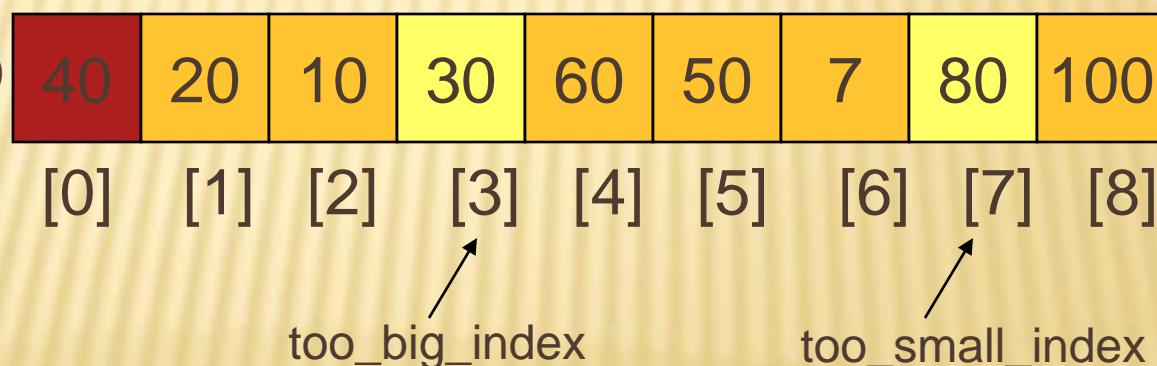
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

`pivot_index = 0`

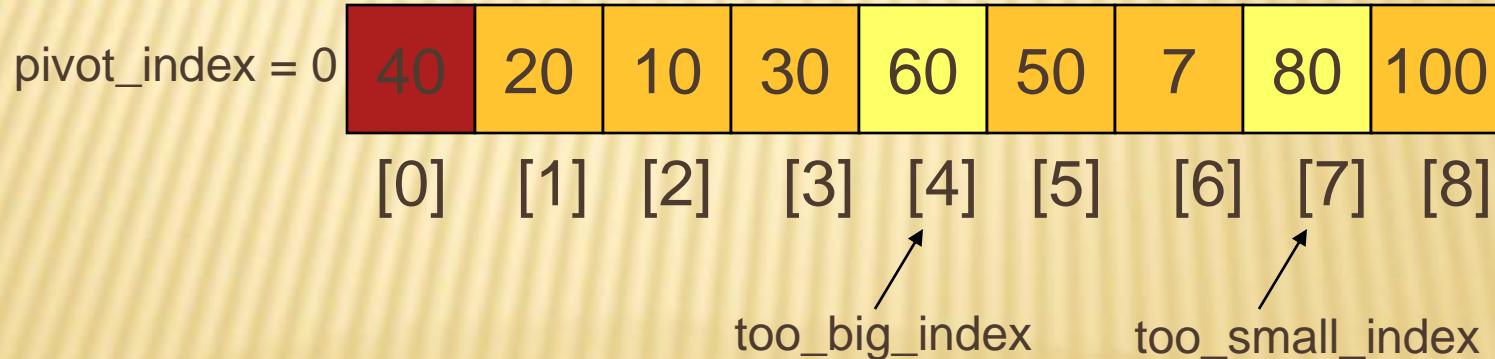


- 1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 - 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 - 3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
 - 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

`pivot_index = 0`

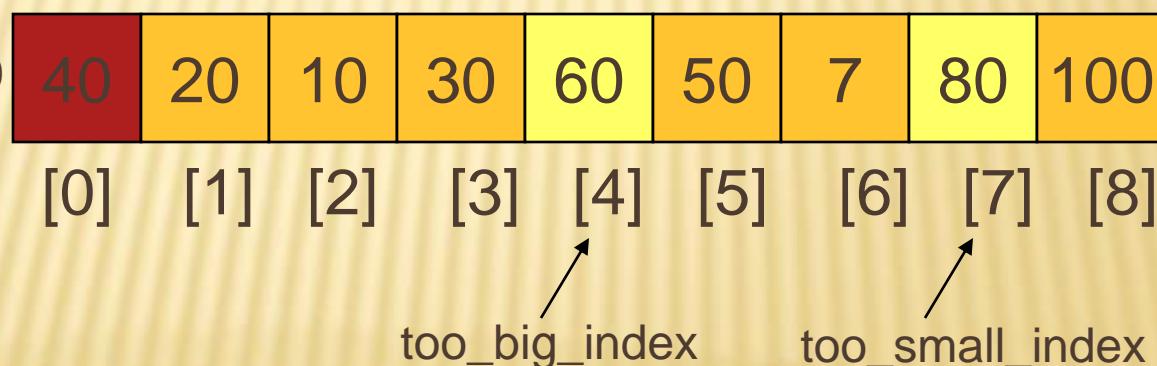


- 1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



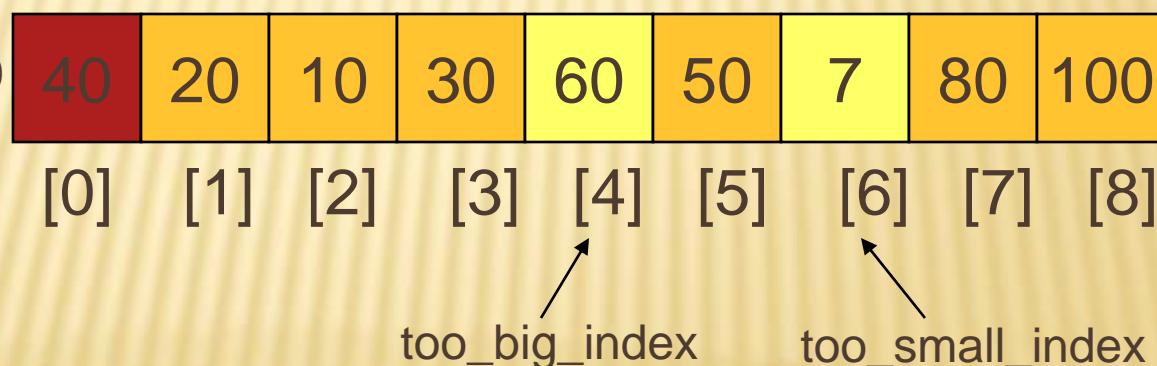
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 - 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

`pivot_index = 0`

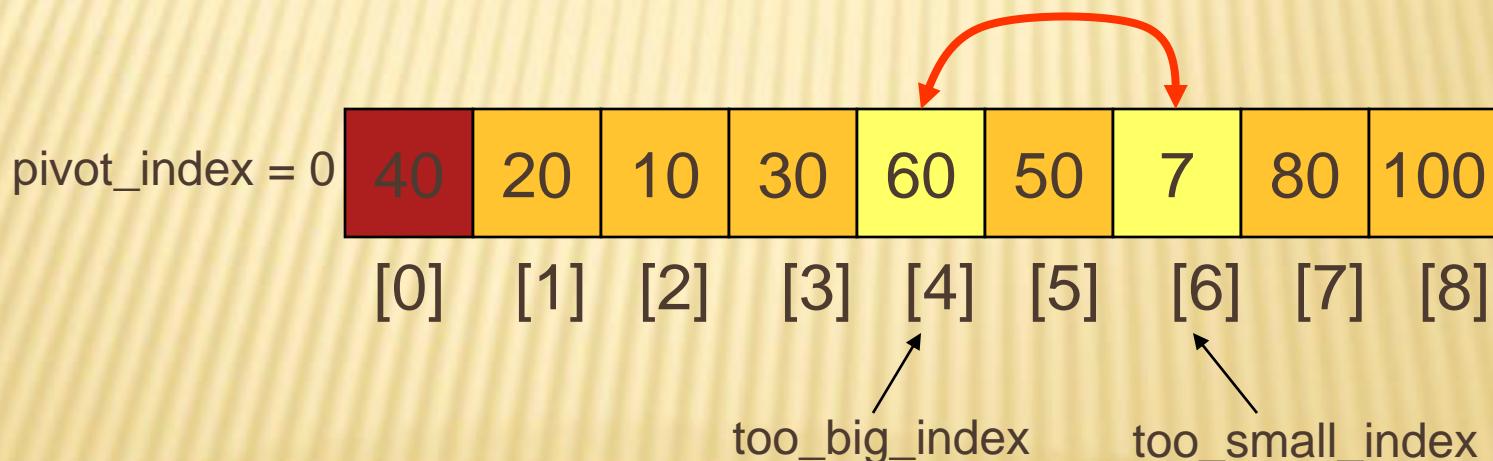


1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 - 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

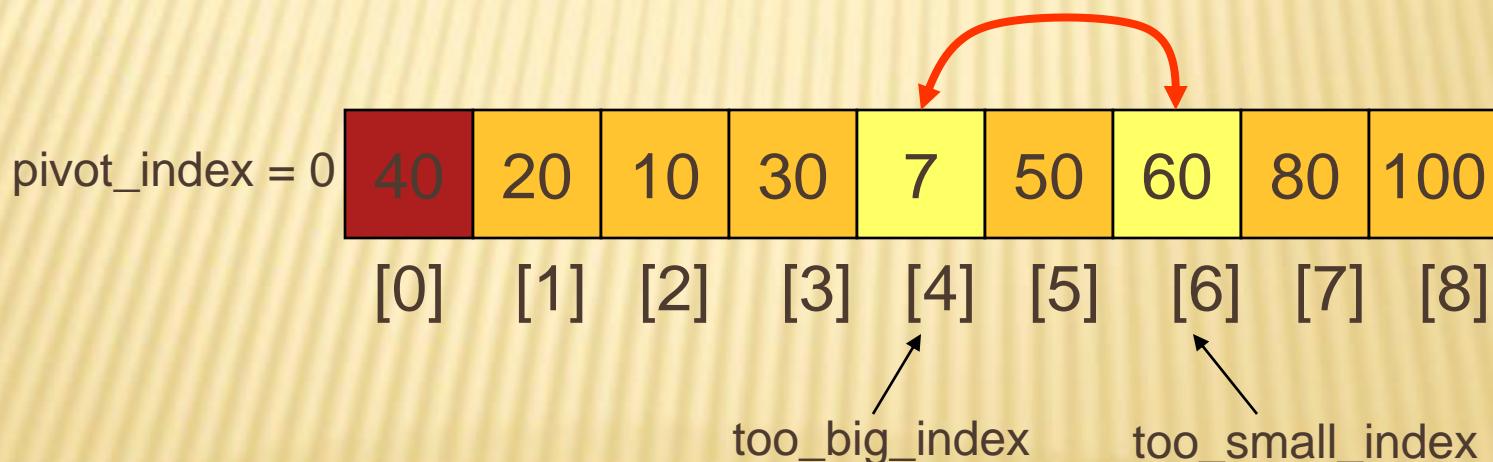
`pivot_index = 0`



1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

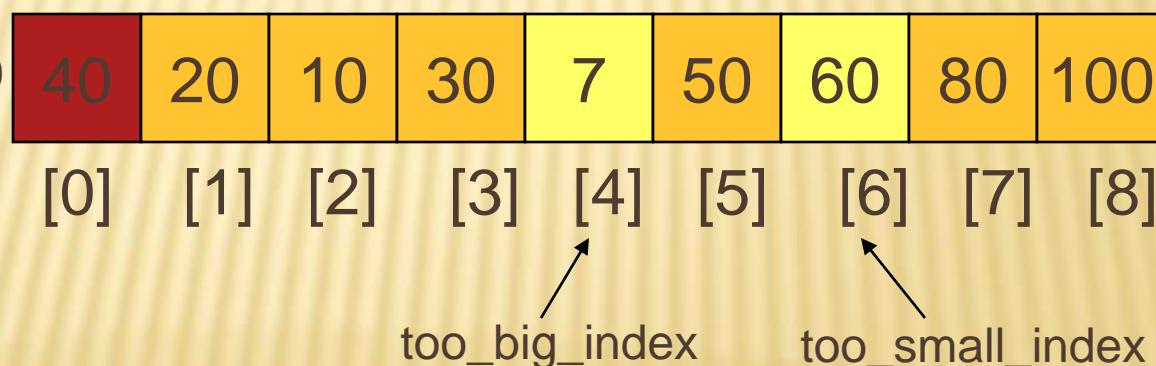


1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



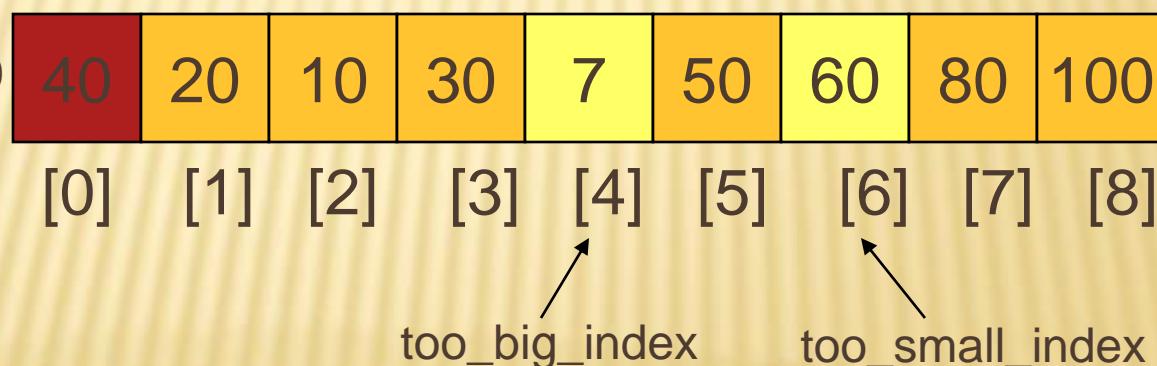
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 - 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

`pivot_index = 0`

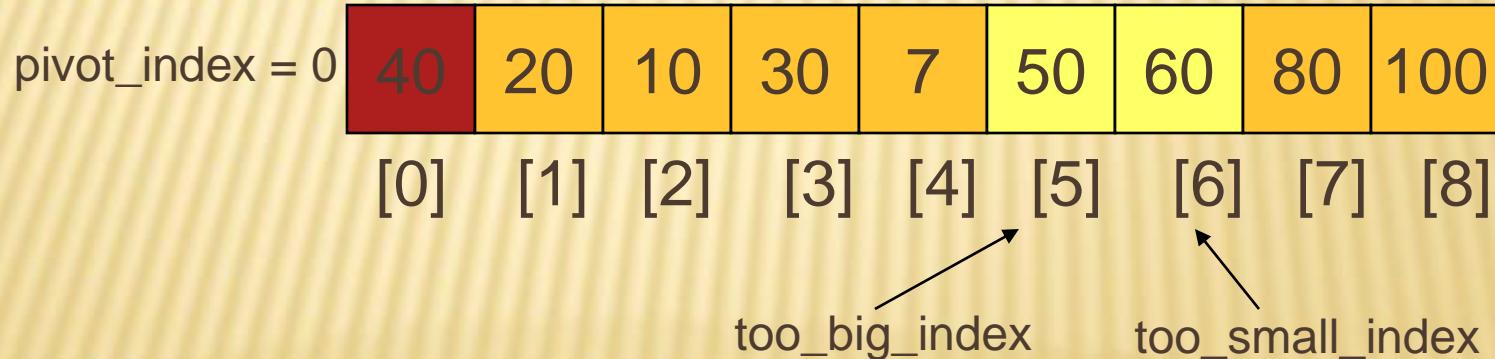


- 1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 - 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 - 3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
 - 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

`pivot_index = 0`

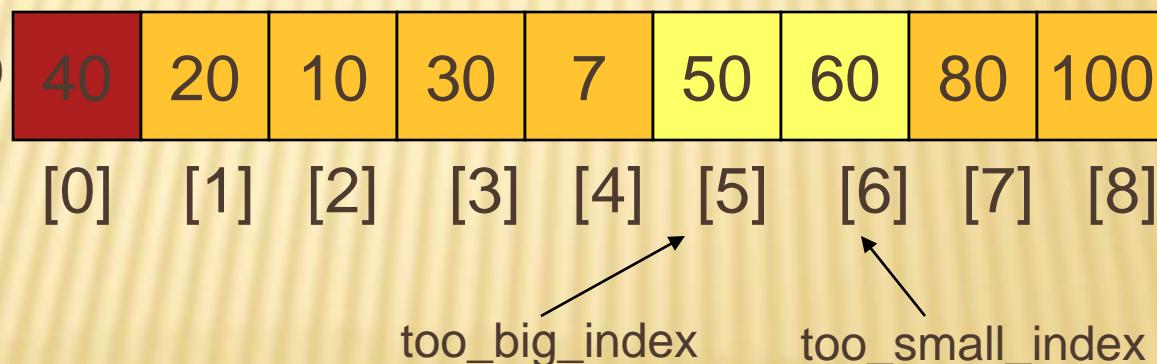


- 1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

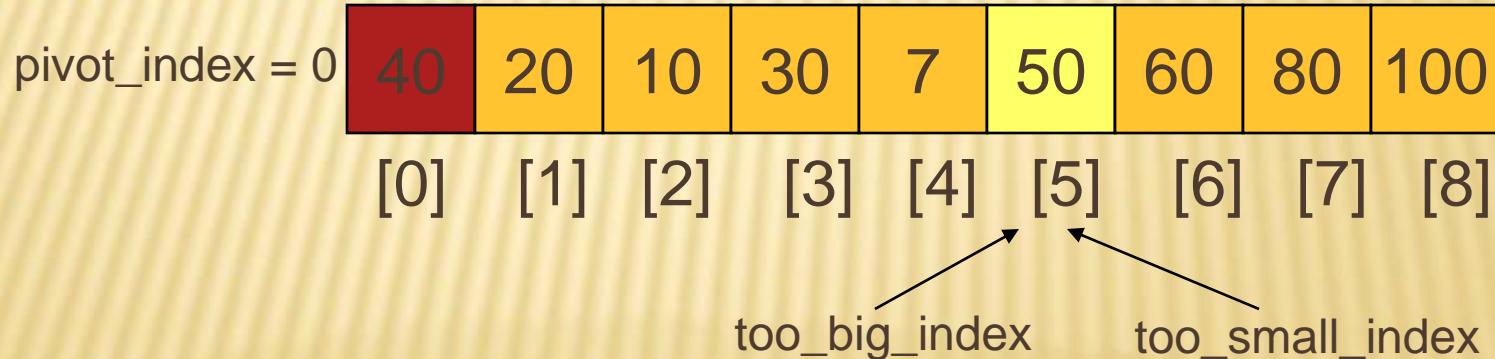


1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad ++\text{too_big_index}$
 - 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad --\text{too_small_index}$
 3. If $\text{too_big_index} < \text{too_small_index}$
 $\quad \quad \quad \text{swap } \text{data}[\text{too_big_index}] \text{ and } \text{data}[\text{too_small_index}]$
 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.

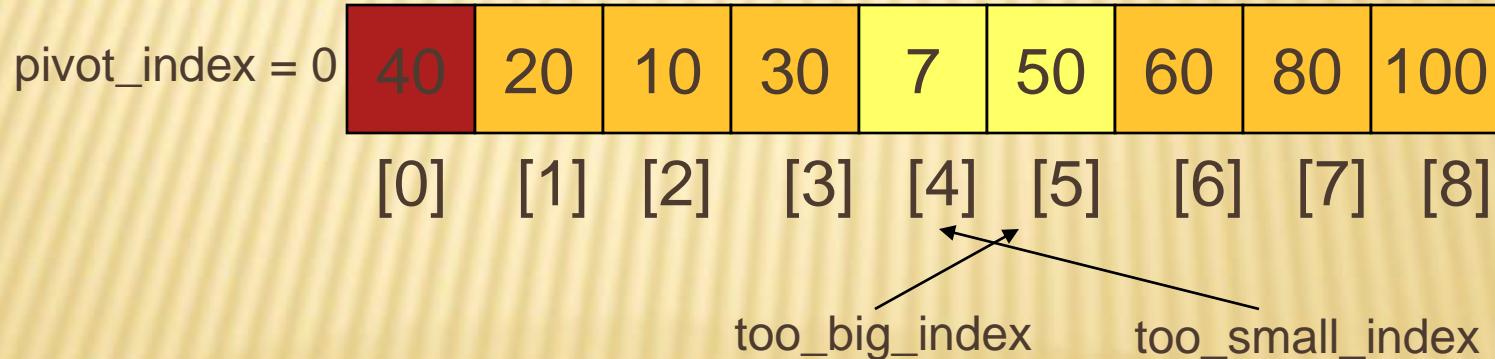
`pivot_index = 0`



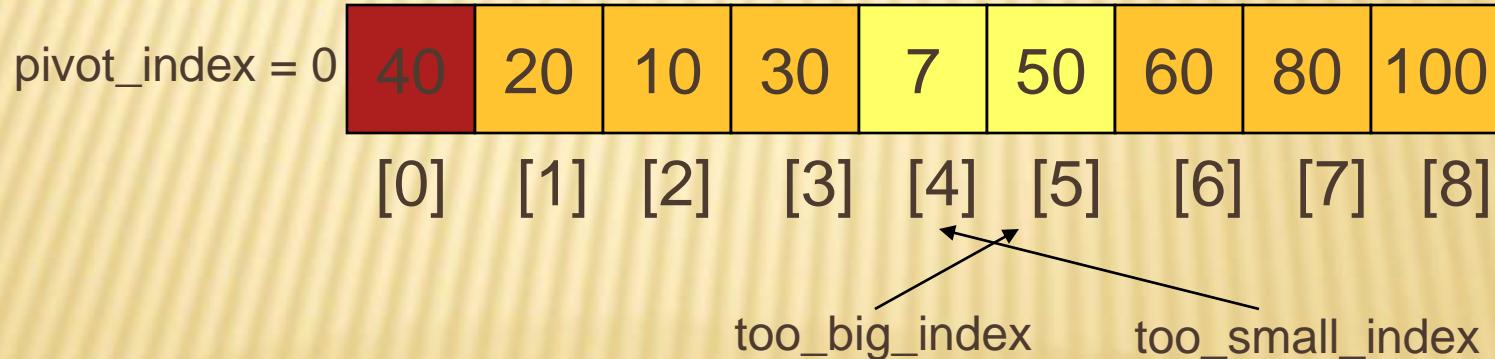
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
- 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



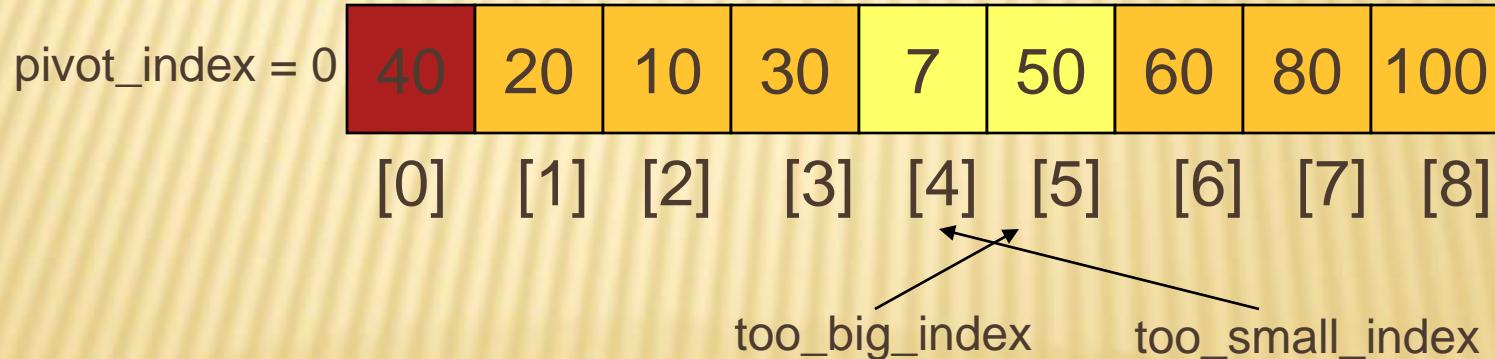
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
- 2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



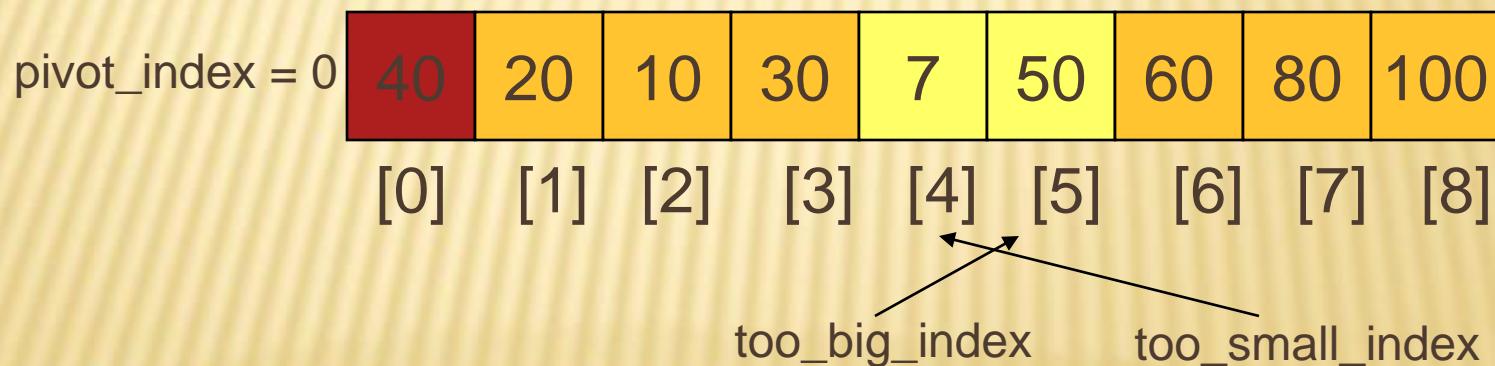
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
- 3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



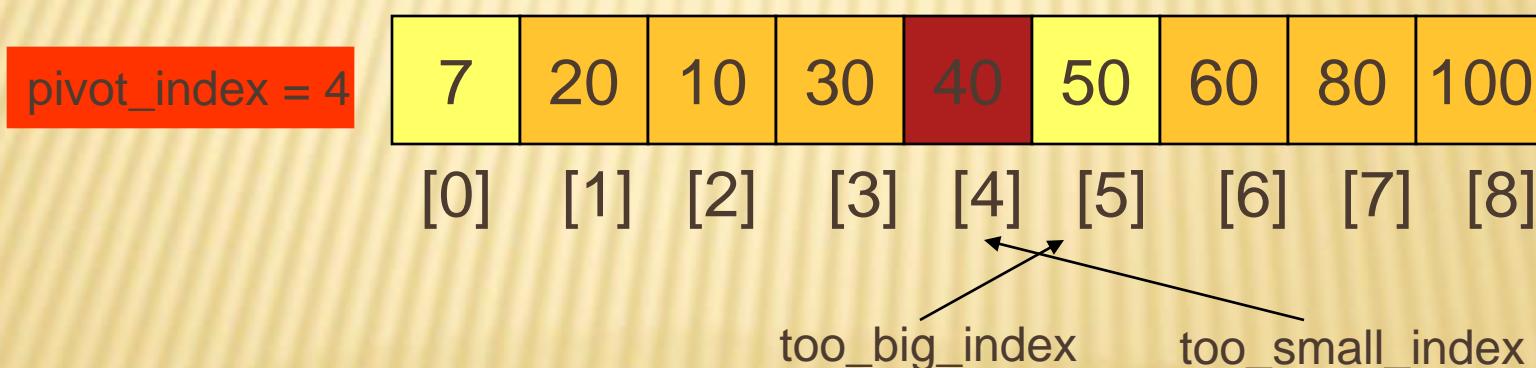
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
- 4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.



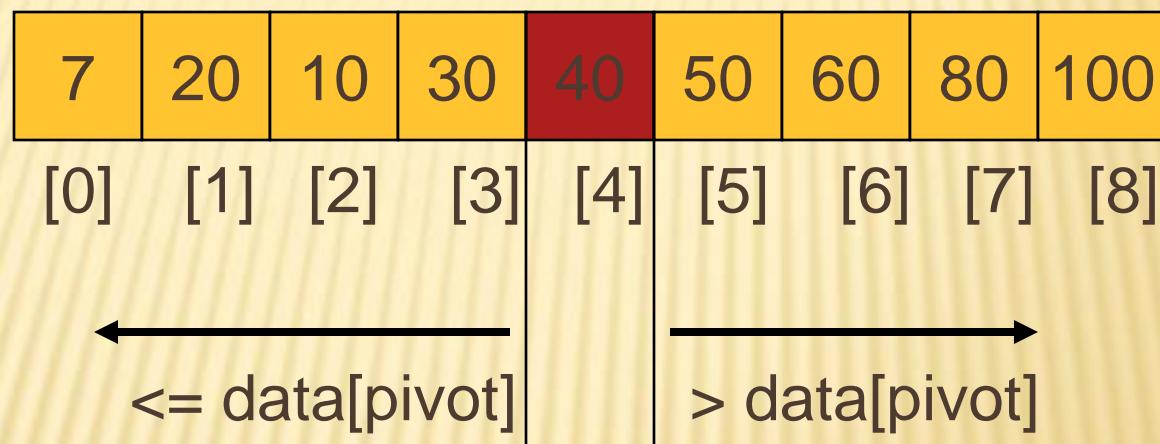
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.
- 5. Swap $\text{data}[\text{too_small_index}]$ and $\text{data}[\text{pivot_index}]$



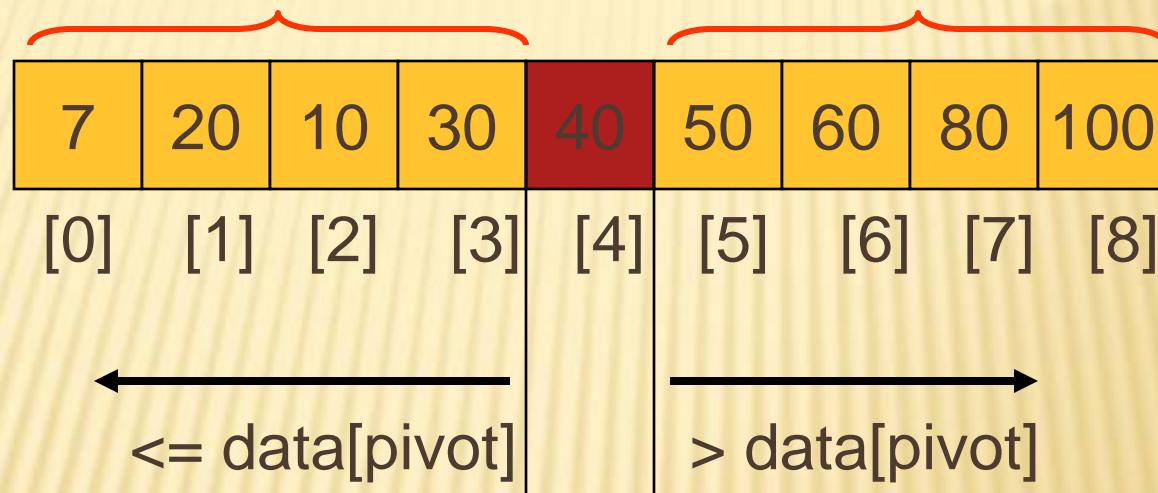
1. While $\text{data}[\text{too_big_index}] \leq \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{++too_big_index}$
2. While $\text{data}[\text{too_small_index}] > \text{data}[\text{pivot}]$
 $\quad \quad \quad \text{--too_small_index}$
3. If $\text{too_big_index} < \text{too_small_index}$
 swap $\text{data}[\text{too_big_index}]$ and $\text{data}[\text{too_small_index}]$
4. While $\text{too_small_index} > \text{too_big_index}$, go to 1.
- 5. Swap $\text{data}[\text{too_small_index}]$ and $\text{data}[\text{pivot_index}]$



Partition Result



Recursion: Quicksort Sub-arrays





Thank You

