Luiss
Libera Università Internazionale degli Studi Sociali Guido Carli

Algorithms A.Y. 2022/2023

Lab - Quick Sort

Irene Finocchi, Flavio Giorgi, Bardh Prenkaj
finocchi@luiss.it, fgiorgi@luiss.it, bprenkaj@luiss.it©

17 February 2023

courtesy of: *Andrea Coletta*

LUISS

Dipartimento di Impresa e Management

Quick sort – a step by step example

Quick sort is another divide-and-conquer recursive algorithm.

It is very efficient.

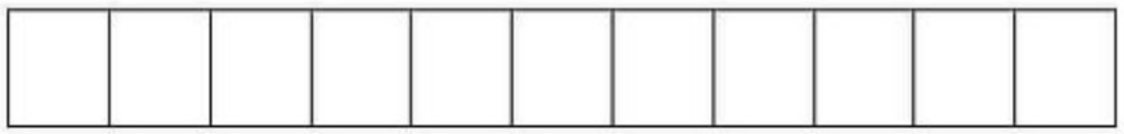
And can be even optimized!

<u>It can be used for the project along with the other algorithm we saw!</u>

LUISS

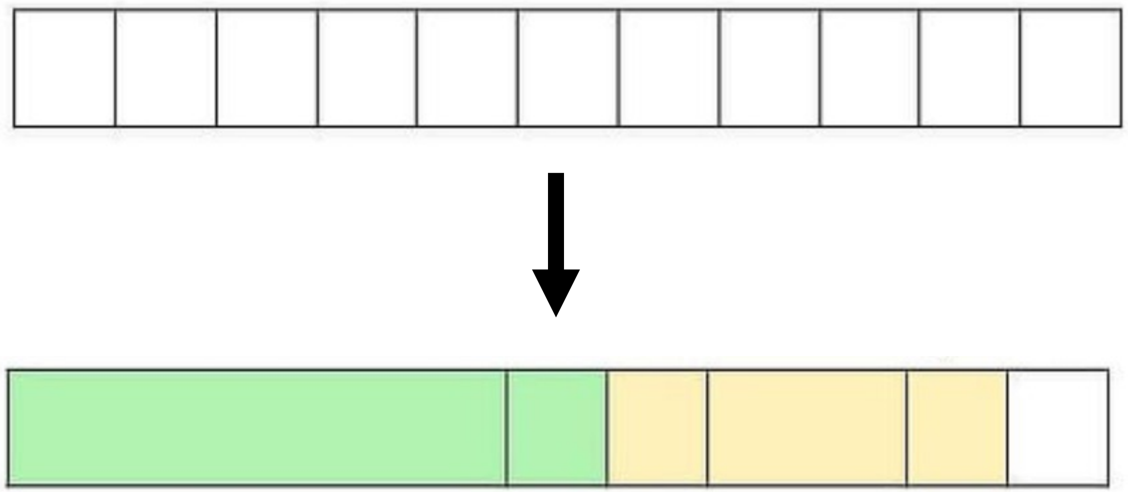
Quick sort – a step by step example

The goal of Quick Sort is to **partition** the list into two sub-list



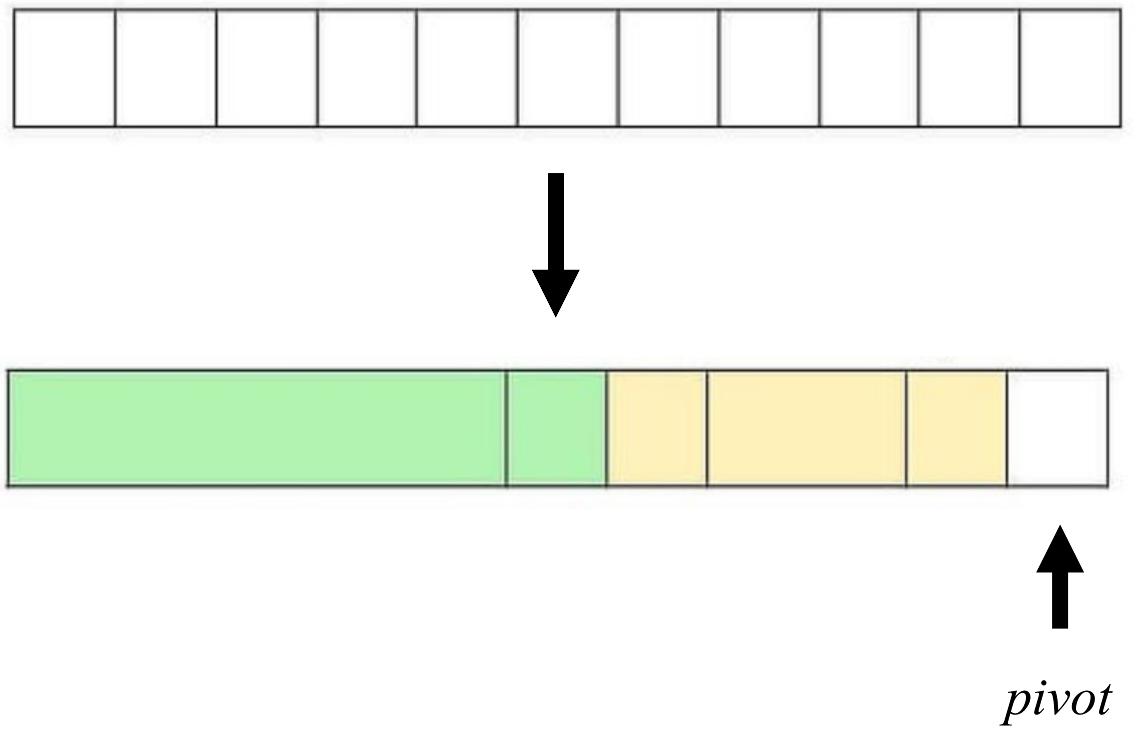
Quick sort – a step by step example

The goal of Quick Sort is to **partition** the list into two sub-list



Quick sort – a step by step example

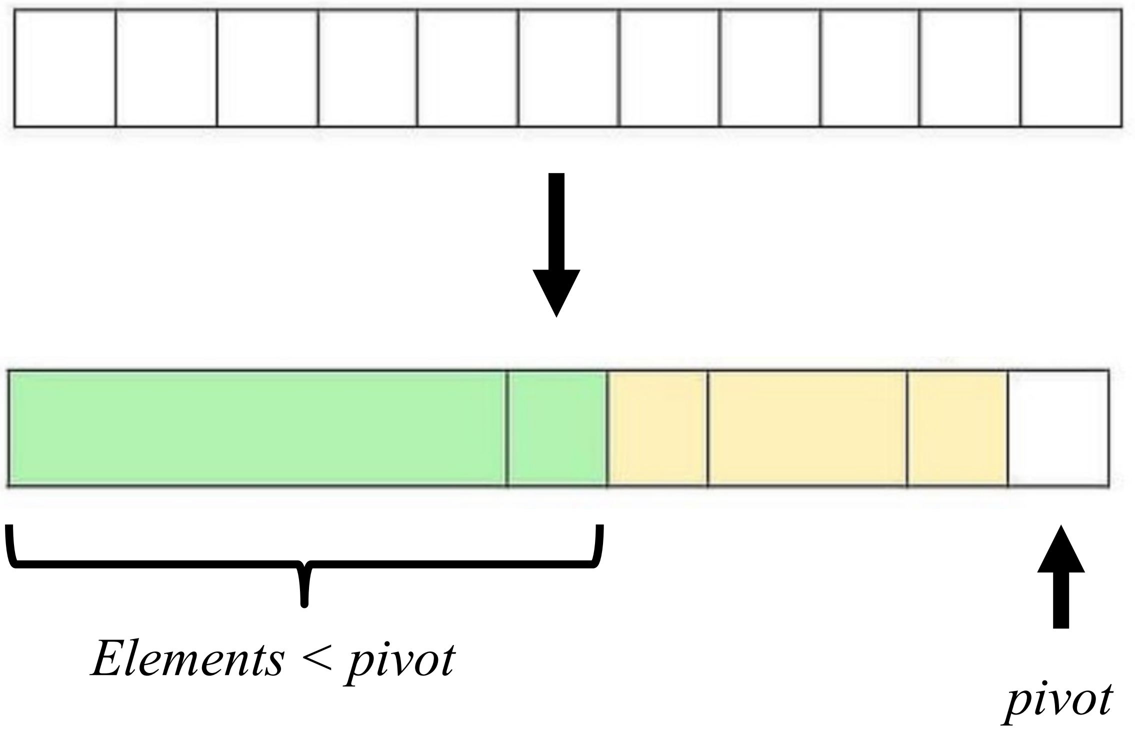
The goal of Quick Sort is to **partition** the list into two sub-list



LUISS

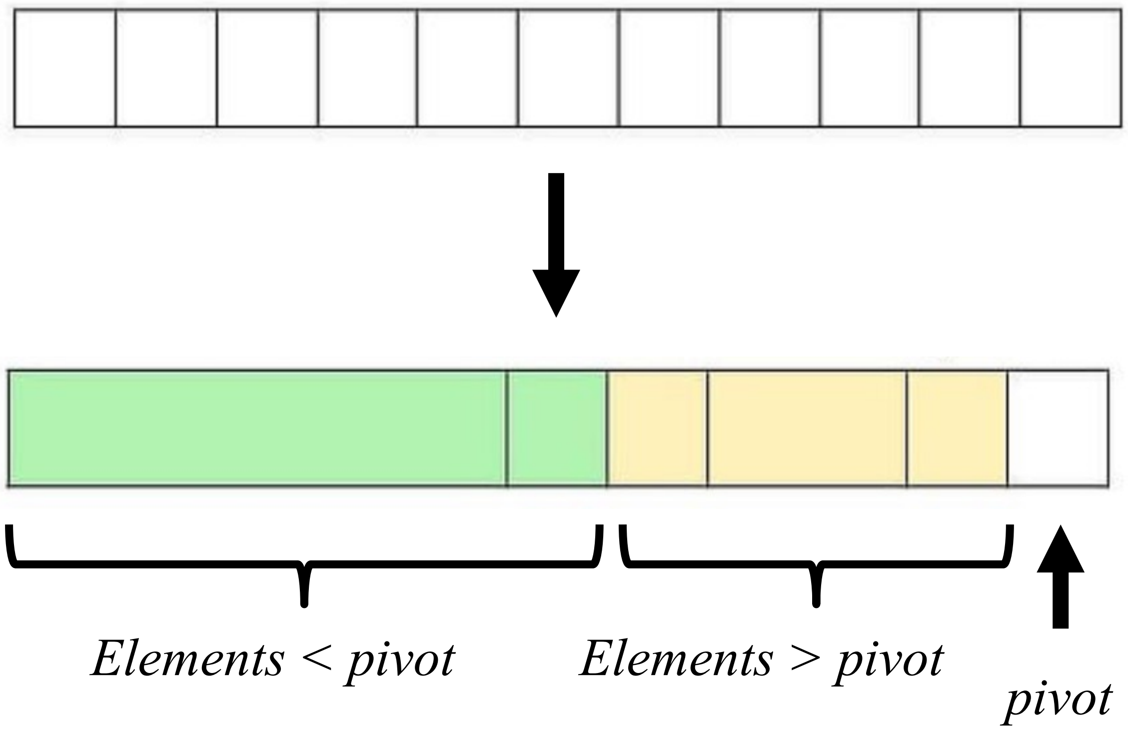
Quick sort – a step by step example

The goal of Quick Sort is to **partition** the list into two sub-list



Quick sort – a step by step example

The goal of Quick Sort is to **partition** the list into two sub-list



Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

pivot:

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

pivot: The pivot can be a random position in the list
Anyway there are clever ways to choose it!

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

pivot: <u>A common choice is the last element of the array!</u>

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

pivot

pivot: 5

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	i					*pivot*

pivot: 5

Now we start comparing the elements in the list with the pivot starting from index i = 0

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

i *pivot*

pivot: 5

Is the element in i = 0 grater or smaller than the element at the pivot?

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	$\uparrow$ *i*					$\uparrow$ *pivot*

pivot: 5

It is grater!

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

i

pivot

pivot: 5

Now we can declare another pointer j to keep track of the element that is grater than the pivot element

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

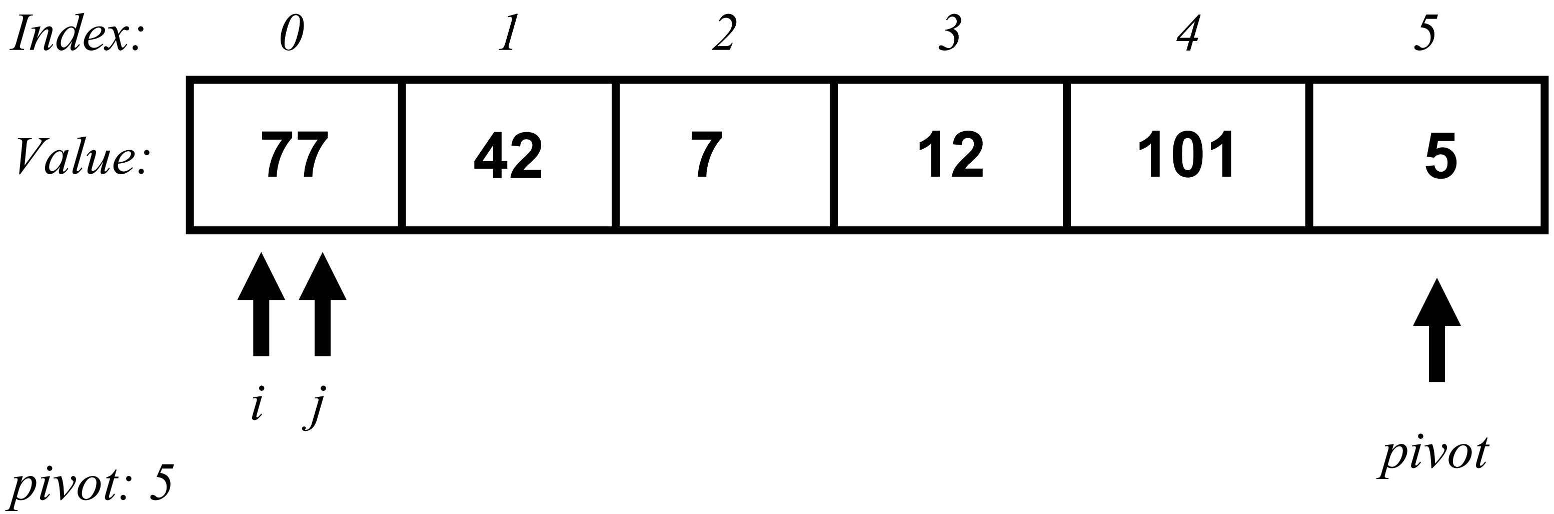
i *j*

pivot

pivot: 5

Now we can declare another pointer j to keep track of the element that is grater than the pivot element

Quick Sort: an example



Now we can declare another pointer j to keep track of the element that is grater than the pivot element

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**

j *i* *pivot*

pivot: 5

Next iteration we increase i = 1

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	j	*i*				*pivot*

pivot: 5

Again is 42 grater or smaller than 5?

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	j	*i*				*pivot*

pivot: 5

Again is 42 grater or smaller than 5?
It is grater so we increase i again

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	j		*i*			*pivot*

pivot: 5

Same here, is 7 grater or smaller than 5?
It is grater!
So we increase again the i pointer

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	↑ *j*				↑ *i*	↑ *pivot*

pivot: 5

Until we reach 101 because the the same reasoning holds for all the elements!

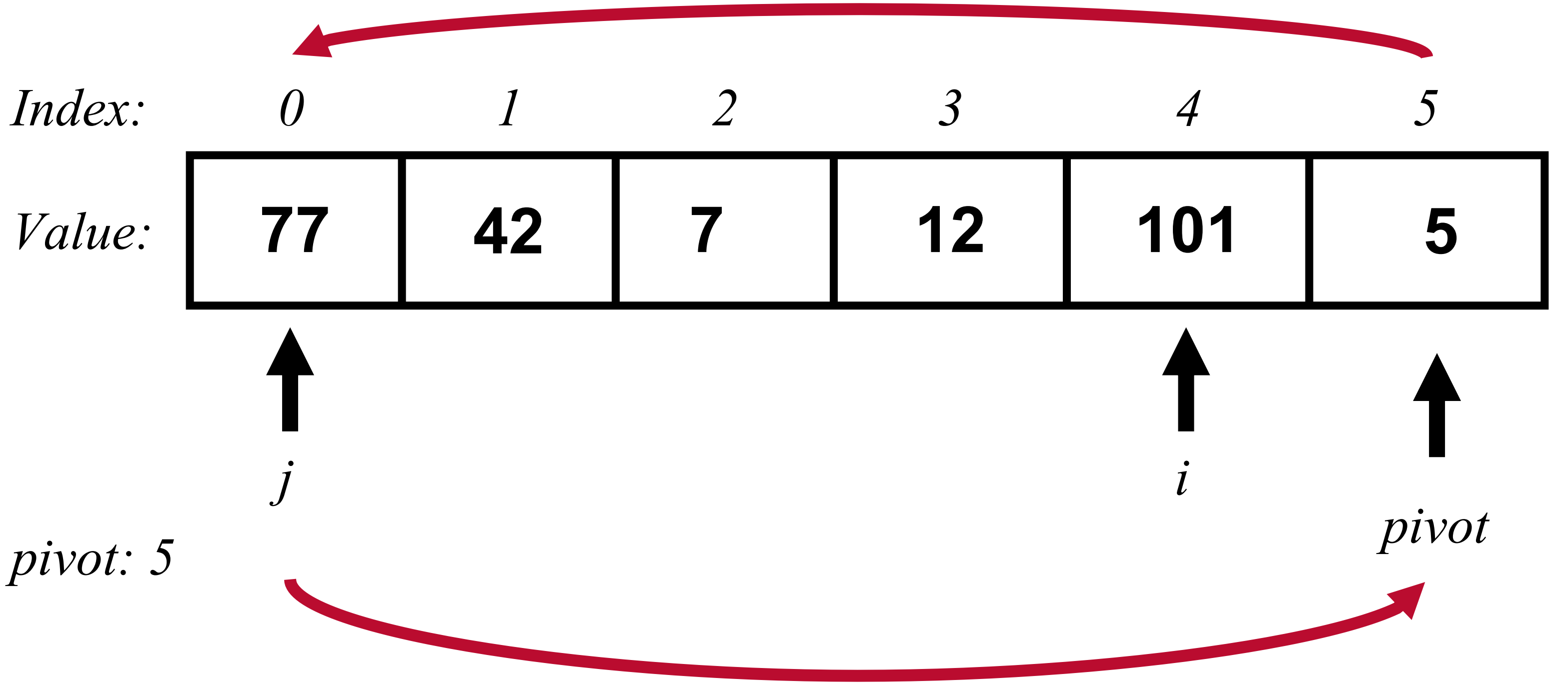
Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**7**	**12**	**101**	**5**
	↑ *j*				↑ *i*	↑ *pivot*

pivot: 5

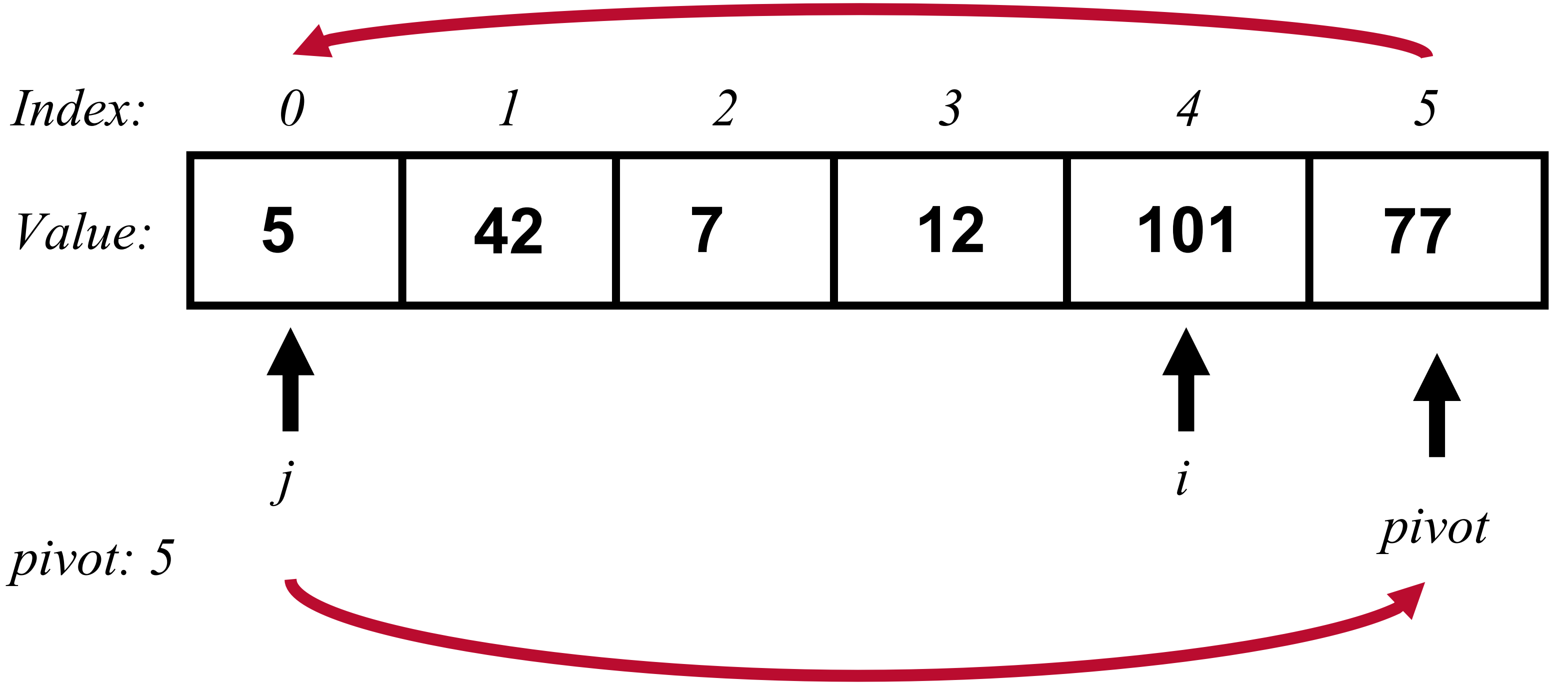
Now, the j pointer points at the first grater number we found in the list!

Quick Sort: an example



In order to order the list we have to swap the element at the *j-th* position with the element at the pivot position!

Quick Sort: an example



In order to order the list we have to **swap** the element at the *j-th* position with the element at the pivot position!

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**42**	**7**	**12**	**101**	**77**

pivot: 5

Now the element that was **the pivot is used to split in two sub-lists** the original list

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**42**	**7**	**12**	**101**	**77**

Elements > pivot

pivot: 5

Since it is at the beginning of the list, **we won't get** the left side sub-list.

Quick Sort: an example

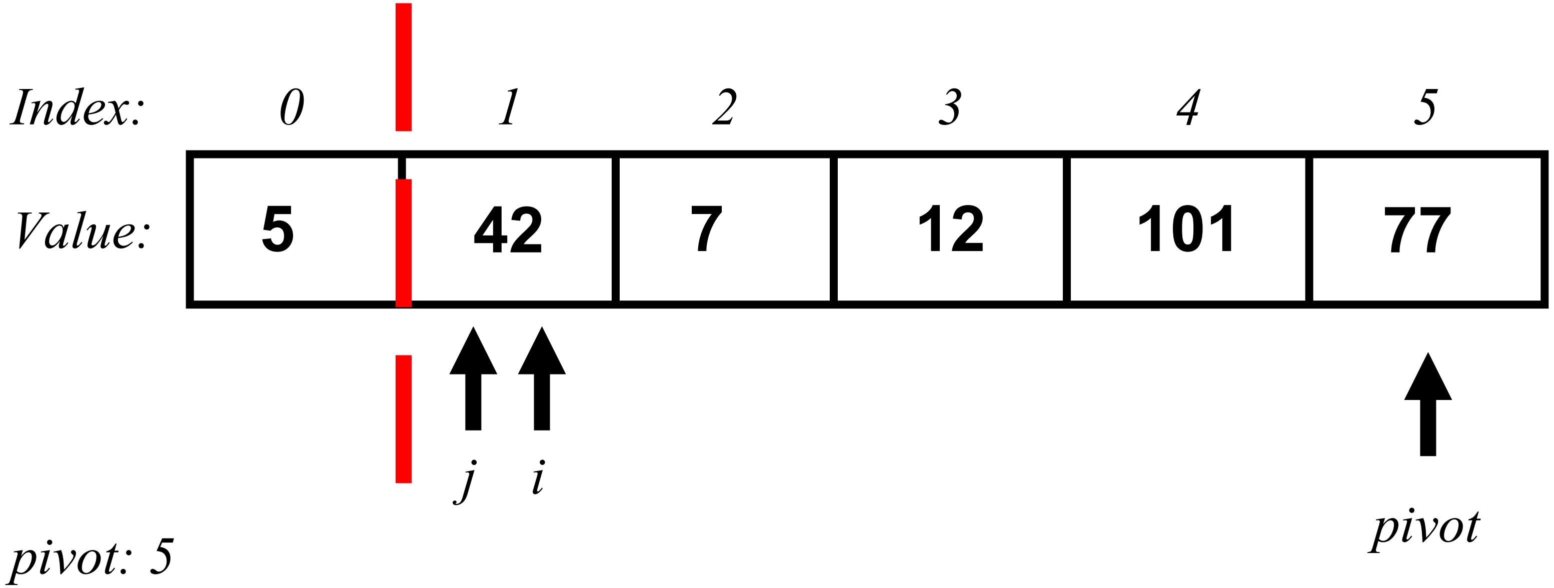
Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**42**	**7**	**12**	**101**	**77**

Elements > pivot

pivot: 5

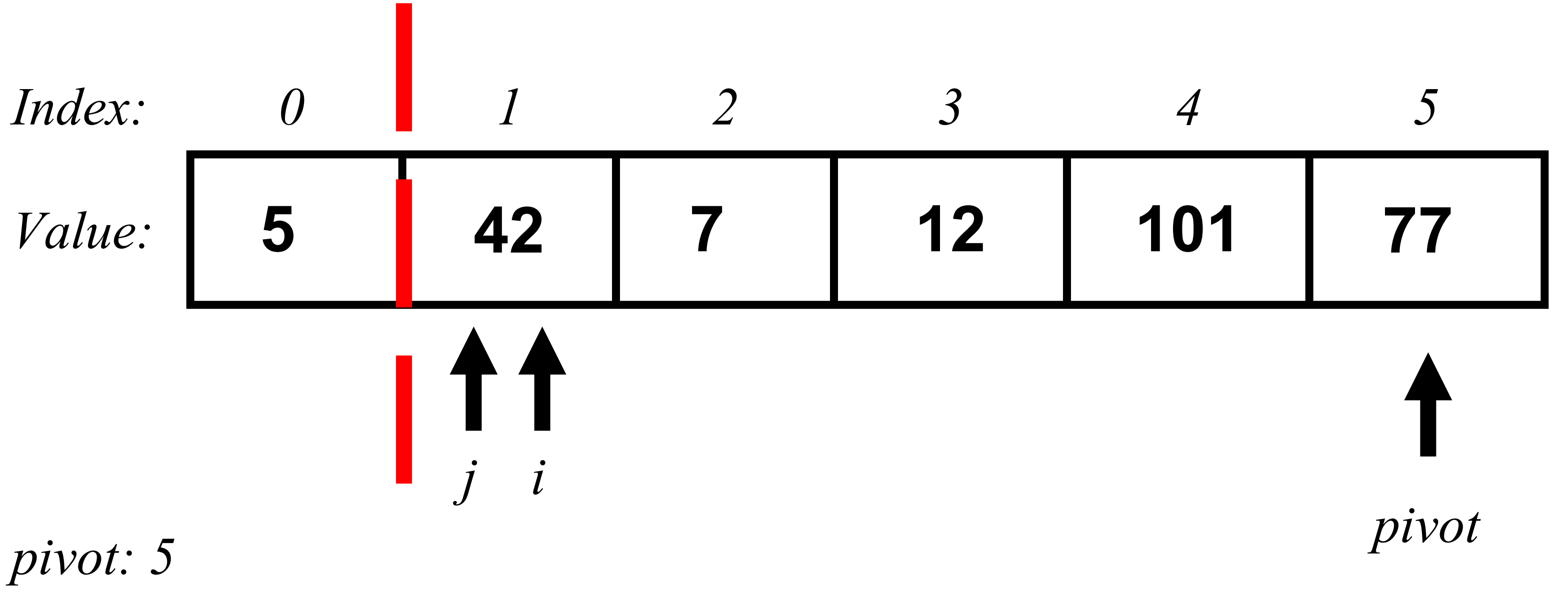
After the first iteration we have three properties:
1) The element that was the **pivot is in its final position**
2) Every element in the **right side** list is **grater** than the pivot
3) Every element in the **left side** of the list is **smaller** than the pivot

Quick Sort: an example



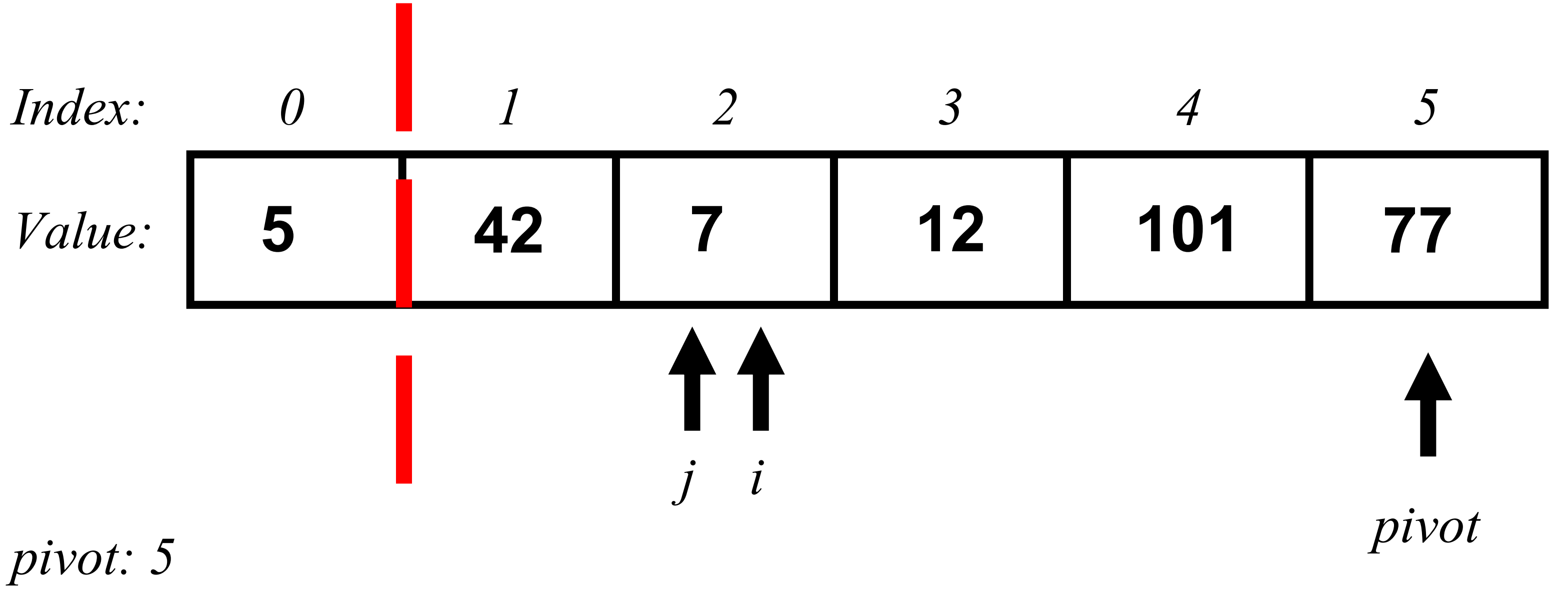
Again, we **select** the **last element** of the list as a **pivot**

Quick Sort: an example



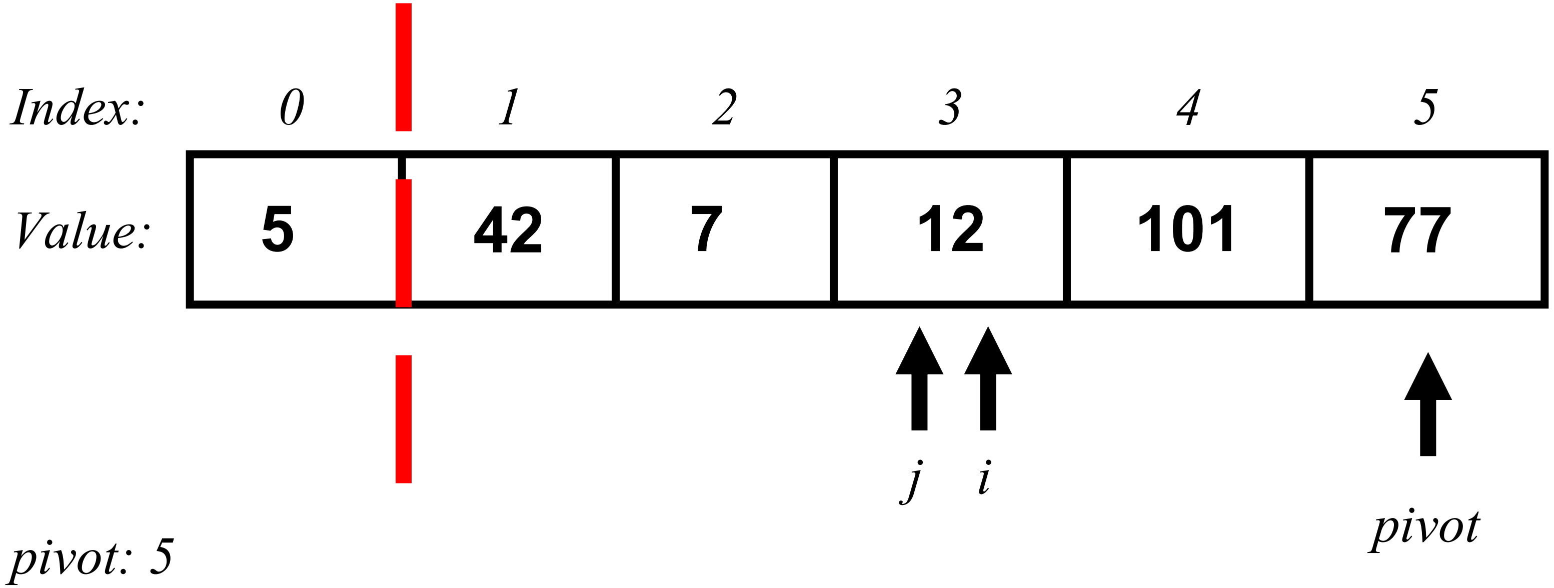
We ask: Is 42 grater or smaller than 77?
It is clearly smaller!
We increase both j and i index

Quick Sort: an example



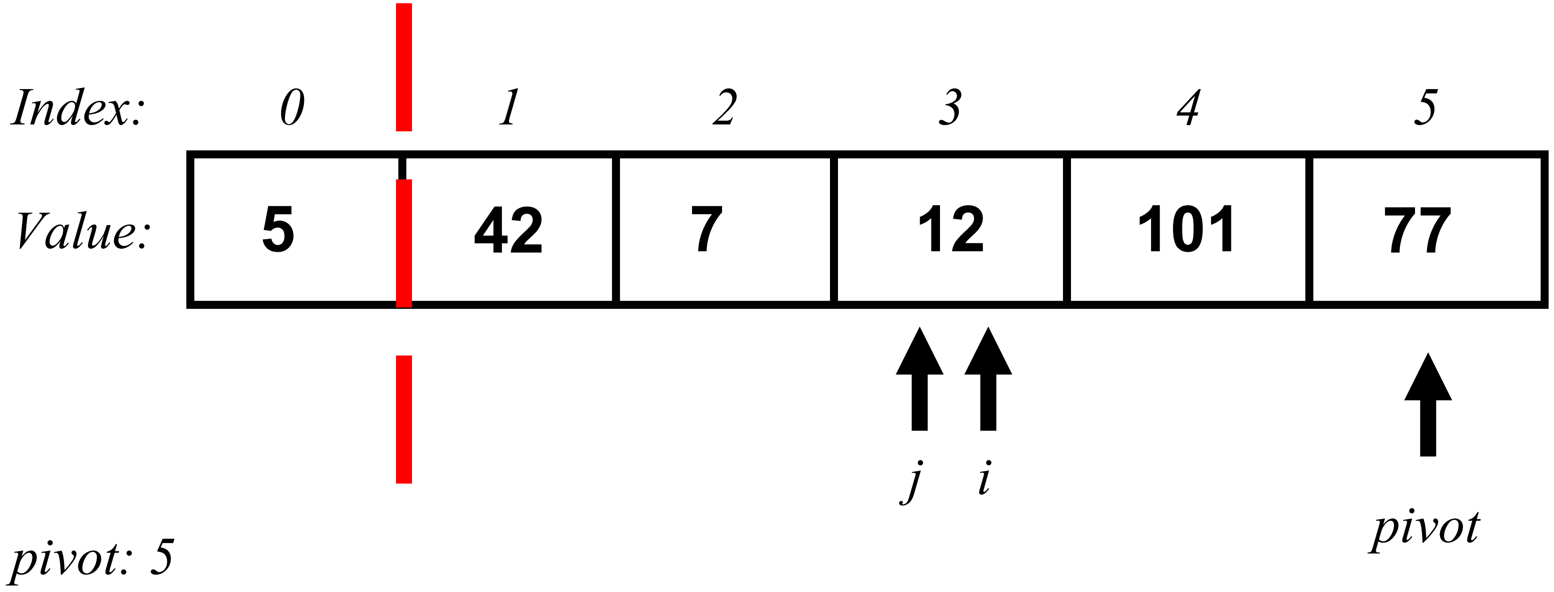
We ask: Is 7 grater or smaller than 77?
It is clearly smaller!
We increase both j and i index again

Quick Sort: an example



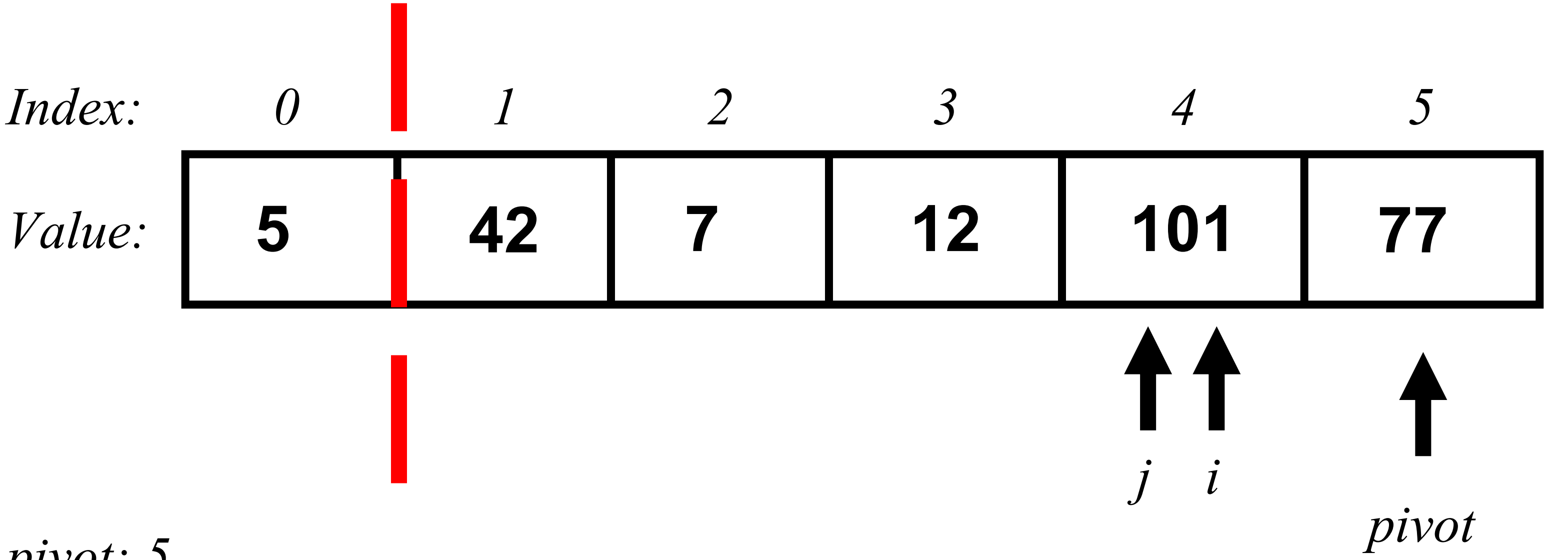
We ask: Is 7 grater or smaller than 77?
It is clearly smaller!
We increase both j and i index again

Quick Sort: an example



We ask: Is 12 grater or smaller than 77?
It is clearly smaller!
We increase both j and i index again

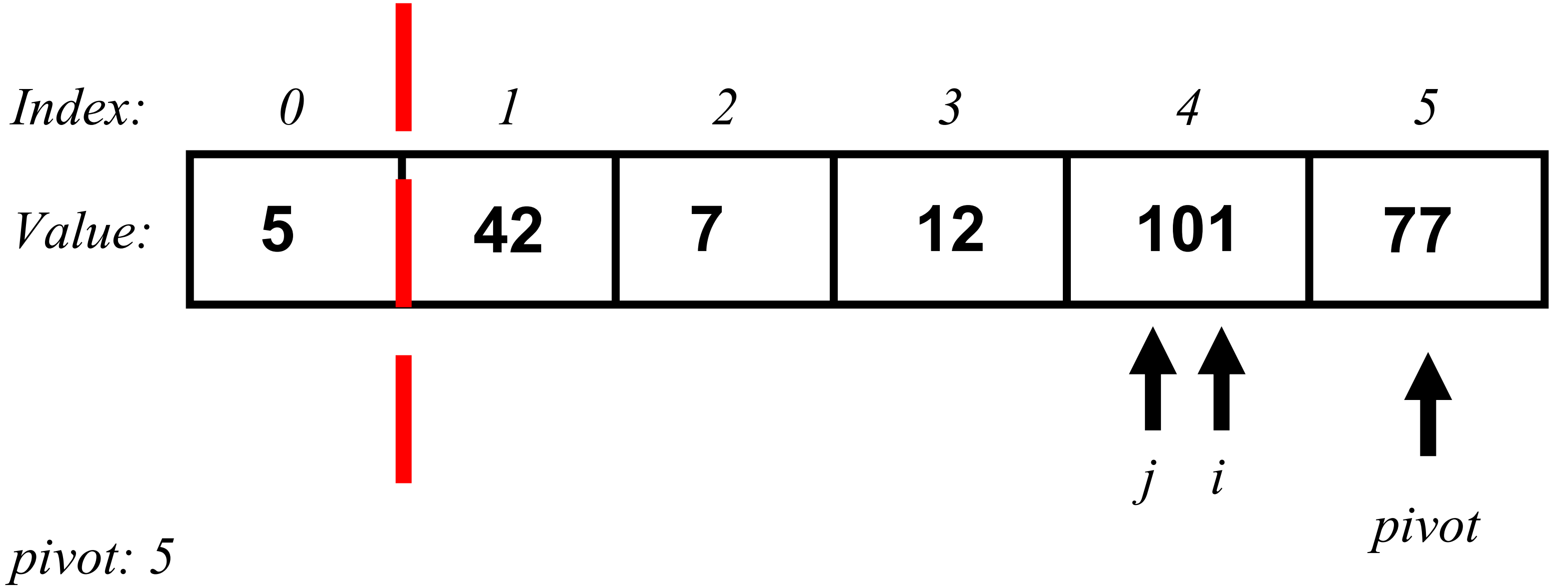
Quick Sort: an example



pivot: 5

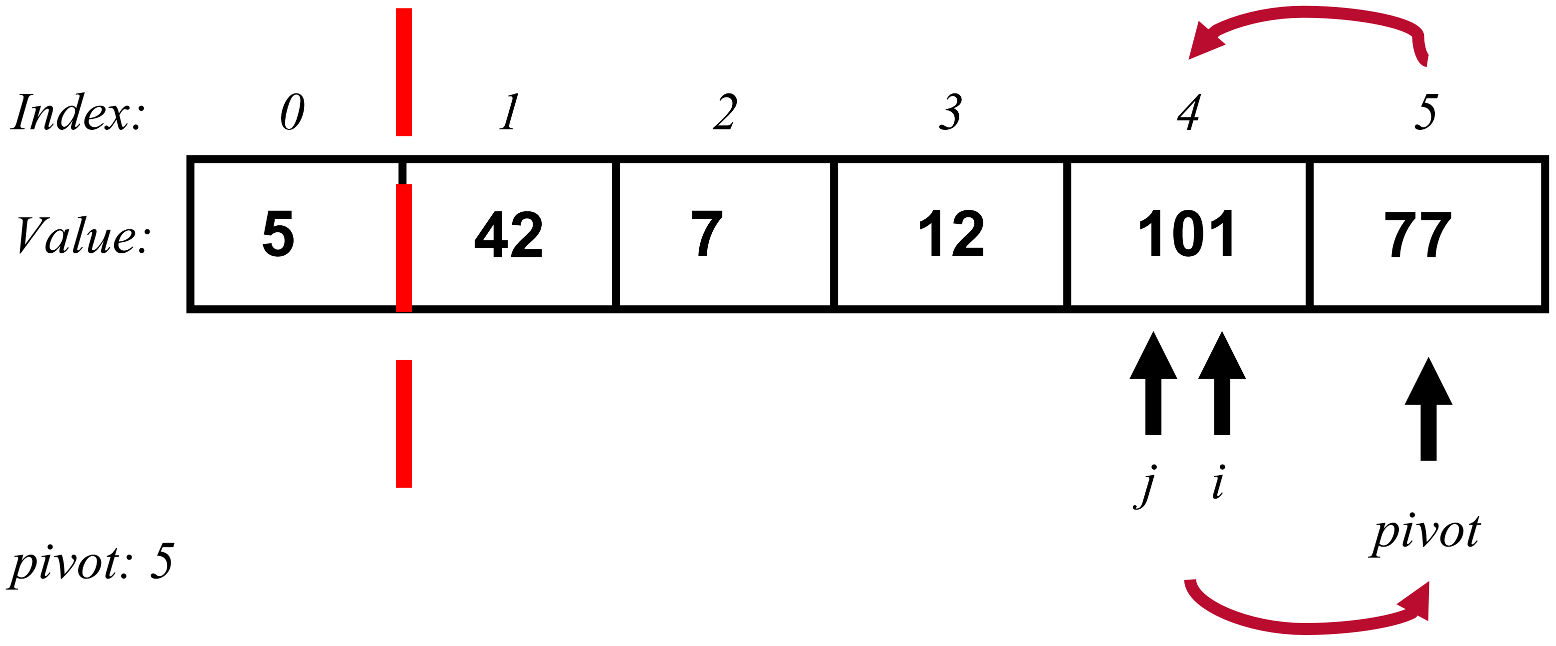
We ask: Is 12 grater or smaller than 77?
It is clearly smaller!
We increase both j and i index again

Quick Sort: an example



We ask: Is 101 grater or smaller than 77?
It is clearly grater!
So j = 4, also i = 4 because we cannot go further!

Quick Sort: an example



Now we swap the pivot with the element at position j=4

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**42**	**7**	**12**	**77**	**101**

Elements < pivot (indices 1–3)

Elements > pivot (index 5)

pivot: 5

Now we get two sub-lists, the left side and the right side.

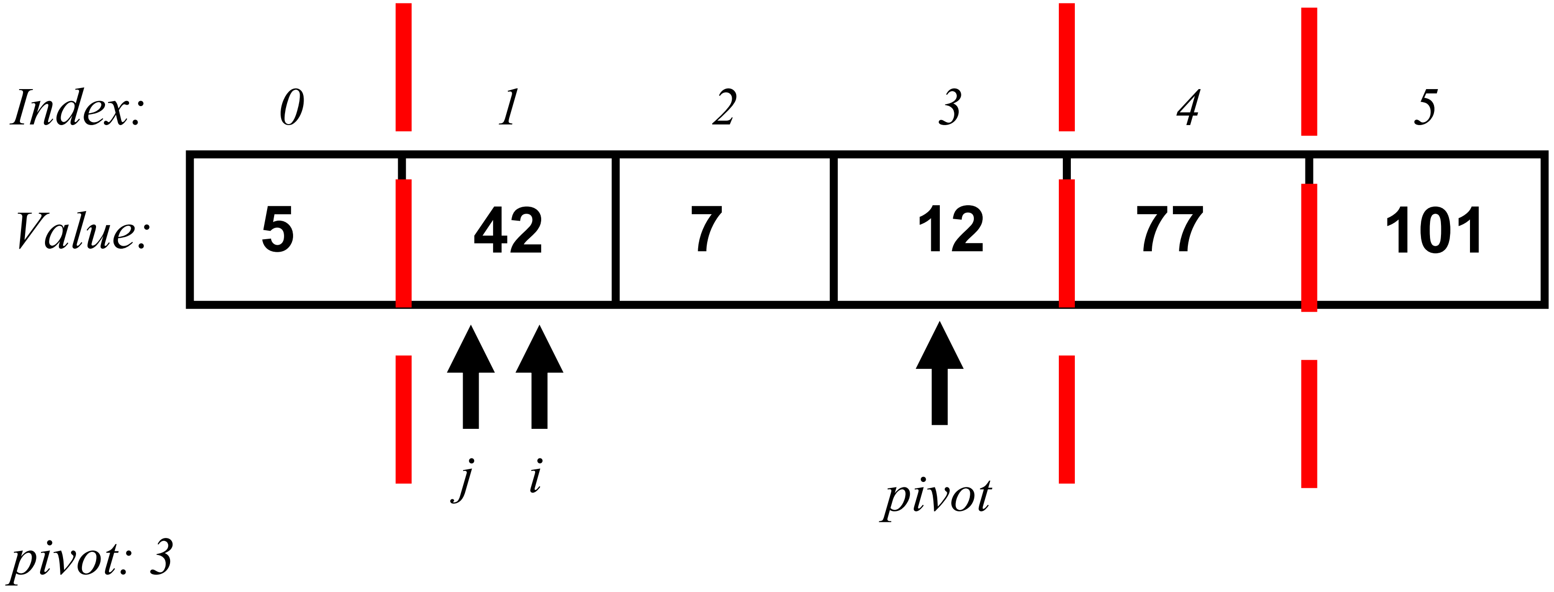
Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**42**	**7**	**12**	**77**	**101**

pivot: 5

Again, the element that was the pivot (77) is now in its final position in the array along with 5

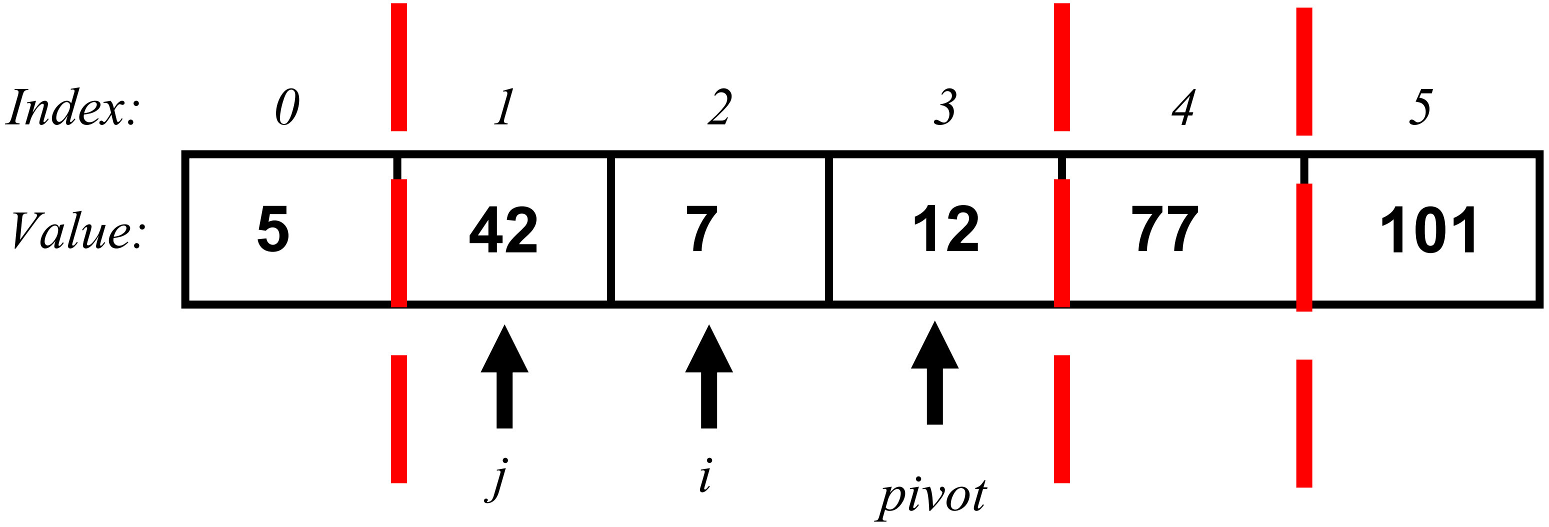
Quick Sort: an example



Starting from the left side list we chose always as pivot the last element.

Remember: it could be any index inside the sub-list!

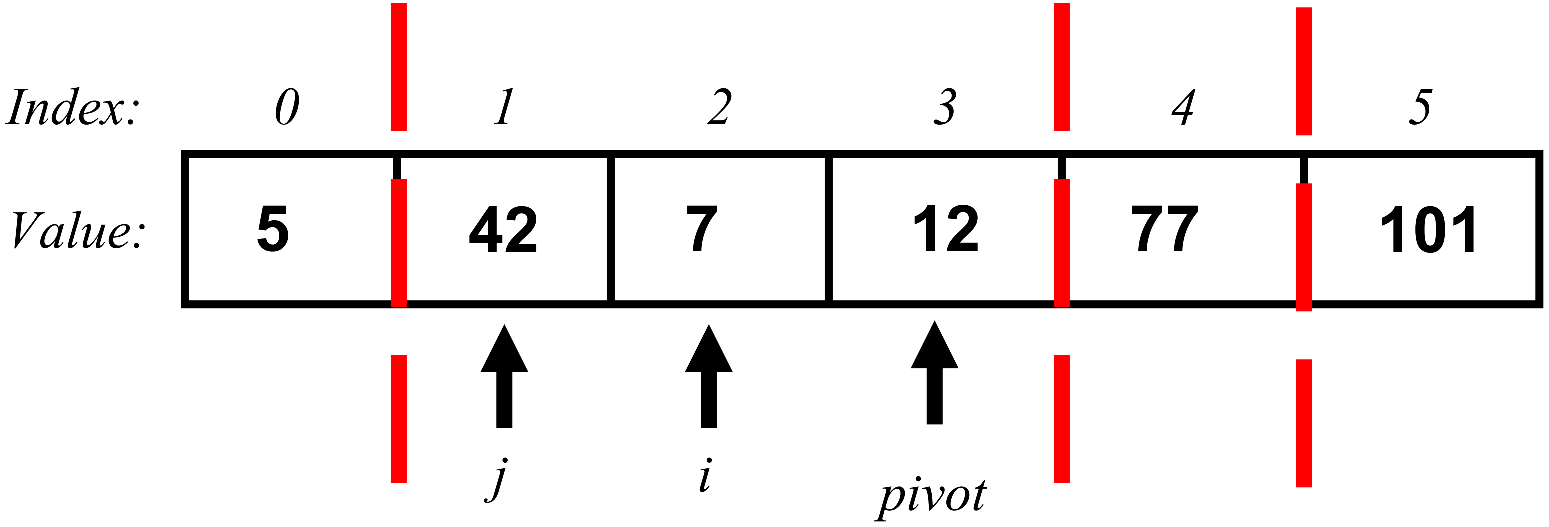
Quick Sort: an example



pivot: 3

Is 42 grater or smaller than 12?
It is grater! So i = 2 j = 1

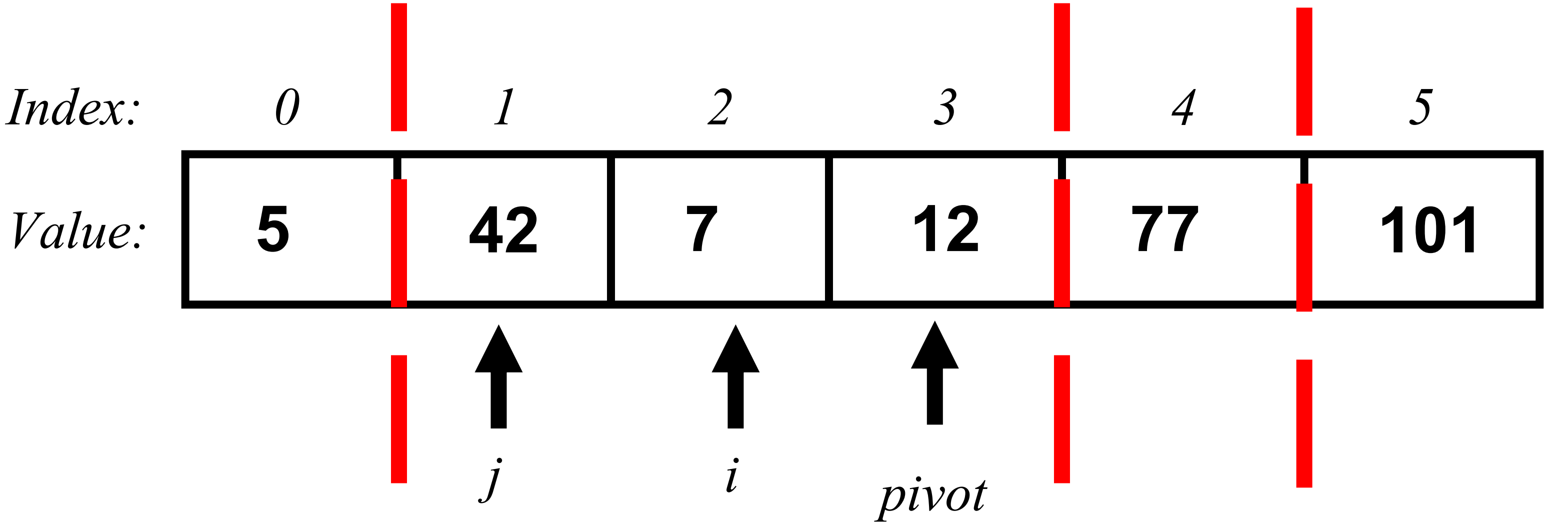
Quick Sort: an example



pivot: 3

Is 7 grater or smaller than 12?
It is smaller!

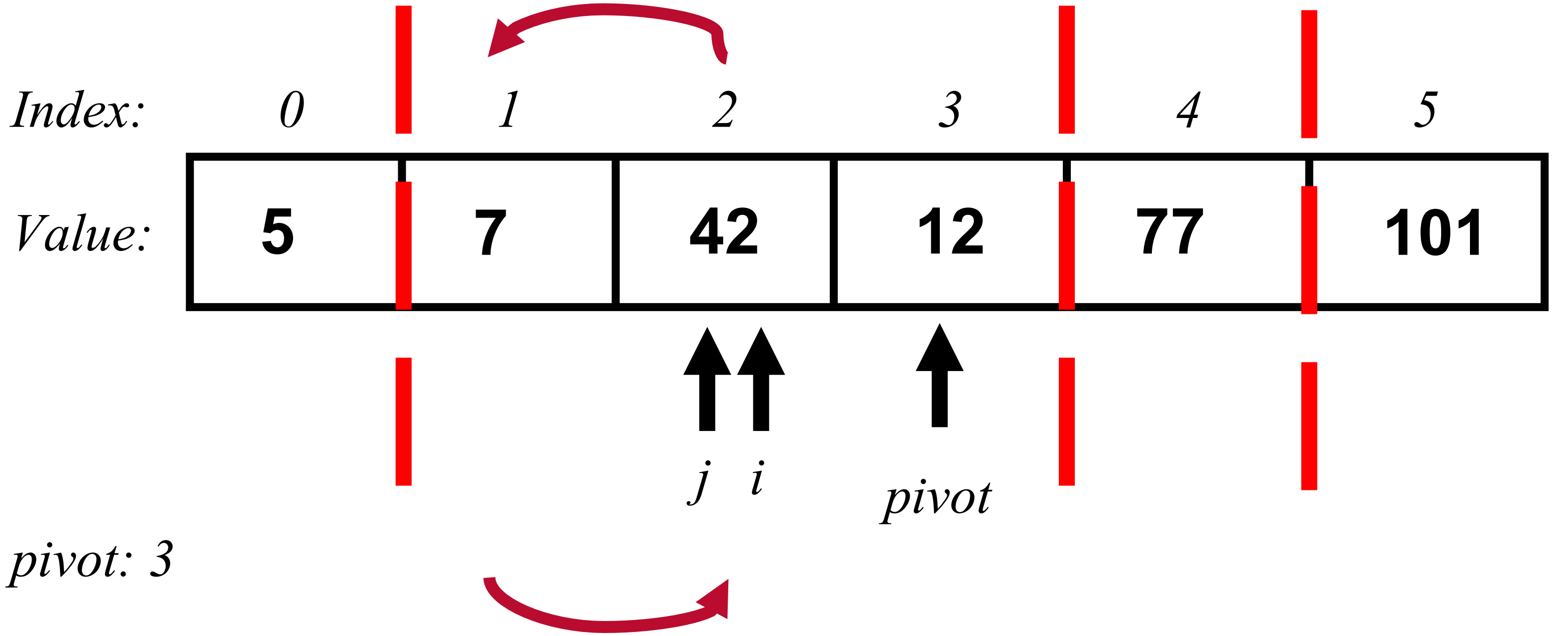
Quick Sort: an example



pivot: 3

So we have to swap 42 and 7 because 42 is larger than 7 and 7 is smaller than 12

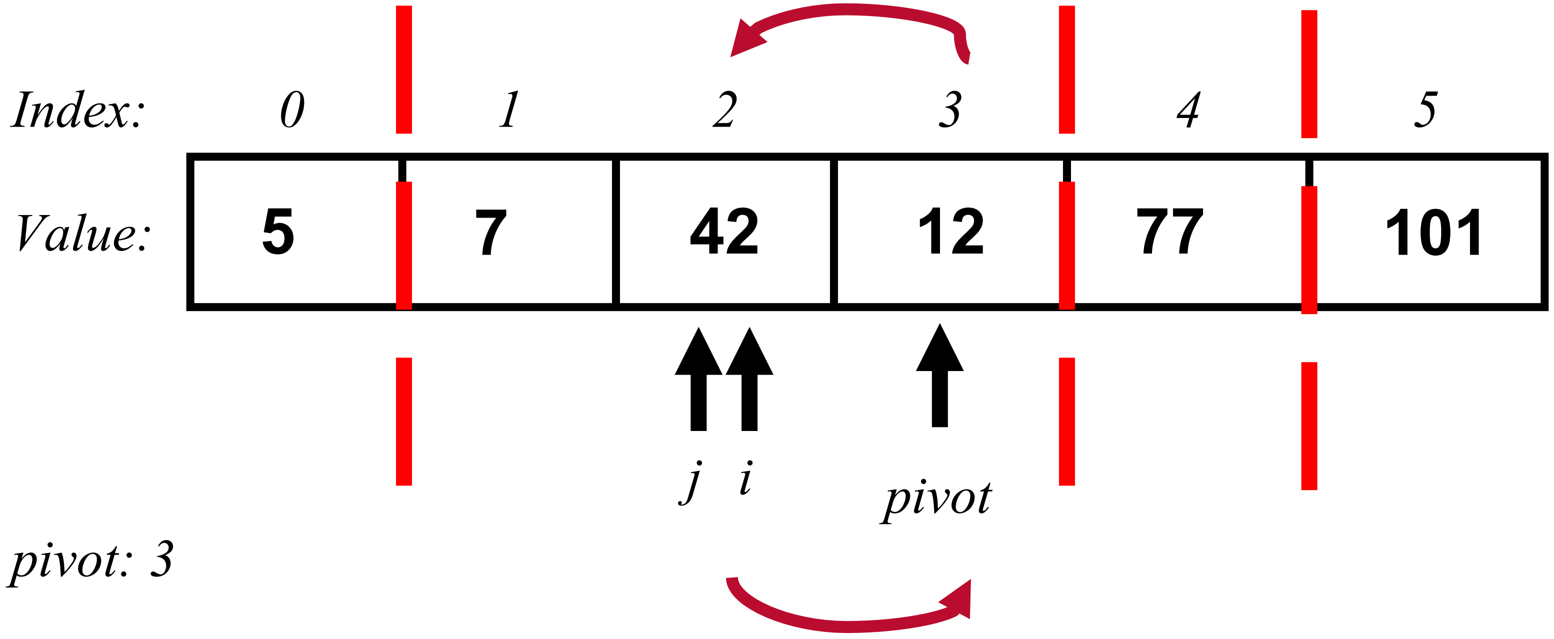
Quick Sort: an example



pivot: 3

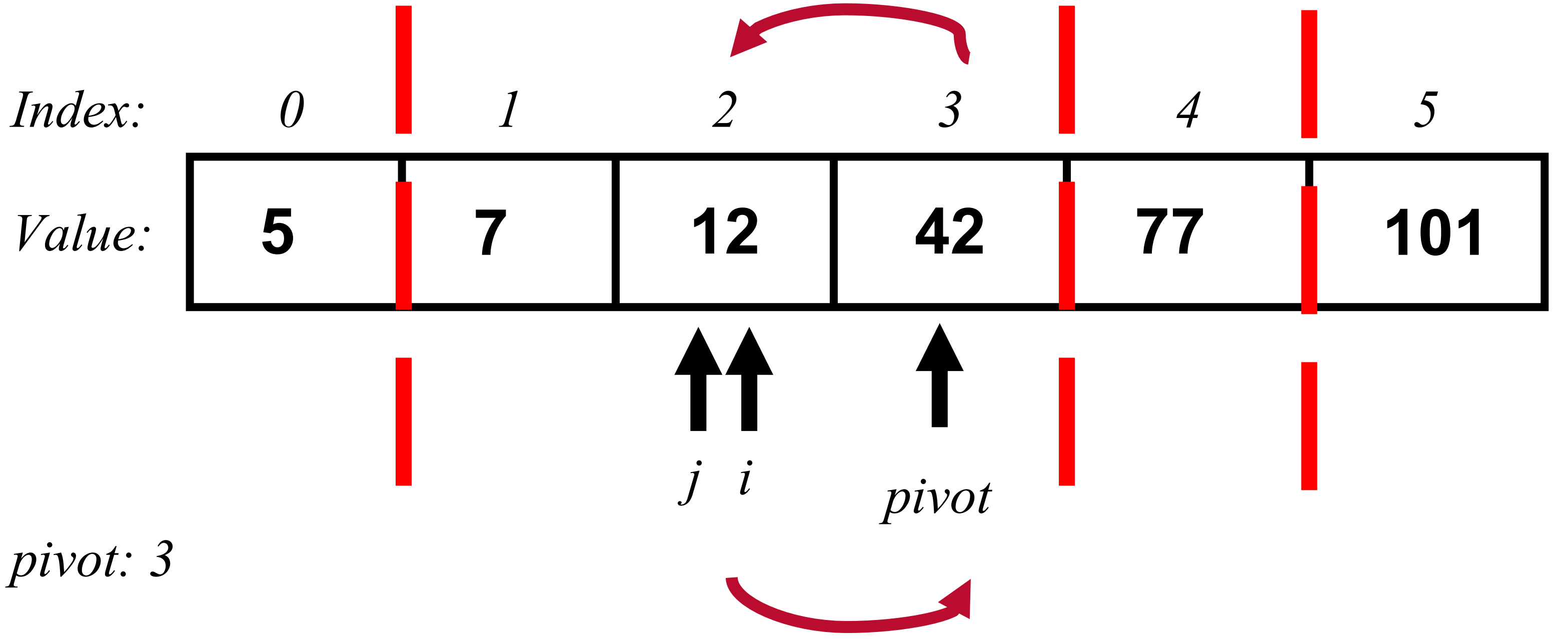
And then we increase the j pointer j = 2

Quick Sort: an example



Since we reached the end of the sub-list so we need to swap the element at position **j** with the **pivot**

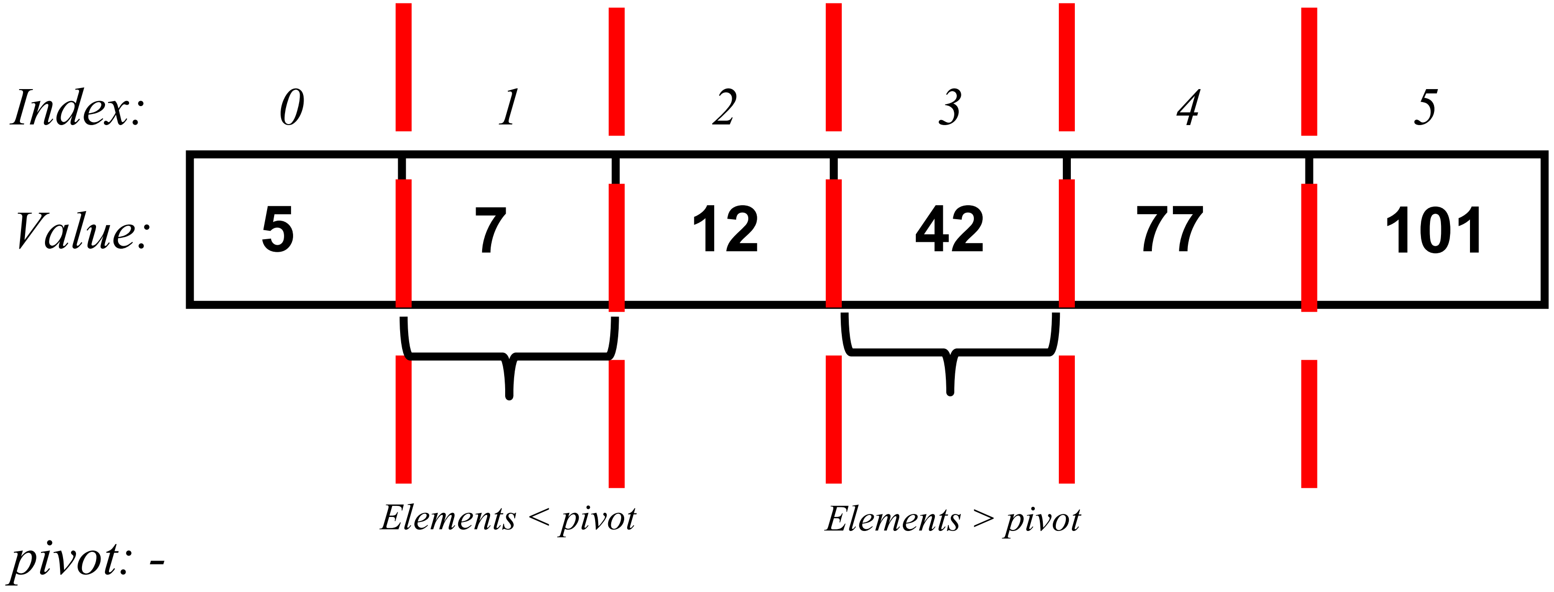
Quick Sort: an example



pivot: 3

Since we reached the end of the sub-list so we need to swap the element at position **j** with the **pivot**

Quick Sort: an example



In both the resulting sub-list we just return the value as it is because they are both a single element list

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**7**	**12**	**42**	**77**	**101**

pivot: 3

Finally, the right side list (101) that we got before is again a single element list and have to be there.

Quick Sort: an example

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**7**	**12**	**42**	**77**	**101**

pivot: 3

The array is finally sorted!

Quick Sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**7**	**12**	**42**	**77**	**101**

Two Questions:

1) **What is the main factor that influences the number of steps we have to do?**

Quick Sort

Index:	0	1	2	3	4	5
Value:	5	7	12	42	77	101

Two Questions:

1) **What is the main factor that influences the number of steps we have to do?**
2) **In the worst case how many steps we have to do to sort the list?**

Quick Sort: clever ways to chose the pivot

The pivot has a huge impact on the performances!

Quick Sort: clever ways to chose the pivot

The pivot has a huge impact on the performances!
We can find a way to chose it wisely!
Our goal is to ideally find a pivot that can split in half the list each time!

Quick Sort: clever ways to chose the pivot

The pivot has a huge impact on the performances!
We can find a way to chose it wisely!
Our goal is to ideally find a pivot that can split in half the list each time!
Why?

Quick Sort: clever ways to chose the pivot

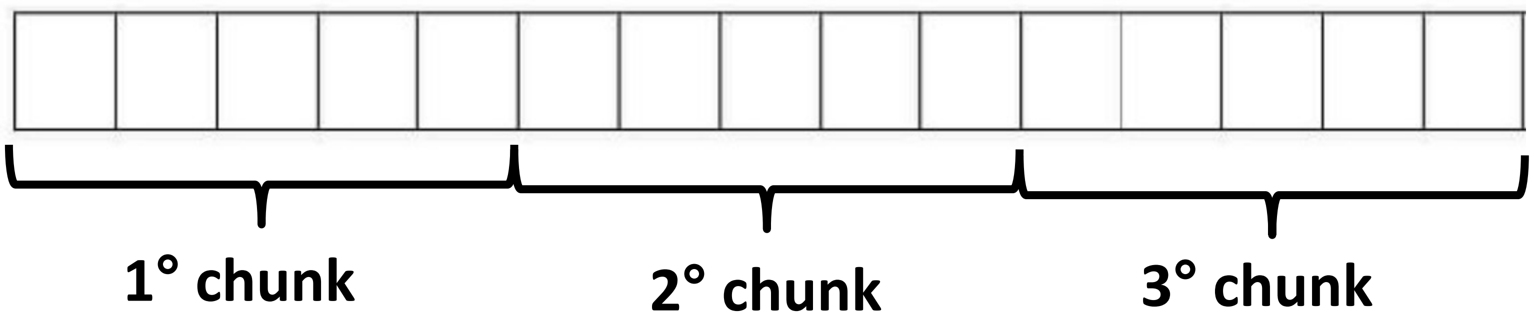
Splitting in half the array each time give us an advantage from a computational perspective!

Quick Sort: clever ways to chose the pivot

In fact the overall complexity in that case would be
$O(n \log n)$

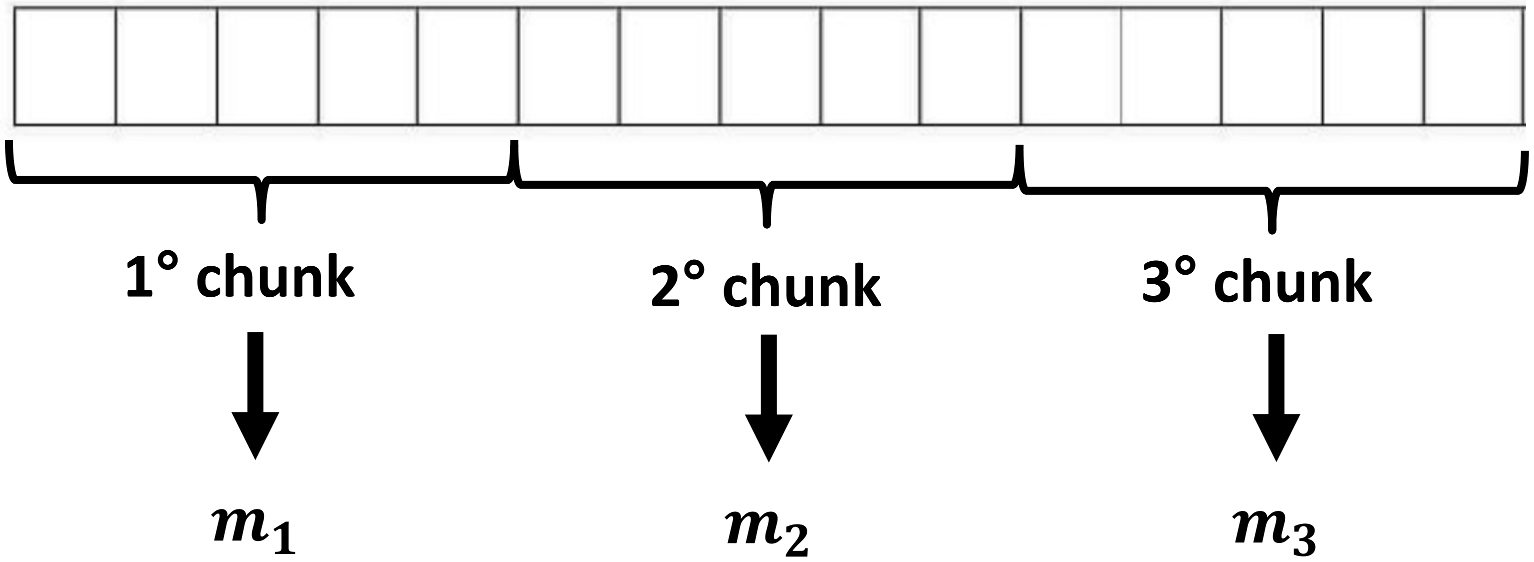
Quick Sort: clever ways to chose the pivot

Given a **list** of **n** elements we can **partition** the list in **chunks** containing 5 elements



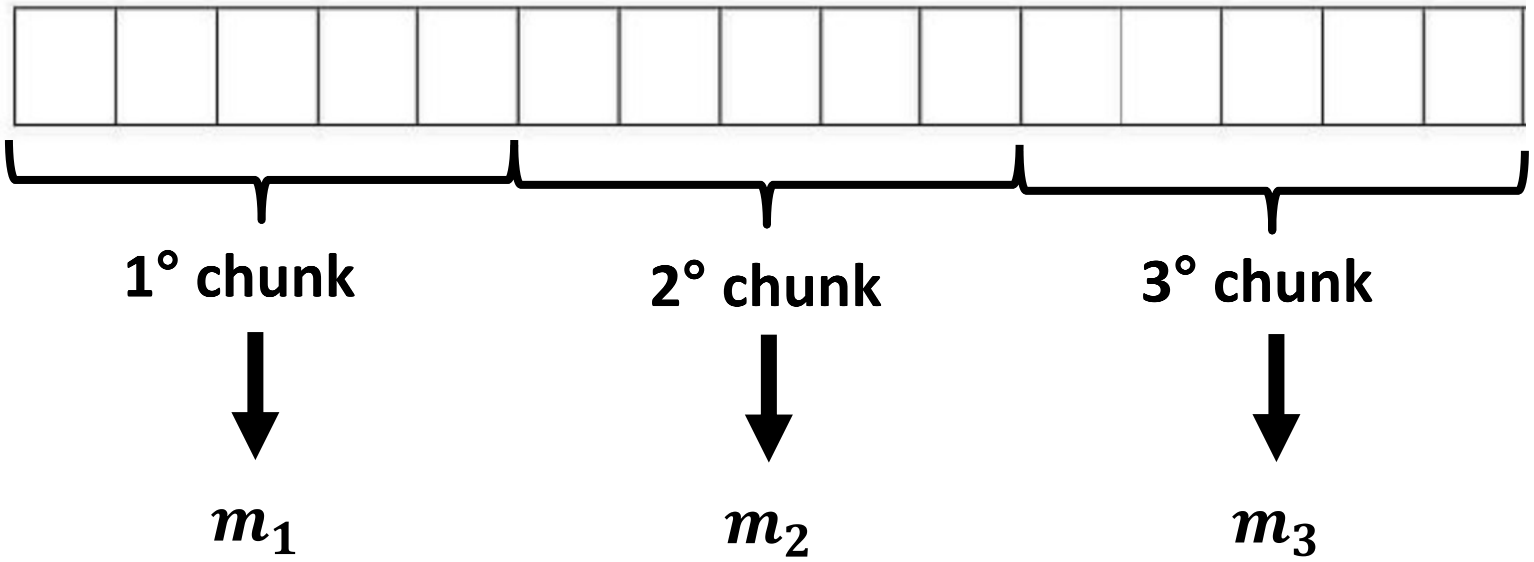
Quick Sort: clever ways to chose the pivot

Then for **each** one of these **chunks** we **compute** the **median** values and we call them m_1, m_2, m_3



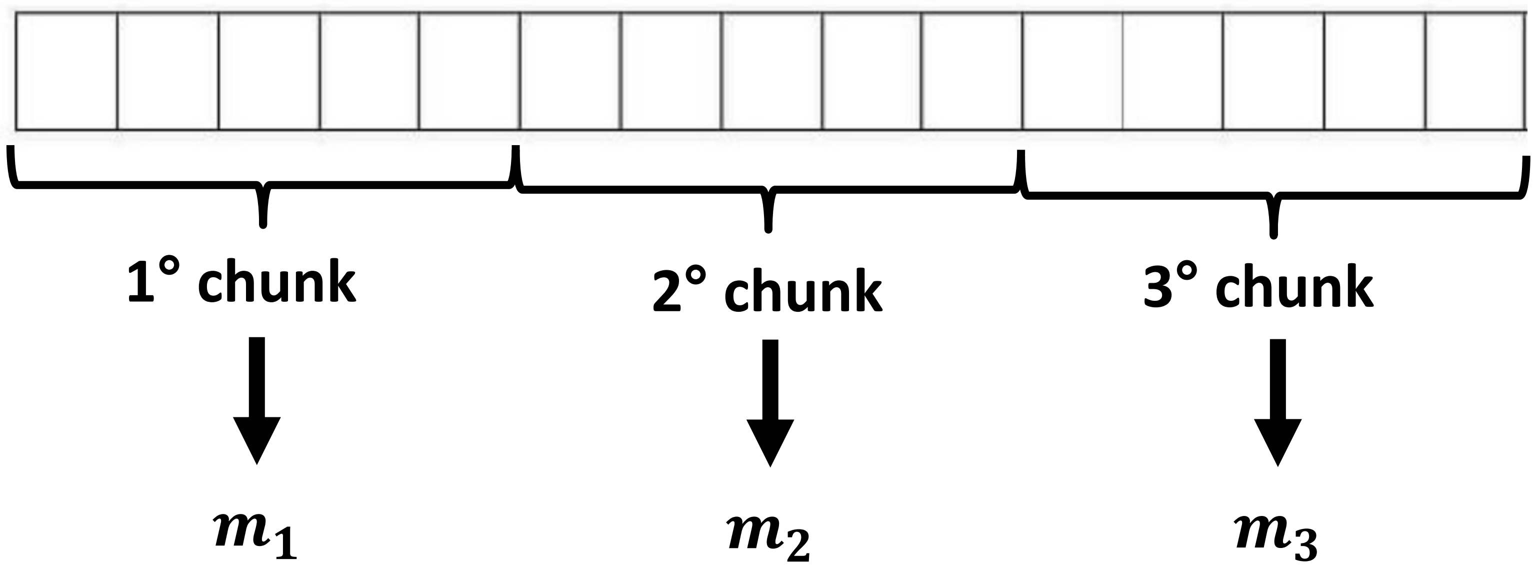
Quick Sort: clever ways to chose the pivot

Once we have m_1, m_2, m_3 we can **compute again the median value among these values and we take the median as pivot**



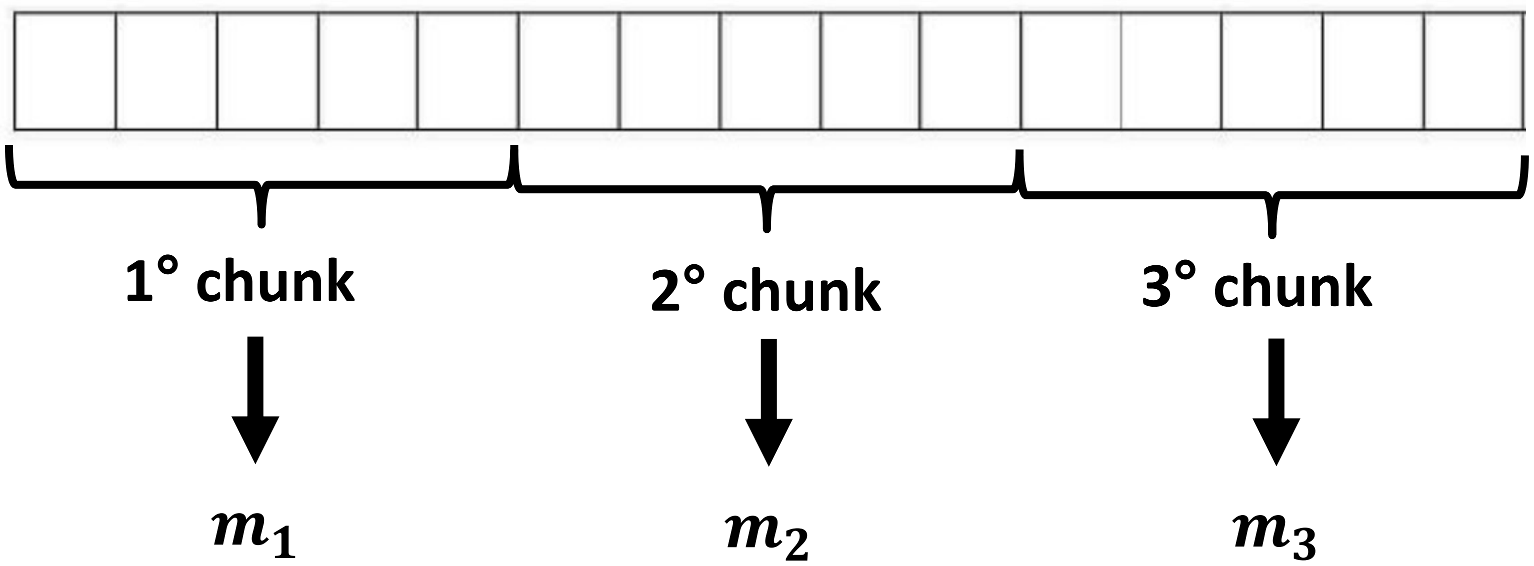
Quick Sort: clever ways to chose the pivot

Doing so quick sort complexity becomes $O(n \log n)$
But why?



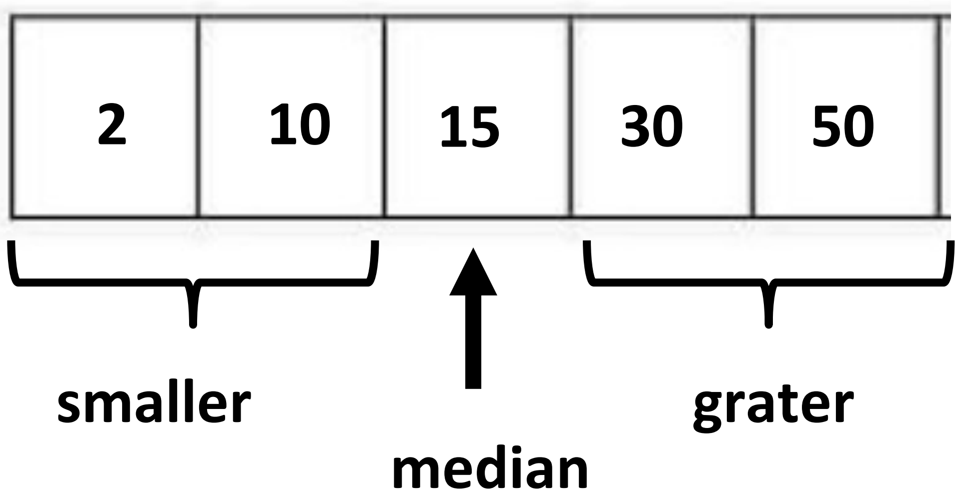
Quick Sort: clever ways to chose the pivot

Doing so quick sort complexity becomes $O(n \log n)$
But why?



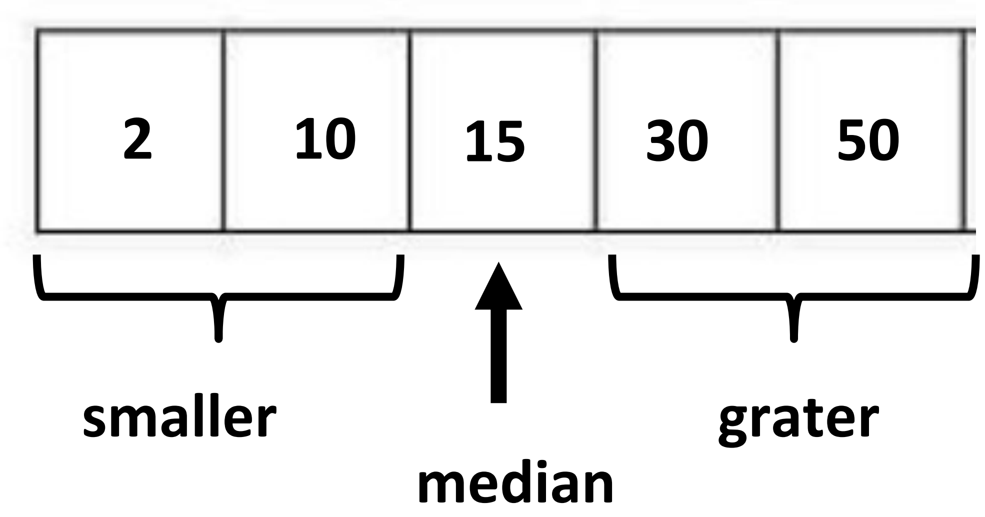
Quick Sort: clever ways to chose the pivot

The median value is the value that separate in half the distribution



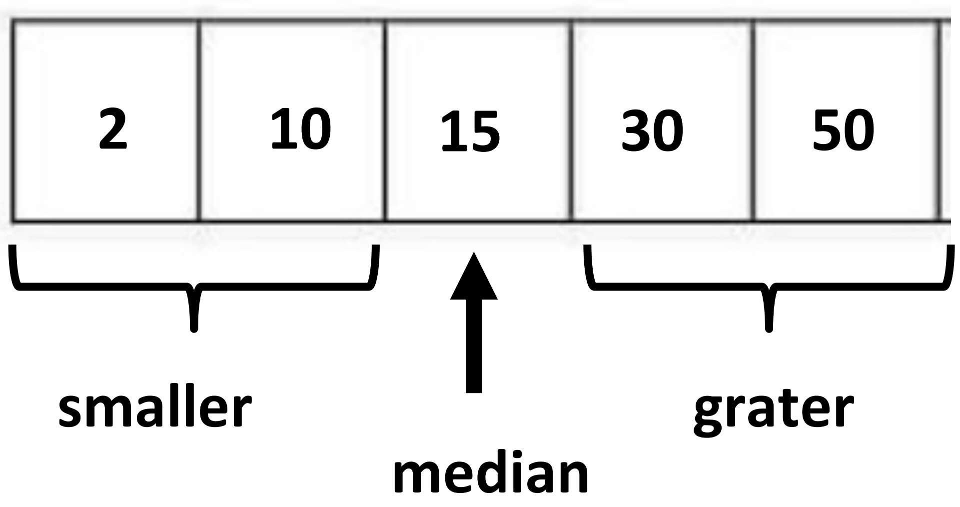
Quick Sort: clever ways to chose the pivot

If we partition a list in chunks with 5 elements and we take the median values we get $\frac{n}{5}$ elements: all the medians



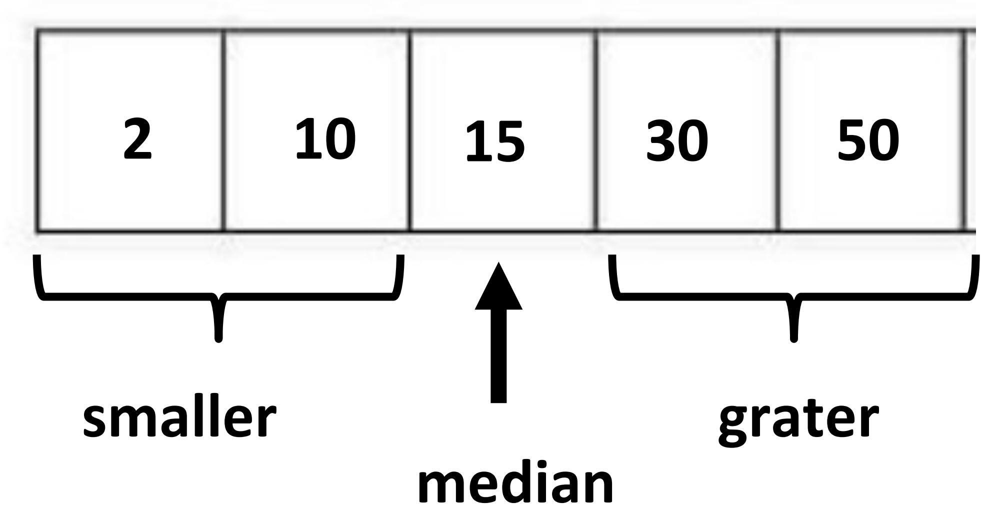
Quick Sort: clever ways to chose the pivot

If we select again the median of the medians, we can conclude that the chosen value is larger than half of the median values, and smaller of the other half.



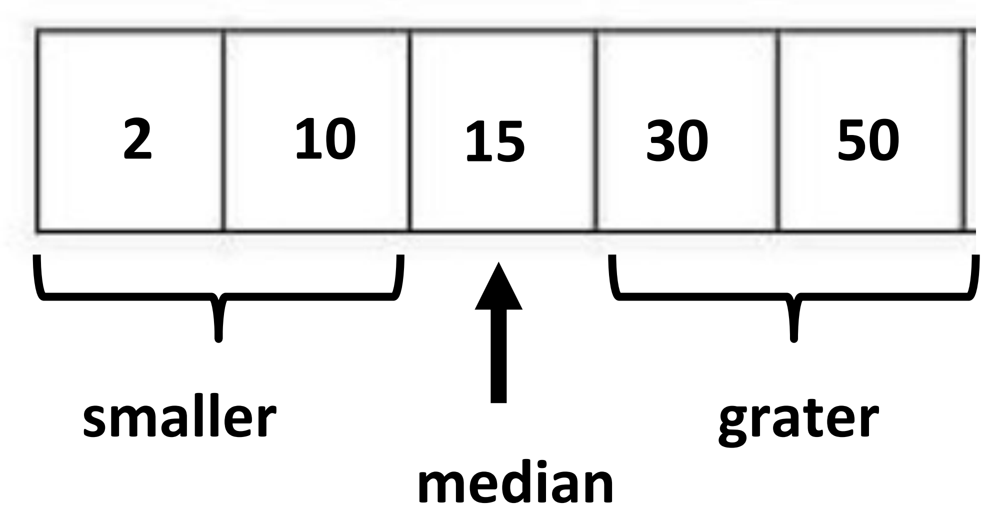
Quick Sort: clever ways to chose the pivot

But this in turn means that it is also larger than (at least) half the elements contained in the chunks with smaller medians



Quick Sort: clever ways to chose the pivot

And it is smaller than (at least) half the elements contained in the chunks with smaller medians



Quick Sort: clever ways to chose the pivot

Doing so we can halve (approximately) each time the size of the input obtaining in this way a complexity of $O(n \log n)$