Advanced Sorting Algorithms (Quick Sort)

Overview

In this topic we will look at quicksort:

- The idea behind the algorithm
- The run time and worst-case scenario
- Strategy for avoiding the worst-case: median-of-three
- Implementing quicksort in place
- Examples

Strategy

We have seen $\Theta(n \ln(n))$ sorting algorithms:

- Merge sort which is faster but requires more memory

We will now look at a recursive algorithm which may be done *almost* in place as well as faster.

- Use an object in the array (a pivot) to divide the two
- Average case: $\Theta(n \ln(n))$ time and $\Theta(\ln(n))$ memory
- Worst case: $\Theta(n^2)$ time and $\Theta(n)$ memory

We will look at strategies for avoiding the worst case

Quicksort

Merge sort splits the array sub-lists and sorts them

The larger problem is split into two sub-problems based on *location* in the array

Consider the following alternative:

- Chose an object in the array and partition the remaining objects into two groups relative to the chosen entry

Quicksort

For example, given

80	38	95	84	66	10	79	**44**	26	87	96	12	43	81	3

we can select the middle entry, 44, and sort the remaining entries into two groups, those less than 44 and those greater than 44:

38	10	26	12	43	3	**44**	80	95	84	66	79	87	96	81

Notice that 44 is now in the correct location if the list was sorted

- Proceed by applying the algorithm to the first six and last eight entries

Run-time analysis

Like merge sort, we can either:

- – Apply insertion sort if the size of the sub-list is sufficiently small, or
- – Sort the sub-lists using quicksort

In the best case, the list will be split into two approximately equal sub-lists, and thus, the run time could be very similar to that of merge sort: $\Theta(n \ln(n))$

What happens if we don't get that lucky?

Worst-case scenario

Suppose we choose the first element as our pivot and we try ordering a sorted list:

80	38	95	84	66	10	79	**2**	26	87	96	12	43	81	3

Using 2, we partition into

2	80	38	95	84	66	10	79	26	87	96	12	43	81	3

We still have to sort a list of size $n - 1$

– Thus, the run time drops from $n \ln(n)$ to n^2

Worst-case scenario

Our goal is to choose the median element in the list as our pivot:

80	38	95	84	**66**	10	79	2	26	87	96	12	43	81	3

Unfortunately, it's difficult to find

Alternate strategy: take the median of a subset of entries

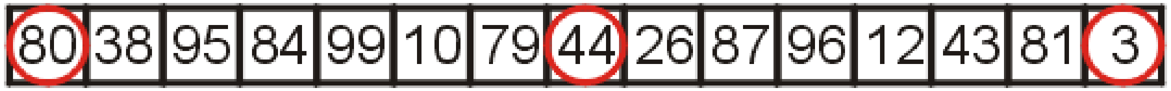
- For example, take the median of the first, middle, and last entries

Median-0f-three

It is difficult to find the median so consider another strategy:

- Choose the median of the first, middle, and last entries in the list

This will usually give a better approximation of the actual median

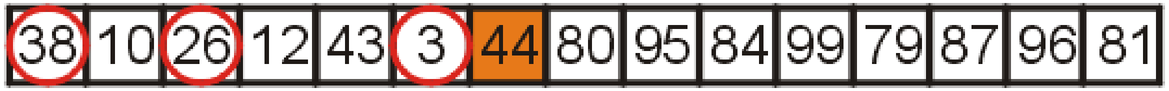


80	38	95	84	99	10	79	44	26	87	96	12	43	81	3

Median-of-three

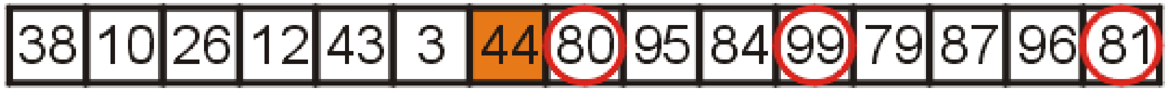
Sorting the elements based on 44 results in two sub-lists, each of which must be sorted (again, using quicksort)

Select the 26 to partition the first sub-list:



38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

Select 81 to partition the second sub-list:



38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

Implementation

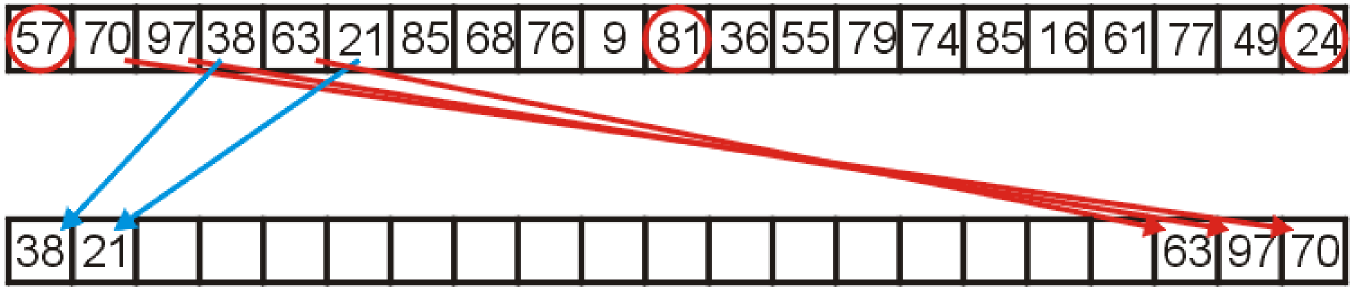
If we choose to allocate memory for an additional array, we can implement the partitioning by copying elements either to the front or the back of the additional array

Finally, we would place the pivot into the resulting hole

Implementation

For example, consider the following:

- 57 is the median-of-three
- we go through the remaining elements, assigning them either to the front or the back of the second array

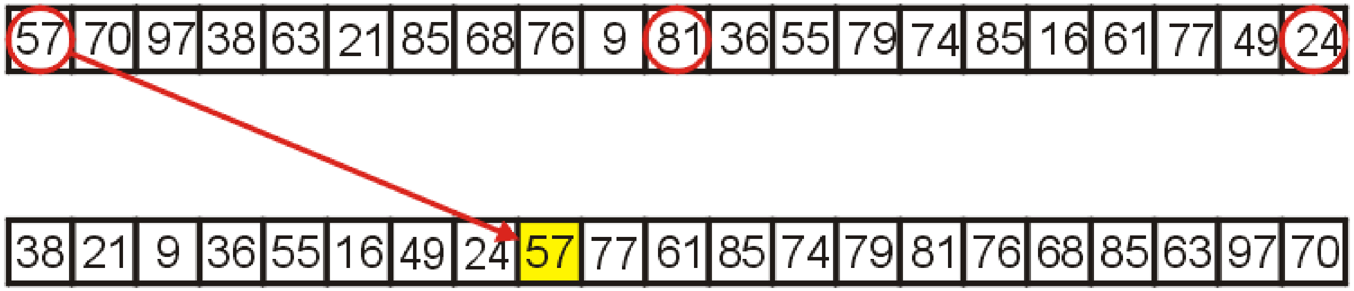


57	70	97	38	63	21	85	68	76	9	81	36	55	79	74	85	16	61	77	49	24

38	21																	63	97	70

Implementation

Once we are finished, we copy the median-of-three, 57, into the resulting hole



57	70	97	38	63	21	85	68	76	9	81	36	55	79	74	85	16	61	77	49	24

38	21	9	36	55	16	49	24	57	77	61	85	74	79	81	76	68	85	63	97	70

Implementation

Note, however, we can do a better job with merge sort, it always divides the numbers being sorted into two equal or near-equal arrays

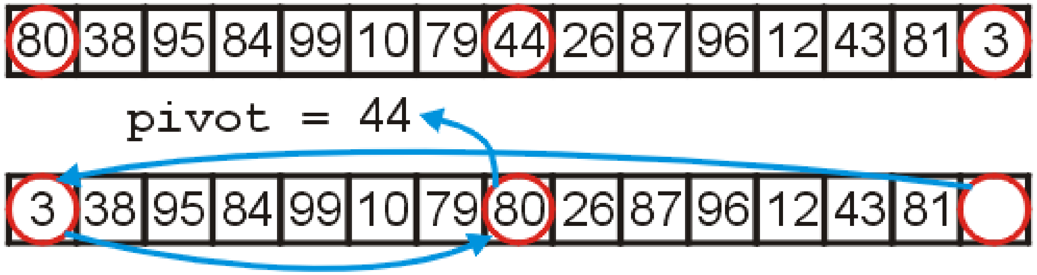
Can we implement quicksort in place?

Implementation

First, we have already examined the first, middle, and last entries and chosen the median of these to be the pivot

In addition, we can:

- move the smallest entry to the first entry
- move the largest entry to the middle entry



Implementation

Next, recall that our goal is to partition all remaining elements based on whether they are smaller than or greater than the pivot

We will find two entries:

- One larger than the pivot (staring from the front)
- One smaller than the pivot (starting from the back)

which are out of order and then swap them

Implementation

Continue doing so until the appropriate entries you find are actually in order

The index to the larger entry we found would be the first large entry in the list (as seen from the left)

Therefore, we could move this entry into the last entry of the list

We can fill this spot with the pivot

Implementation

The implementation is straight-forward

```
template <typename Type>
void quicksort( Type *array, int first, int last ) {
    if ( last - first <= N ) {
        insertion_sort( array, first, last );
    } else {
        Type pivot = find_pivot( array, first, last );
        int low  =     find_next( pivot, array, first + 1 );
        int high = find_previous( pivot, array,  last - 2 );

        while ( low < high ) {
            std::swap( array[low], array[high] );
            low  =     find_next( pivot, array,  low + 1 );
            high = find_previous( pivot, array, high - 1 );
        }

        array[last – 1] = array[low];
        array[low] = pivot;
        quicksort( array, first,  low );
        quicksort( array,  high, last );
    }
}
```

Quicksort Example

Consider the following unsorted array of 25 entries

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

We will call insertion sort if the list being sorted of size $N = 6$ or less

Quicksort Example

We call `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	57	99	17	32	94	28	15	55	7	51	88	97	62

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	**57**	99	17	32	94	28	15	55	7	51	88	97	**62**

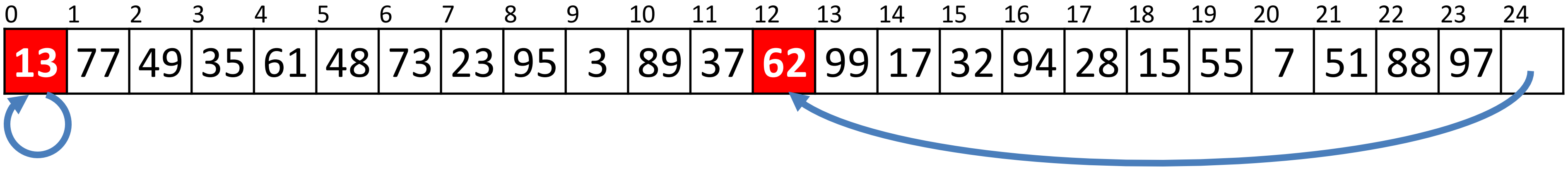
First, $25 - 0 > 6$, so find the midpoint and the pivot

```
midpoint = (0 + 25)/2; // == 12
```

`quicksort( array,  0, 25 )`

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	**62**	99	17	32	94	28	15	55	7	51	88	97	

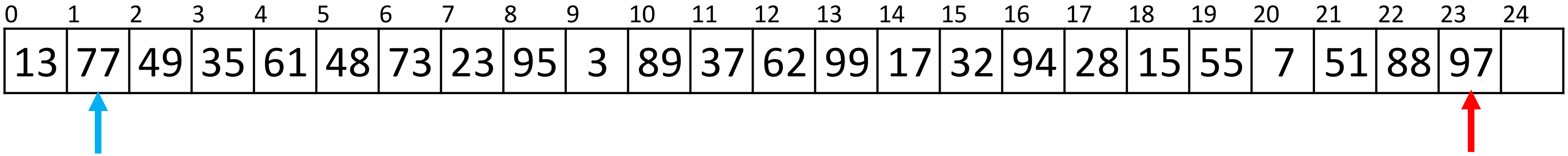
First, $25 - 0 > 6$, so find the midpoint and the pivot

```
midpoint = (0 + 25)/2; // == 12
pivot = 57;
```

`quicksort( array,  0, 25 )`

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	77	49	35	61	48	73	23	95	3	89	37	62	99	17	32	94	28	15	55	7	51	88	97	

Starting from the front and back:

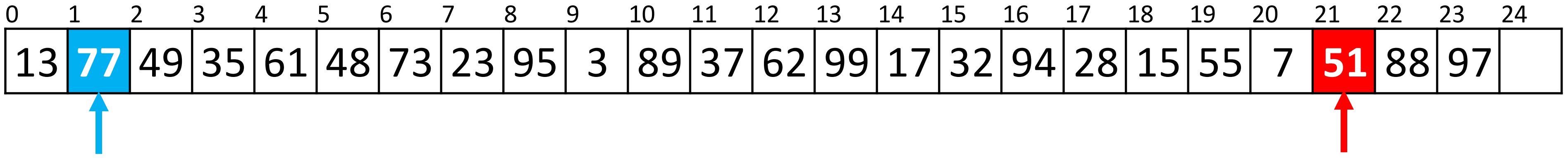
- Find the next element greater than the pivot
- The last element less than the pivot

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	**77**	49	35	61	48	73	23	95	3	89	37	62	99	17	32	94	28	15	55	7	**51**	88	97	

Searching forward and backward:

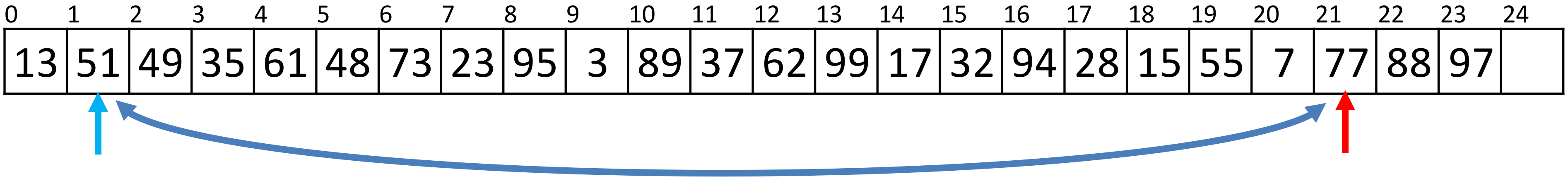
```
low = 1;
high = 21;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	61	48	73	23	95	3	89	37	62	99	17	32	94	28	15	55	7	77	88	97	

Searching forward and backward:

```
low = 1;
high = 21;
```

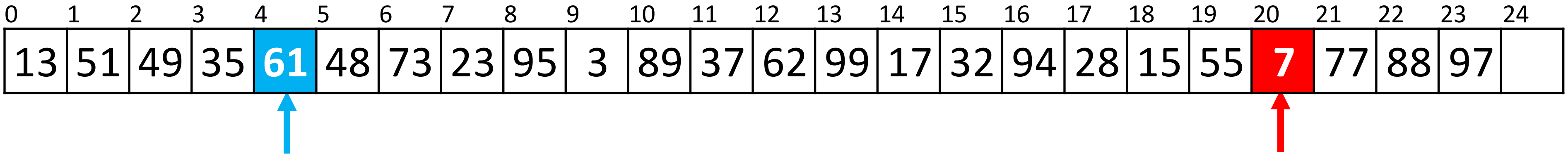
Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	**61**	48	73	23	95	3	89	37	62	99	17	32	94	28	15	55	**7**	77	88	97	

Continue searching

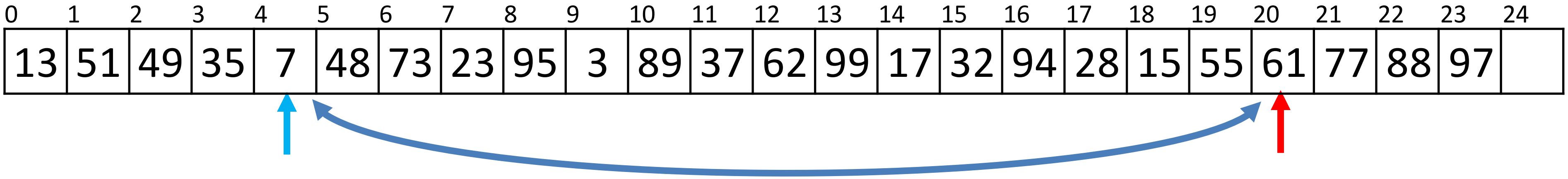
```
low = 4;
high = 20;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	73	23	95	3	89	37	62	99	17	32	94	28	15	55	61	77	88	97	

Continue searching

```
low = 4;
high = 20;
```

Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	**73**	23	95	3	89	37	62	99	17	32	94	28	15	**55**	61	77	88	97	

Continue searching

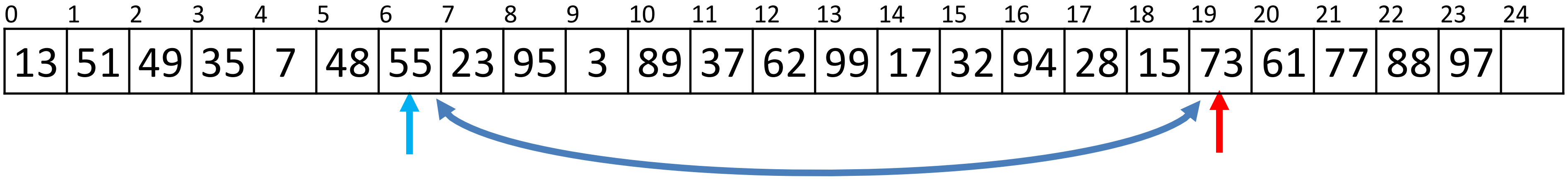
```
low = 6;
high = 19;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	95	3	89	37	62	99	17	32	94	28	15	73	61	77	88	97	

Continue searching

```
low = 6;
high = 19;
```

Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	**95**	3	89	37	62	99	17	32	94	28	**15**	73	61	77	88	97	

Continue searching

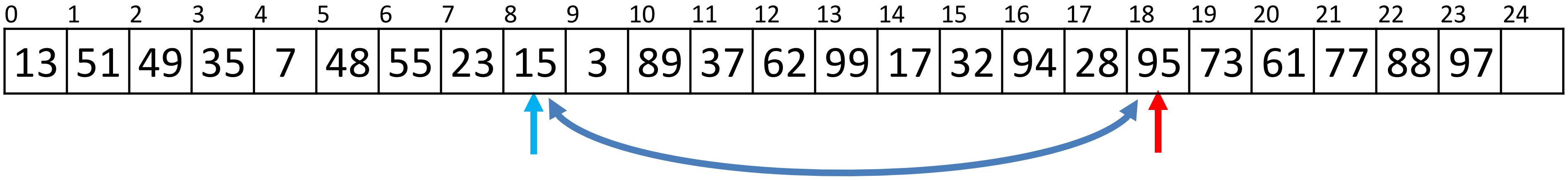
```
    low = 8;
    high = 18;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	89	37	62	99	17	32	94	28	95	73	61	77	88	97	

Continue searching

```
low = 8;
high = 18;
```

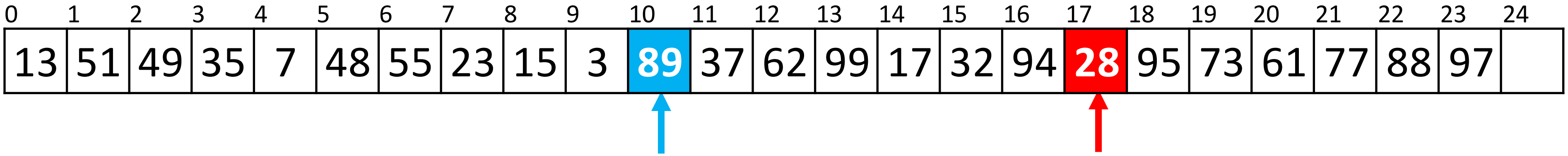
Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	**89**	37	62	99	17	32	94	**28**	95	73	61	77	88	97	

Continue searching

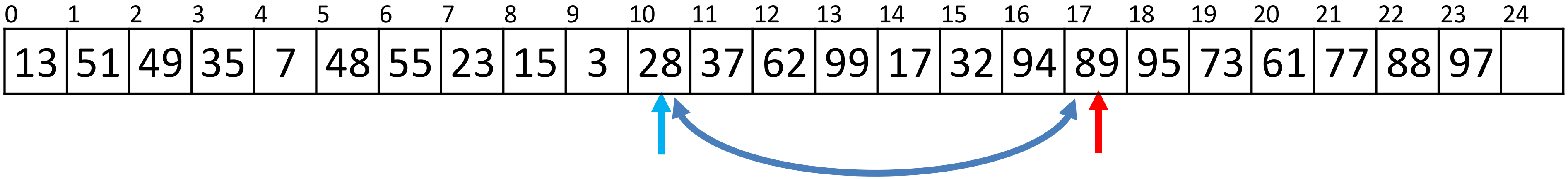
```
low = 10;
high = 17;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	62	99	17	32	94	89	95	73	61	77	88	97	

Continue searching

```
    low = 10;
    high = 17;
```

Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	62	99	17	32	94	89	95	73	61	77	88	97	

Continue searching

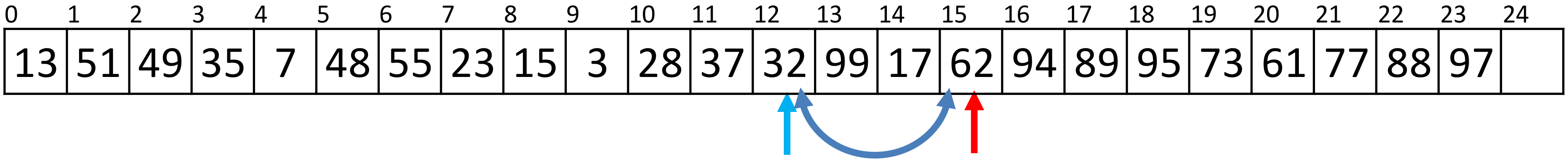
```
low = 12;
high = 15;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	99	17	62	94	89	95	73	61	77	88	97	

Continue searching

```
low = 12;
high = 15;
```

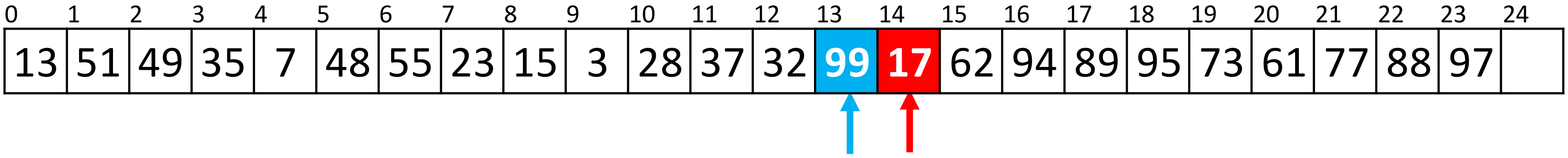
Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	99	17	62	94	89	95	73	61	77	88	97	

Continue searching

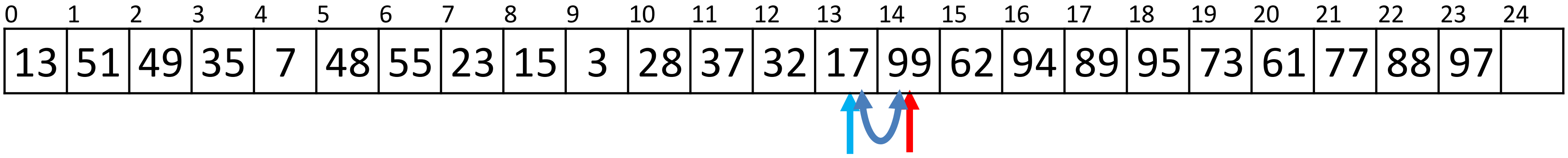
```
low = 13;
high = 14;
```

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	17	99	62	94	89	95	73	61	77	88	97	

Continue searching

```
    low = 13;
    high = 14;
```

Swap them

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	**17**	**99**	62	94	89	95	73	61	77	88	97	

Continue searching

```
    low = 14;
    high = 13;
```

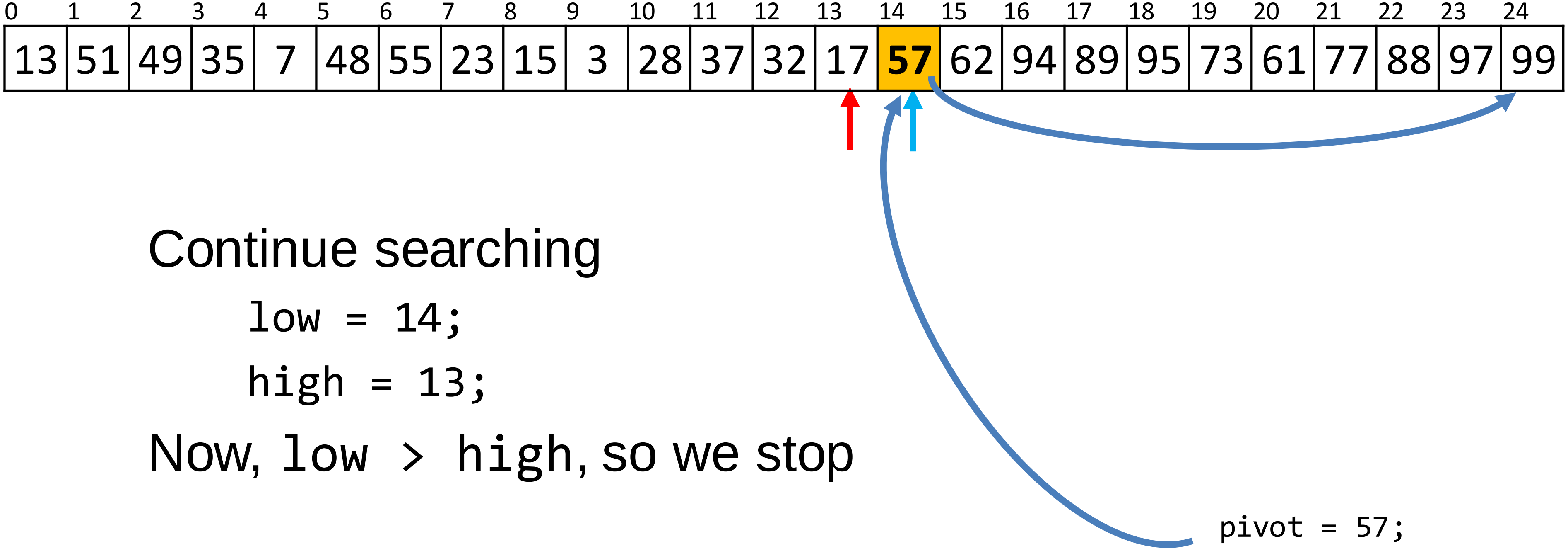
Now, `low > high`, so we stop

```
pivot = 57;
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are calling `quicksort( array, 0, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	17	**57**	62	94	89	95	73	61	77	88	97	99

Continue searching

```
low = 14;
high = 13;
```

Now, `low > high`, so we stop

`pivot = 57;`

`quicksort( array,  0, 25 )`

Quicksort Example

We are calling `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	17	**57**	62	94	89	95	73	61	77	88	97	99

We now begin calling quicksort recursively on the first half

```
    quicksort( array, 0, 14 );
```

```
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32	17	57	62	94	89	95	73	61	77	88	97	99

First, $14 - 0 > 6$, so find the midpoint and the pivot

```
    midpoint = (0 + 14)/2; // == 7
```

```
quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	**23**	15	3	28	37	32	**17**	57	62	94	89	95	73	61	77	88	97	99

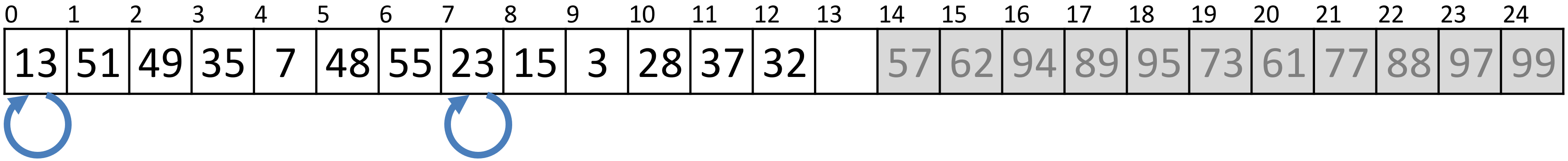
First, $14 - 0 > 6$, so find the midpoint and the pivot

```
midpoint = (0 + 14)/2; // == 7
pivot = 17
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32		57	62	94	89	95	73	61	77	88	97	99

First, $14 - 0 > 6$, so find the midpoint and the pivot

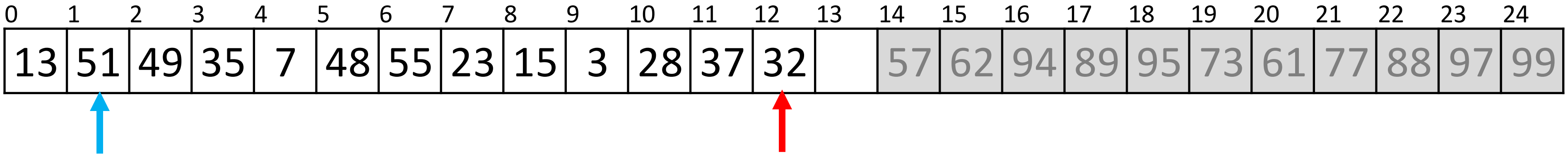
```
midpoint = (0 + 14)/2; // == 7
```

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	51	49	35	7	48	55	23	15	3	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Starting from the front and back:

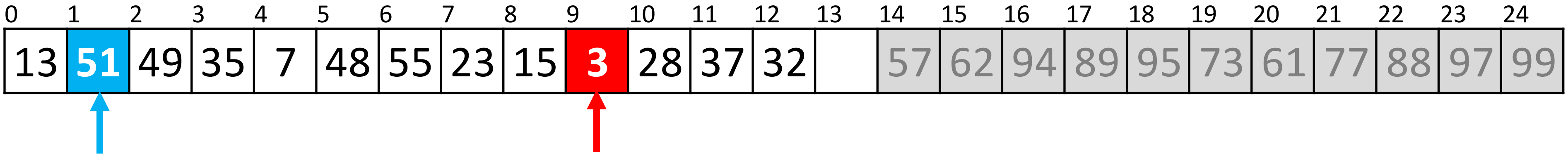
- Find the next element greater than the pivot
- The last element less than the pivot

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	**51**	49	35	7	48	55	23	15	**3**	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

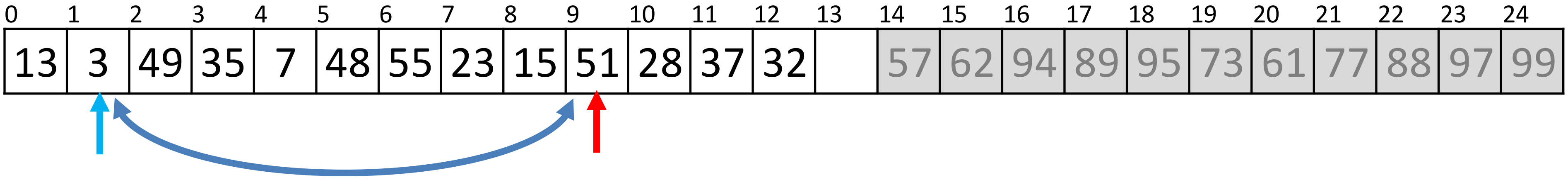
```
low = 1;
high = 9;
```

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	49	35	7	48	55	23	15	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

```
low = 1;
high = 9;
```

Swap them

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	**49**	35	7	48	55	23	**15**	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

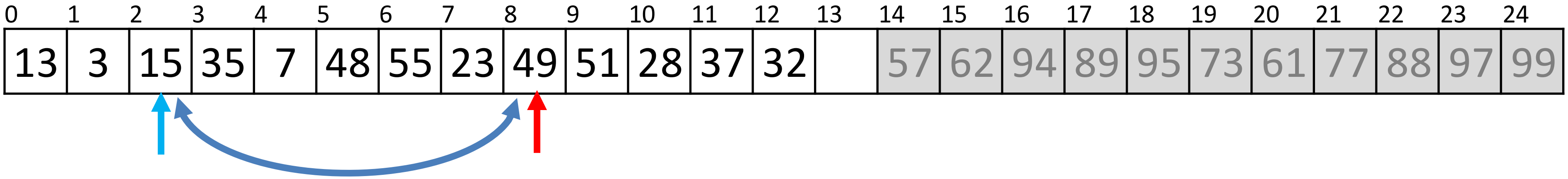
```
low = 2;
high = 8;
```

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	35	7	48	55	23	49	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

```
low = 2;
high = 8;
```

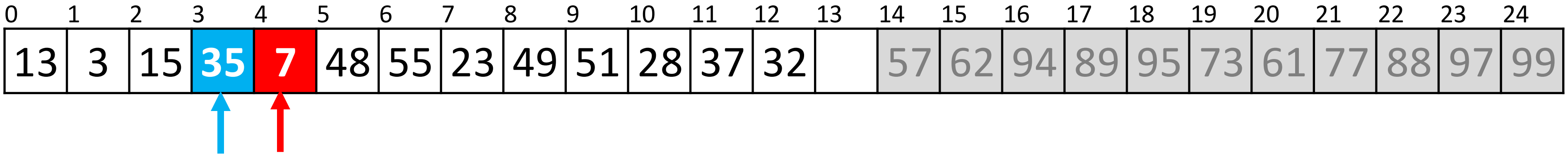
Swap them

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	35	7	48	55	23	49	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

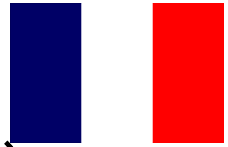
Searching forward and backward:

```
low = 3;
high = 4;
```

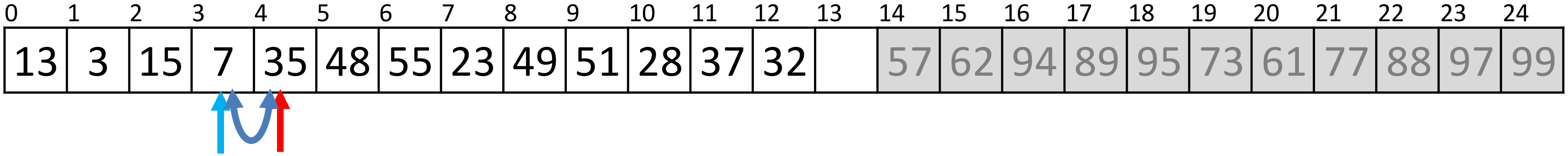
```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example



We are executing `quicksort( array, 0, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	7	35	48	55	23	49	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

```
low = 3;
high = 4;
```

Swap them

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	**7**	**35**	48	55	23	49	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

```
low = 4;
high = 3;
```

Now, `low > high`, so we stop

```
pivot = 17;
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	7	**17**	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

We continue calling quicksort recursively

```
quicksort( array, 0, 4 );
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 0, 4 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	7	17	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

Now, $4 - 0 \le 6$, so find we call insertion sort

```
quicksort( array,  0,  4 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from `0` to `3`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	3	15	7	17	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

```
insertion_sort( array, 0, 4 )

quicksort( array,  0,  4 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from `0` to `3`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

– This function call completes and so we exit

```
insertion_sort( array, 0, 4 )

quicksort( array,  0,  4 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

```
quicksort( array,  0,  4 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 0, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	**17**	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

We continue calling quicksort recursively on the second half

```
quicksort( array, 0,  4 );
quicksort( array, 5, 14 );
```

```
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	48	55	23	49	51	28	37	32	35	57	62	94	89	95	73	61	77	88	97	99

First, $14 - 5 > 6$, so find the midpoint and the pivot

```
    midpoint = (5 + 14)/2; // == 9
```

```
quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	**48**	55	23	49	**51**	28	37	32	**35**	57	62	94	89	95	73	61	77	88	97	99

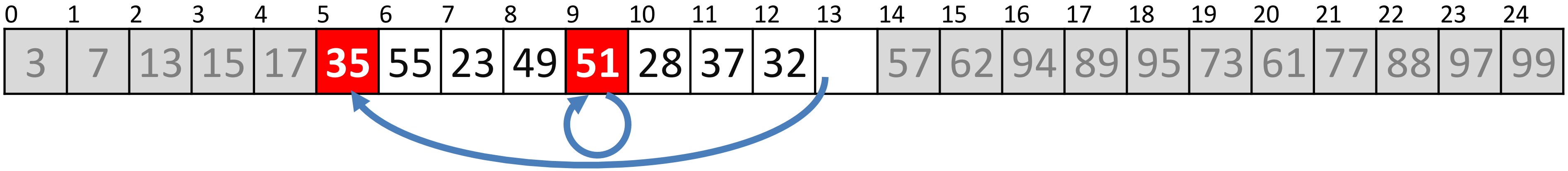
First, $14 - 5 > 6$, so find the midpoint and the pivot

```
midpoint = (5 + 14)/2; // == 9
pivot = 48
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	**35**	55	23	49	**51**	28	37	32		57	62	94	89	95	73	61	77	88	97	99

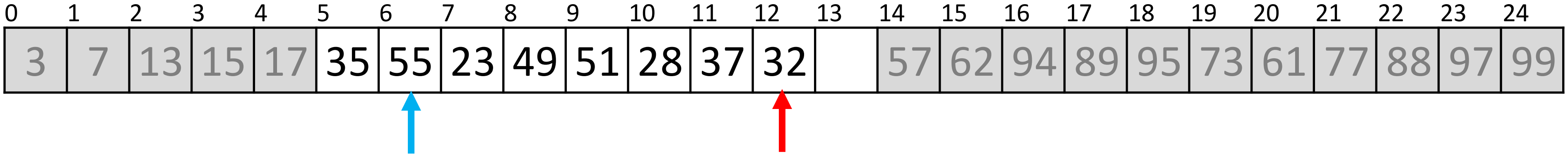
First, $14 - 5 > 6$, so find the midpoint and the pivot

```
midpoint = (5 + 14)/2; // == 9
pivot = 48
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	55	23	49	51	28	37	32		57	62	94	89	95	73	61	77	88	97	99

Starting from the front and back:

- Find the next element greater than the pivot
- The last element less than the pivot

```
pivot = 48;
```

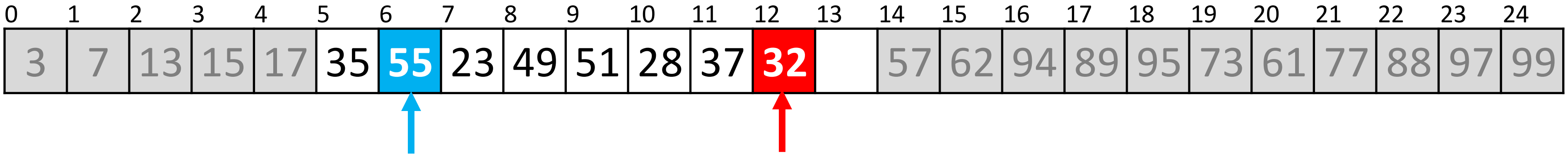
```
quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	**55**	23	49	51	28	37	**32**		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

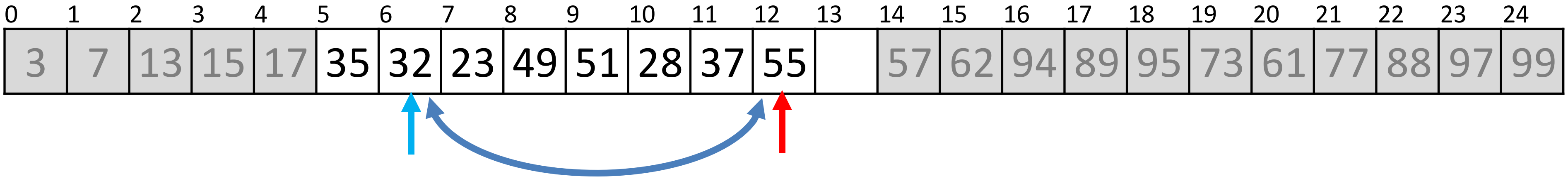
```
low = 6;
high = 12;
```

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	49	51	28	37	55		57	62	94	89	95	73	61	77	88	97	99

Searching forward and backward:

```
low = 6;
high = 12;
```

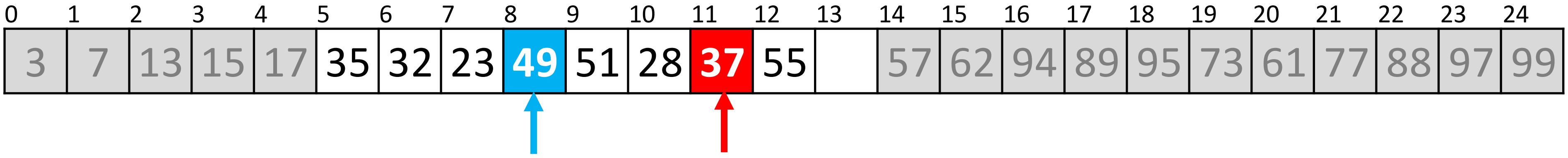
Swap them

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	**49**	51	28	**37**	55		57	62	94	89	95	73	61	77	88	97	99

Continue searching

```
    low = 8;
    high = 11;
```

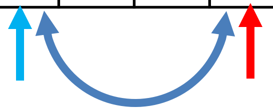
```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	51	28	49	55		57	62	94	89	95	73	61	77	88	97	99



Continue searching

```
low = 8;
high = 11;
```

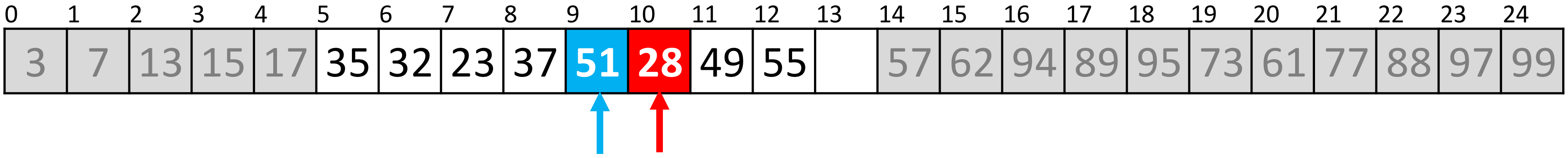
Swap them

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	51	28	49	55		57	62	94	89	95	73	61	77	88	97	99

Continue searching

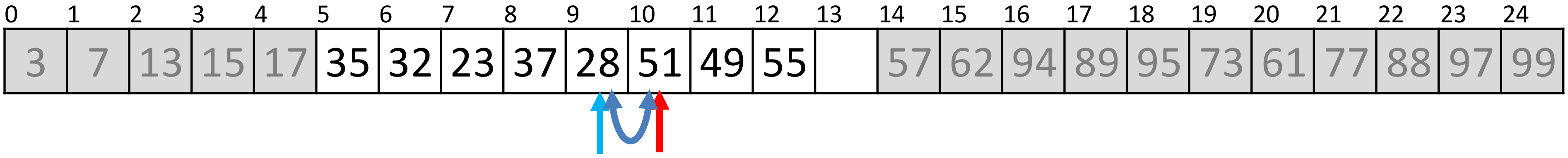
```
low = 8;
high = 11;
```

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	51	49	55		57	62	94	89	95	73	61	77	88	97	99

Continue searching

```
low = 8;
high = 11;
```

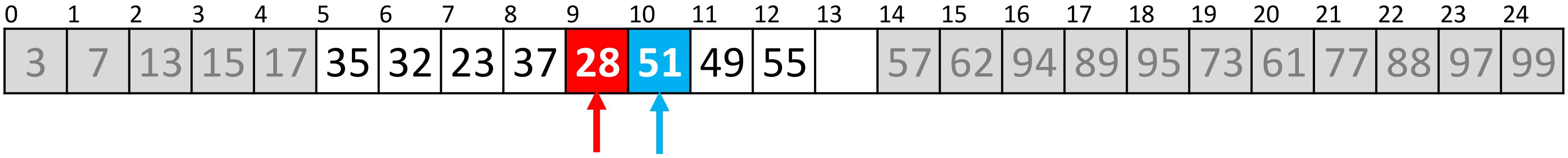
Swap them

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	51	49	55		57	62	94	89	95	73	61	77	88	97	99

Continue searching

```
    low = 8;
    high = 11;
```

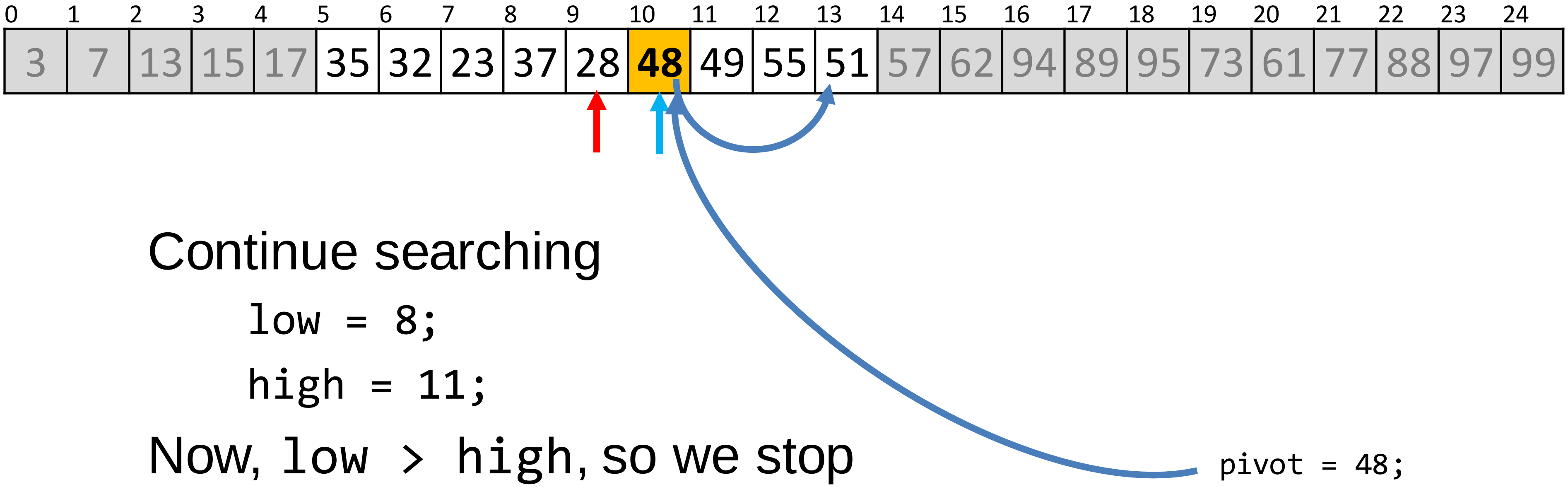
Now, `low > high`, so we stop

```
pivot = 48;
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	**48**	49	55	51	57	62	94	89	95	73	61	77	88	97	99

Continue searching

```
low = 8;
high = 11;
```

Now, `low > high`, so we stop

`pivot = 48;`

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	**48**	49	55	51	57	62	94	89	95	73	61	77	88	97	99

We now begin calling quicksort recursively on the first half

```
quicksort( array, 5, 10 );
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We now are calling `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	**48**	49	55	51	57	62	94	89	95	73	61	77	88	97	99

We now begin calling quicksort recursively

```
quicksort( array, 5, 10 );
```

```
quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 5, 10 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

Now, $10 - 5 \leq 6$, so find we call insertion sort

```
quicksort( array,  5, 10 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 5 to 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	35	32	23	37	28	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

```
insertion_sort( array, 5, 10 )

quicksort( array,  5, 10 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 5 to 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

– This function call completes and so we exit

```
insertion_sort( array, 5, 10 )

quicksort( array,  5, 10 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

```
quicksort( array,  5, 10 )
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 5, 14 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	**48**	49	55	51	57	62	94	89	95	73	61	77	88	97	99

We continue calling quicksort recursively on the second half

```
quicksort( array, 5, 10 );
quicksort( array, 6, 14 );
```

```
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 11, 15 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

Now, $15 - 11 \le 6$, so find we call insertion sort

```
quicksort( array,  6, 14 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 11 to 14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	55	51	57	62	94	89	95	73	61	77	88	97	99

```
insertion_sort( array, 11, 14 )

quicksort( array, 11, 14 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 11 to 14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	62	94	89	95	73	61	77	88	97	99

– This function call completes and so we exit

```
insertion_sort( array, 11, 14 )

quicksort( array, 11, 14 )

quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	62	94	89	95	73	61	77	88	97	99

```
quicksort( array, 11, 14 )
quicksort( array,  5, 14 )
quicksort( array,  0, 14 )
quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	62	94	89	95	73	61	77	88	97	99

```
quicksort( array,  5, 14 )

quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	62	94	89	95	73	61	77	88	97	99

```
quicksort( array,  0, 14 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 0, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	**57**	62	94	89	95	73	61	77	88	97	99

We continue calling quicksort recursively on the second half

```
quicksort( array, 0, 14 );
quicksort( array, 15, 25 );
```

`quicksort( array,  0, 25 )`

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	62	94	89	95	73	61	77	88	97	99

First, $25 - 15 > 6$, so find the midpoint and the pivot

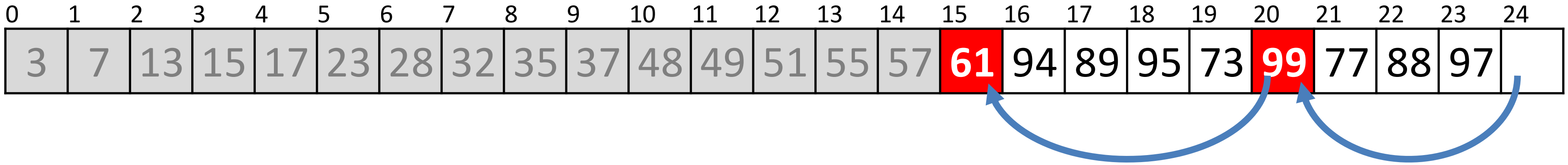
```
midpoint = (15 + 25)/2; // == 20
```

```
quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	**61**	94	89	95	73	**99**	77	88	97	

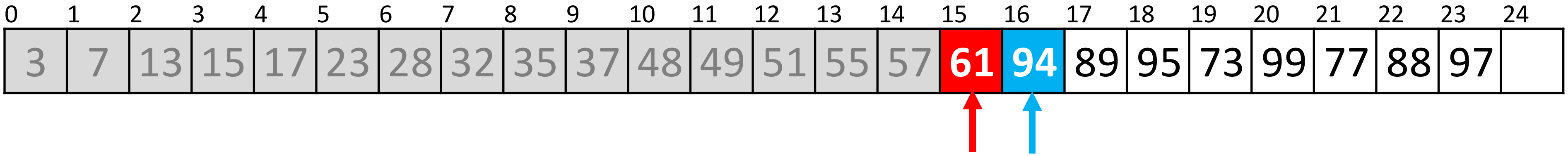
First, $25 - 15 > 6$, so find the midpoint and the pivot

```
midpoint = (15 + 25)/2; // == 20
pivot = 62;
```

```
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	**61**	**94**	89	95	73	99	77	88	97	

Searching forward and backward:

```
low = 16;
high = 15;
```

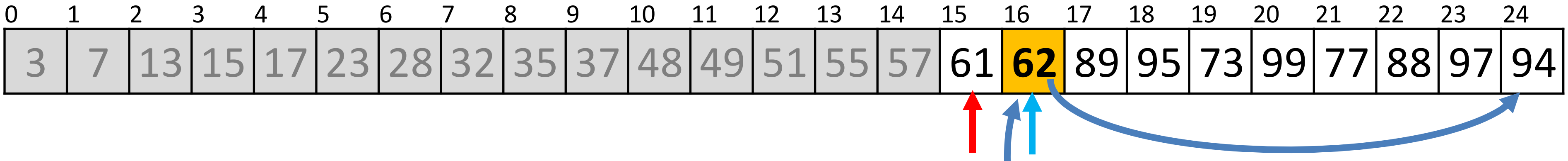
Now, `low > high`, so we stop

```
pivot = 62;
```

```
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	**62**	89	95	73	99	77	88	97	94

Searching forward and backward:

```
low = 16;
high = 15;
```

Now, `low > high`, so we stop

- Note, this is the worst-case scenario
- The pivot is the second smallest element

```
pivot = 62;
```

```
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	89	95	73	99	77	88	97	94

We continue calling quicksort recursively on the first half

```
quicksort( array, 15, 16 );
```

```
quicksort( array, 15, 16 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are executing `quicksort( array, 15, 16 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	89	95	73	99	77	88	97	94

Now, $16 - 15 \le 6$, so find we call insertion sort

```
quicksort( array, 15, 16 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort immediately returns

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	89	95	73	99	77	88	97	94

```
insertion_sort( array, 15, 16 )

quicksort( array, 15, 16 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	89	95	73	99	77	88	97	94

```
quicksort( array, 15, 16 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 15, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	**62**	89	95	73	99	77	88	97	94

We continue calling quicksort recursively on the second half

```
quicksort( array, 15, 16 );
quicksort( array, 17, 25 );
```

```
quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	89	95	73	99	77	88	97	94

First, $25 - 17 > 6$, so find the midpoint and the pivot

```
    midpoint = (17 + 25)/2; // == 21
```

```
quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	**89**	95	73	99	**77**	88	97	**94**

First, $25 - 17 > 6$, so find the midpoint and the pivot

```
midpoint = (17 + 25)/2; // == 21
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	**89**	95	73	99	**77**	88	97	**94**

First, $25 - 17 > 6$, so find the midpoint and the pivot

```
midpoint = (17 + 25)/2; // == 21
pivot = 89
```

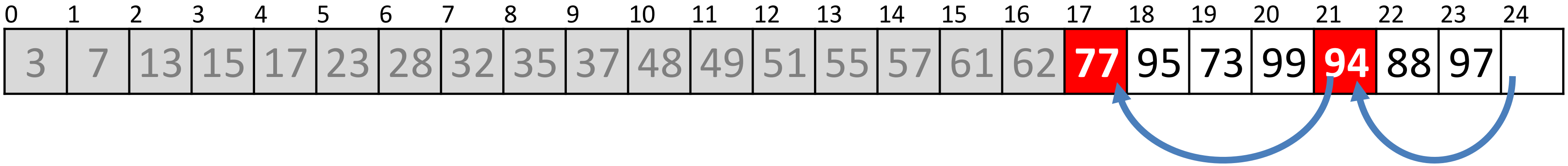
```
quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	**77**	95	73	99	**94**	88	97	

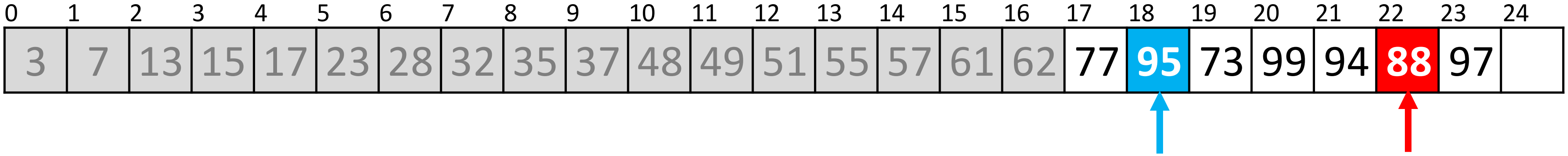
First, $25 - 17 > 6$, so find the midpoint and the pivot

```
midpoint = (17 + 25)/2; // == 21
pivot = 89
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	**95**	73	99	94	**88**	97	

Searching forward and backward:

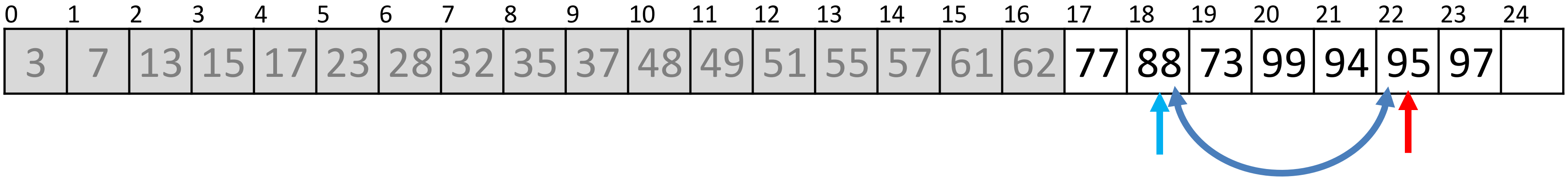
```
low = 18;
high = 22;
```

```
pivot = 89;
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	99	94	95	97	

Searching forward and backward:

```
low = 18;
high = 22;
```

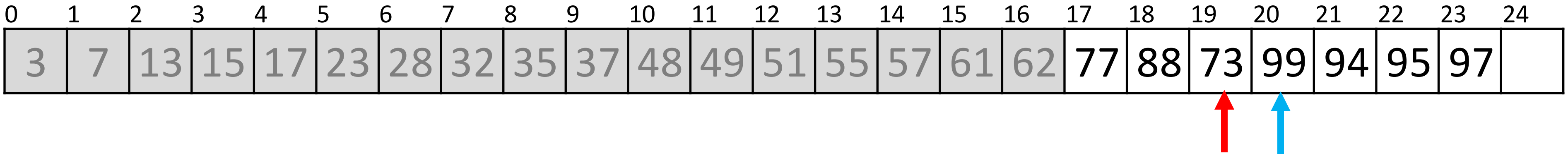
Swap them

```
pivot = 89;
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	99	94	95	97	

Searching forward and backward:

```
    low = 20;
    high = 19;
```

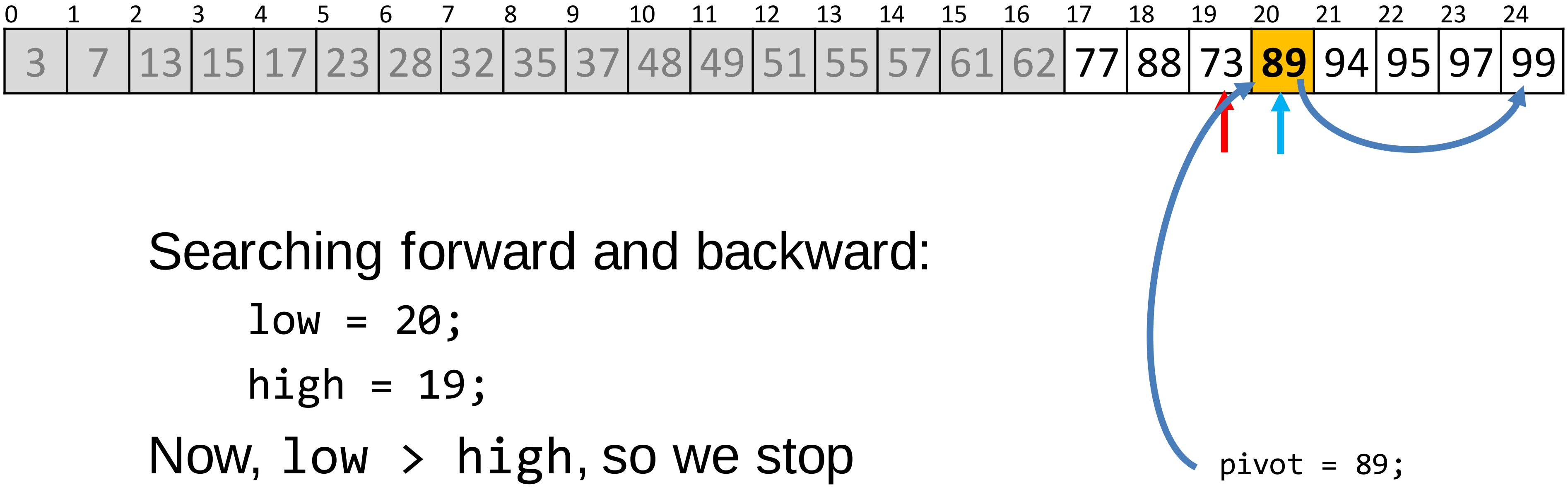
Now, `low > high`, so we stop

```
pivot = 89;
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	**89**	94	95	97	99

Searching forward and backward:

```
low = 20;
high = 19;
```

Now, `low > high`, so we stop

`pivot = 89;`

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	**89**	94	95	97	99

We start by calling quicksort recursively on the first half

```
    quicksort( array, 17, 20 );
```

```
quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are now executing `quicksort( array, 17, 20 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	89	94	95	97	99

Now, $4 - 0 \le 6$, so find we call insertion sort

```
quicksort( array, 17, 20 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 17 to 19

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	77	88	73	89	94	95	97	99

```
insertion_sort( array, 17, 20 )

quicksort( array, 17, 20 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 17 to 19

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

– This function call completes and so we exit

```
insertion_sort( array, 17, 20 )

quicksort( array, 17, 20 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
quicksort( array, 17, 20 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	**89**	94	95	97	99

```
quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

We are back to executing `quicksort( array, 17, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	**89**	94	95	97	99

We continue by calling quicksort on the second half

```
quicksort( array, 17, 20 );
quicksort( array, 21, 25 );
```

```
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

We are now calling `quicksort( array, 21, 25 )`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

Now, $25 - 21 \le 6$, so find we call insertion sort

```
quicksort( array, 21, 25 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 21 to 24

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
insertion_sort( array, 21, 25 )
quicksort( array, 21, 25 )
quicksort( array, 17, 25 )
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

Insertion sort just sorts the entries from 21 to 24

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

– In this case, the sub-array was already sorted
– This function call completes and so we exit

```
insertion_sort( array, 21, 25 )

quicksort( array, 21, 25 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
quicksort( array, 21, 25 )

quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
quicksort( array, 17, 25 )

quicksort( array, 15, 25 )

quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
quicksort( array, 15, 25 )
quicksort( array,  0, 25 )
```

Quicksort Example

This call to `quicksort` is now also finished, so it, too, exits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

```
quicksort( array,  0, 25 )
```

Quicksort Example

We have now used quicksort to sort this array of 25 entries

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	7	13	15	17	23	28	32	35	37	48	49	51	55	57	61	62	73	77	88	89	94	95	97	99

Summary

- This topic covered quicksort
  - On average faster than heap sort or merge sort
  - Uses a pivot to partition the objects
  - Using the median of three pivots is a reasonably means of finding the pivot
  - Average run time of $\Theta(n \ln(n))$ and $\Theta(\ln(n))$ memory
  - Worst case run time of $\Theta(n^2)$ and $\Theta(n)$ memory