ESC101: Introduction to Computing

Sorting

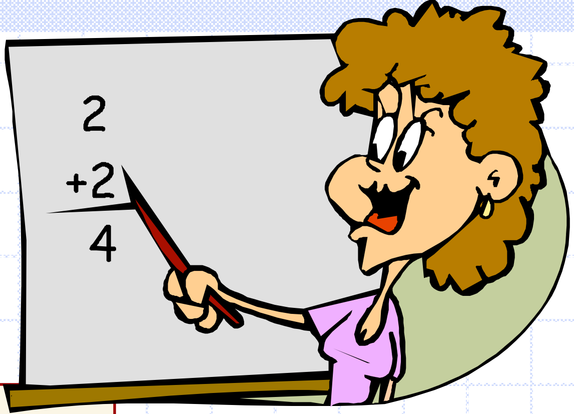
Sorting

- Given a list of integers (in an array), arrange them in ascending order.
  - Or descending order

INPUT ARRAY	5	6	2	3	1	4
OUTPUT ARRAY	1	2	3	4	5	6

- Sorting is an extremely important problem in computer science.
  - A common problem in everyday life.
  - Example:
    - Contact list on your phone.
    - Ordering marks before assignment of grades.

What's easy to do in a Sorted Array?



Clearly, searching for a key is fast.

Rank Queries: find the k^{th} largest/smallest value.
Quantile: 90%ile—the key value in the array such that 10% of the numbers are larger than it.

40	50	55	60	70	75	80	85	90	92

Marks in an exam: sorted

90 percentile : 90
80 percentile : 85
10 percentile: 40
50 percentile: 70
(also called median)

Sorted array have difficulty with

- inserting a new element while preserving the sorted structure.
- deleting an existing element (while preserving the sorted structure.
- In both cases, there may be need to shift elements to the right or left of the index corresponding to insertion or deletion.

40	50	55	60	70	75	80	85	90	92

Example: Insert 65.
1. Find index where 65 needs to be inserted

40	50	55	60	65	70	75	80	85	90	92

2. Shift right from index 5 to create space.

3. Insert 65

May have to shift n-1 elements in the worst case.

Sorting

- Many well known sorting Algorithms
  - Selection sort
  - Merge sort
  - Quick sort
  - Bubble sort
  - …
- Special cases also exist for specific problems/data sets
- Different runtime
- Different memory requirements

Selection Sort

- Select the largest element in your array and swap it with the first element of the array.
- Consider the sub-array from the second element to the last, as your current array and repeat Step 1.
- Stop when the array has only one element.
  - Base case, trivially sorted

Selection Sort: Pseudo code

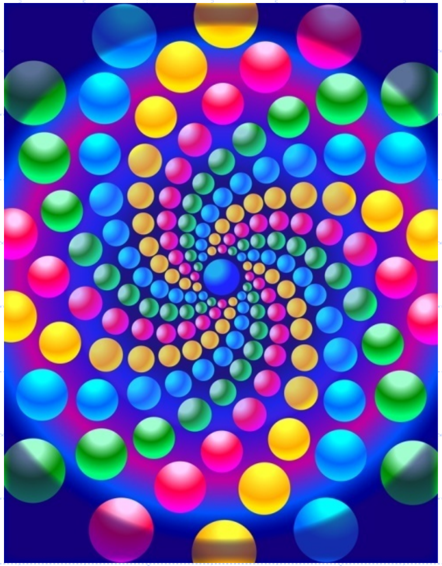
```
selection_sort(a, start, end) {
   if (start == end) /* base case, one elt => sorted */
      return;

   idx_max = find_idx_of_max_elt(a, start, end);
   swap(a, idx_max, start);
   selection_sort(a, start+1, end);
}
```

```
swap(a, i, j) {
   tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;
}
```

```
main() {
   arr[] = { 5, 6, 2, 3, 1, 4 };
   selection_sort(arr, 0, 5);
   /* print arr */
}
```

Selection Sort: Properties



- Is the pseudo code iterative or recursive?
- What is the estimated run time when input array has **n** elements
  - for swap **Constant**
  - for find_idx_of_max_elt **∝ n**
  - for selection_sort **On next slide**
- **Practice**: Write C code for iterative and recursive versions of selection sort.

Selection Sort: Time Estimate

- Recurrence

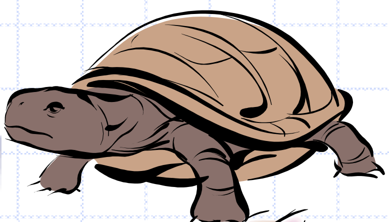
$$T(n) = T(n-1) + k_1 \times n + k_2$$

- Solution

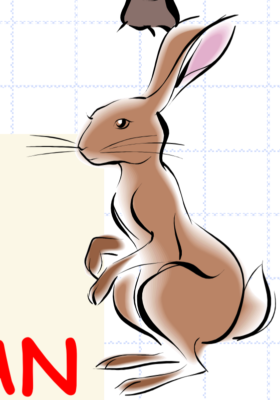
$$T(n) \propto n(n+1)$$

- Or simply

$$T(n) \propto n^2$$



Selection sort runs in time proportional to the **square of the size of the array** to be sorted.

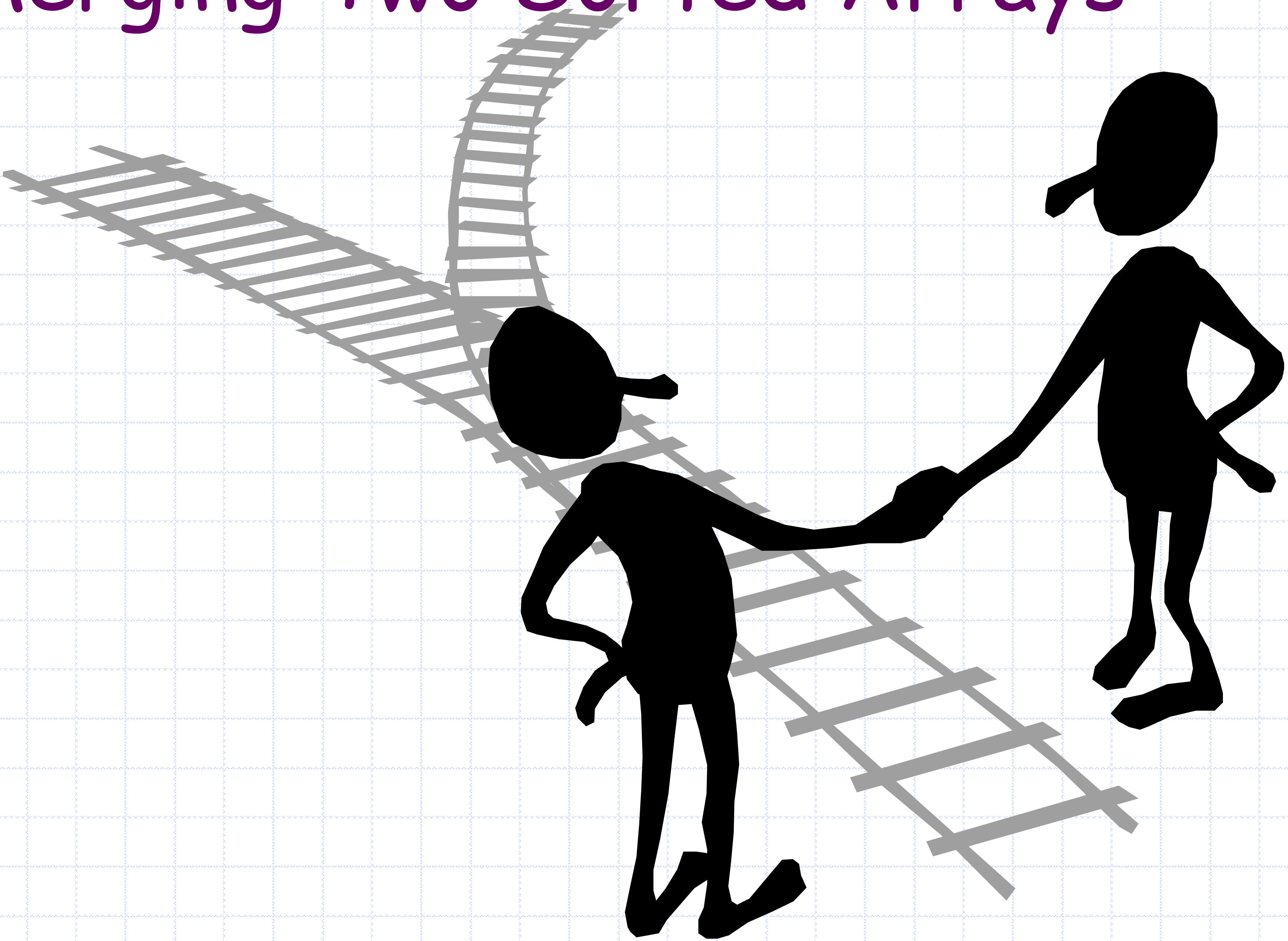


Can we do better?
YES WE CAN

Merging Two Sorted Arrays

- Input: Array A of size n & array B of size m.
- Create an empty array C of size n + m.
- Variables i , j and k
  - array variables for the arrays A, B and C resp.
- At each iteration
  - compare the i^{th} element of A (say u) with the j^{th} element of B (say v)
  - if u is smaller, copy u to C; increment i and k,
  - otherwise, copy v to C; increment j and k,

Merging Two Sorted Arrays



Time Estimate

- Number of steps $\propto 3(n + m)$.
  - The constant 3 is not very important as it does not vary with different sized arrays.
- Now suppose A and B are halves of an array of size n (both have size n/2).
- Number of steps = 3n.

$$T(n) \propto n$$

MergeSort

- Merge function can be used to sort an array
  - recursively!
- Given an array C of size n to sort
  - Divide it into Arrays A and B of size n/2 each (approx.)
  - Sort A into A′ using MergeSort
  - Sort B into B′ using MergeSort
  - Merge A′ and B′ to give C′ ≡ C sorted
- Can we reduce #of extra arrays (A′, B′, C′)?

Recursive calls. Base case?

n <= 1

```
/*Sort ar[start, …, start+n-1] in place */
void merge_sort(int ar[], int start, int n) {
   if (n>1) {
      int half = n/2;
      merge_sort(ar, start, half);
      merge_sort(ar, start+half, n-half);
      merge(ar, start, n);
   }
}
```

```
int main() {
   int arr[]={2,5,4,8,6,9,8,6,1,4,7};
   merge_sort(arr,0,11);
   /* print array */
   return 0;
}
```

```
void merge(int ar[], int start, int n) {
  int temp[MAX_SZ], k, i=start, j=start+n/2;
  int lim_i = start+n/2, lim_j = start+n;
  for(k=0; k<n; k++) {
    if ((i < lim_i) && (j < lim_j)) {// both active
      if (ar[i] <= ar[j]) { temp[k] = ar[i]; i++; }
      else { temp[k] = ar[j]; j++; }
    } else if (i == lim_i) // 1st half done
      { temp[k] = ar[j]; j++; } // copy 2nd half
    else // 2nd half done
      { temp[k] = ar[i]; i++; } // copy 1st half
  }
  for (k=0; k<n; k++)
    ar[start+k]=temp[k]; // in-place
}
```

Time Estimate

`void merge_sort(int a[], int s, int n) {`	$T(n)$
`if (n>1) {`	c
`int h = n/2;`	c
`merge_sort(a, s, h);`	$T(n/2)$
`merge_sort(a, s+h, n-h);`	$T(n-n/2) \approx T(n/2)$
`merge(a, s, n);`	$\approx 4n$
`}`	
`}`	

Time Estimate

$T(n) = 2T(n/2) + 4n$

$= 2(2T(n/4) + 4n/2) + 4n = 2^2T(n/4) + 8n$

$= 2^2(2T(n/8) + 4n/4) + 4n = 2^3T(n/8) + 12n$

= … // keep going for k steps

$= 2^kT(n/2^k) + k*4n$

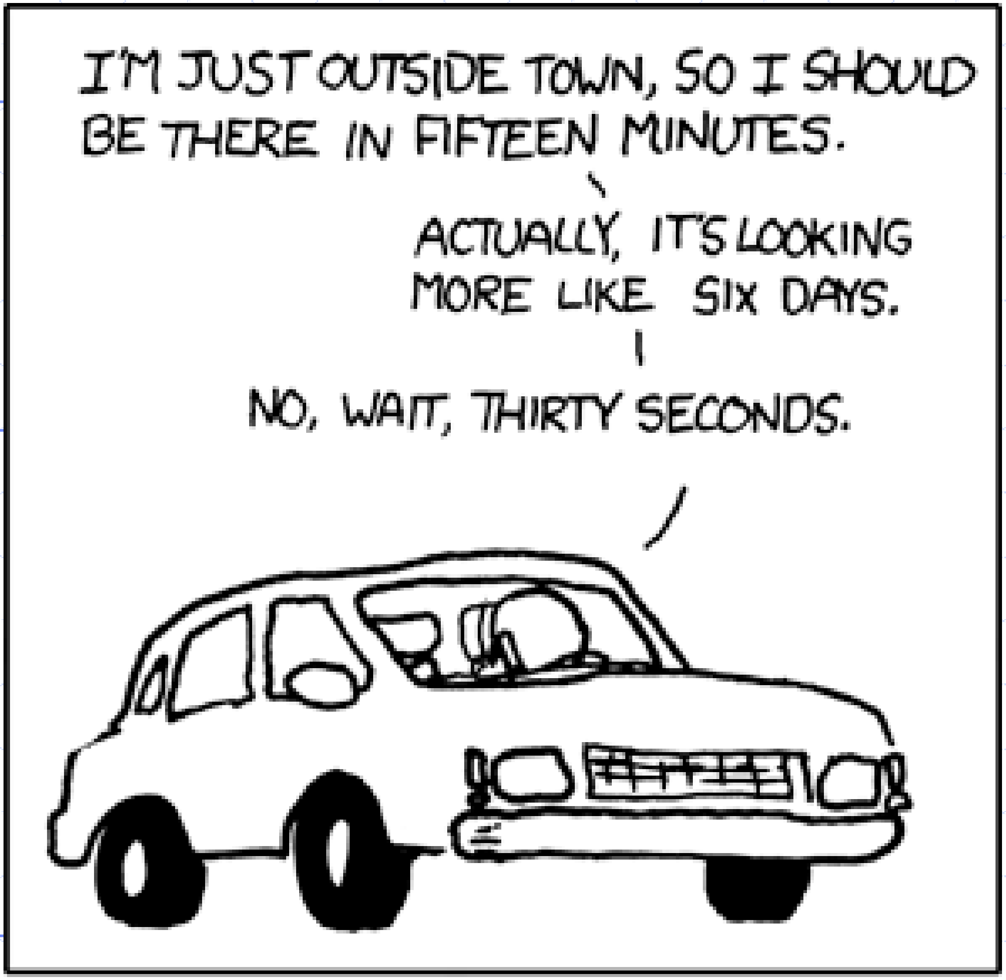
Assume $n = 2^k$ for some k. Then,

$$T(n) = n*T(1) + 4n*\log_2 n$$

$$T(n) \propto n \log_2 n$$

Order Notation

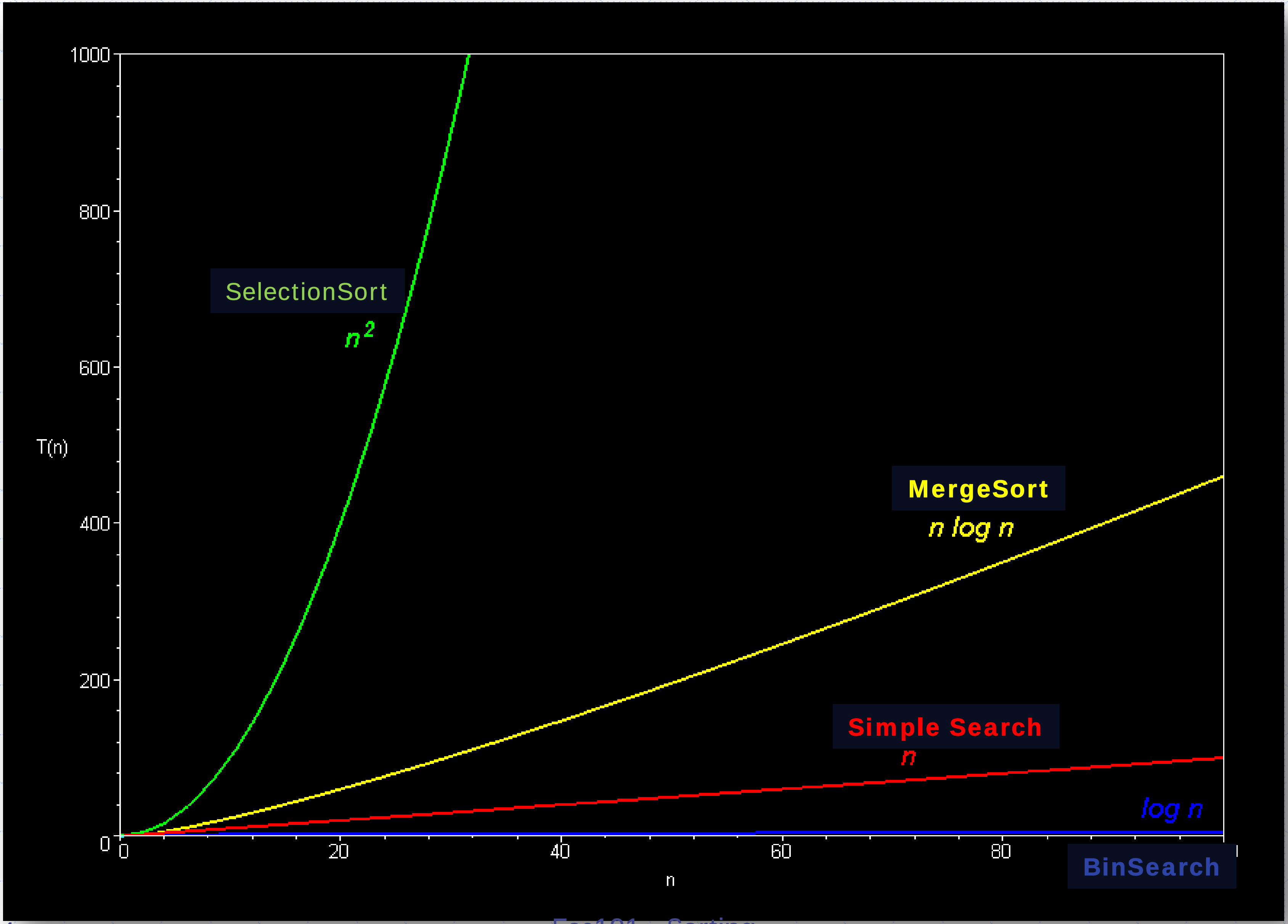
Why worry about O(n) vs O(n²) vs O(…) algo?



THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.

http://xkcd.com/612/

Time Estimates…



Source: http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/

QuickSort-- Partition Routine

A useful sub-routine (function) for many problems, including quicksort(), the most popularly used sorting method.

1. Partition takes an array a[] of size n and a value called the pivot.
2. The pivot is an element in the array is usually chosen as a[0].
3. Partition re-arranges the array elements into two parts:
   a) the left part has all elements <= pivot.
   b) the right part has all elements >= pivot.
4. Partition returns the index of the beginning of the right part.

Let us see an example.

1. **Partition** takes an array a[] of size n and a value called the pivot.
2. The pivot is an element in the array is usually chosen as a[0].
3. Partition re-arranges the array elements into two parts:
   a) all elements in the left part are <= pivot
   b) all elements in the right part are >= pivot

Input Array a[], size is n : 11

31	4	10	35	59	31	3	25	35	11	0

Call to partition(a,11). **Pivot** element is assumed to be a[0]: 31

0	4	10	11	25	3	31	31	35	35	31

left partition

right partition

return value is 6

COMMENTS

Multiple "partitions" of an array are possible, even for the same pivot. They all would satisfy the above specification.

Note: Partition **DOES NOT** sort the array. It is "weaker" than sorting. But it is useful step towards sorting (useful for other problems also).

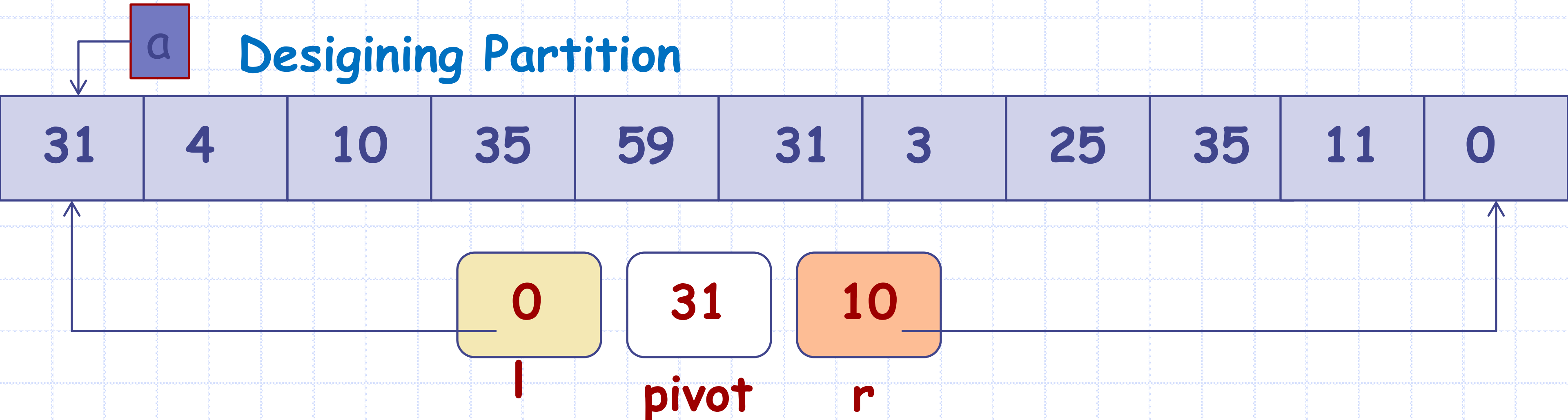
1. partition(int a[], int n). pivot will be a[0].
2. Partition re-arranges the array elements into two parts:
   a) the left part has all elements <= pivot
   b) the right part has all elements >= pivot
3. Partition should return either the first index of the right part or the last index of the left part. (Both answers would be acceptable).

Designing partition: Goal is to have **linear time complexity**, meaning that the number of comparisons and exchanges of items must be linear in the size.

Also, we will do partition ***in place*** – that is, without using extra arrays.

Can you do it easily if you have extra arrays?

Desigining Partition



1. Keep two integer variables denoting indices: l starts at the left end and r starts at the right end.
2. pivot is a[0] which is 31.
3. Value of pivot will not change during partition.

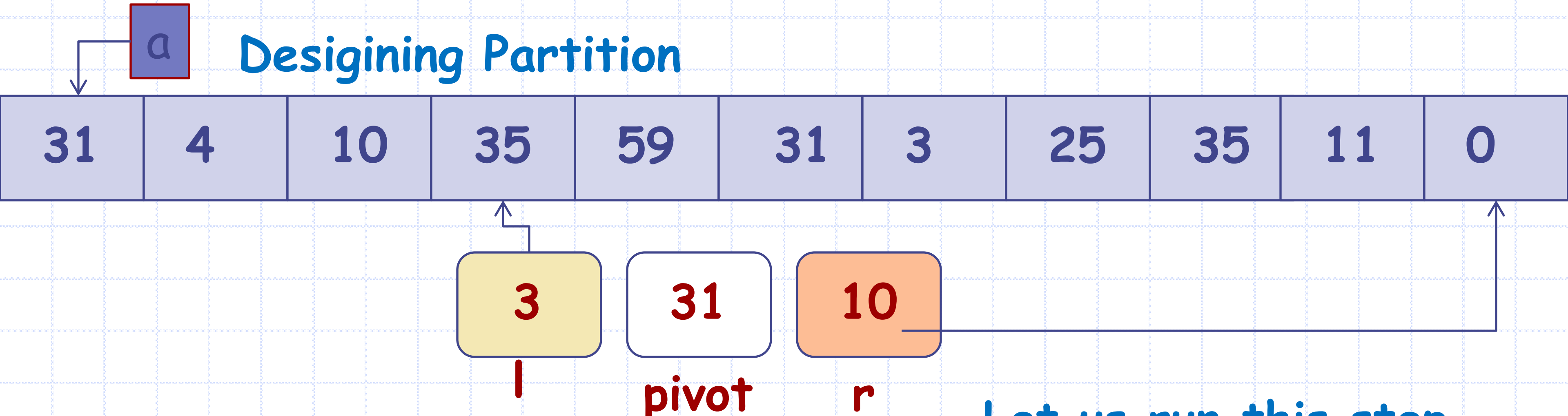
Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

As long as a[r] >= pivot, decrease r by 1.

If l < r, Exchange a[l] with a[r].
advance l by 1; decrement r by 1

Desigining Partition



Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

As long as a[r] >= pivot, decrease r by 1.

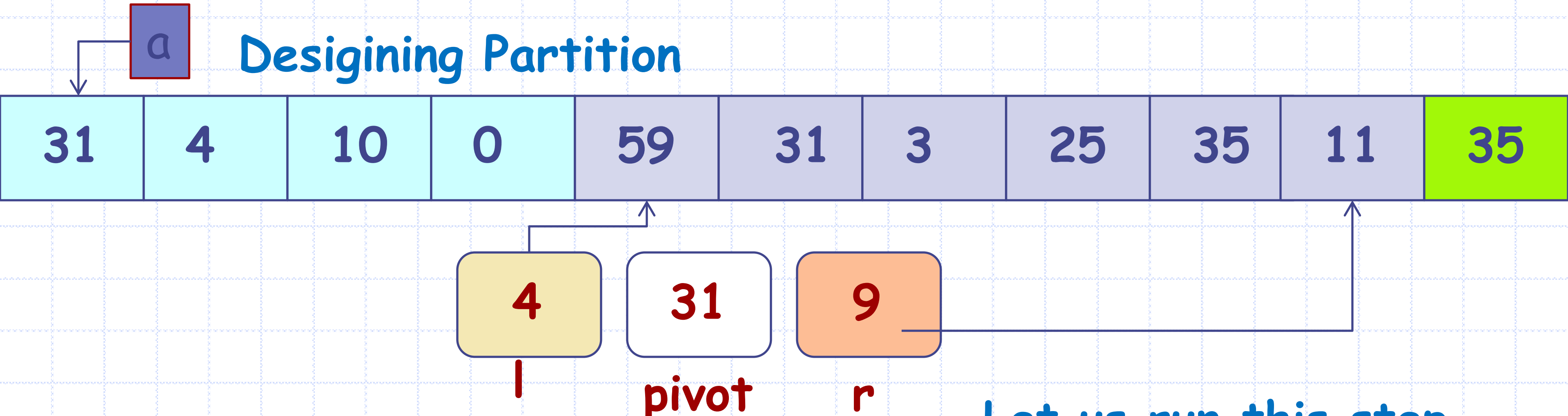
If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Let us run this step on the above array

1. First loop terminates, with l as 3.
2. Second loop terminates immediately, with r as 10.

Now we swap a[3] with a[10]

Desigining Partition



Let us run this step on the above array

Basic Step in Partition Loop:

As long as a[l] < pivot, advance l by 1.

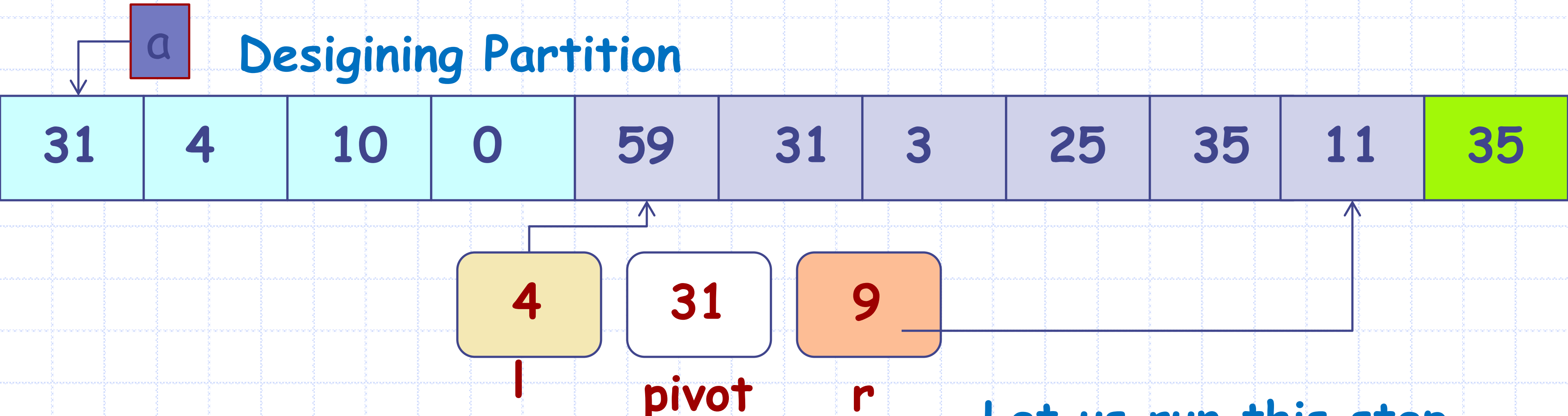
As long as a[r] > pivot, decrease r by 1.

Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Swap and Advance

1. swap a[3] with a[10]
2. Advance l to 4
3. Decrement r to 9

Desigining Partition



Basic Step in Partition Loop:

Let us run this step on the above array

As long as a[l] < pivot, advance l by 1.

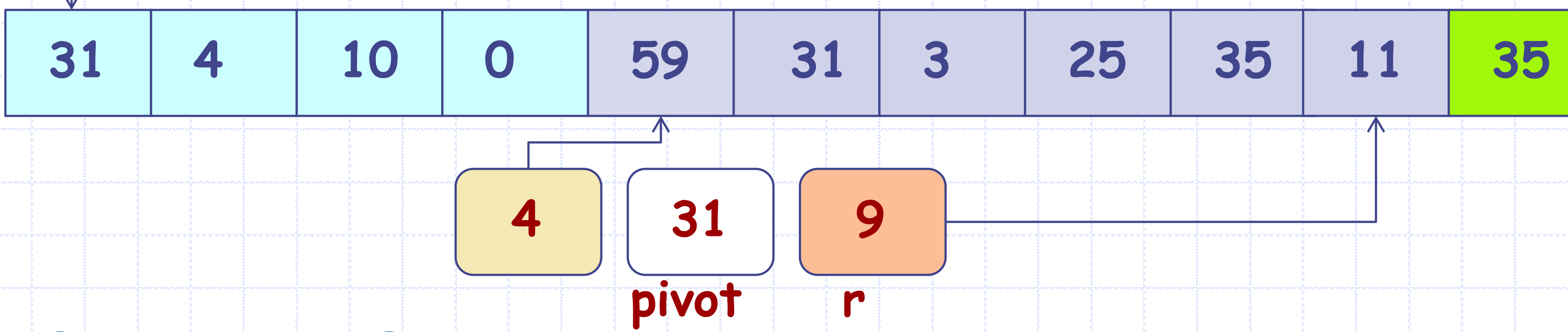
As long as a[r] > pivot, decrease r by 1.

Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Invariant

1. a[0]…a[l-1] are all <= pivot.
2. a[r+1]…a[n-1] are all >= pivot.

Desigining Partition



Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

As long as a[r] >= pivot, decrease r by 1.

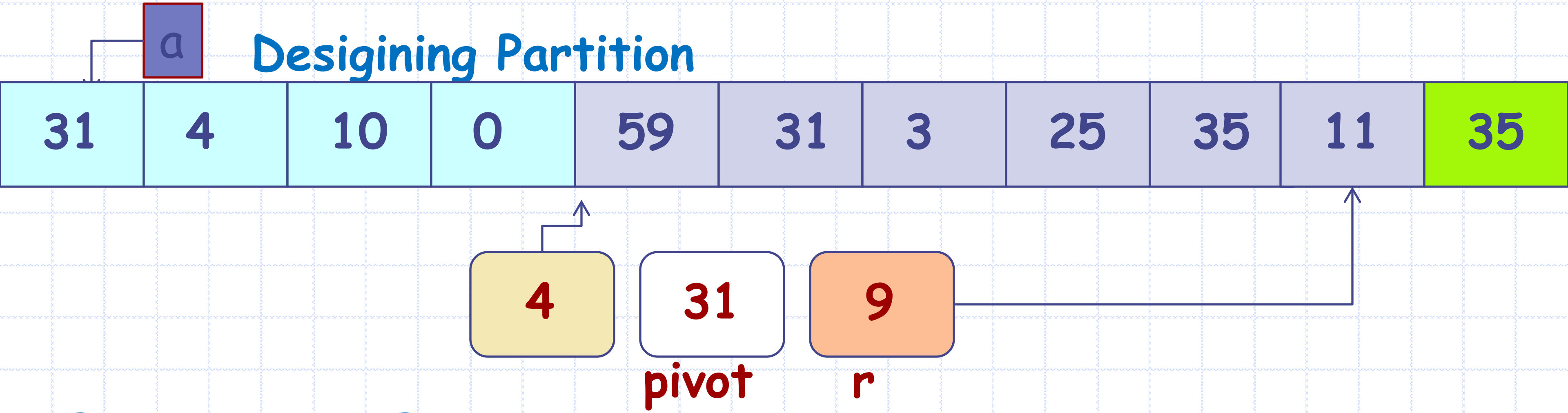
If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Now what should we do? We can run the basic step again.

Loop for l

1. The loop for l terminates immediately since 59 > 31.

Desigining Partition



Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

As long as a[r] >= pivot, decrease r by 1.

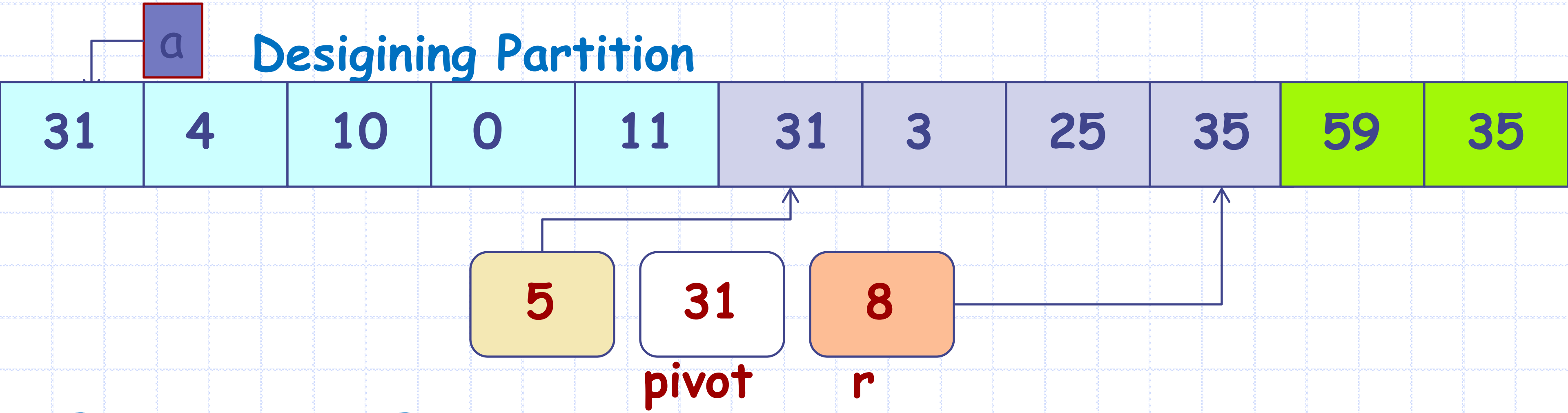
If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Now what should we do?
We can run the basic step again.

Swap and advance

1. Swap 59 with 11
2. Increment l by 1
3. Decrement r by 1

Desigining Partition



Basic Step in Partition Loop:

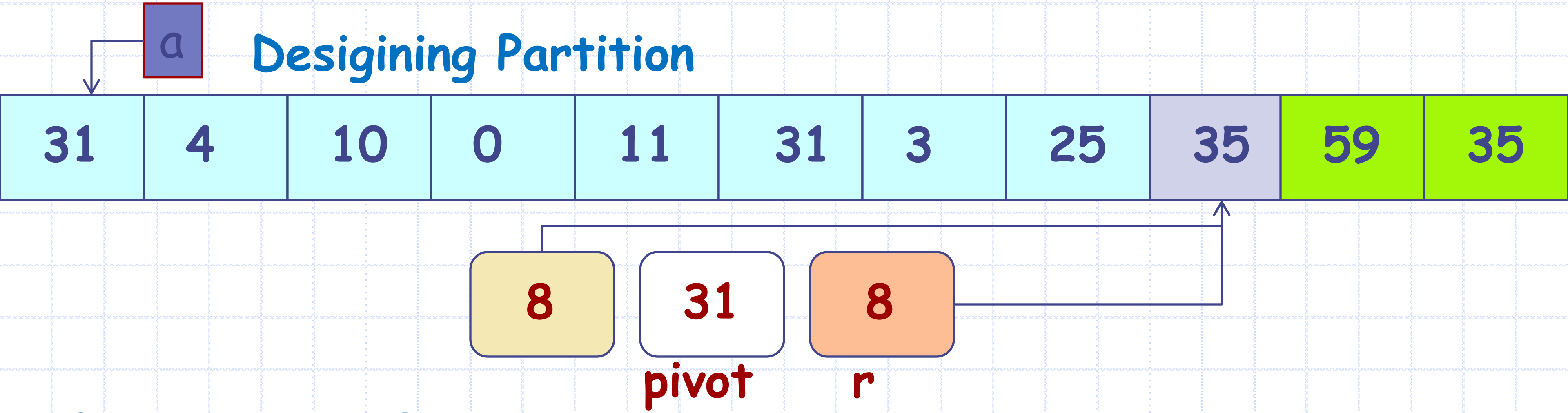
As long as a[l] <= pivot, advance l by 1.

As long as a[r] >= pivot, decrease r by 1.

If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Now what should we do? We can run the basic step again.

Desigining Partition



Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

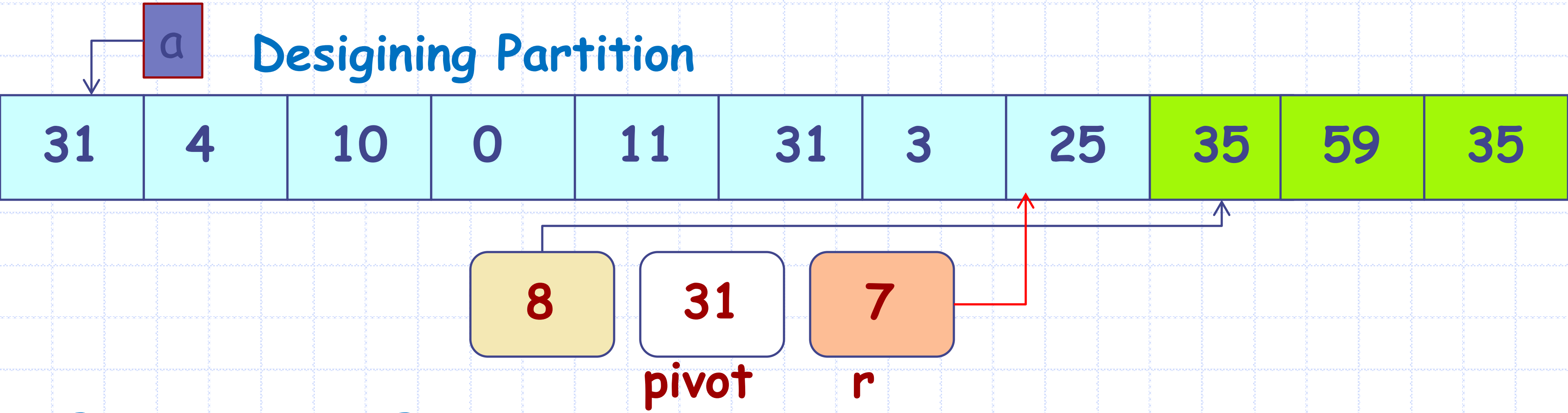
As long as a[r] >= pivot, decrease r by 1.

If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Loop for l

1. The loop for l terminates at l=8.

Desigining Partition



Basic Step in Partition Loop:

As long as a[l] <= pivot, advance l by 1.

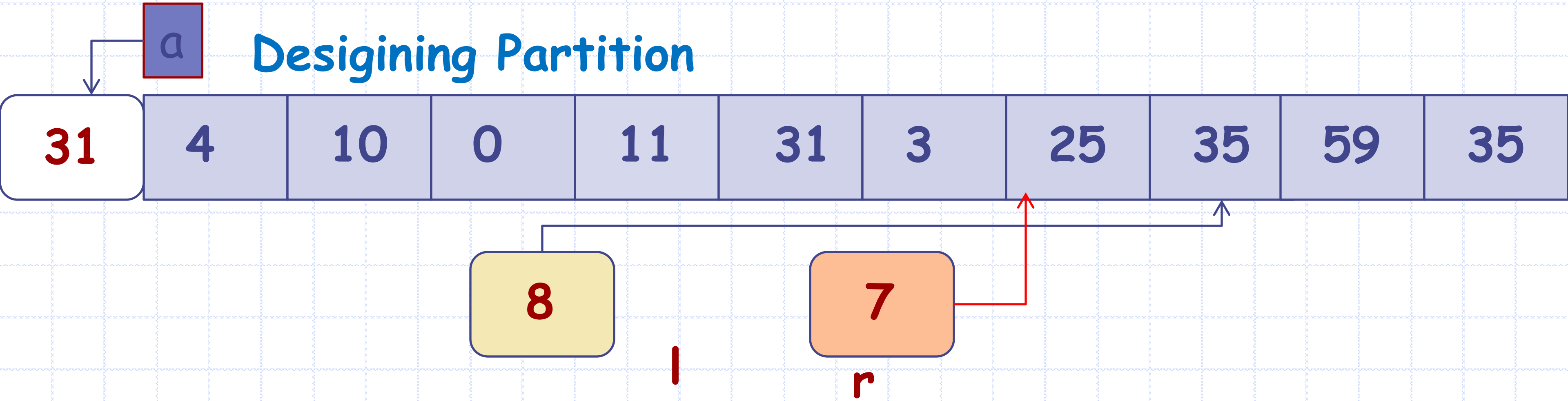
As long as a[r] >= pivot, decrease r by 1.

If l < r, Swap a[l] with a[r].
advance l by 1;
decrement r by 1

Loop for r

1. The loop for l terminates at r=7.

Desigining Partition



Partition is *almost* done!

Basic Step in Partition Loop:

As long as a[l] < pivot, advance l by 1.

As long as a[r] > pivot, decrease r by 1.

Swap a[l] with a[r]. advance l by 1; decrement r by 1

Why? From invariant:

1. What should we return?
2. We can return r.

```
int partition(int a[], int n) {
      int l = 0, r = n-1, pivot = a[0];
      while (l <=n-1 && r>=0) {
            while (a[l] <= pivot) { l=l+1; }
            while (a[r] >= pivot) { r=r-1; }
            if(l<r) {
              swap(a, l, r);
              l = l+1; r = r-1;
            } else {
              /* move pivot to its position */
             swap(a, l-1, 0);
             return l-1;
            }
      }
}
```

The Partition function

We designed a function **int partition(int a[], int n)** that returns an index **pindex** of the array a[] such that for any a[] with n >=2, all the following are true.

1. pindex lies between 0 and n-2,
2. all items in a[0..pindex] are <= pivot,
3. all items in a[pindex+1…n-1] are >= pivot,
4. Number of operations required by partition is O(n), that is bounded by c•n for some constant c. Required only a single pass over the array: each element is touched once.

Pivoting choices

Pivot may be chosen to be any value of a[]. Some choices are

1. Pivot is a[0]: simple choice.
2. Pivot is some random member of a[]: randomized pivot choice.
3. Pivot is the median element of a[]. This gives the most equal sized partitions, but is much more complicated.

Suppose we wish to sort the array a[].

After the call **pindex = partition(a,n)**

1. each element of a[0..pindex-1] <= pivot.
2. each element of a[pindex..n-1] >= pivot.

So after the call to partition(), to sort a[], we can just

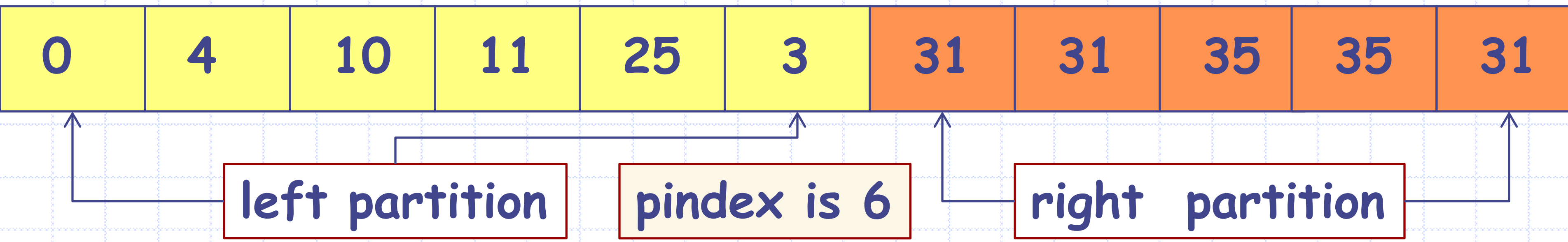
1. sort the array a[0..pindex-1], and,
2. sort the array a[pindex…n-1].

For example, consider the array.

Input Array a[], size is n : 11

31	4	10	35	59	31	3	25	35	11	0

After call to partition(a,11). Pivot element is assumed to be a[0]: 31



Obvious from diagram: to sort a[], we can sort the left partition and the right partition independently.

How should we do the sorting: Any way we wish, but… how about choosing the same algorithm, that is, run partition on each half again (and then again on smaller parts—this is **recursion**)

After call to partition(a,11). **Pivot** element is assumed to be a[0]: 31

0	4	10	11	25	3	31	31	35	35	31

left partition

pindex is 6

right partition

Writing formally:

```
void qsort(int a[], int n) {
        int pindex;
        if (n<=1) return;   /* nothing to sort */
        else {
           pindex = partition(a,n);
           qsort(a,pindex);
           qsort(a+pindex+1, n-pindex-1);
     }
}
```

This is a recursive program

These are recursive calls.

31	4	10	35	59	31	3	25	35	11	31

a

Let us now run qsort on the above array: n is 11.

STACK

Function called	Called by function	n	pindex	Return address
main()	---	---	---	---
qsort(a,11)	main()	11	-	main().??

```
void qsort(int a[], int n) {
    int pindex;
    if (n<=1) return;
    pindex = partition(a,n);
    qsort(a,pindex);
    qsort(a+pindex+1, n-pindex-1);
}
```

Quicksort function

calls partition (a,11)

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	---	---
qsort(a,11)	main()	11	5	main().??

```
void qsort(int a[], int n) {
    int pindex;
    if (n<=1) return;
    pindex = partition(a,n);
    qsort(a,pindex);
    qsort(a+pindex+1, n-pindex-1);
}
```

partition(a,11) returns pindex as 5

Now a call is made to qsort(a,6).

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6		qsort().6

```
1.  void qsort(int a[], int n) {
2.      int pindex;
3.      if (n<=1) return;
4.      pindex = partition(a,n);
5.      qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
7.  }
```

in call qsort(a,11)

calls qsort(a,pindex). return address is qsort().6

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6		qsort().6

The call to qsort(a,6)

Makes a call to partition(a,6) here.

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6		qsort().6
partition(a,6)	qsort(a,6)	6	--	qsort().4

return address for partition(a,6) is this.

```
1. void qsort(int a[], int n) {
…
4. pindex = partition(a,n);

5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

Calling partition(a,6) returns 0 and changes the array as follows.

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6		qsort().6
partition(a,6)	qsort(a,6)	6	--	qsort().4

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6		qsort().6
partition(a,6)	qsort(a,6)	6	--	qsort().4

1. partition(a,6)
2. returns.
3. Return value is 0,
4. pindex is set to 0.
5. qsort(a,6)
6. resumes at line 4.

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a,1)	qsort(a,6)	1	--	qsort().6

In call qsort(a,6)

qsort(a,6) now has pindex as 1.
Now calls qsort(a,1).
Return addr. qsort() line 6.

```
1. void qsort(int a[], int n) {
2.      int pindex;
3.      if (n<=1) return;
4.      pindex = partition(a,n);
5       qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a,1)	qsort(a,6)	1	--	qsort().6

In call qsort(a,1)

Since n is 1, qsort(a,1) returns immediately.

```
1. void qsort(int a[], int n) {
2.     int pindex;
3      if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6

Returns to executing qsort(a,6), line 6

In call qsort(a,6), line 6

Now calls qsort(a+pindex+1, n-pindex-1).
Calls qsort(a+1, 5)
return addr. qsort.7

```
1. void qsort(int a[], int n) {
2.      int pindex;
3.      if (n<=1) return;
4.      pindex = partition(a,n);
5.      qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	?	qsort().7

In call qsort(a+1,5), line 3

Now calls partition(a+1,5)

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	4	10	11	25	3	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	?	qsort().7
partition(a+1,5)	qsort(a+1,5)	5	--	qsort().4

In partition(a+1,5):

Pivot is (a+1)[0] which is 4, so after partition is over, the array would be like this...

0	3	10	11	25	4	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	?	qsort().7

In qsort(a+1,5):

partition(a+1,5) returns 0. So pindex is 0. execution resumes at line 4.

```
1. void qsort(int a[], int n) {
2.     ...
4      pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	10	11	25	4	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	?	qsort().7

In qsort(a+1,5):

Calls qsort(a+1, 1)

To summarize, this will terminate immediately, with no change to a[].

```
1. void qsort(int a[], int n) {
2.     …
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6      qsort(a+pindex+1, n-pindex-1);
7  }
```

0	3	10	11	25	4	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().6

In qsort(a+1,5):

in line 6: pindex is 0
Calls qsort(a+2, 4).

```
1.  void qsort(int a[], int n) {
2.      …
4.      pindex = partition(a,n);
5.      qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
7
```

0	3	10	11	25	4	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	?	qsort().7

in line 6: pindex is 0
Calls qsort(a+2, 4).
return addr is
qsort().line 7

```
1.  void qsort(int a[], int n) {
2.      …
4.      pindex = partition(a,n);
5.      qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
7.
```

0	3	10	11	25	4	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	?	qsort().7
partition(a+2,4)	qsort(a+2,4)	4	--	qsort().4

In partition (a+2, 4)

pivot will be (a+2)[4] which is 10. After partition, the state of the array is as shown…

0	3	4	11	25	10	31	31	35	35	31

a

After partition (a+2, 4)

a[] is as above, return value 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	?	qsort().7

In qsort(a+2,4) line 4

pindex is set to 0.

```
1. void qsort(int a[], int n) {
2.     …
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
```

0	3	4	11	25	10	31	31	35	35	31

a

After partition (a+2, 4)

a[] is as above, return value 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7

In qsort(a+2,4) line 5

calls qsort(a+2,1).
This returns
immediately. No change

```
1. void qsort(int a[], int n) {
2.     ...
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
```

0	3	4	11	25	10	31	31	35	35	31

a

After partition (a+2, 4)

a[] is as above, return value 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7

In qsort(a+2,4) line 6

calls qsort(a+3,3). since, n-pindex-1 is 4-0-1 which is 3.

```
1. void qsort(int a[], int n) {
2.      …
5.      qsort(a,pindex);
6.      qsort(a+pindex+1, n-pindex-1);
```

0	3	4	11	25	10	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	??	qsort().7

In qsort(a+3,3) line 4

calls partition(a+3,3)

```
1. void qsort(int a[], int n) {
2.       …
4.    pindex = partition(a,n);
…
```

0	3	4	11	25	10	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	??	qsort().7
partition(a+3,3)	qsort(a+3,3)	3	--	qsort().4

In partition (a+3,3)

pivot is (a+3)[0] which is 11.

State of array after partition becomes

0	3	4	10	25	11	31	31	35	35	31

a

partition (a+3,3) returns 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	??	qsort().7

In qsort(a+3,3) line 4

pindex is set to 0.

```
5.   qsort(a,pindex);
6.   qsort(a+pindex+1, n-pindex-1);
```

0	3	4	10	25	11	31	31	35	35	31

a

partition (a+3,3) returns 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	??	qsort().7

calls qsort(a+3,1)

returns immediately.

```
5.    qsort(a,pindex);
6.    qsort(a+pindex+1, n-pindex-1);
```

0	3	4	10	25	11	31	31	35	35	31

a

partition (a+3,3) returns 0

pindex is set to 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7

calls qsort(a+4,2)

```
5.    qsort(a,pindex);
6. => qsort(a+pindex+1, n-pindex-1);
```

0	3	4	10	25	11	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	??	qsort().7
partition(a+4,2)	qsort(a+4,2)	2	--	qsort().4

calls partition (a+4,2)

pivot is 25

array becomes…

0	3	4	10	11	25	31	31	35	35	31

a

partition (a+4,2) returns 0

pindex is 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	??	qsort().7

qsort(a+4,2) resumes at line 7. But this is the last line…

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

Line 7 terminates the call to qsort(a,n).

So stack changes as follows.

0	3	4	10	11	25	31	31	35	35	31

a

partition (a+4,2) returns 0

pindex is 0

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	0	qsort().7

qsort(a+4,2) resumes at line 4, pindex is 0.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	0	qsort().7

qsort(a+4,2) line 5: Calls qsort(a+5,1) which terminates immediately.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	0	qsort().7

qsort(a+4,2) line 6: Calls qsort(a+6,1) which terminates immediately.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7
qsort(a+4,2)	qsort(a+3,3)	2	0	qsort().7

qsort(a+4,2) line 7: qsort(a+4,2) terminates now.
Control returns to its calling fn: qsort(a+3,3) at line 7.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7
qsort(a+3,3)	qsort(a+2,4)	3	0	qsort().7

qsort(a+3,3) resumes at line 7 and terminates. Control retuns to calling fn qsort(a+2,4) at line 7.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7
qsort(a+2,4)	qsort(a+1,5)	4	0	qsort().7

qsort(a+2,4) resumes at line 7 and terminates. Control retuns to calling fn qsort(a+1,5) at line 7.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7

qsort(a+1,5) resumes at line 7 and terminates. Control retuns to calling fn qsort(a,6) at line 7.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6
qsort(a+1,5)	qsort(a,6)	5	0	qsort().7

qsort(a+1,5) resumes at line 7 and terminates. Control retuns to calling fn qsort(a,6) at line 7.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a,6)	qsort(a,11)	6	0	qsort().6

qsort(a,6) resumes at line 7 and terminates. Control retuns to calling fn qsort(a,11) at line 6.

Note that qsort(a,6) has terminated and the array a[0..5] has been sorted.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??

qsort(a,11)
resumes at line 6.

Calls qsort(a+6,5)

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5.     qsort(a,pindex);
6      qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	??	qsort().7

In qsort(a+6,5)

Calls partition(a+6,5)

```
1. void qsort(int a[], int n) {
2.     int pindex;
3.     if (n<=1) return;
4.     pindex = partition(a,n);
5      qsort(a,pindex);
6.     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	??	qsort().7
partition(a+6,5)	qsort(a+6,5)	5	--	qsort().4

In partition(a+6,5)

pivot is (a+6)[0] which is 31.

After partition, array looks like this...

0	3	4	10	11	25	31	31	35	35	31

a

partition(a+6,5) returns 1

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	??	qsort().7

In qsort(a+6,5)

pindex is 1.
Calls qsort(a+6,2).

```
1. void qsort(int a[], int n) {
...
5.        qsort(a,pindex);
6.        qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+6,2)	qsort(a+6,5)	2	??	qsort().6

In qsort(a+6,5)

pindex is 1.
Calls qsort(a+6,2).

```
1. void qsort(int a[], int n) {
...
5.         qsort(a,pindex);
6.         qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+6,2)	qsort(a+6,5)	2	0	qsort().6

In qsort(a+6,2) | Calls partition(a+6,2)

1. Partition is called for the array

31	31

2. partition returns 0.
3. Control returns to qsort(a+6,2) line 4, with pindex set to 0.
4. line 5: Calls qsort(a+6,1) which returns immediately.
5. line 6: Calls qsort(a+7,1) which returns immediately.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+6,2)	qsort(a+6,5)	2	0	qsort().6

In qsort(a+6,2)

Calls made:

1. partition(a+6,2) is called for the array

31	31

2. partition(a+6,2) returns 0.
3. Control returns to qsort(a+6,2) line 4, with pindex set to 0.
4. line 5: Calls qsort(a+6,1) which returns immediately.
5. line 6: Calls qsort(a+7,1) which returns immediately.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+6,2)	qsort(a+6,5)	2	0	qsort().6

In qsort(a+6,2): Terminates at line 7.

Control returns to qsort(a+6,5) line 6.

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7

In qsort(a+6,5) line 6.

calls qsort(a+8,3)

```
1. void qsort(int a[], int n) {
...
5.          qsort(a,pindex);
6.          qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	??	qsort().7

In qsort(a+8,3).

calls partition (a+8,3)

```
1. void qsort(int a[], int n) {
...
5.         qsort(a,pindex);
6.   ⇒     qsort(a+pindex+1, n-pindex-1);
7. }
```

0	3	4	10	11	25	31	31	35	35	31

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	??	qsort().7
partition(a+8,3)	qsort(a+8,3)	3	--	qsort().4

In partition(a+8,3)

pivot is (a+8)[0] which is 35.
Partition returns 1.

state of array after partition(a+8,3) is...

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	1	qsort().7
qsort(a+8,2)	qsort(a+8,3)	2	??	qsort().6
partition(a+8,2)	qsort(a+8,2)	1	--	qsort().4

In partition(a+8,2)

1. pivot is 31.
2. returns 0. No change to array.

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	1	qsort().7
qsort(a+8,2)	qsort(a+8,3)	2	??	qsort().6

In qsort(a+8,2) line 4:

1. calls partition(a+8,2).

1. pivot is 31.
2. returns 0. No change to array.

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	1	qsort().7
qsort(a+8,2)	qsort(a+8,3)	2	0	qsort().6

In qsort(a+8,2)

1. line 4: pindex is set to 0.
2. line 5: calls qsort(a+8,1).
3. this returns immediately.

1. line 6: calls qsort(a+9,1).
2. Returns immediately.
3. line 7: qsort(a+8,2) returns.

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	1	qsort().7
qsort(a+8,2)	qsort(a+8,3)	2	0	qsort().6

In qsort(a+8,2)

1. line 4: pindex is set to 0.
2. line 5: calls qsort(a+8,1).
3. this returns immediately.

1. line 6: calls qsort(a+9,1).
2. Returns immediately.
3. line 7: qsort(a+8,2) returns to call qsort(a+8,3) line 6.

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7
qsort(a+8,3)	qsort(a+6,5)	3	1	qsort().7

In qsort(a+8,3) line 6:

1. calls qsort(a+10,1)
2. returns immediately.
3. qsort(a+8,3) returns to line 7 in call qsort(a+6,5)

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7

In qsort(a+6,5) line 7: returns to line 7 of qsort(a,11).

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??
qsort(a+6,5)	qsort(a,11)	5	1	qsort().7

In qsort(a+6,5) line 7: returns to line 7 of qsort(a,11).

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---
qsort(a,11)	main()	11	5	main().??

In qsort(a,11) line 7: returns to the calling function main().

0	3	4	10	11	25	31	31	31	35	35

a

STACK

Function called	Called by function	n	pindex	Return address
main()	---	--	--	---

ARRAY a[] is SORTED.