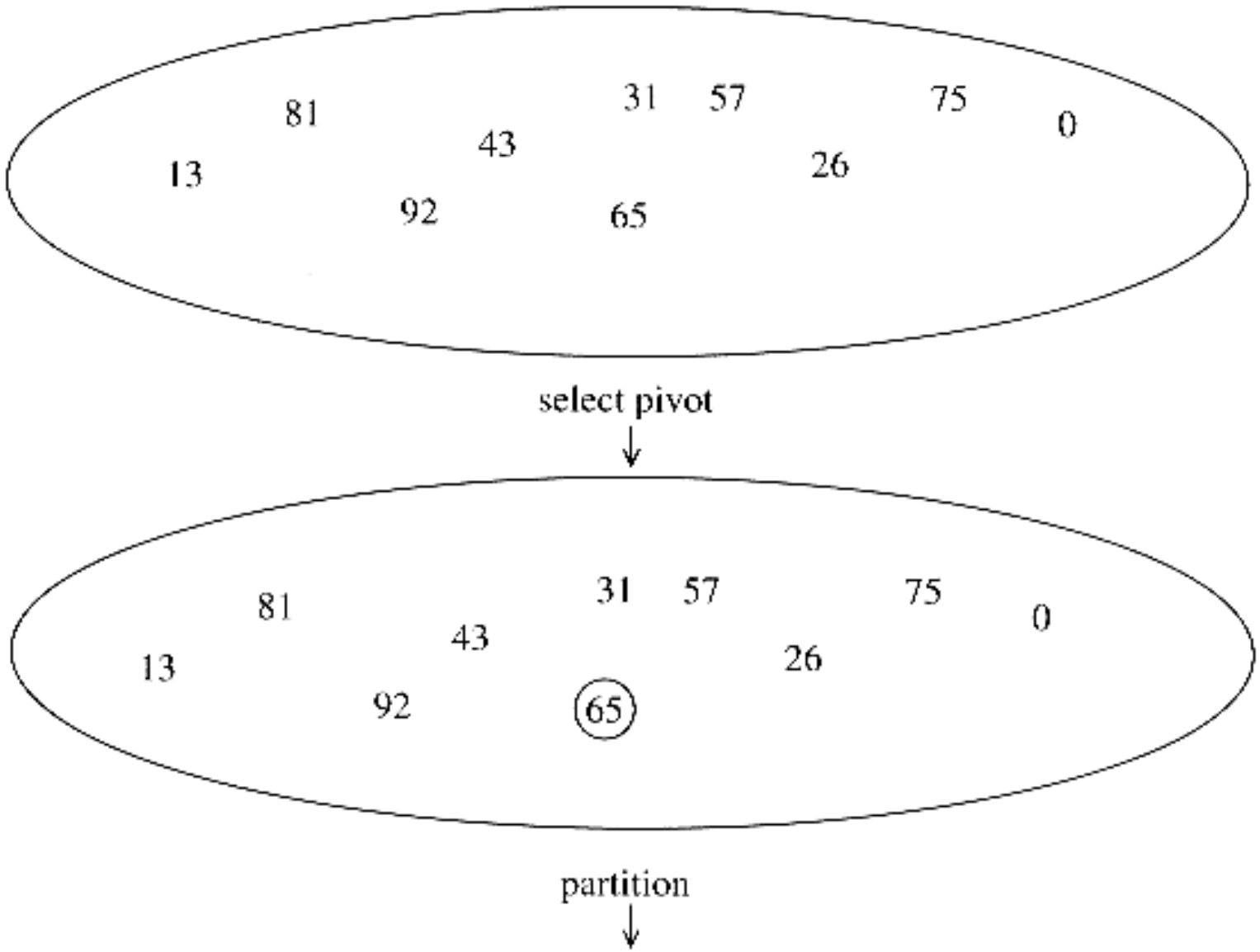
Sorting

Dr. Abdallah Karakra

Computer Science Department

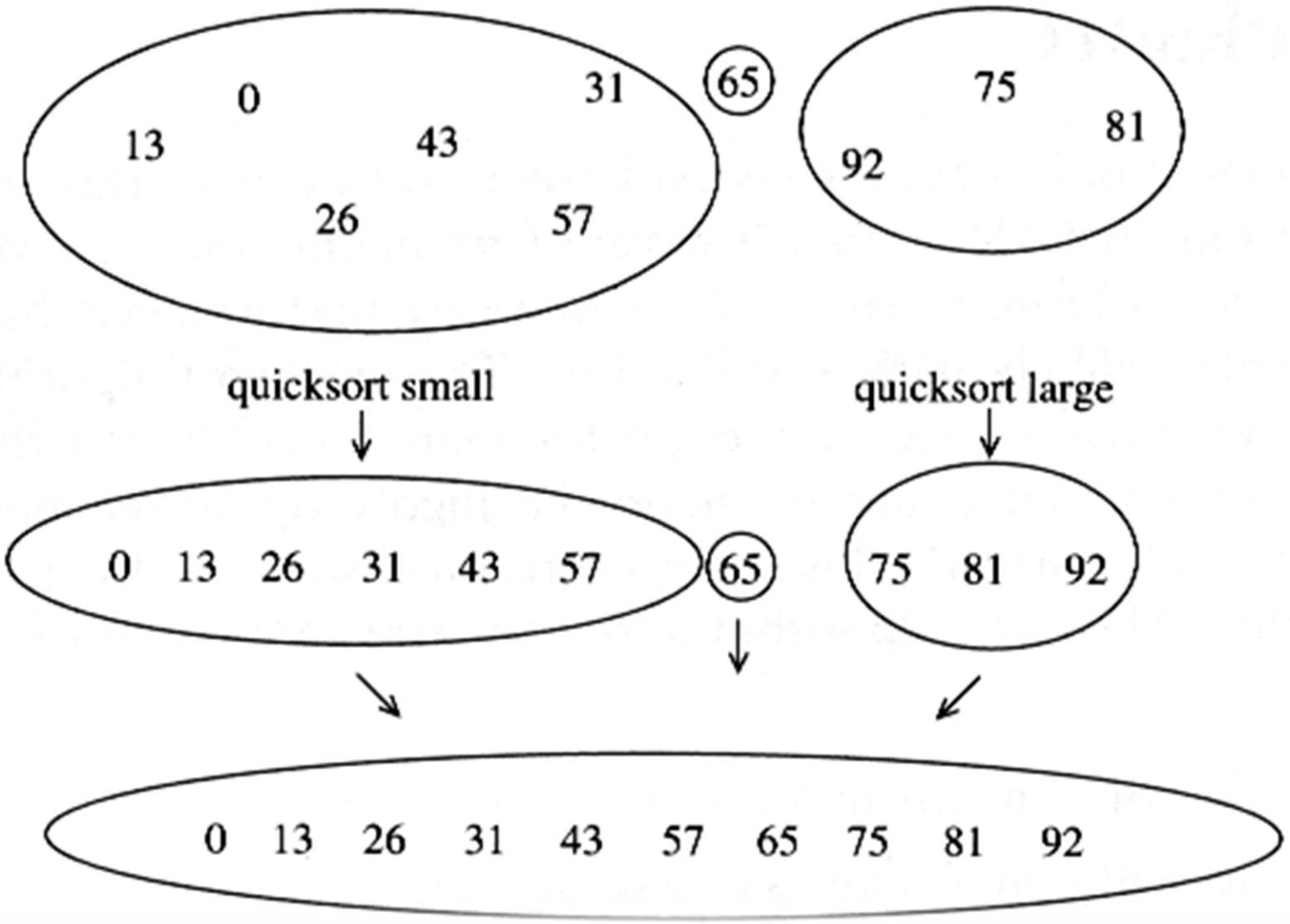
COMP242

Quick_sort


81
31
57
75
0
43
13
26
92
65
select pivot
81
31
57
75
0
43
13
26
92
65
partition

To Select Pivot

1. **Choose the pivot randomly**
2. **Use the first element as pivot**
3. **Use the last element as pivot**
4. **Median-of-three partitioning**



To Select Pivot

```
if (A[left]>A[center])
   swap (A[left],A[center])
if (A[left]>A[right])
   swap (A[left],A[right])
if (A[center]>A[right])
  swap (A[center],A[right])

Swap (A[center],A[right-1])
```

How it works

While (Left>right)
- **moves i right, skipping over elements smaller than the pivot**
- **moves j left, skipping over elements greater than the pivot**

When both I && j have stopped
- **swap (A[i], A[j])**

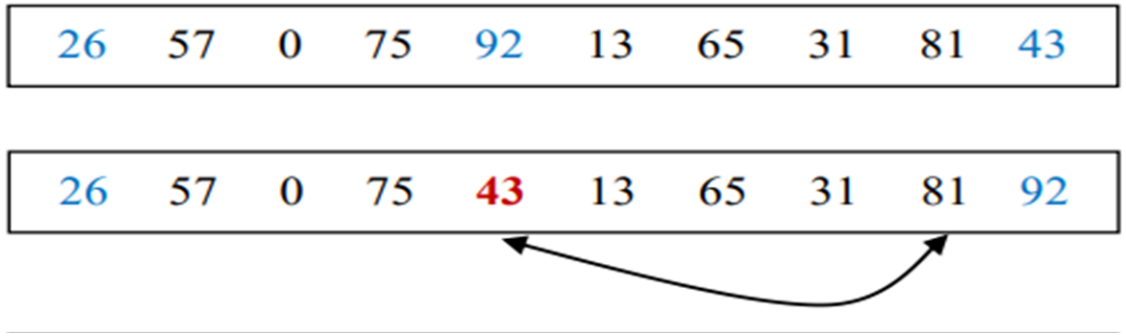
To Select Pivot

26	57	0	75	92	13	65	31	81	43

```
if (A[left]>A[center])
   swap (A[left],A[center])
if (A[left]>A[right])
   swap (A[left],A[right])
if (A[center]>A[right])
  swap (A[center],A[right])

Swap (A[center],A[right-1])
```

To Select Pivot

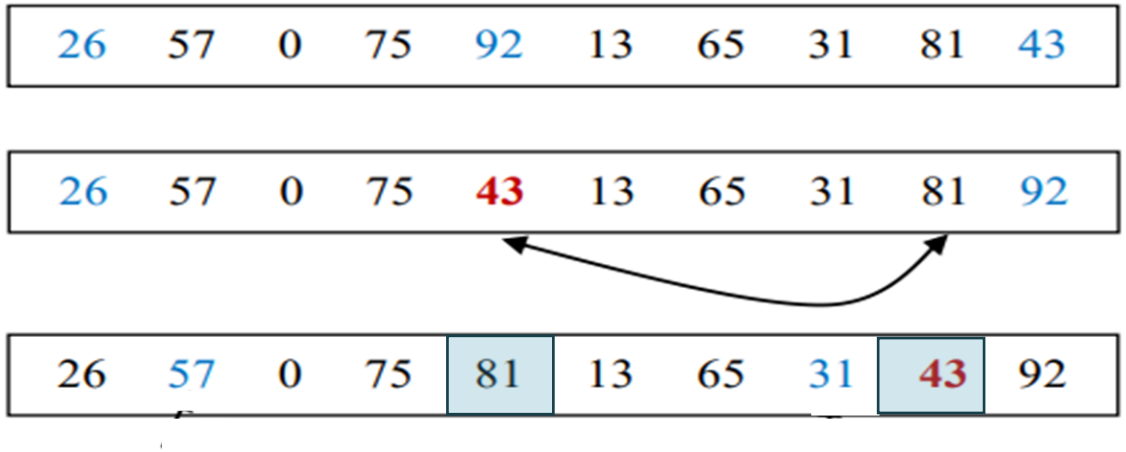
```
if (A[left]>A[center])
   swap (A[left],A[center])
if (A[left]>A[right])
   swap (A[left],A[right])
if (A[center]>A[right])
  swap (A[center],A[right])

Swap (A[center],A[right-1])
```



To Select Pivot

```
if (A[left]>A[center])
   swap (A[left],A[center])
if (A[left]>A[right])
   swap (A[left],A[right])
if (A[center]>A[right])
  swap (A[center],A[right])
```

Swap (A[center],A[right-1])



How it works

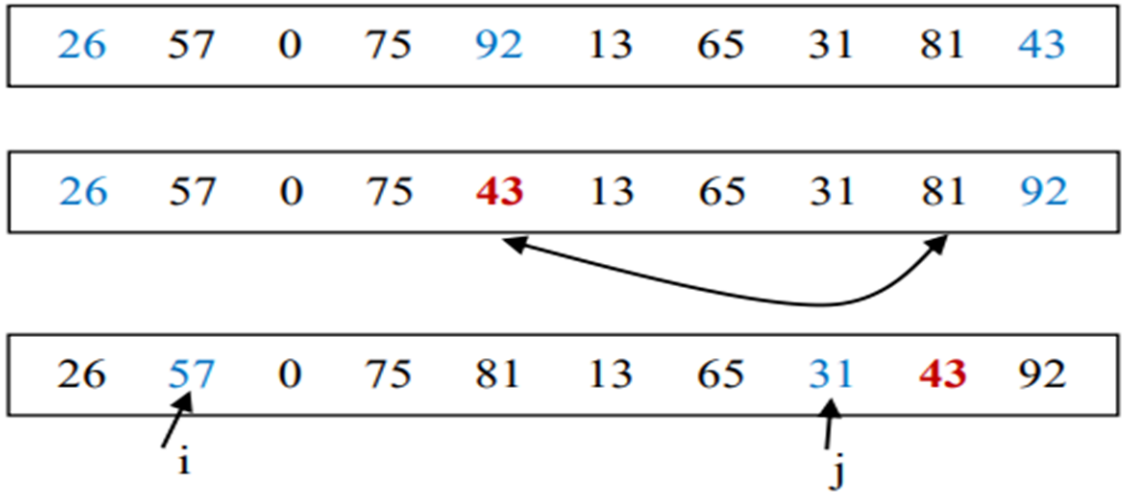
While (Left<right)

moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])



How it works

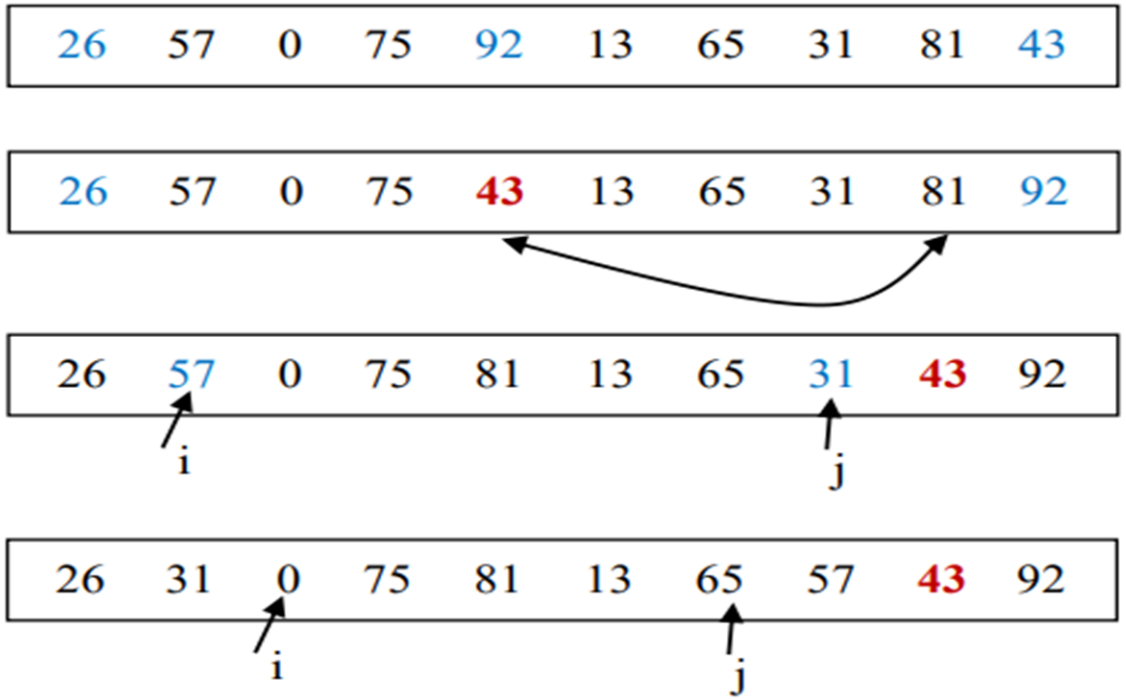
While (Left<right)

moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])



How it works

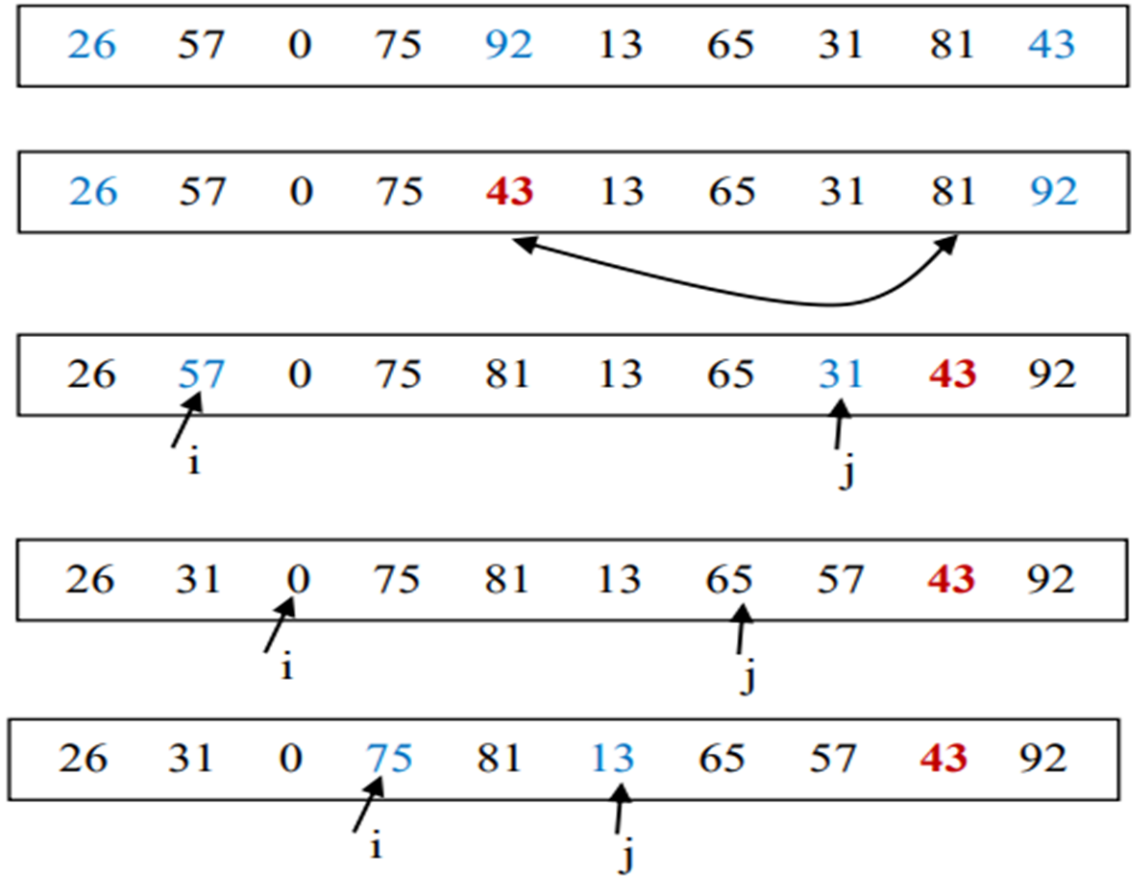
While (Left<right)

moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])



How it works

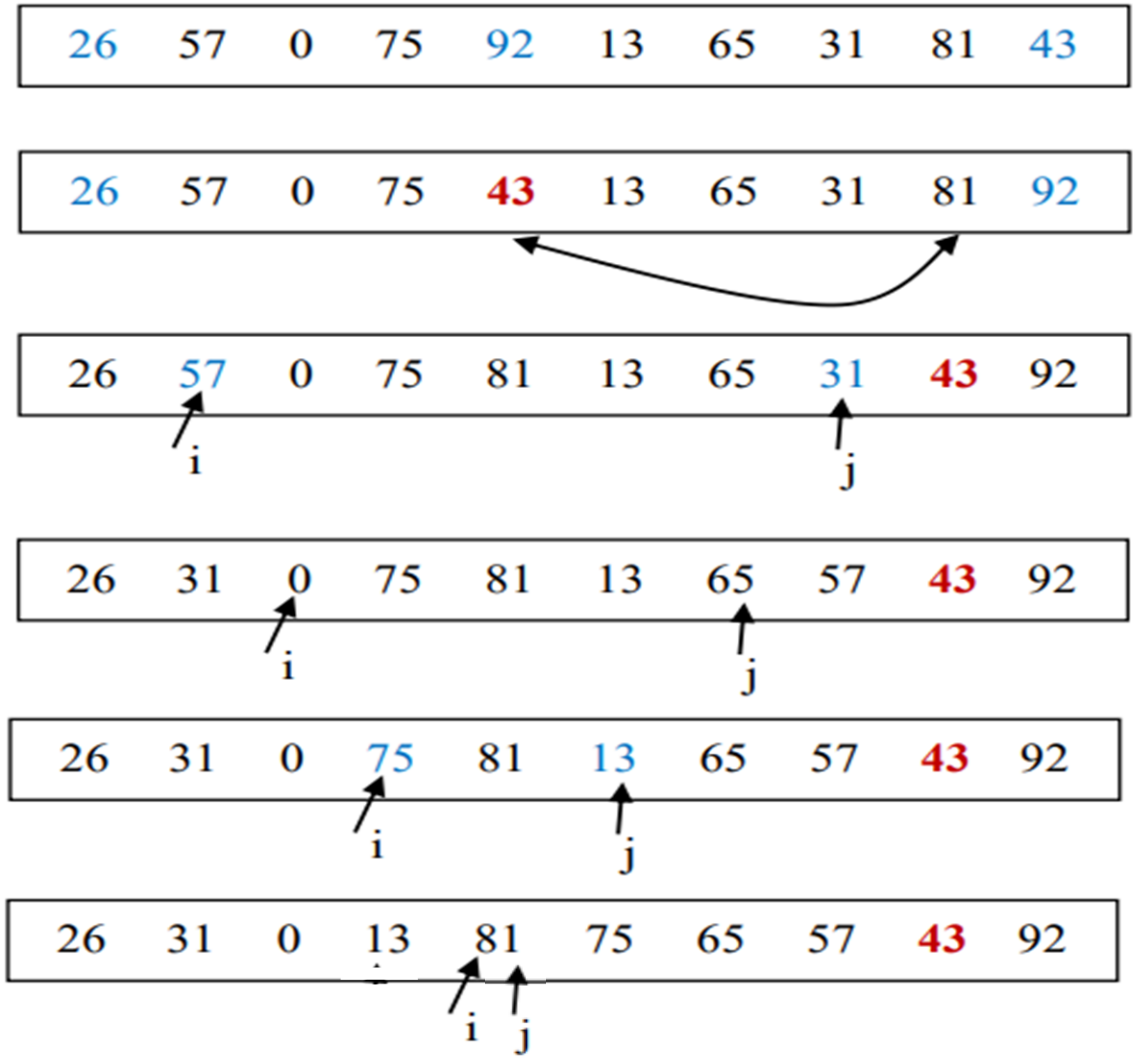
While (Left<right)

moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])



How it works

While (Left<right)

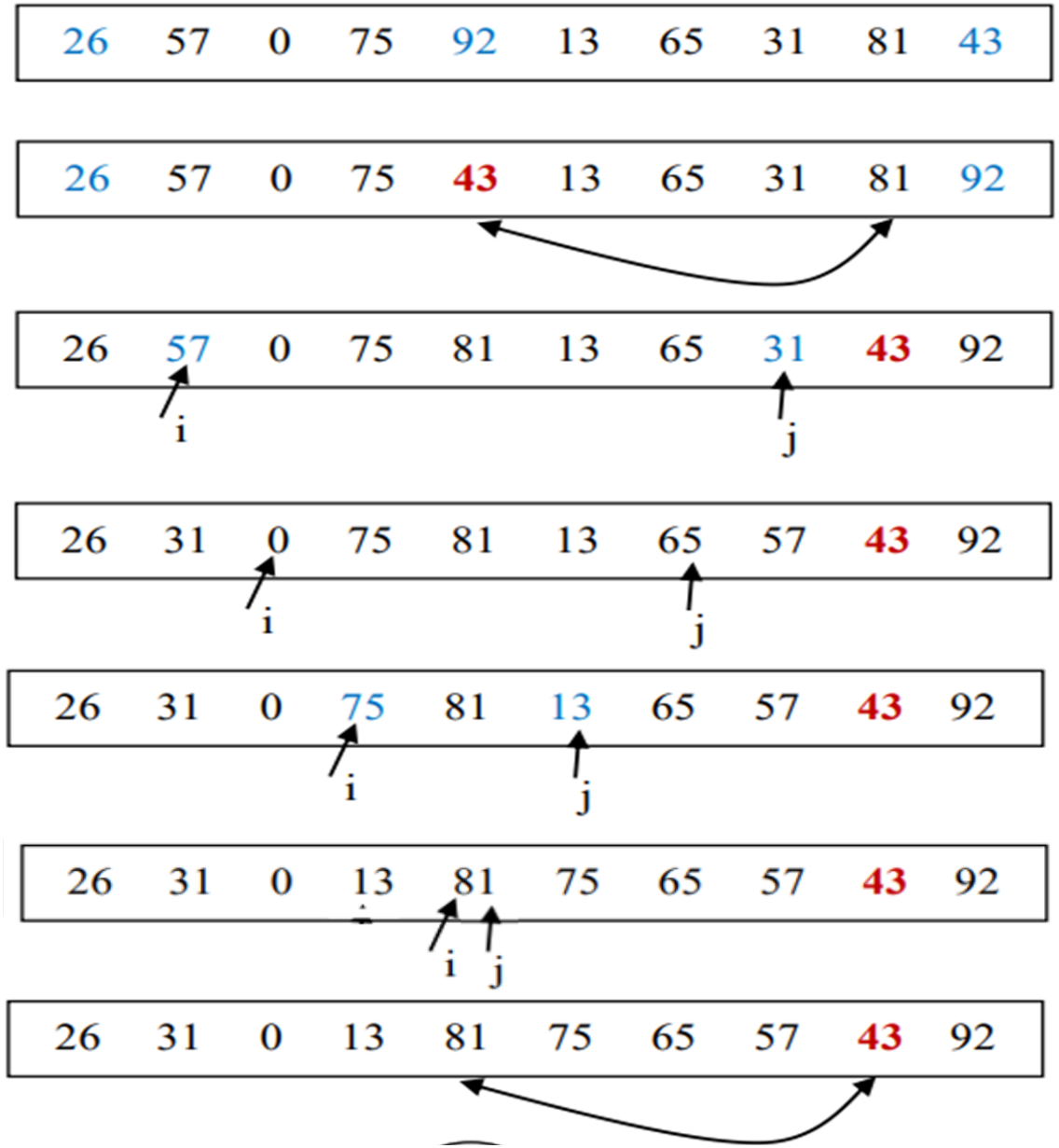
moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])

swap (A[i], A[right-1])



How it works

While (Left<right)

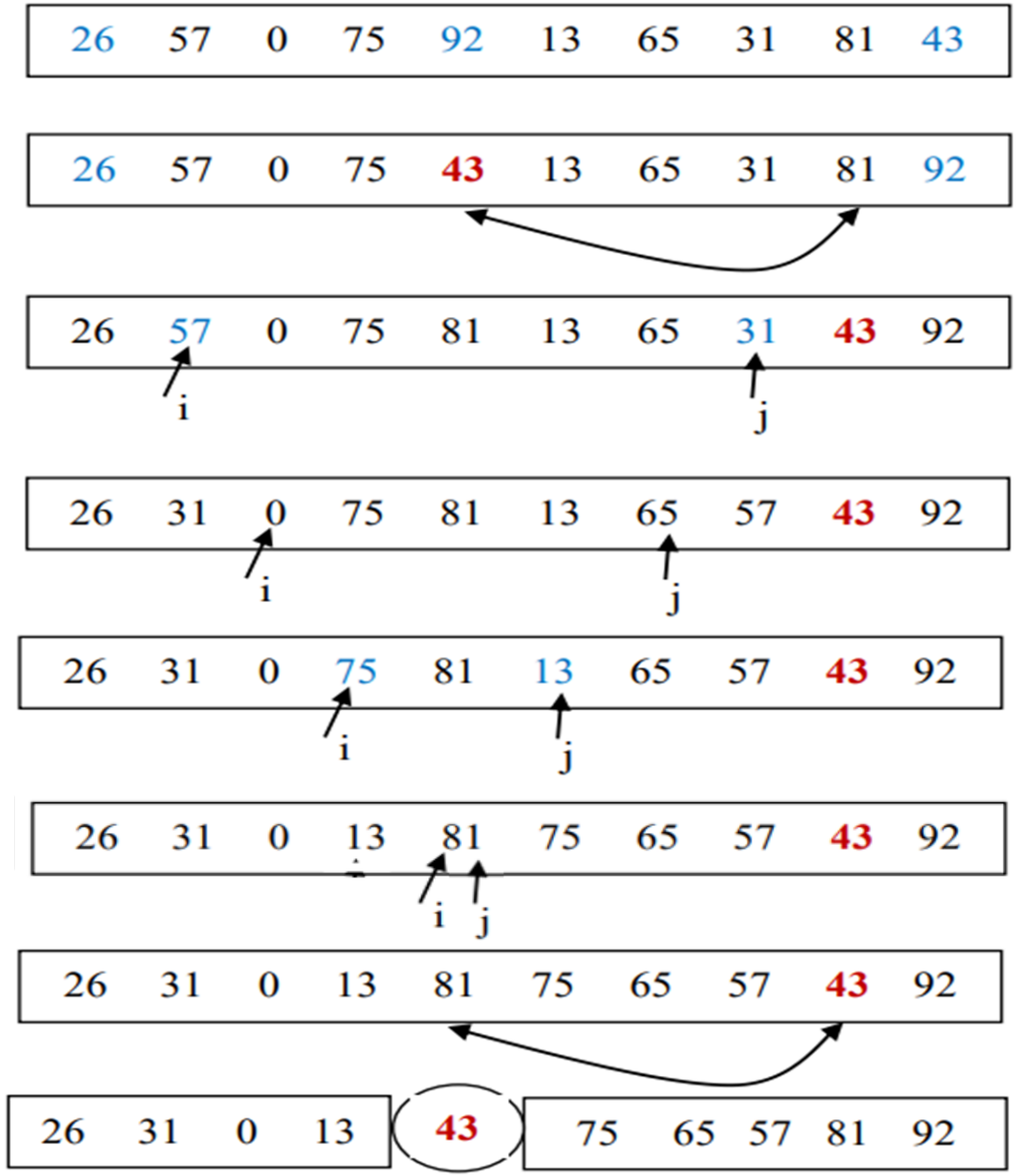
moves i right, skipping over elements smaller than the pivot

moves j left, skipping over elements greater than the pivot

When both i && j have stopped

swap (A[i], A[j])

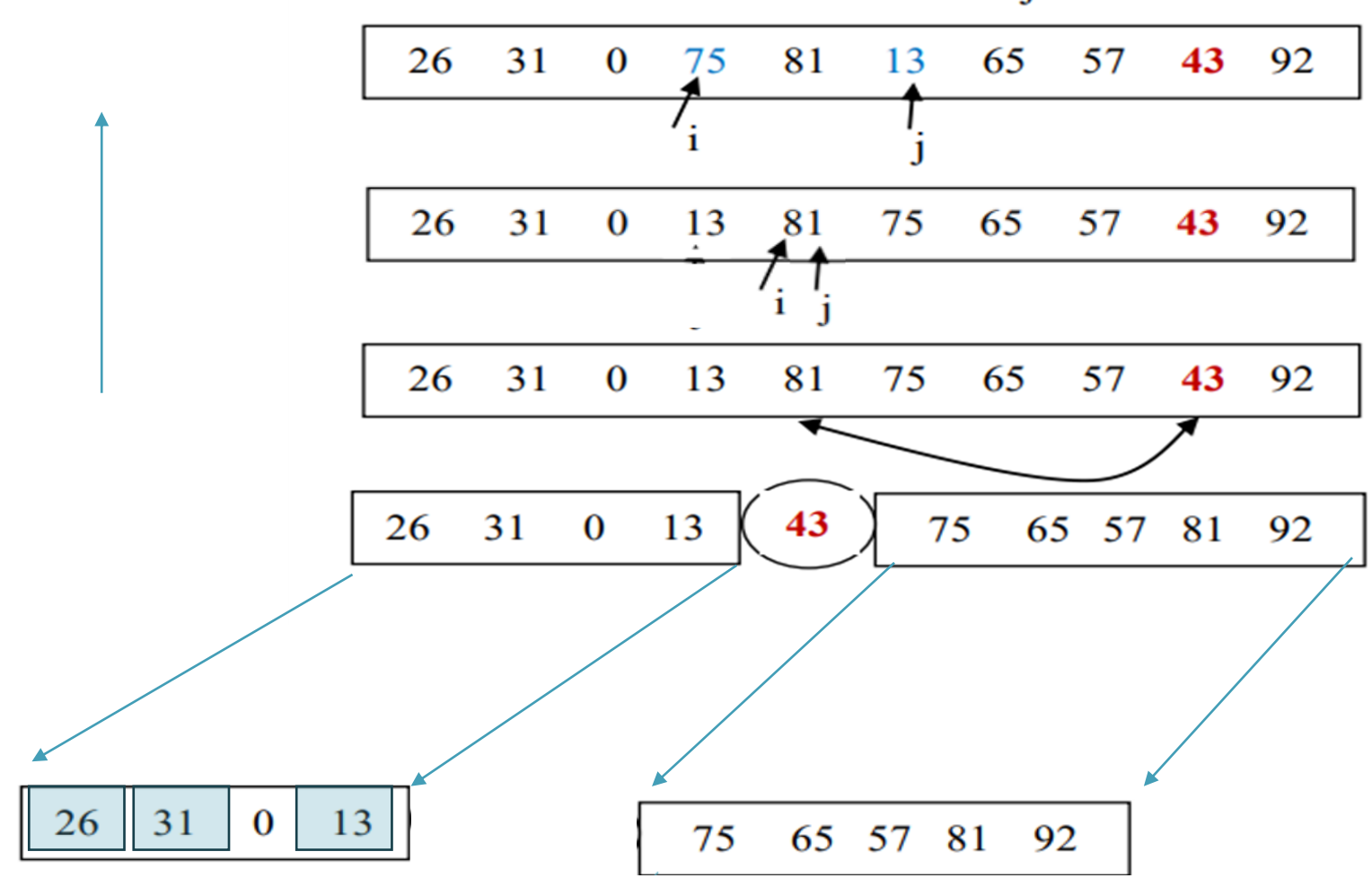
swap (A[i], A[right-1])



To Select Pivot

```
if (A[left]>A[center])
   swap (A[left],A[center])
if (A[left]>A[right])
   swap (A[left],A[right])
if (A[center]>A[right])
  swap (A[center],A[right])
```

Swap (A[center],A[right-1])



Quick_sort

```
void q_sort ( int [] A, int left, int right){
      int i,j,pivot;
      if (left<right){
               pivot=median3 (A,left,right);
               i=left;
               j=right-1;

               for ( ;    ;) {
                     while (A[++i]<pivot);
                     while (A[--j]>pivot);
                        if (i<j)
                           swap (A[i],A[j])
                        else
                            break:

              }
               swap (A[i],Aright-1]);
               q_sort (A,left,i-1);
               q_sort (A,i+1,right);

      }
}
```

```
int median3 ( int [] A, int left, int right){
 int center=(left+right)/2;
 if (A[left]>A[center])
    swap (A[left],A[center])
  if (A[left]>A[right])
     swap (A[left],A[right])
   if (A[center]>A[right])
      swap (A[center],A[right])

    swap (A[center],A[right-1])

    return A[right-1]
}
```

Quick sort

- **Worst case**

  T(n) = T(n-1) + Cn n>1

  .

  .

  .

  O(n^2)

- **Best case**

  T(n) = 2 T(n/2) + Cn

  .

  .

  .

  T(n) = O(n log n)

- **Average case**

  T(n) = T(i) + T(n-i-1) + Cn

  .

  .

  .

  T(n) = O(n log n)

Insertion Sort

Dr. Abdallah Karakra

Example:

We color a sorted part in **green**, and an unsorted part in **black**.

Here is an insertion sort step by step. We take an element from unsorted part and compare it with elements in sorted part, moving form right to left.

29, 20, 73, 34, 64

29, 20, 73, 34, 64
20, 29, 73, 34, 64
20, 29, 73, 34, 64
20, 29, 34, 73, 64
20, 29, 34, 64, 73

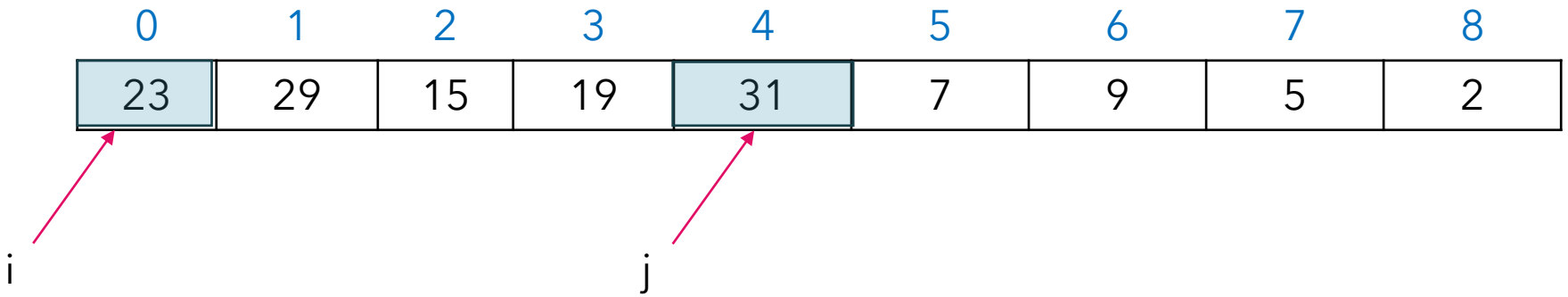
Case Study: Analysis Of Insertion Sort

```
void insertionSort(int[] ar)
{           int index,j;
           for (int i=1; i ‹ ar.length; i++)
          {
                index = ar[i];
                j = i;
                while (j > 0 && ar[j-1] > index)
               {
                    ar[j] = ar[j-1];
                     j--;
               }
                ar[j] = index;
          }
}
```

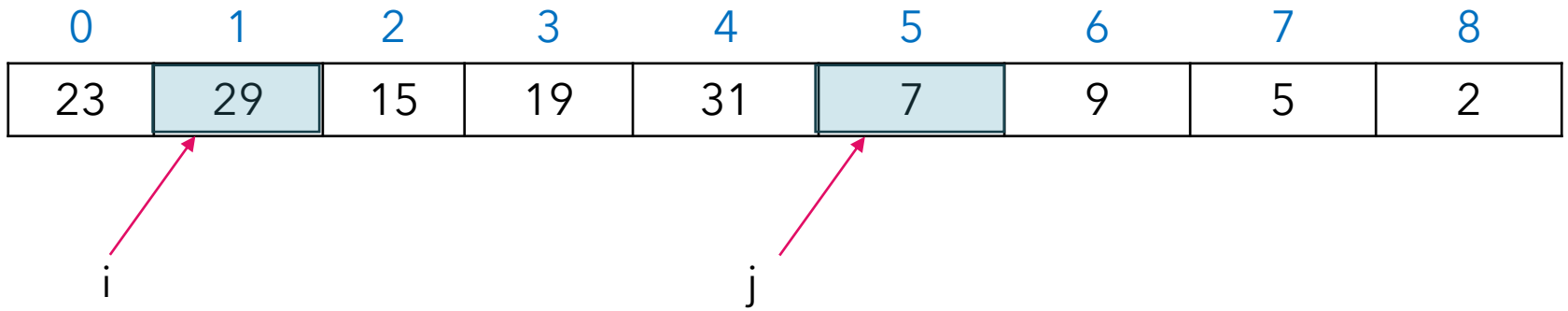
Shell sort

Dr. Abdallah Karakra

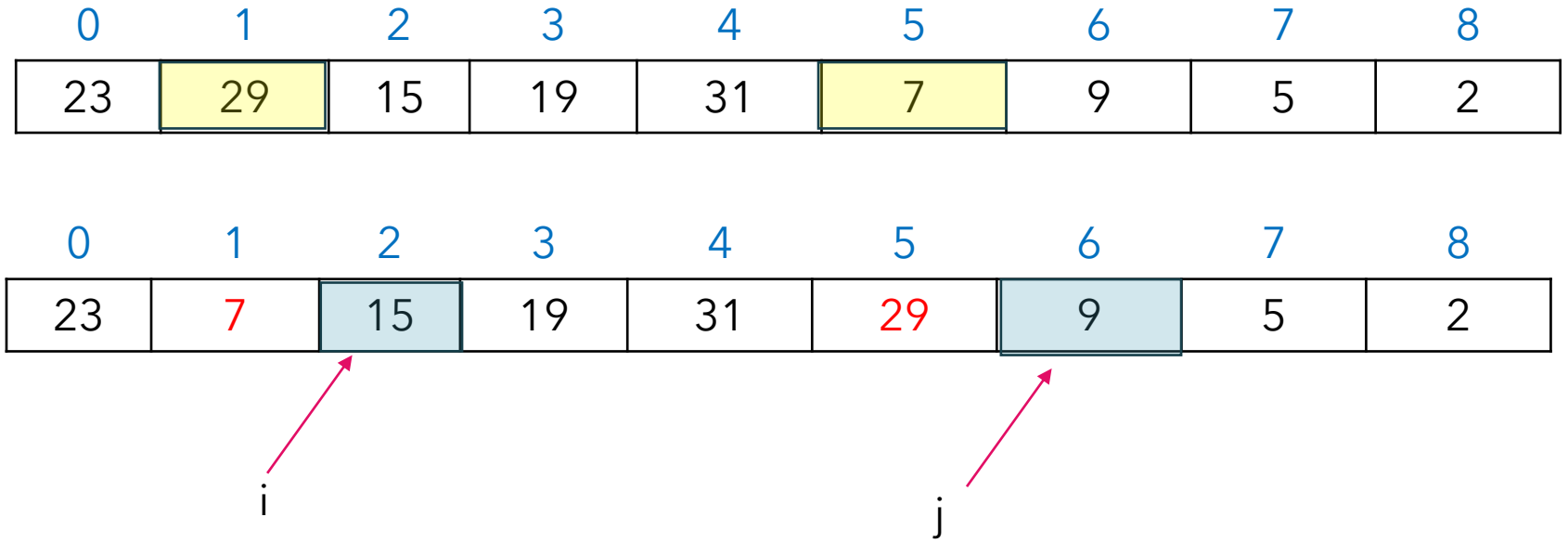
Gap(increment)=N/2
Gap=9/2=4



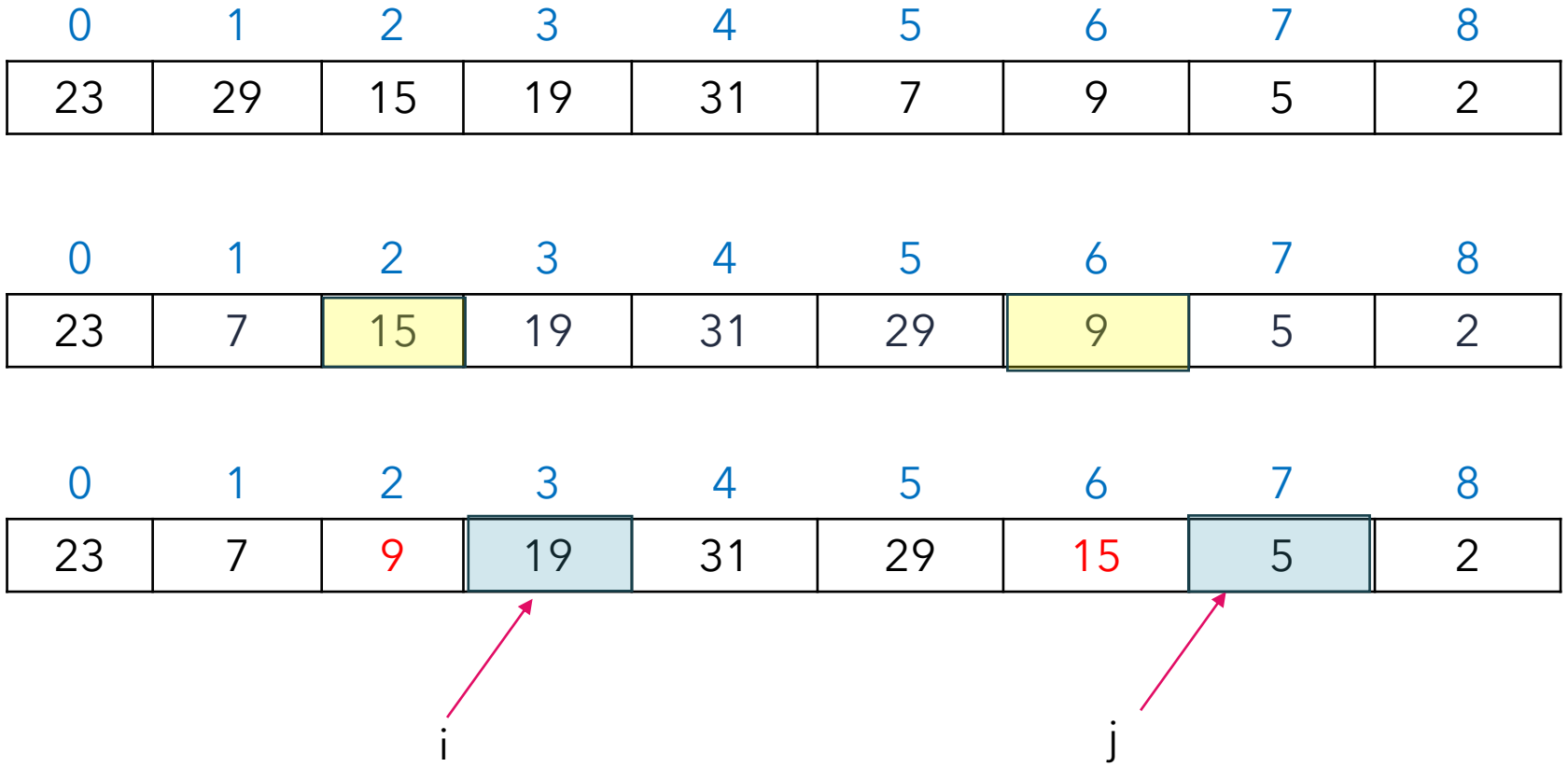
Gap=N/2
Gap=9/2=4



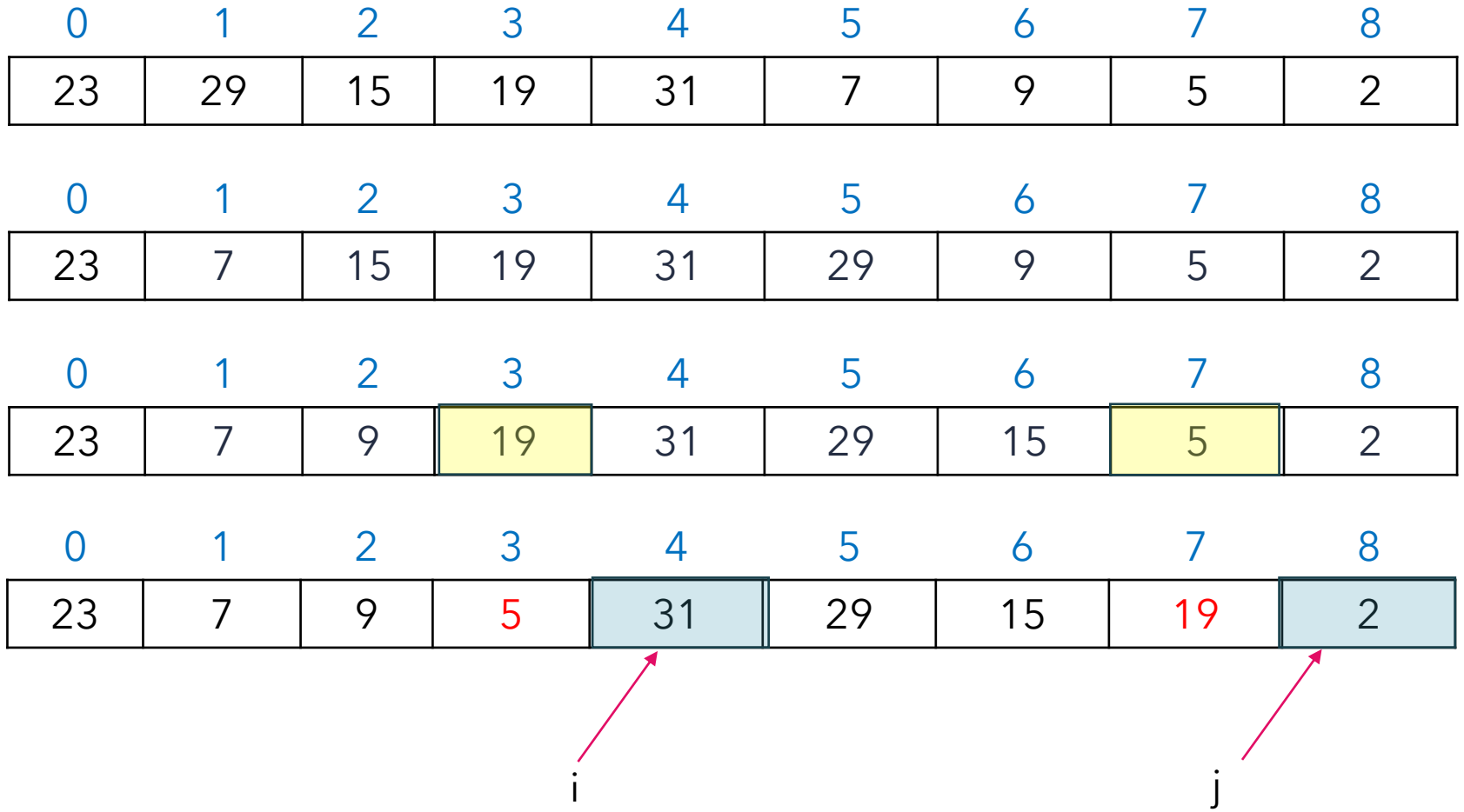
Gap=N/2
Gap=9/2=4



Gap=N/2
Gap=9/2=4



Gap=N/2
Gap=9/2=4



Gap=N/2
Gap=9/2=4

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2

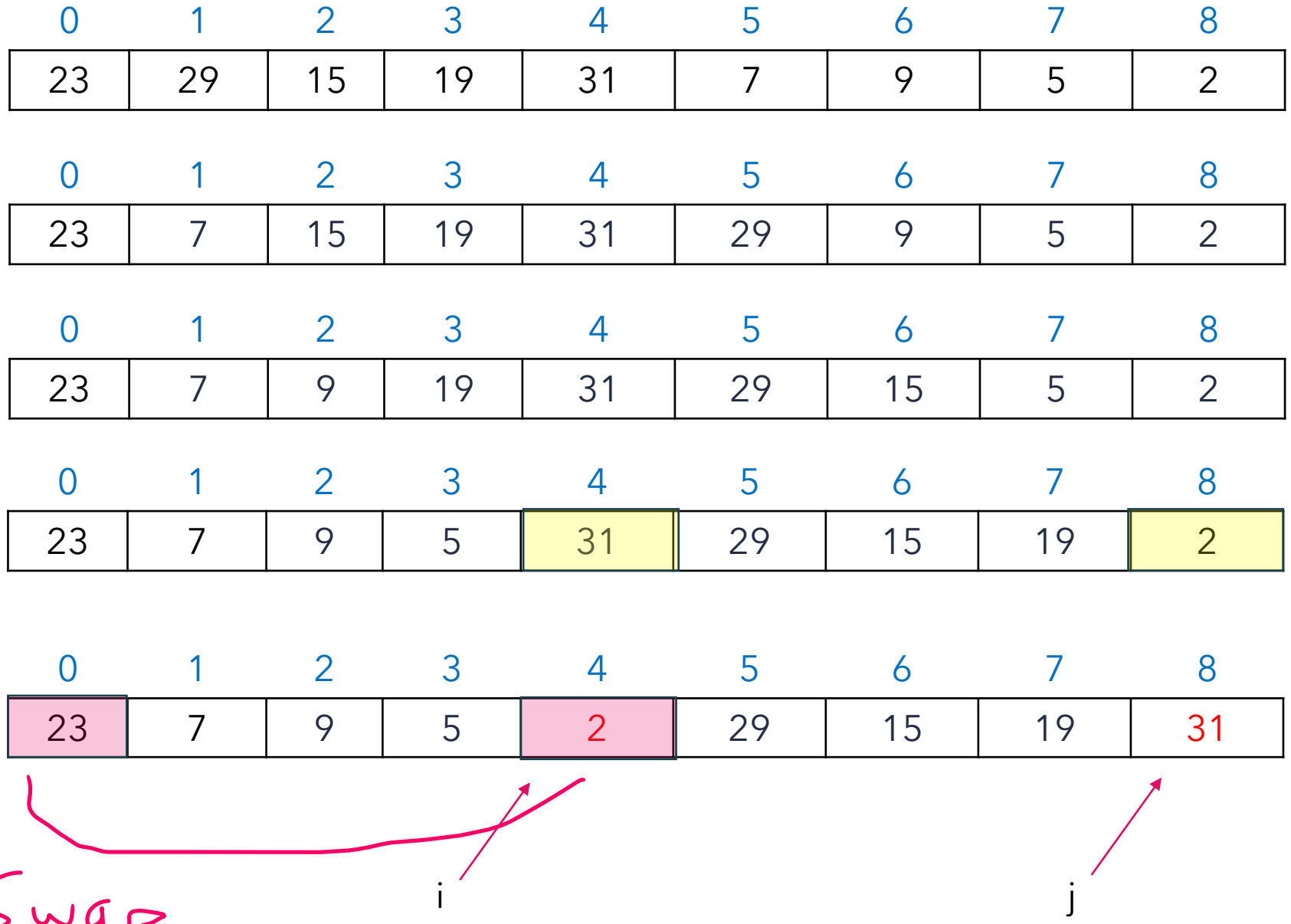
0	1	2	3	4	5	6	7	8
23	7	15	19	31	29	9	5	2

0	1	2	3	4	5	6	7	8
23	7	9	19	31	29	15	5	2

0	1	2	3	4	5	6	7	8
23	7	9	5	31	29	15	19	2

0	1	2	3	4	5	6	7	8
23	7	9	5	2	29	15	19	31

Gap=N/2
Gap=9/2=4



Gap=N/2
Gap=9/2=4

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2

0	1	2	3	4	5	6	7	8
23	7	15	19	31	29	9	5	2

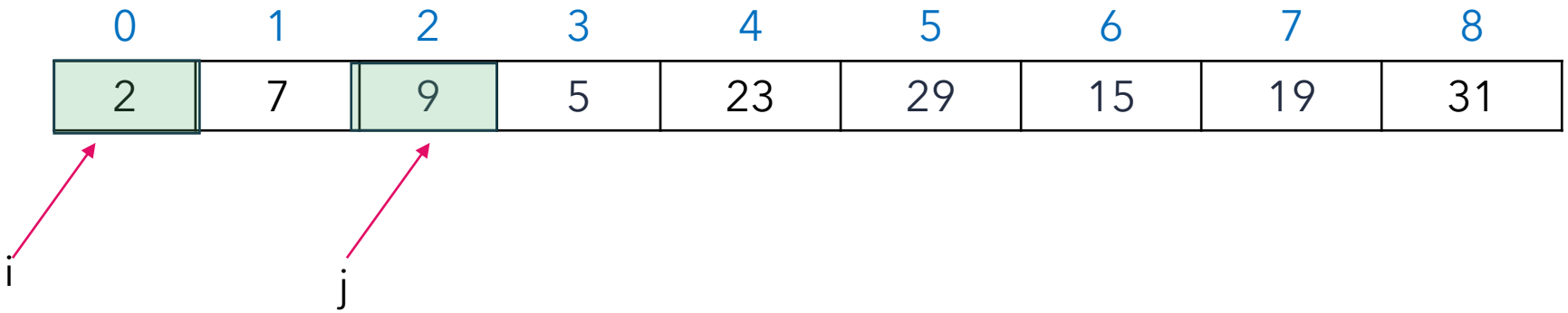
0	1	2	3	4	5	6	7	8
23	7	9	19	31	29	15	5	2

0	1	2	3	4	5	6	7	8
23	7	9	5	31	29	15	19	2

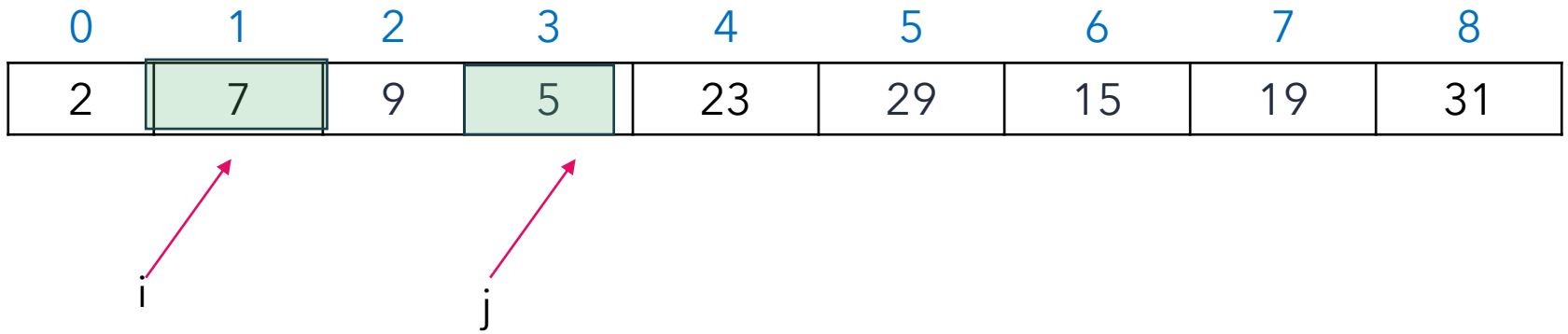
0	1	2	3	4	5	6	7	8
23	7	9	5	2	29	15	19	31

0	1	2	3	4	5	6	7	8
2	7	9	5	23	29	15	19	31

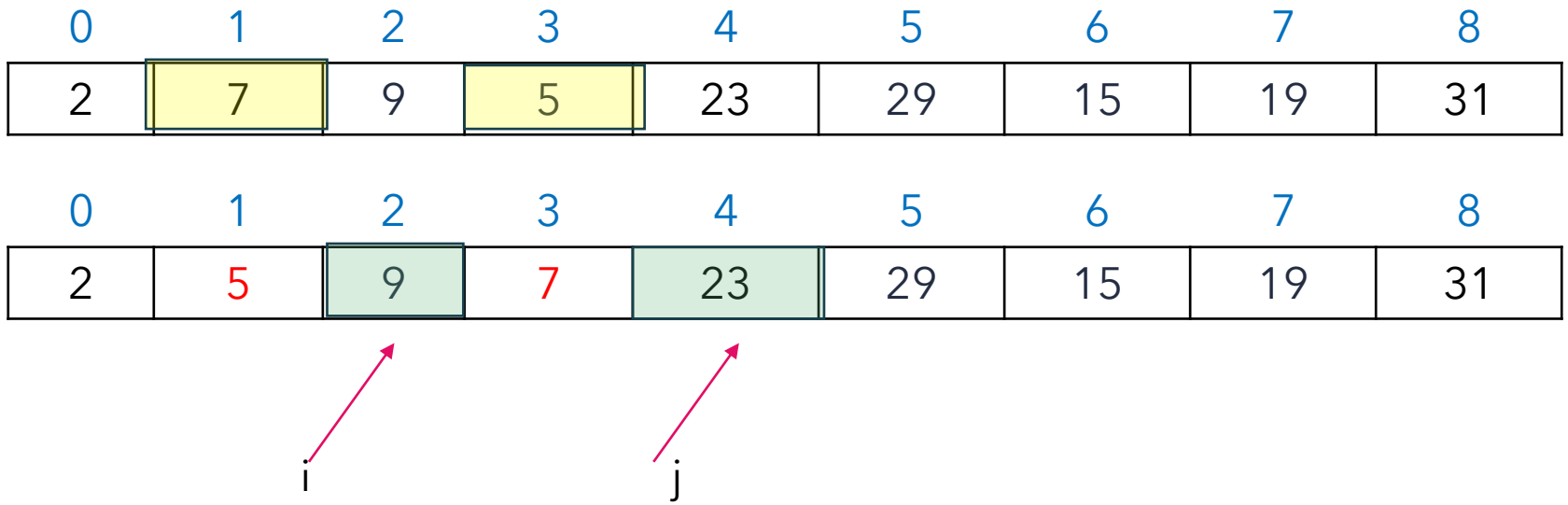
Gap=N/2
Gap=4/2=2



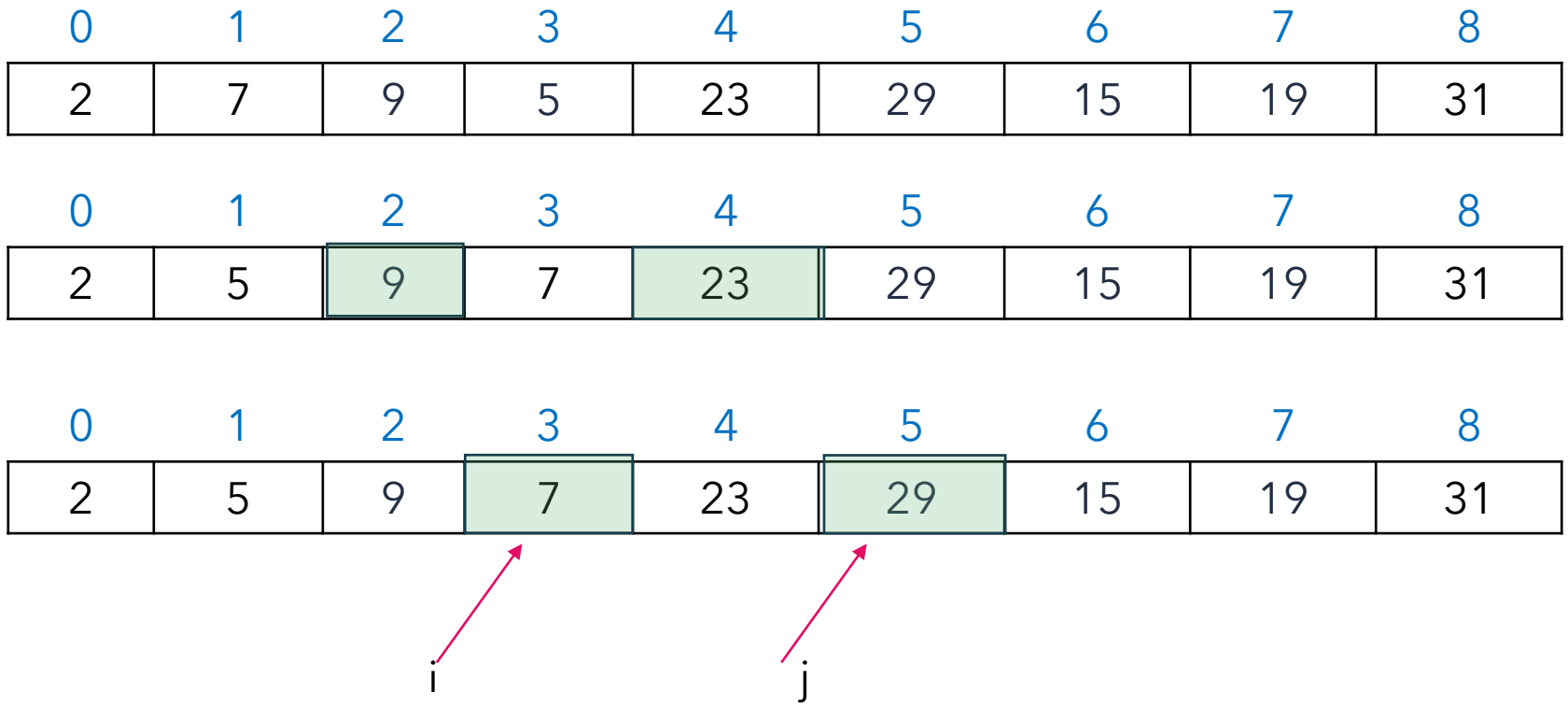
Gap=N/2
Gap=4/2=2



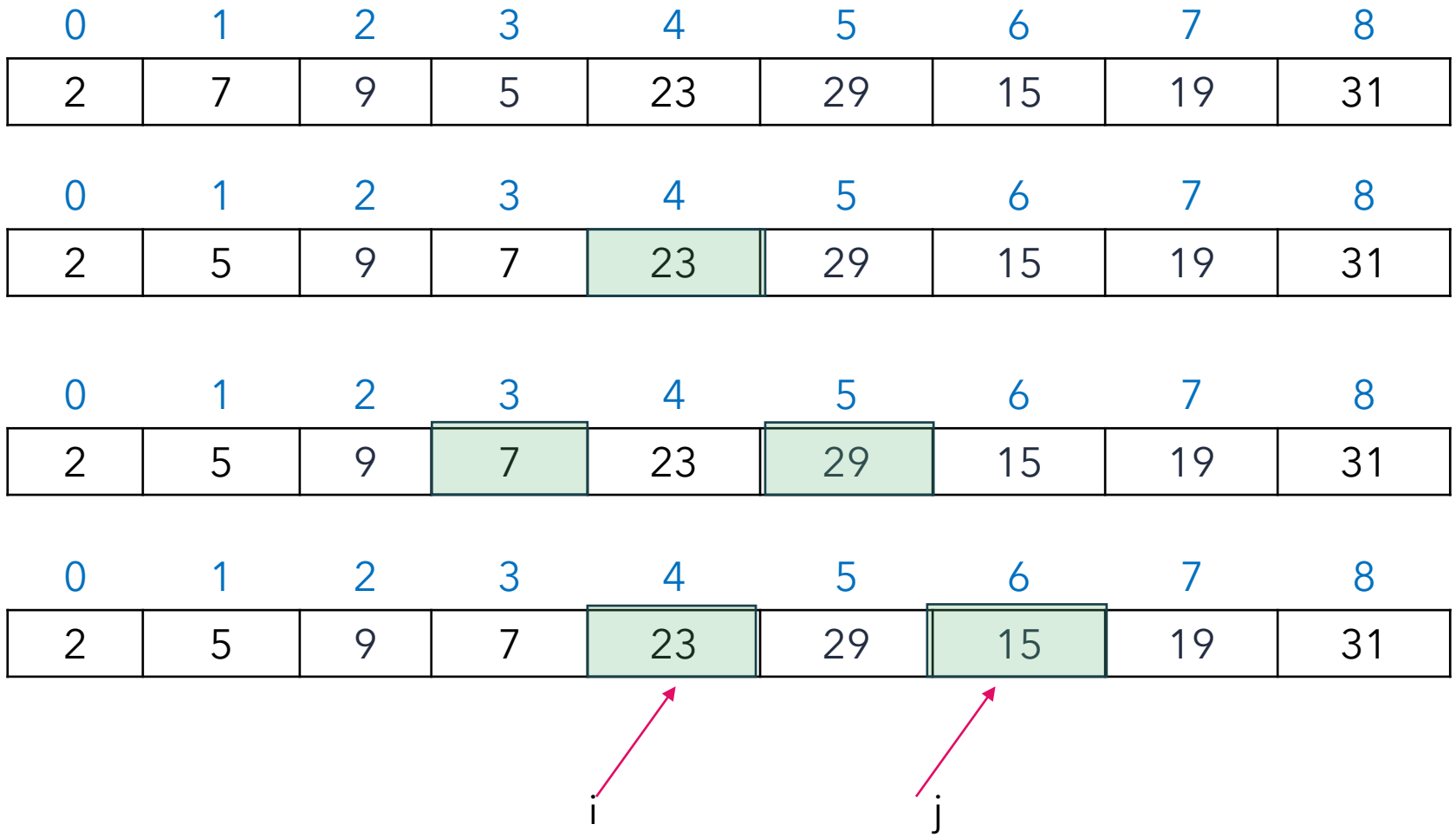
Gap=N/2
Gap=4/2=2



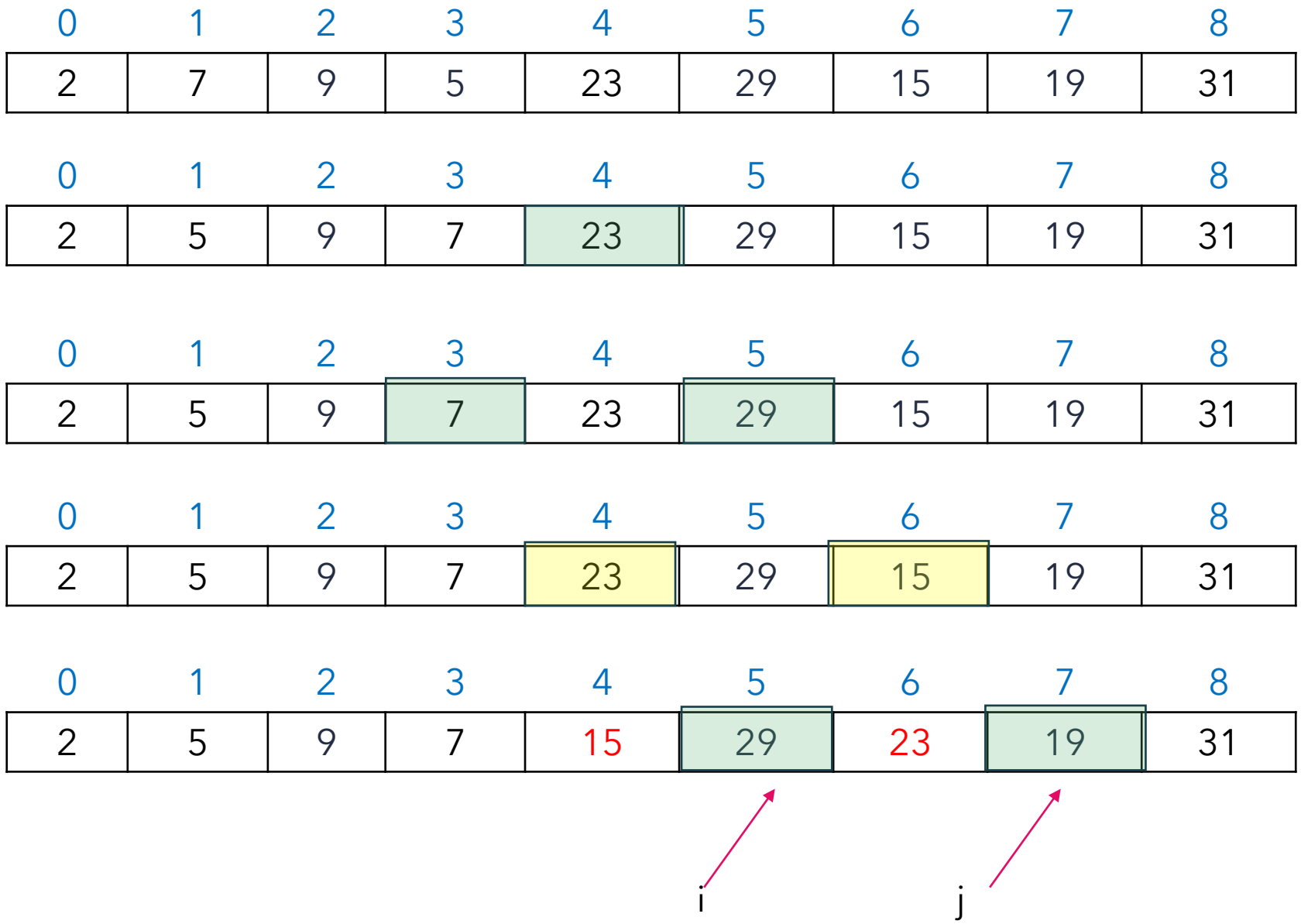
Gap=N/2
Gap=4/2=2



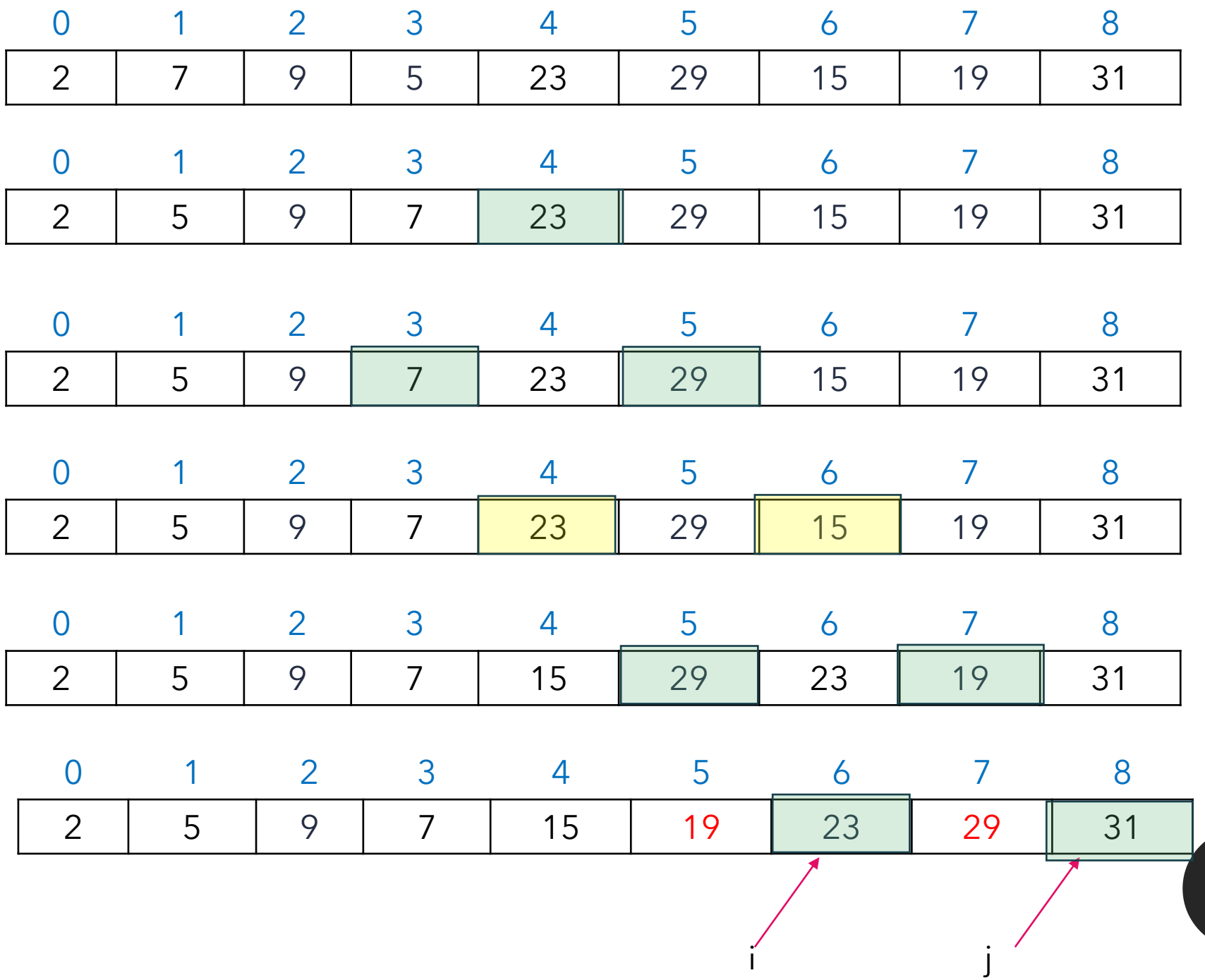
Gap=N/2
Gap=4/2=2



Gap=N/2
Gap=4/2=2

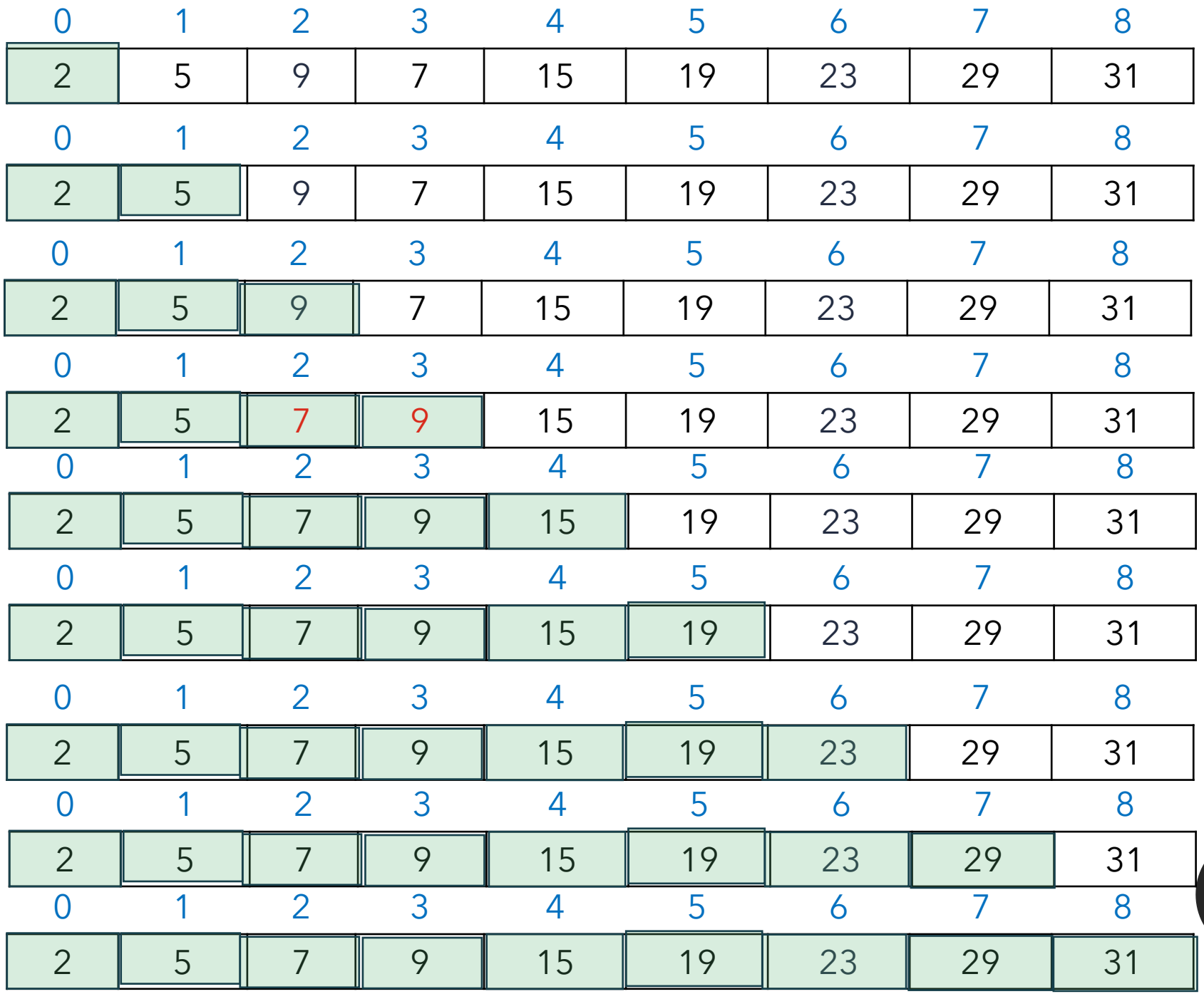


Gap=N/2
Gap=4/2=2



Gap=N/2
Gap=2/2=1

Insertion sort



N=13

Example:

	0	1	2	3	4	5	6	7	8	9	10	11	12
inc\Data	**81**	**94**	**11**	**96**	**12**	**35**	**17**	**95**	**28**	**58**	**41**	**75**	**15**
6	17	94	11	58	12	35	81	95	28	96	41	75	
	15						17						81
	15	94	11	**58**	12	35	17	95	28	96	41	75	81
3	15	12	11	58	94	35							
	15	12	11	17	94	28	58	95	35	96			
					41			94			95	75	
										81			96
	15	**12**	11	17	41	28	58	94	35	81	95	75	96
1	12	15											
	11	12	15	17	41								
					28	41	58	94					
						35	41	58	94				
									81	94	95		
									75	81	94	95	96
	11	12	15	17	28	35	41	58	75	81	94	95	96

13/2

6/2

3/2

Shell sort

```
for (gap=N/2; gap>=1; gap=gap/2){     //No. of passes

   for (j=gap; j<N; j++){
        for (i=j-gap;i>=0;i=i-gap){  //backward
             if (a[i+gap]>a[i]) {
                  break;
             }
             else{
                swap (a[i+gap],a[i])
            }
        }
   }

}
```


Time complexity when gap=N/2 is O(N^2)

External Sorting

Dr. Abdallah Karakra

All the internal sorting algorithms require that the input fit into main memory. There are, however, applications where the input is much too large to fit into memory. For those external sorting algorithms, which are designed to handle very large inputs

The basic external sorting algorithm uses the merge routine from merge sort. Suppose we have four tapes, Ta1, Ta2, Tb1, Tb2, which are two input and two output tapes. Depending on the point in the algorithm, the a and b tapes are either input tapes or output tapes. Suppose the data is initially on Ta1. Suppose further that the internal memory can hold (and sort) m records at a time. A natural first step is to read m records at a time from the input tape, sort the records internally, and then write the sorted records alternately to Tb1 and Tb2. We will call each set of sorted records a run. When this is done, we rewind all the tapes. Suppose we have the same input as our example for Shell sort

T_{a1}	81	94	11	96	12	35	17	99	28	58	41	75	15
T_{a2}													
T_{b1}													
T_{b2}													

If $m = 3$, then after the runs are constructed, the tapes will contain the data indicated in the following figure.

T_{a1}													
T_{a2}													
T_{b1}		11		81		94			17		28		99				15
T_{b2}		12		35		96			41		58		75				

T_{a1}	11	12	35	81	94	96	15						
T_{a2}	17	28	41	58	75	99							
T_{b1}													
T_{b2}													

T_{a1}													
T_{a2}													
T_{b1}	11	12	17	28	35	51	58	75	81	94	96	99	
T_{b2}	15												

T_{a1}	11	12	15	17	28	35	41	58	75	81	94	96	99
T_{a2}													
T_{b1}													
T_{b2}													

Summary

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(n)