GLS UNIVERSITY

0301302 DATA STRUCTURES.
UNIT– V

Searching

- Searching means to find whether a particular value is present in an array or not.
- If the value is present in the array, then searching is said to be successful and the searching process gives the location of that value in the array.
- There are two searching methods:
    - **Linear Search**
    - **Binary Search**

Linear Search

- Linear search, also called as sequential search, is a very simple method for searching an array for a particular value.
- It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found.
- Linear search is mostly used to search an unordered list of elements.

Linear Search Algorithm

Step 1 : SET POS =-1

Step 2 : SET I =1

Step 3 : Repeat Step 4 while I<=N

Step 4 : If A[I] = VAL

SET POS = I

PRINT POS

Go to Step 6

[End of If]

SET I = I + 1

Step 5 : IF POS = -1

PRINT “VALUE is NOT PRESENT”

Step 6: EXIT

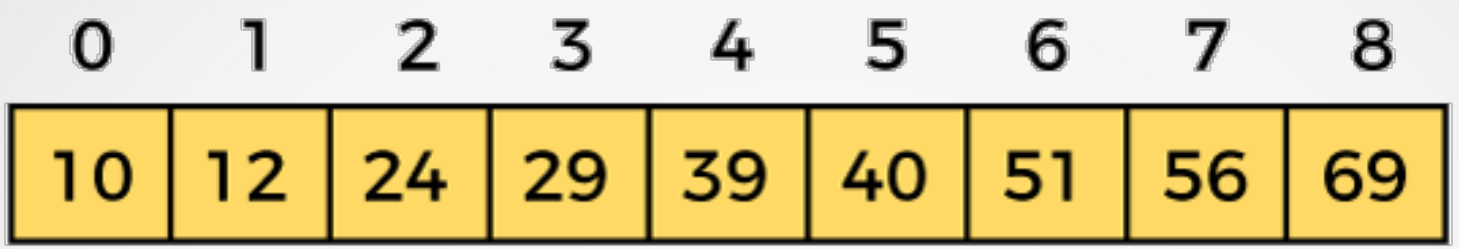
Binary Search

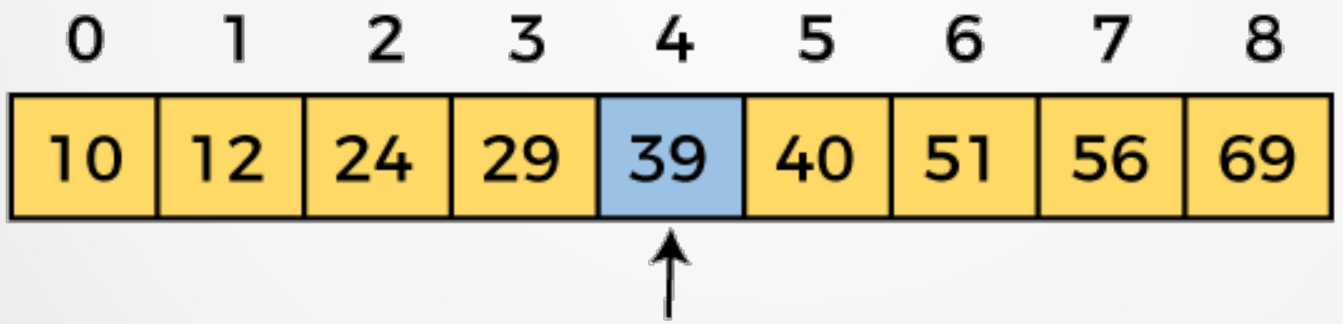
- Binary search is a searching algorithm that works efficiently with a sorted list.
- The mechanism of binary search can be better understood by an analogy of a telephony directory or dictionary.

Binary Search Algorithm

Step 1 : SET BEG = lower_bound, END= upper_bound, POS= -1
Step 2: Repeat Step3 and Step 4 while BEG<=END
Step 3 : SET MID = (BEG + END)/2
Step 4 : IF A[MID] = VAL
SET POS = MID
PRINT POS
Go to Step 6
ELSE IF A[MID] >VAL
SET END =MID – 1
ELSE
SET BEG = MID +1
[END OF IF]
[END of LOOP]
Step 5 : IF POS = -1
PRINT “Value is not present”
[END OF IF]
Step 6 : EXIT

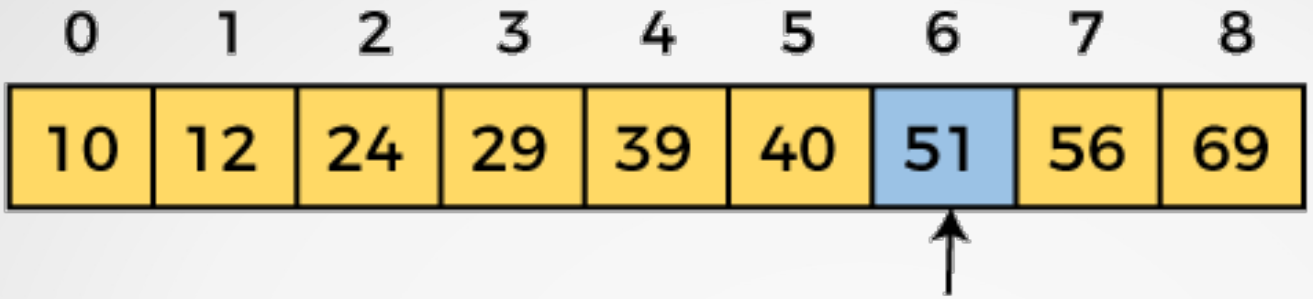
Binary Search



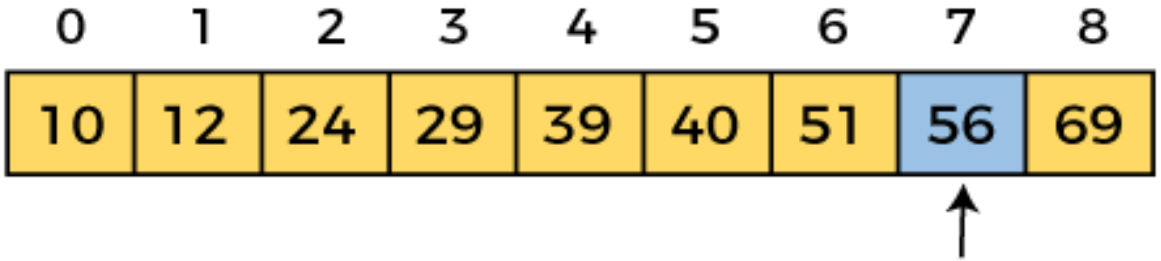


A[mid] = 39
A[mid] < K (or,39 < 56)
So, beg = mid + 1 = 5, end = 8
Now, mid =(beg + end)/2 = 13/2 = 6

Binary Search



A[mid] = 51
A[mid] < K (or, 51 < 56)
So, beg = mid + 1 = 7, end = 8
Now, mid =(beg + end)/2 = 15/2 = 7



A[mid] = 56
A[mid] = K (or, 56 = 56)
So, location = mid
Element found at 7th location of the array

Sorting

- A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements.
- Sorting method can be implemented in different ways - by selection, insertion method, or by merging.

Bubble Sort

- In Bubble sort, Each element of the array is compared with its adjacent element.
- A list with n elements requires n-1 passes for sorting.
- Consider an array A of n elements whose elements are to be sorted by using Bubble sort.

Bubble Sort

The algorithm processes like following:

- In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
- In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
- In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Bubble Sort

- Step 1: Repeat Step 2 For i = 0 to N-1
- Step 2: Repeat For J = 0 to N - i
- Step 3: IF A[J] > A[J+1]
- SWAP A[J] and A[J+1]

  [END OF INNER LOOP]

  [END OF OUTER LOOP
- Step 4: EXIT

Bubble Sort

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 6	0	1	2	**3**					
		1	**2**						

Selection Sort

- In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.
- First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

Selection Sort

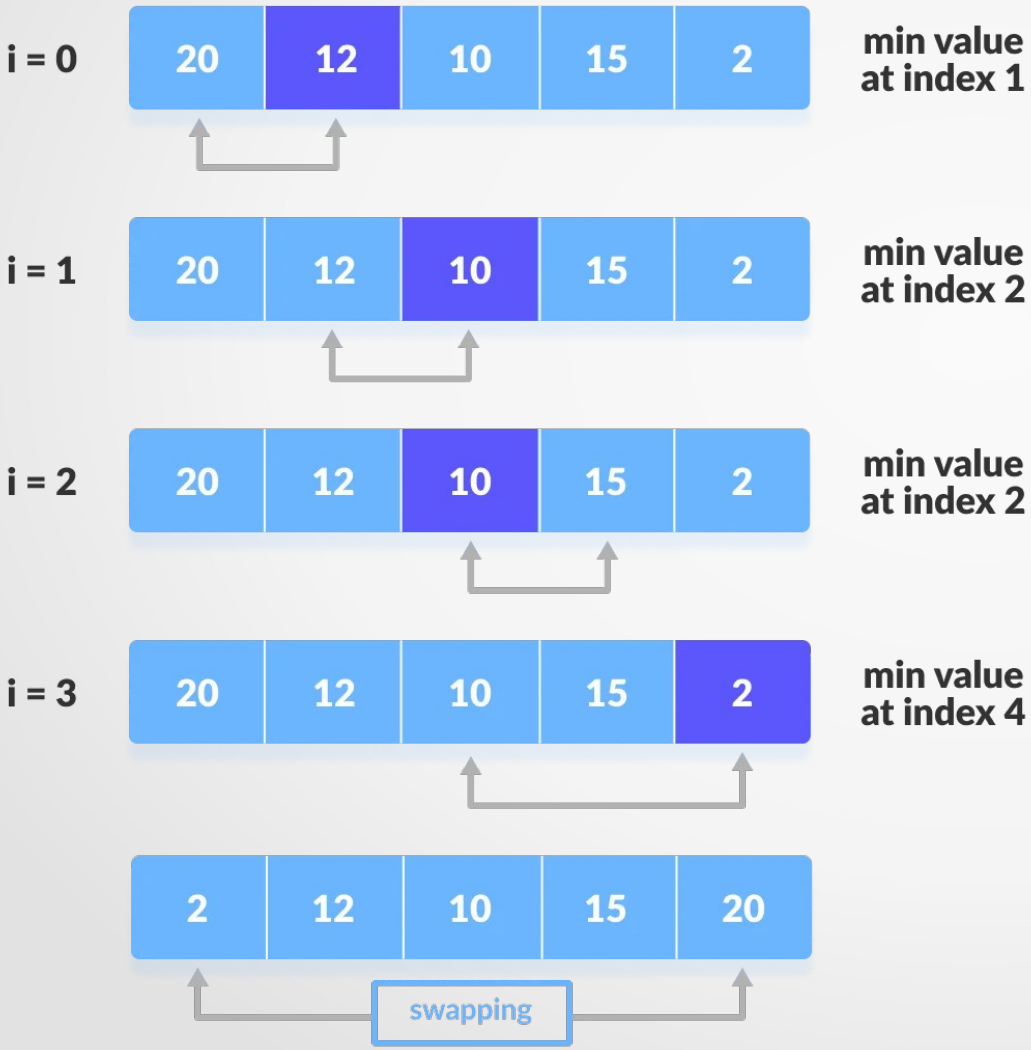
The array with n elements is sorted by using n-1 pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index pos. then, swap A[0] and A[pos]. Thus A[0] is sorted, we now have n -1 elements which are to be sorted.
- In 2nd pas, position pos of the smallest element present in the sub-array A[n-1] is found. Then, swap, A[1] and A[pos]. Thus A[0] and A[1] are sorted, we now left with n-2 unsorted elements.
- In n-1th pass, position pos of the smaller element between A[n-1] and A[n-2] is to be found. Then, swap, A[pos] and A[n-1].

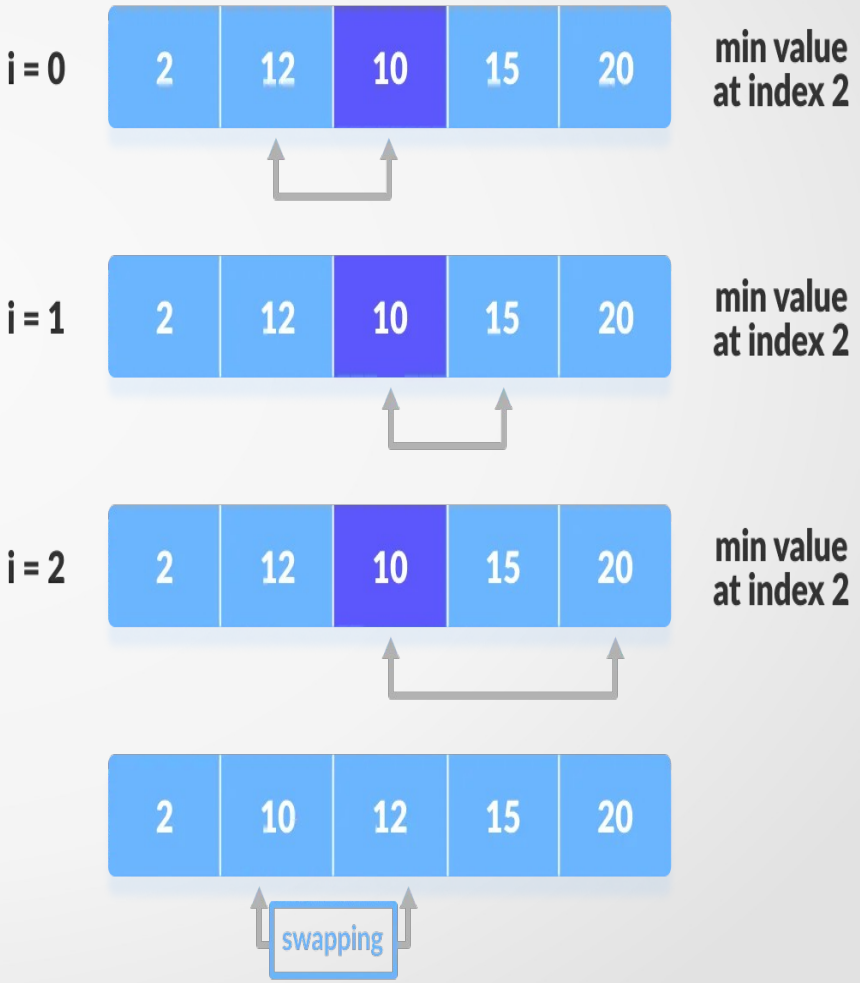
Therefore, by following the above explained process, the elements A[0], A[1], A[2],...., A[n-1] are sorted.

Selection Sort

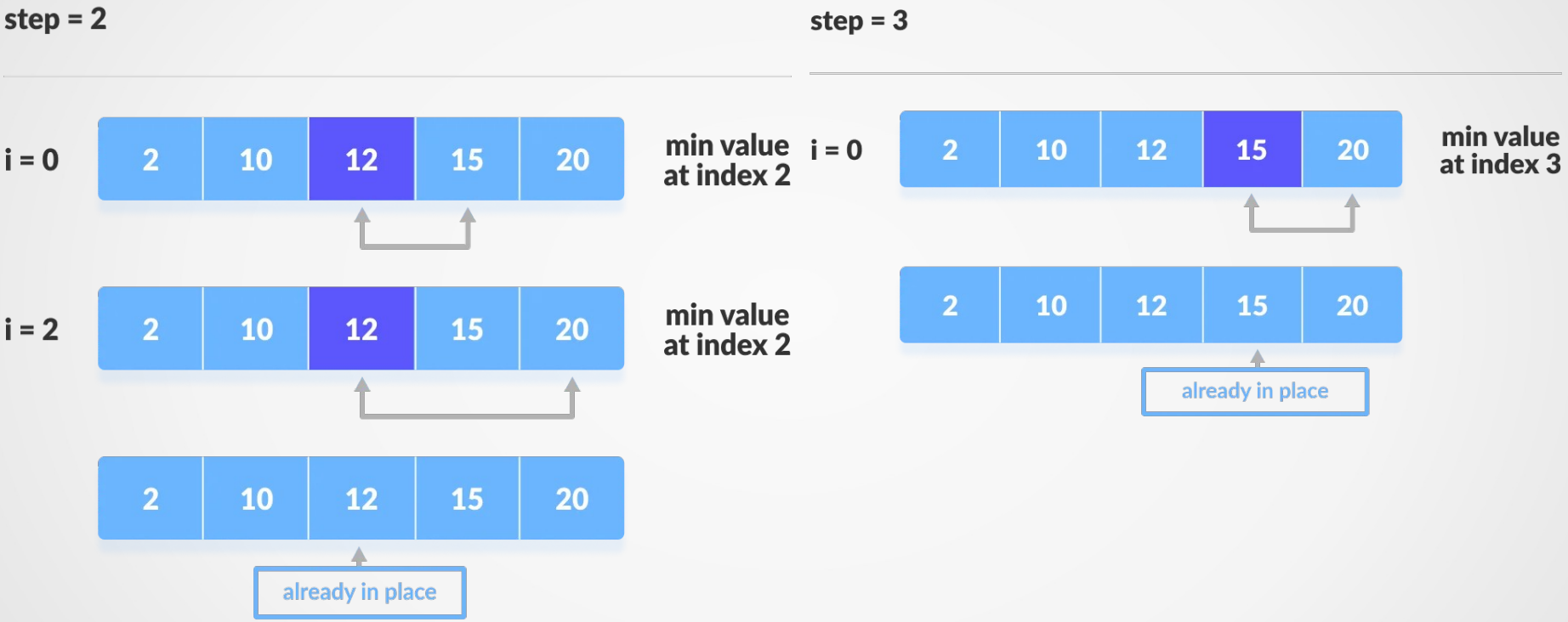
step = 0



step = 1



Selection Sort



Selection Sort

Step 1: For i = 1 to n-1
step 2: Set min = arr[i]
step 3: Set position = i
step 4: For j = i+1 to n-1 repeat:
if (min > arr[j])
Set min = arr[j]
Set position = j
[end of if]
[end of loop]
step 5: swap a[i] with arr[position]
[end of loop]
step 6: END

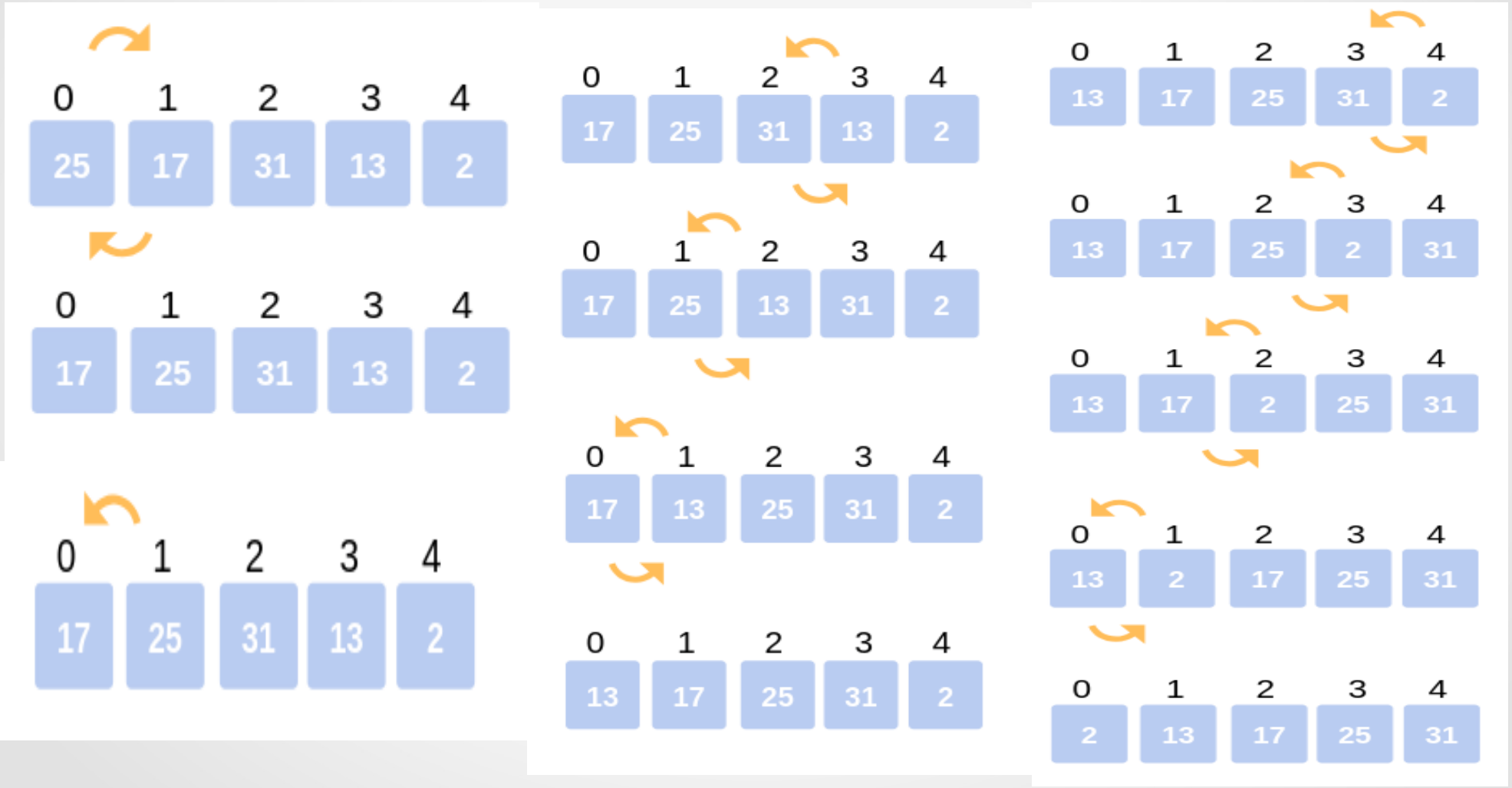
Insertion Sort

- Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.
- Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position i.e, the position to which it belongs in a sorted array.

Insertion Sort

Step 1: For i = 1 to n-1
Step 2: Set temp = array[i]
Step 3: Set j = i-1
Step 4: while temp <= array[j]
Set array[j + 1] = array[j]
Set j = j-1
Step 5: Set array[j + 1] = temp
Step 6: END

Insertion Sort



Merge Sort

- Merge sort is a sorting algorithm that uses divide, conquer and combine algorithmic paradigm.
- **Divide** means partitioning the n-element array to be sorted into two sub-arrays of n/2 elements.
- If there are more than one elements in the array, divide A into two sub-arrays, A1 and A2, each containing about half of the elements of A.
- **Conquer** means sorting the two sub-arrays recursively using merge sort.
- **Combine** means merging the two sorted sub-arrays of size n/2 to produce the sorted array of n elements.

Merge Sort

- A smaller list takes fewer steps and thus less time to sort than a large list.
- As number of steps is relatively less,thus less time is needed.
- If the array is of length 0 or 1, then it is already sorted.
- Otherwise, divide the unsorted array into two sub-arrays of about half of the size.
- Use merge sort algorithm to sort each sub-array.
- Merge the two sub-arrays to form a single sorted list.

Merge Sort

MergeSort(arr[], l, r)

If l < r

1. Find the middle point to divide the array into two halves:

   middle m = (l+r)/2

2. Call mergeSort for first half:

   Call mergeSort(arr, l, m)

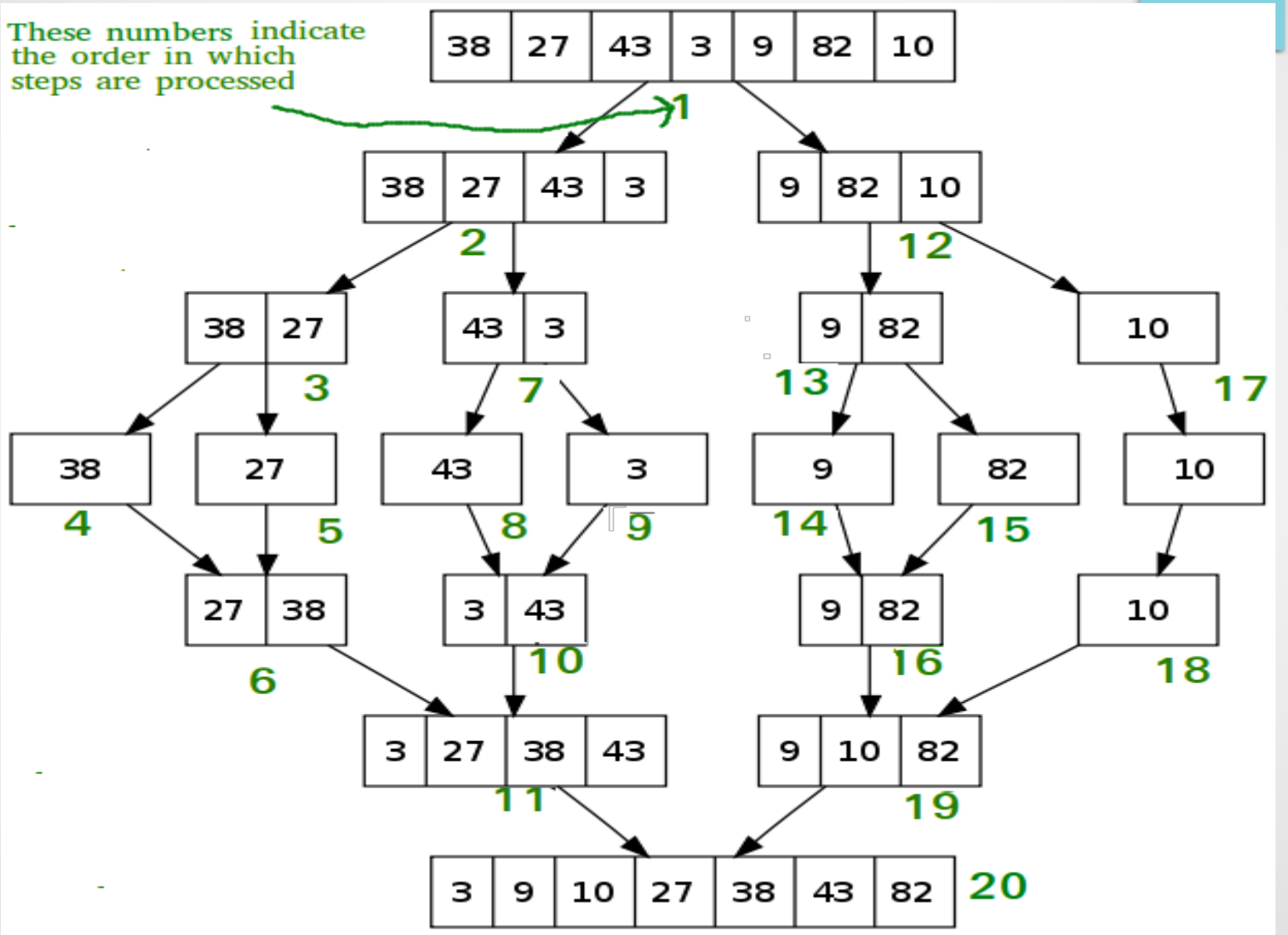
3. Call mergeSort for second half:

   Call mergeSort(arr, m+1, r)

4. Merge the two halves sorted in step 2 and 3:

   Call merge(arr, l, m, r)

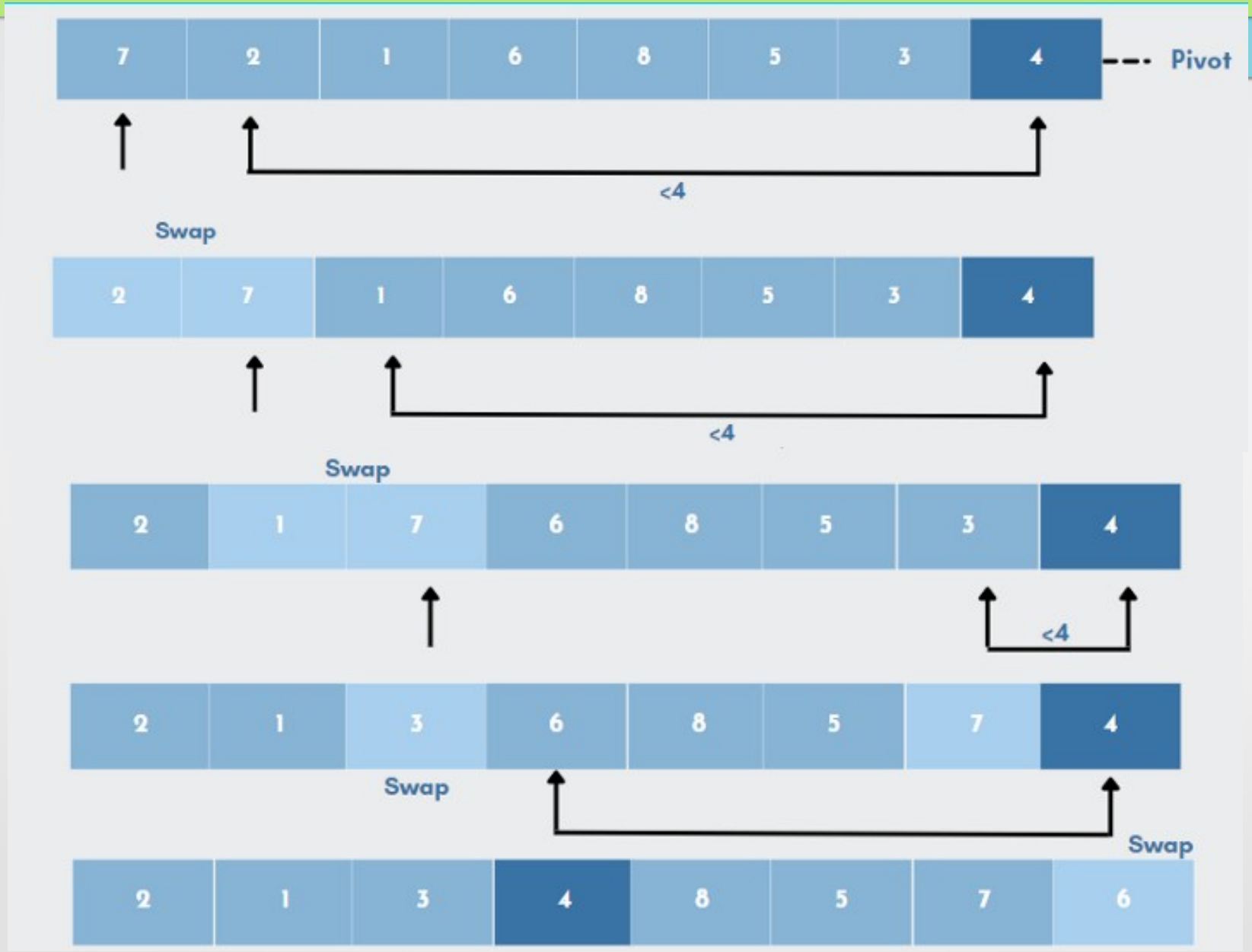
Merge Sort


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20

Quick Sort

- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.
- A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quick Sort

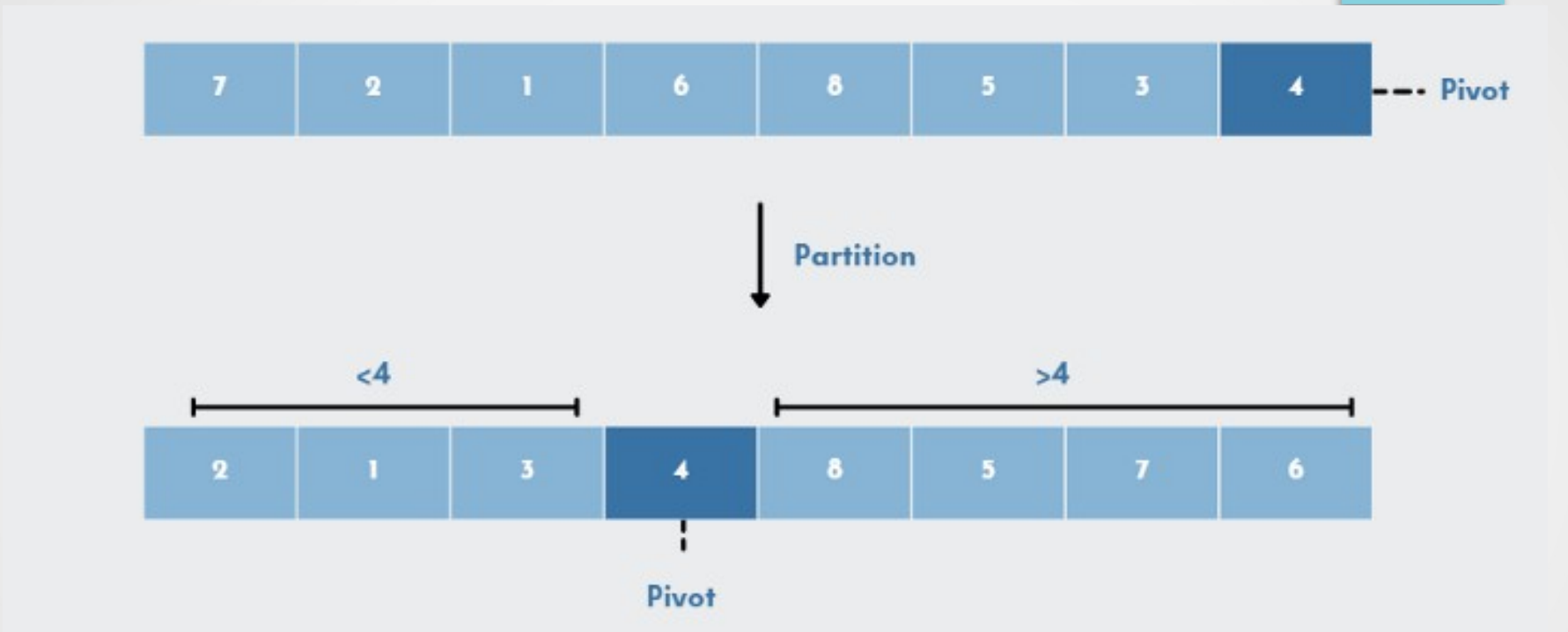

7 2 1 6 8 5 3 4
Pivot
<4
Swap
2 7 1 6 8 5 3 4
<4
Swap
2 1 7 6 8 5 3 4
<4
2 1 3 6 8 5 7 4
Swap
Swap
2 1 3 4 8 5 7 6

Quick Sort

Let’s simplify the above example,

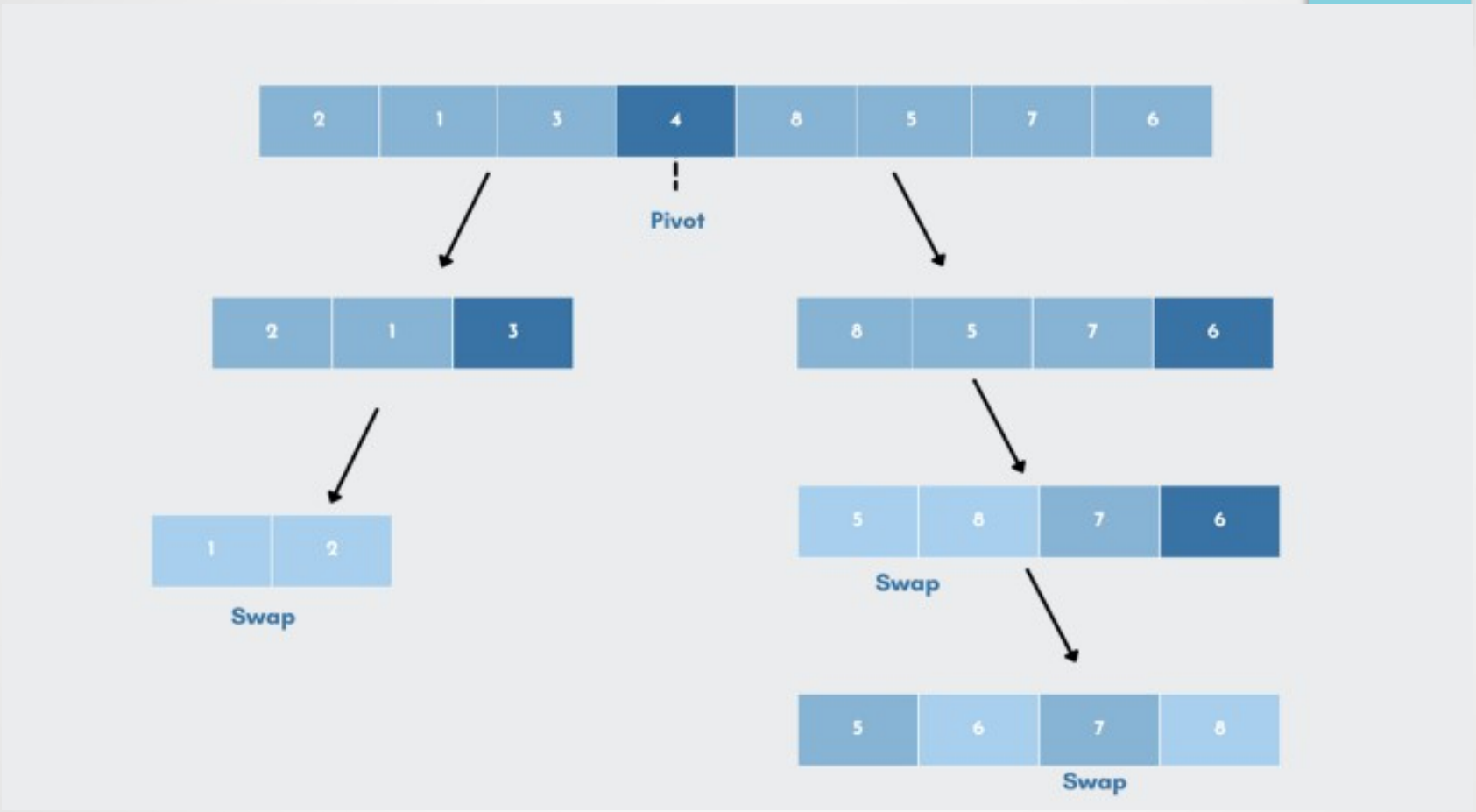
- Every element, starting with 7, will be compared to the pivot(4). A second pointer will be placed at 7 because 7 is bigger than 4.
- The next element, element 2 will now be compared to the pivot. As 2 is less than 4, it will be replaced by the bigger figure 7 which was found earlier.
- The numbers 7 and 2 are swapped. Now, pivot will be compared to the next element, 1 which is smaller than 4.
- So once again, 7 will be swapped with 1.
- The procedure continues until the next-to-last element is reached, and at the end the pivot element is then replaced with the second pointer. Here, number 4(pivot) will be replaced with number 6.

Quick Sort



Pivot in its sorted position

Quick Sort



Sorting the sub-arrays

Quick Sort

Quick Sort Algorithm:

Step 1 − Array[Right] = pivot

Step 2 − Apply partition algorithm over data items using pivot element

Step 3 − quicksort(left of pivot)

Step 4 − quicksort(right of pivot)

Quick Sort

Partition Algorithm:

Step 1: Choose the highest index value i.e. the last element of the array as a pivot point

Step 2: Point to the 1st and last index of the array using two variables.

Step 3: Left points to the low index and Right points to the high

Step 4: while Array[Left] < pivot

Move Right

Step 5: while Array[Right] > pivot

Move Left

Step 6: If no match found in step 5 and step 6, swap Left and Right

Step 7: If Left ≥ Right, their meeting point is the new pivot