Sorting and Algorithm

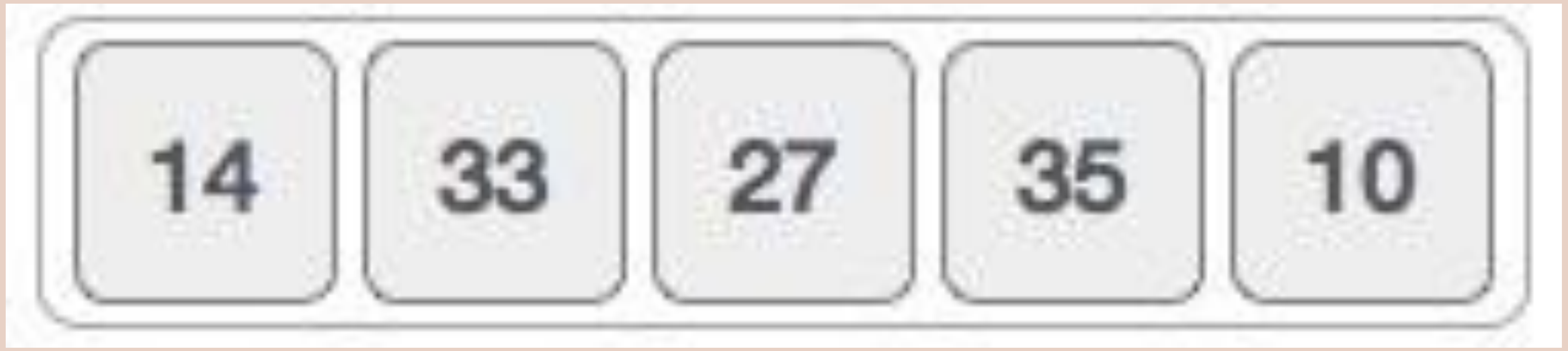
GROUP REPORT

Bubble Sort

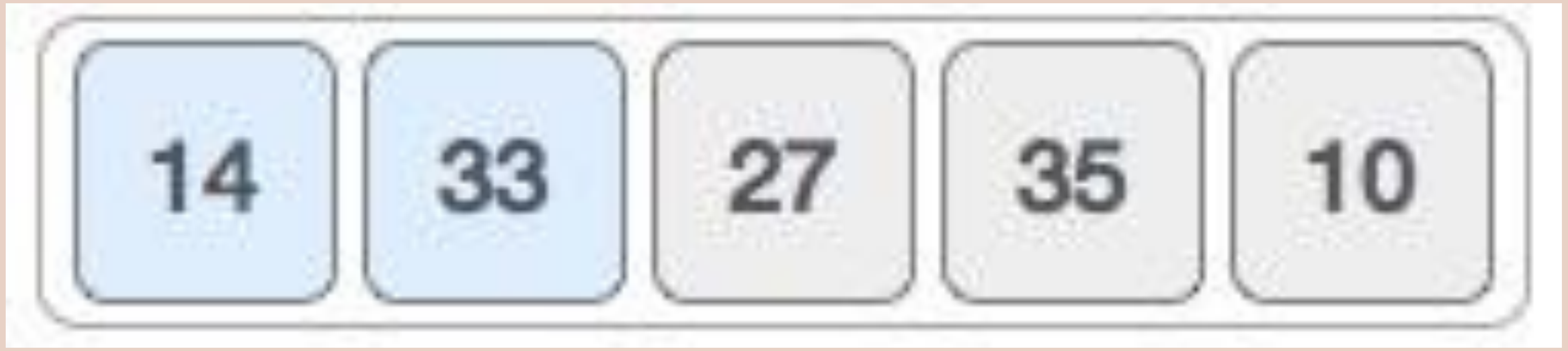
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes O(n2) time so we're keeping it short and precise.

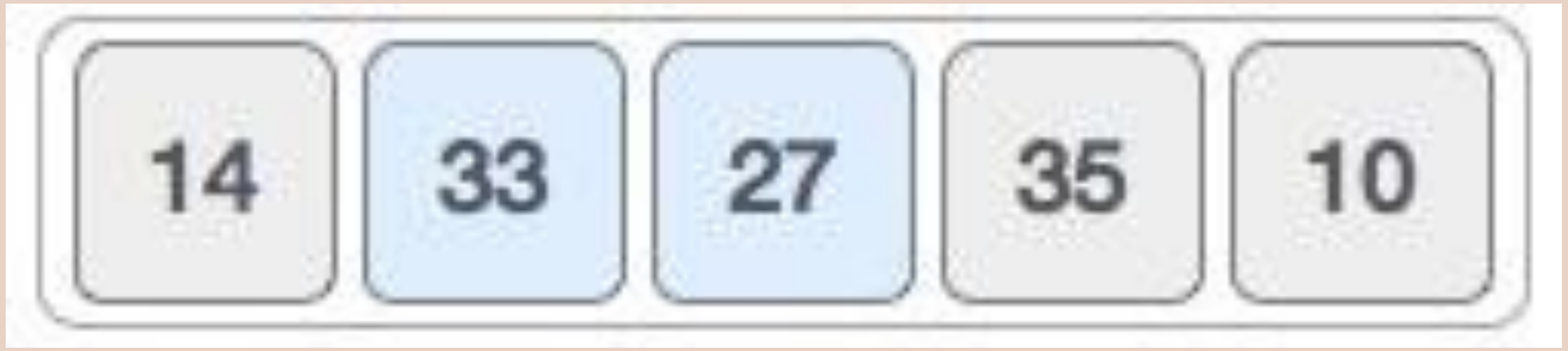


Bubble sort starts with very first two elements, comparing them to check which one is greater.

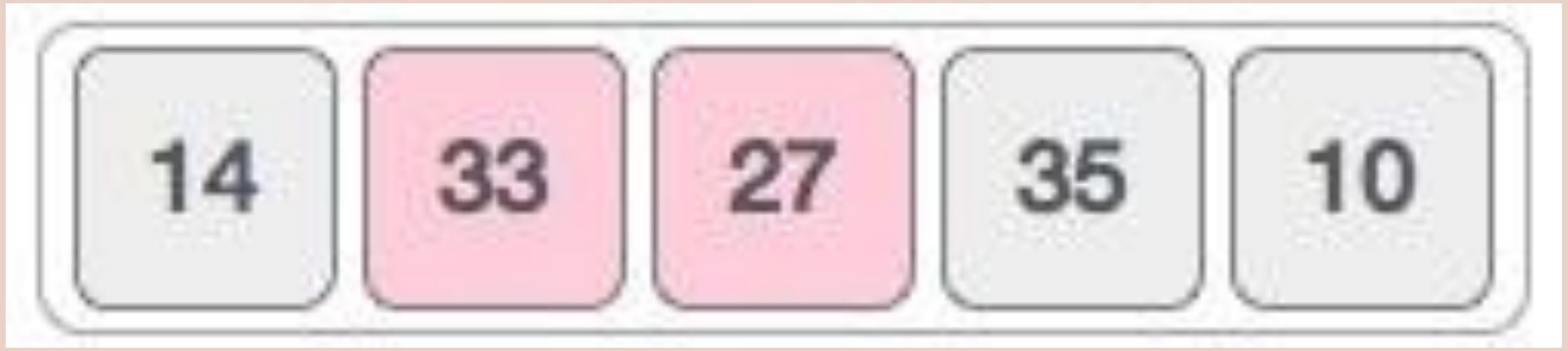


How Bubble Sort Works?

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

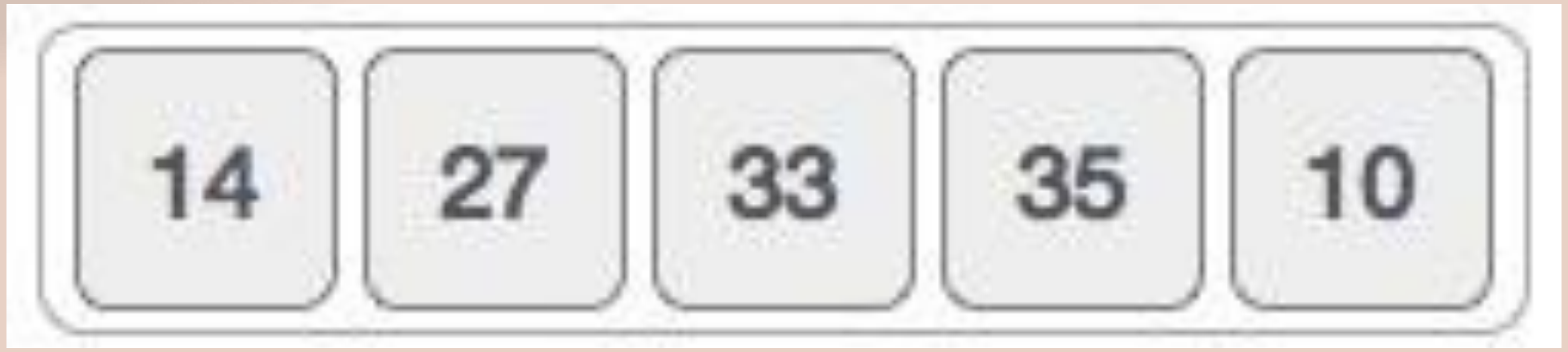


We find that 27 is smaller than 33 and these two values must be swapped.

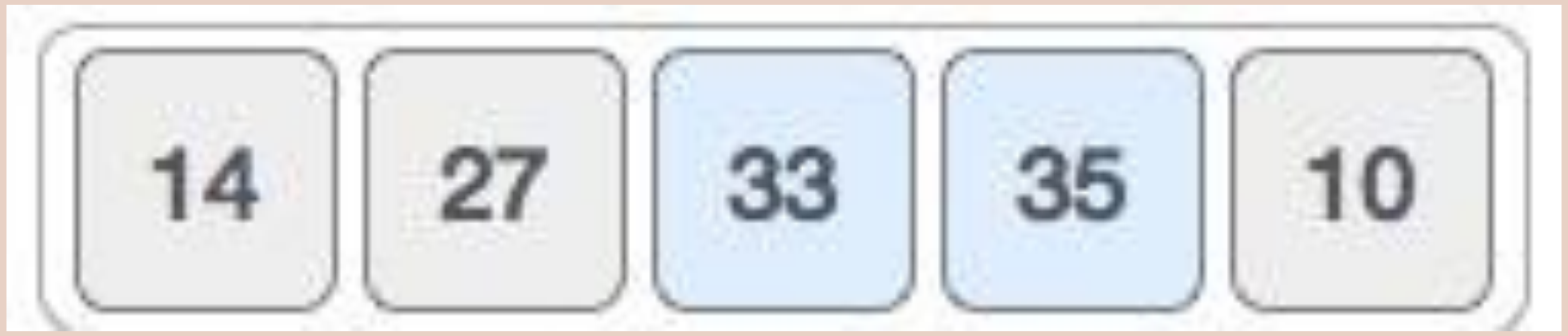


How Bubble Sort Works?

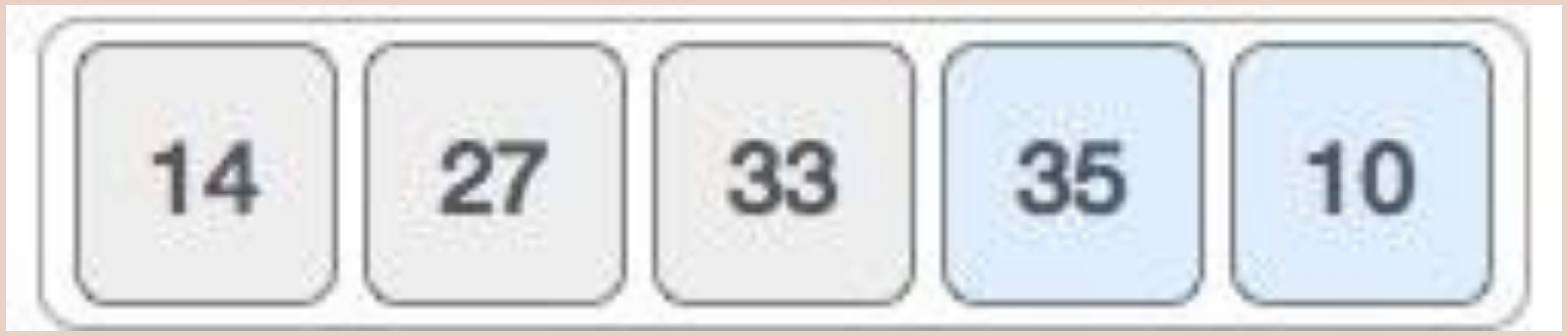
The new array should look like this –



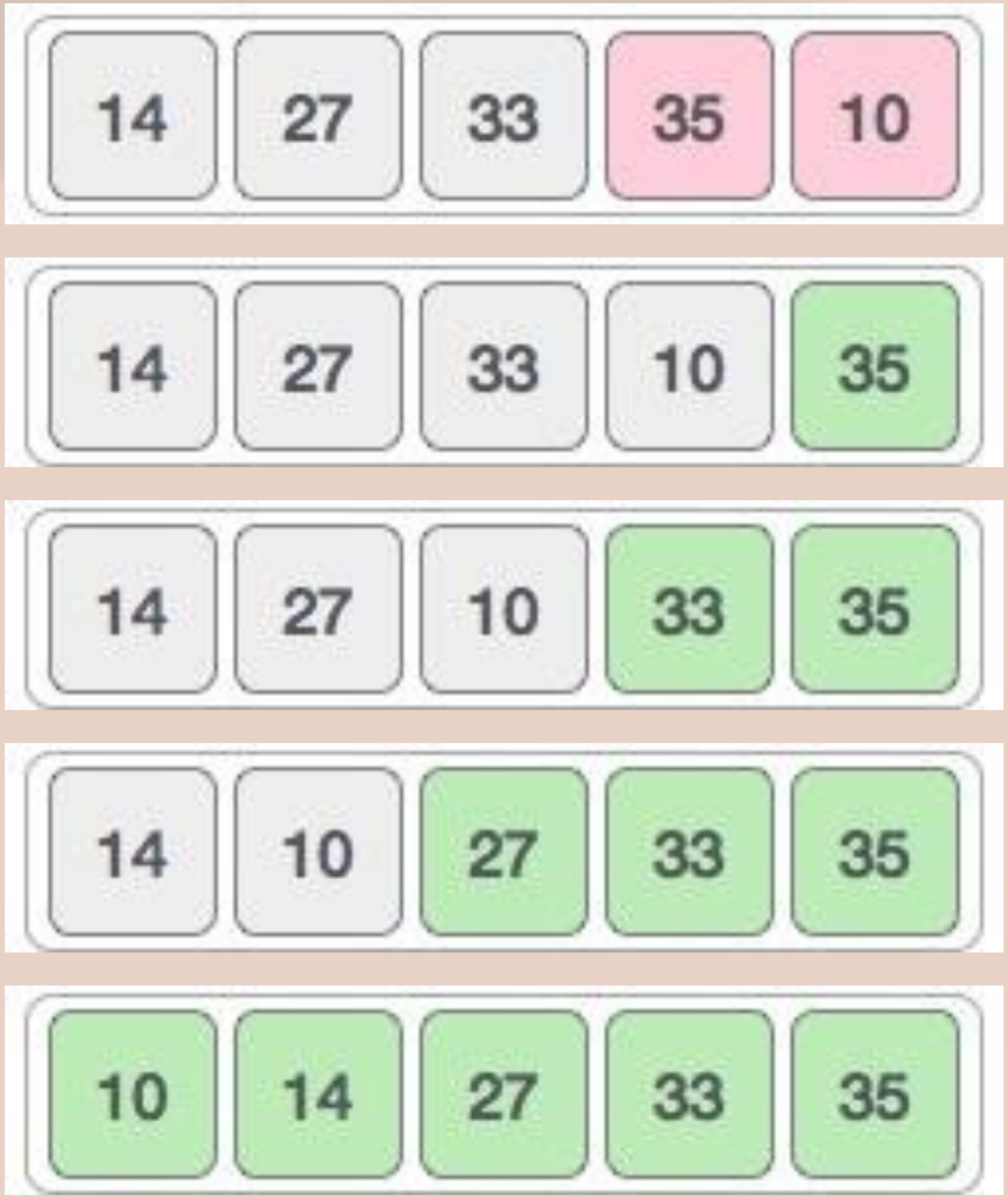
Next we compare 33 and 35. We find that both are in already sorted positions.



Then we move to the next two values, 35 and 10.



How Bubble Sort Works?

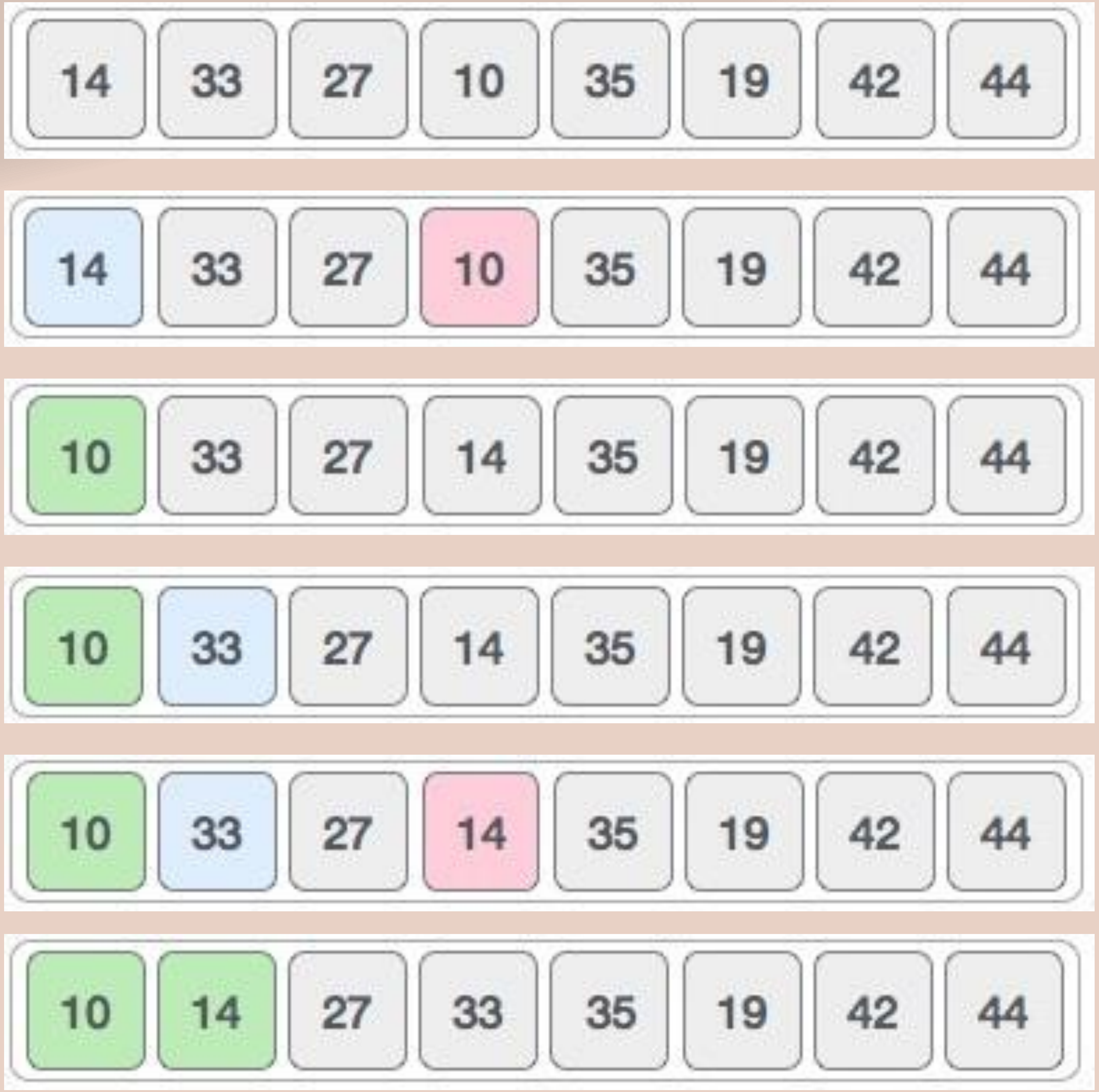


Selection Sort

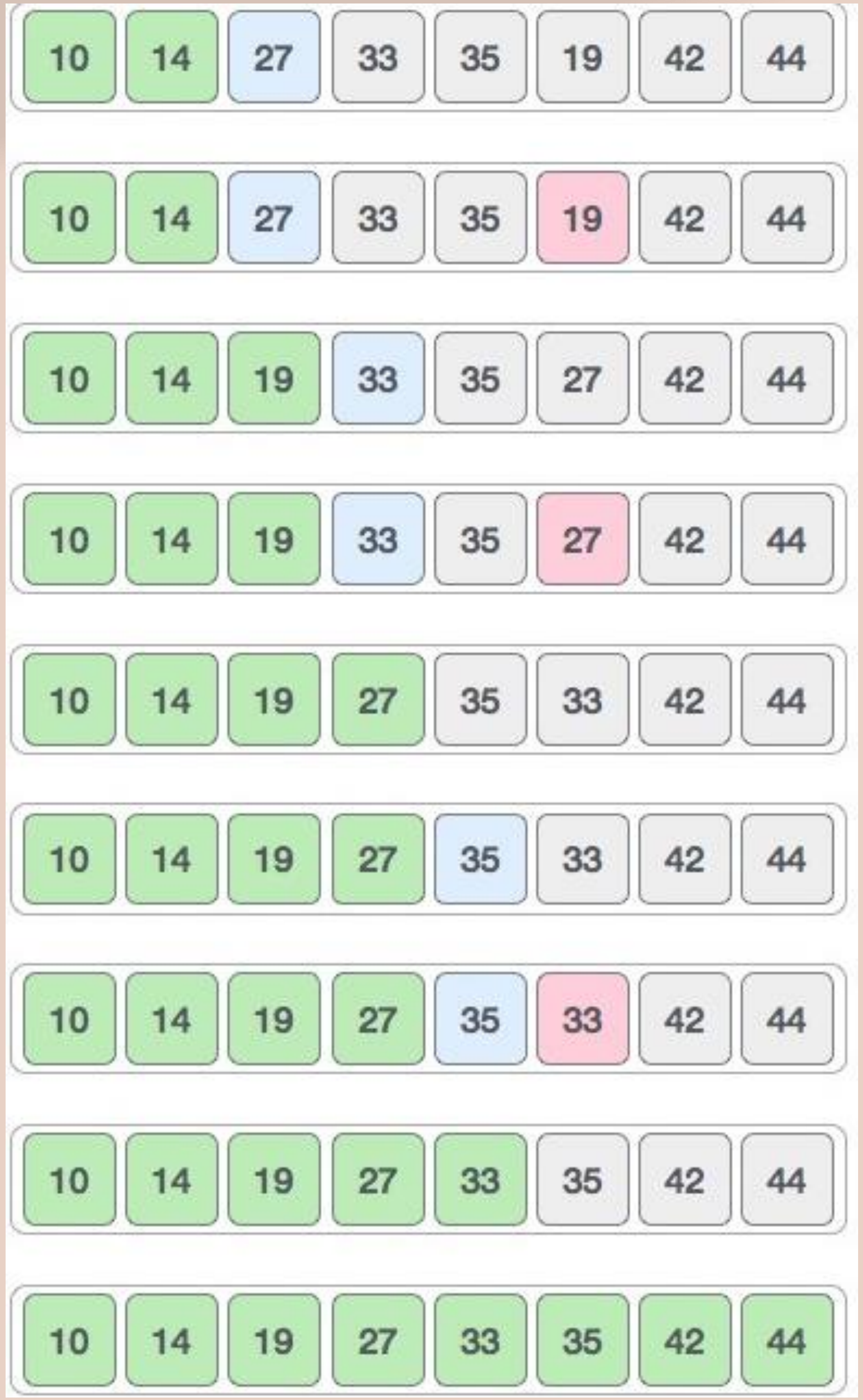
Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

How Selection Sort Works?



Following is a pictorial depiction of the entire sorting process –



Insertion Sort

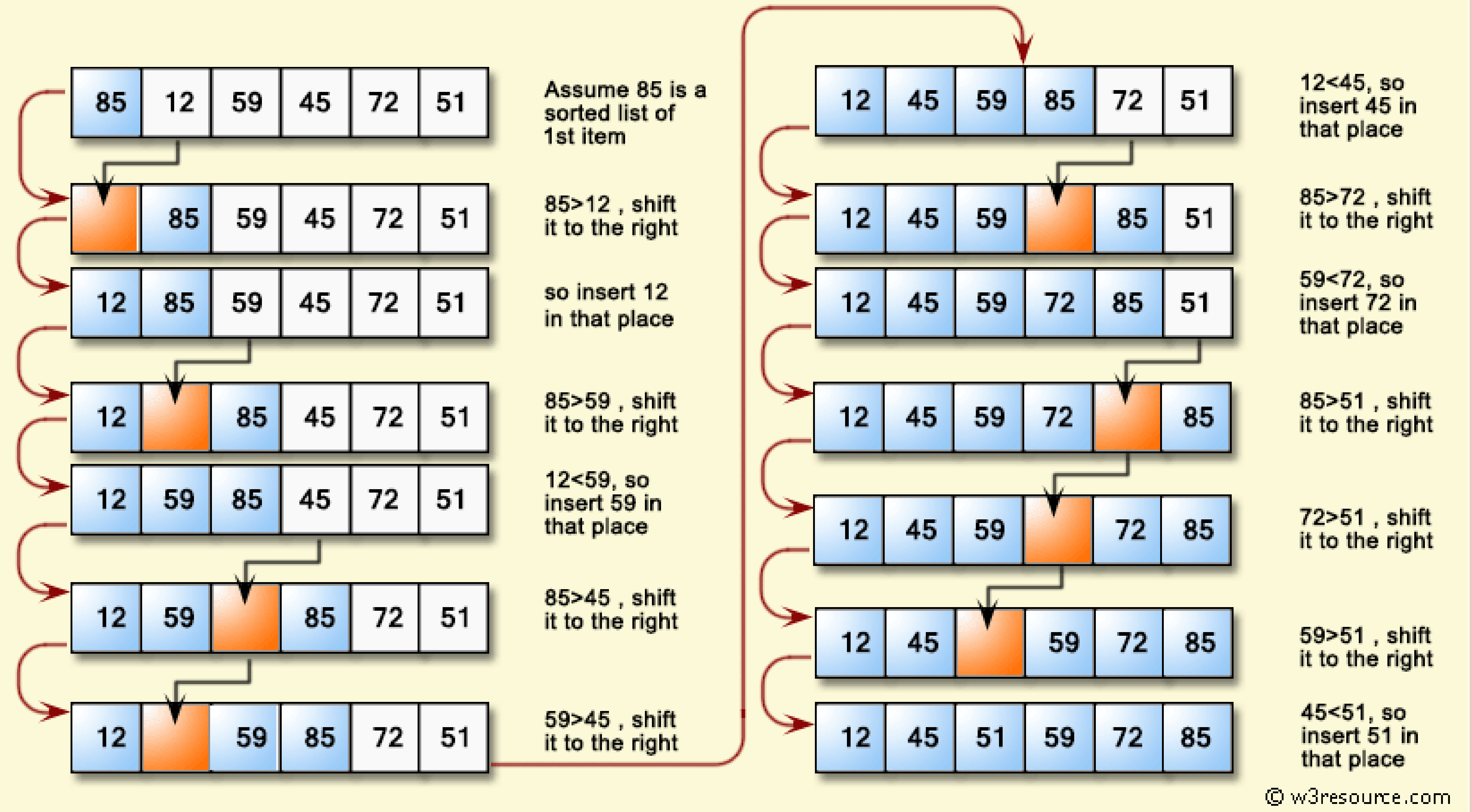
Insertion sort is the simple sorting algorithm that virtually splits the given array into sorted and unsorted parts, then the values from the unsorted parts are picked and placed at the correct position in the sorted part.

1. Iterate from arr[1] to arr[n] over the array.

2. Compare the current element (key) to its predecessor.

3. If the key element is smaller than its predecessor, compare its elements before. Move the greater elements one position up to make space for the swapped element

Insertion Sort



85 12 59 45 72 51
Assume 85 is a sorted list of 1st item
85 59 45 72 51
85>12 , shift it to the right
12 85 59 45 72 51
so insert 12 in that place
12 85 45 72 51
85>59 , shift it to the right
12 59 85 45 72 51
12<59, so insert 59 in that place
12 59 85 72 51
85>45 , shift it to the right
12 59 85 72 51
59>45 , shift it to the right
12 45 59 85 72 51
12<45, so insert 45 in that place
12 45 59 85 51
85>72 , shift it to the right
12 45 59 72 85 51
59<72, so insert 72 in that place
12 45 59 72 85
85>51 , shift it to the right
12 45 59 72 85
72>51 , shift it to the right
12 45 59 72 85
59>51 , shift it to the right
12 45 51 59 72 85
45<51, so insert 51 in that place

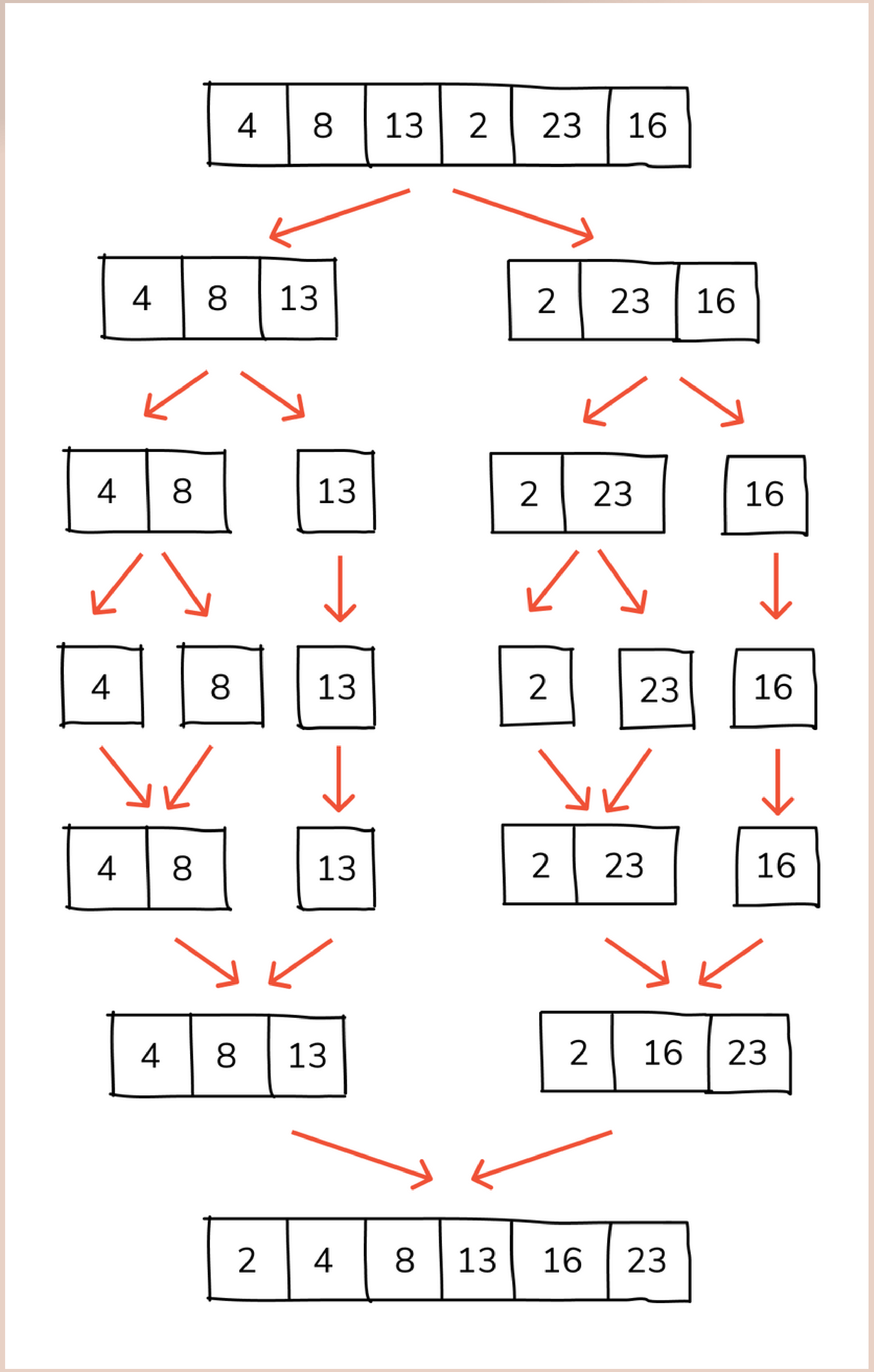
© w3resource.com

Merge Sort

The merge sort divides the entire array into two equal parts iteratively until the single values are reached.

- Step 1: Find the middle index of the array. Middle
- Step 2: Divide the array from the middle.
- Step 3: Call merge sort for the first half of the array
MergeSort (array, first, middle)
- Step 4: Call merge sort for the second half of the
array. ...
- Step 5: Merge the two sorted halves into a single
sorted array.

How Merge Sort Works?



Quick Sort

Quick Sort is one of the different Sorting Technique which is based on the concept of Divide and Conquer, just like merge sort. But in quick sort all the heavy lifting(major work) is done while dividing the array into subarrays, while in case of merge sort, all the real work happens during merging the subarrays. In case of quick sort, the combine step does absolutely nothing.

Quick Sort

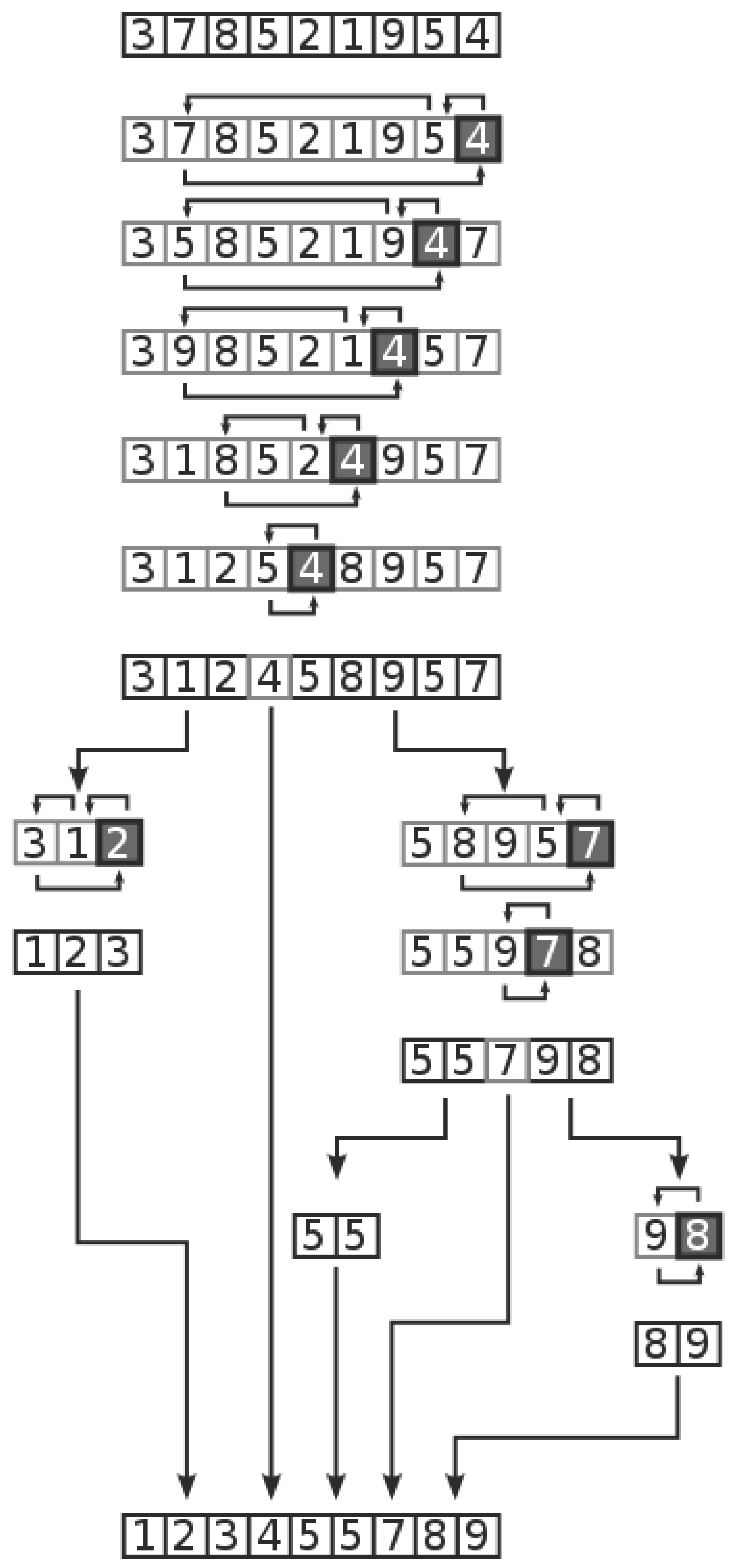
It is also called partition-exchange sort. This algorithm divides the list into three main parts:

1. Elements less than the Pivot element
2. Pivot element(Central element)
3. Elements greater than the pivot element.

Steps

- Step 1: Consider an element as a pivot element.
- Step 2: Assign the lowest and highest index in the array to low and high variables and pass it in the QuickSort function.
- Step 3: Increment low until array [low] greater than pivot then stop.
- Step 4: Decrement high until array [high] less than pivot then stop.

How Quick Sort Works?

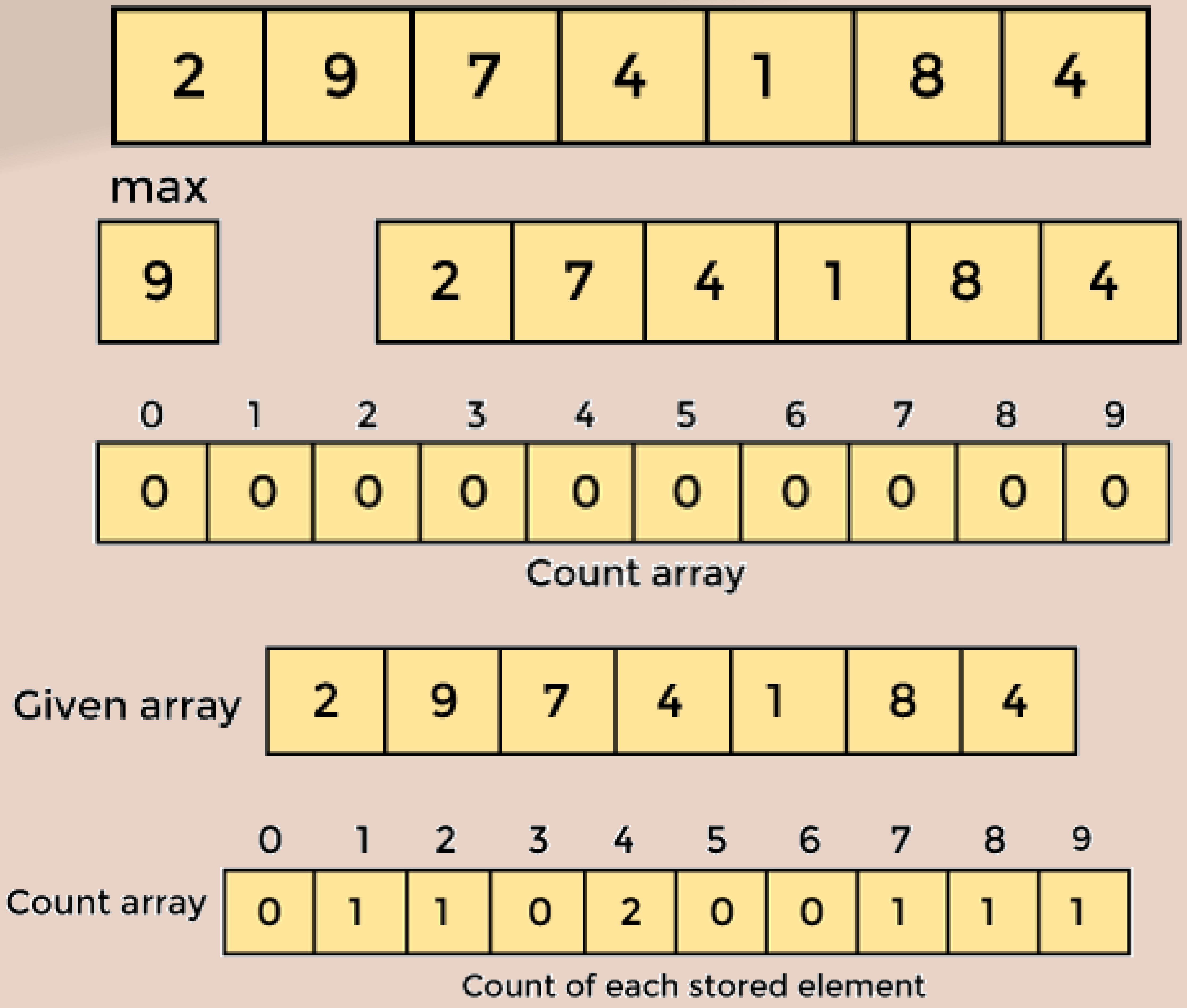


Counting Sort

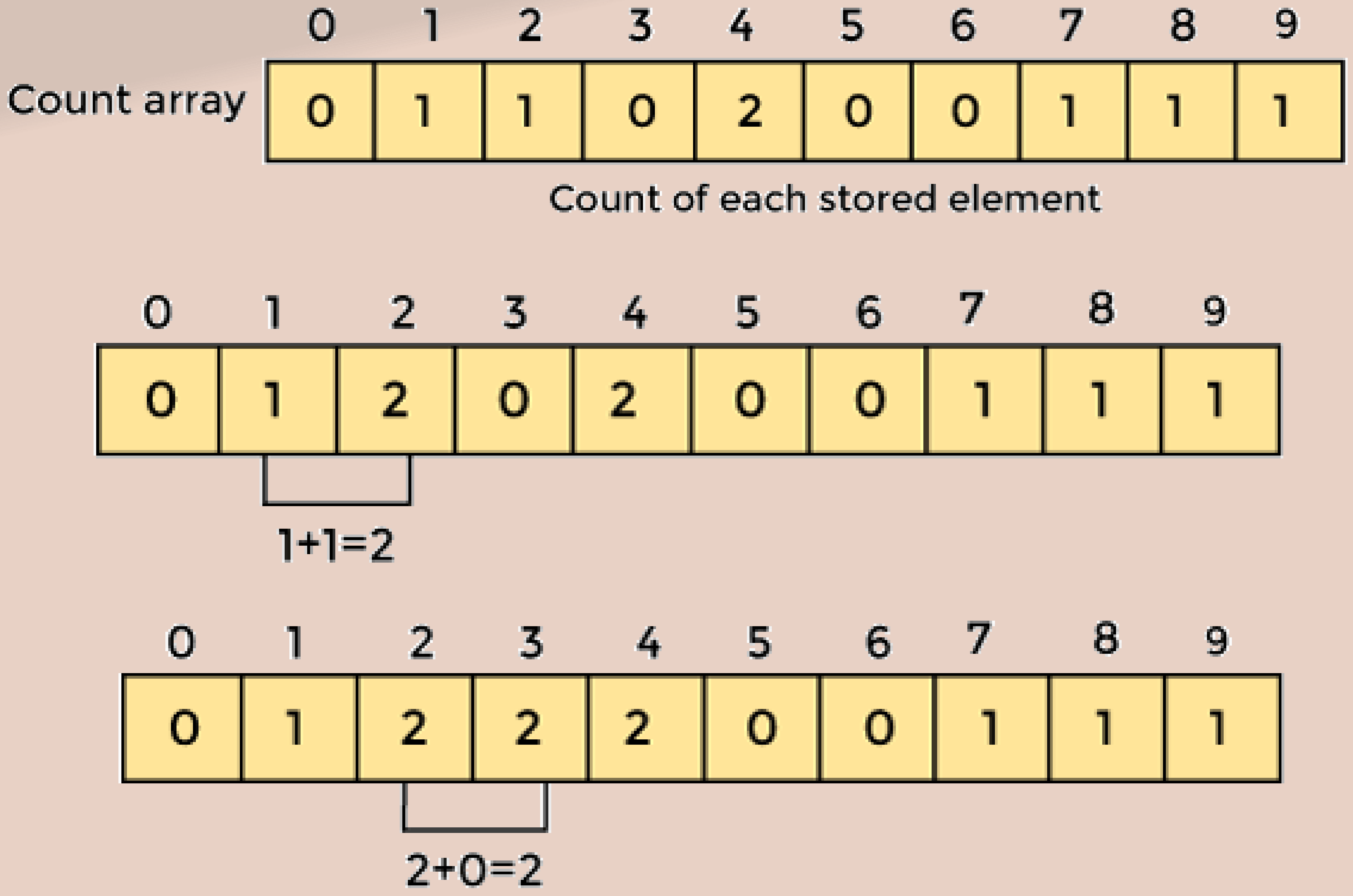
Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array.

This sorting technique doesn't perform sorting by comparing elements. It performs sorting by counting objects having distinct key values like hashing.

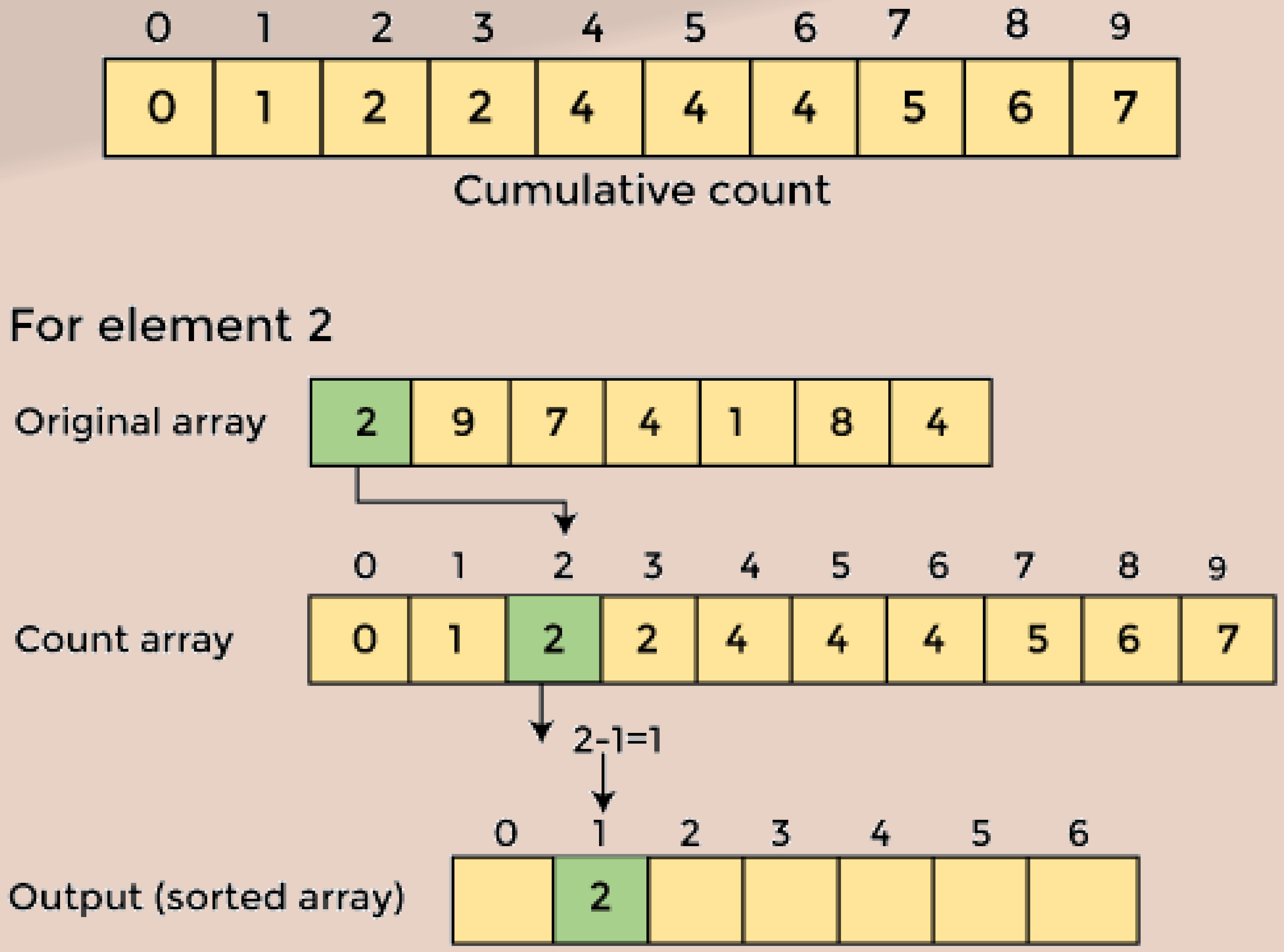
How Counting Sort Works?



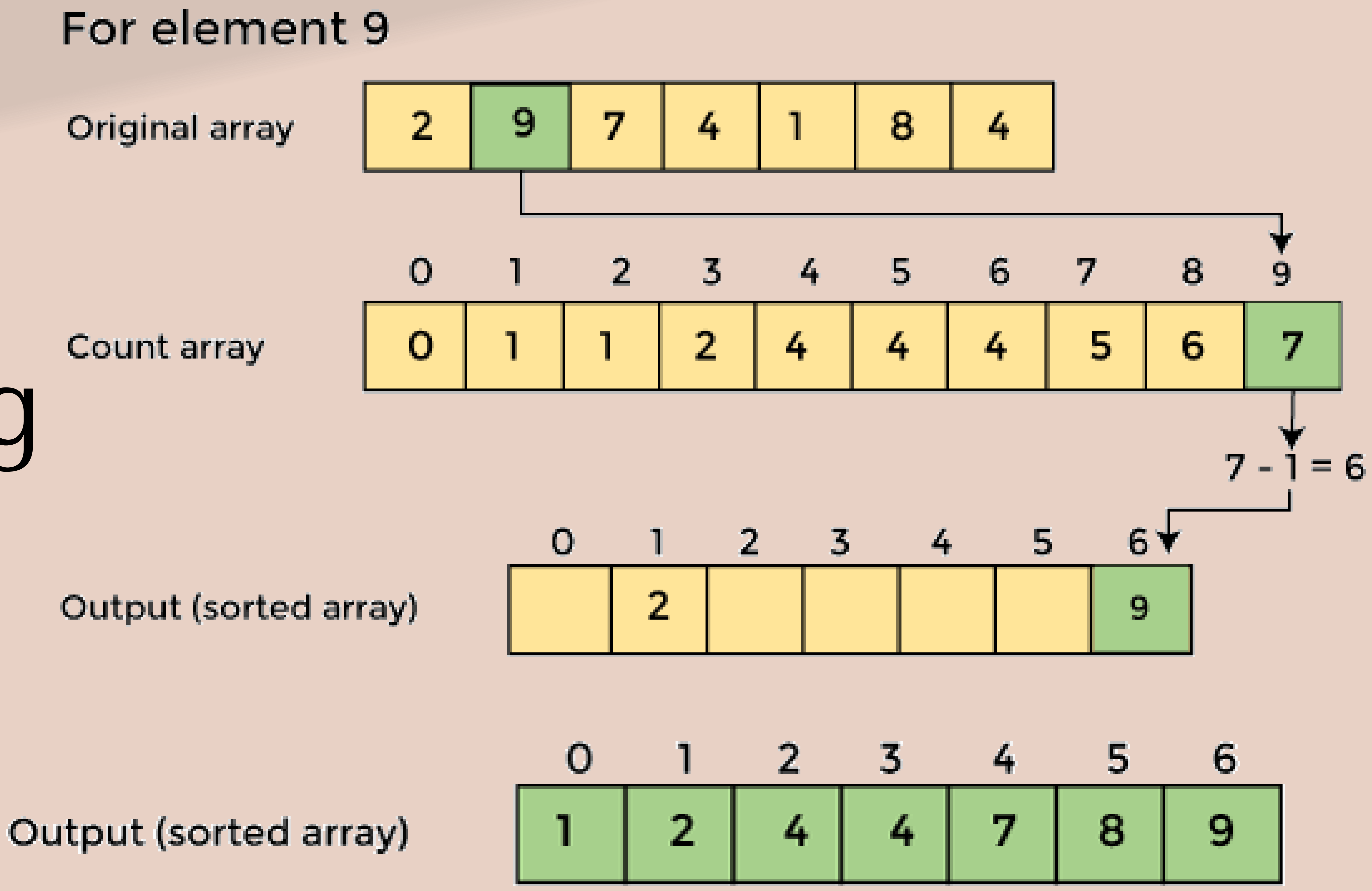
How Counting Sort Works?



How Counting Sort Works?



How Counting Sort Works?

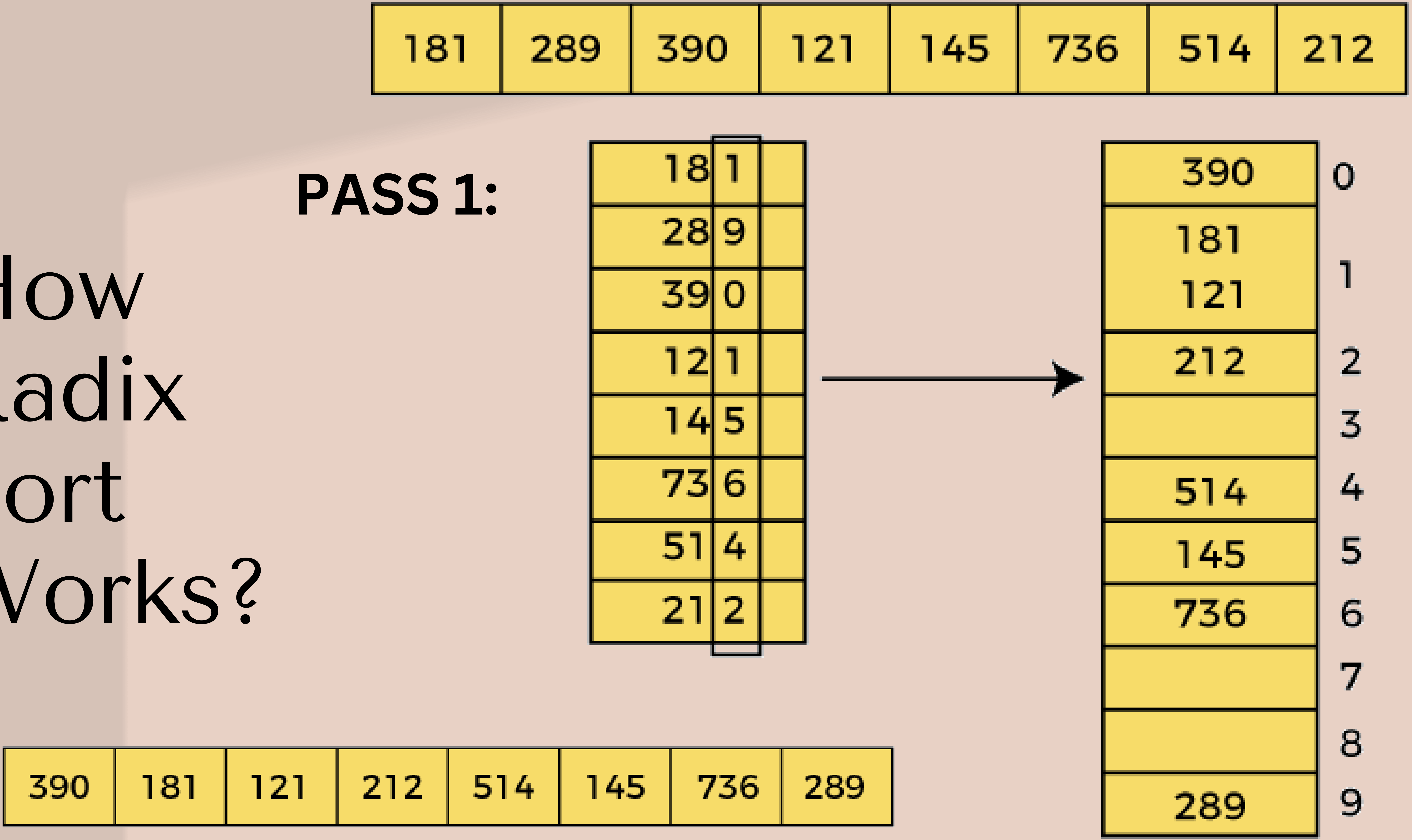


Radix Sort

Radix sort is a sorting algorithm that sorts the elements by first grouping the individual digits of the same place value. Then, sort the elements according to their increasing/decreasing order.

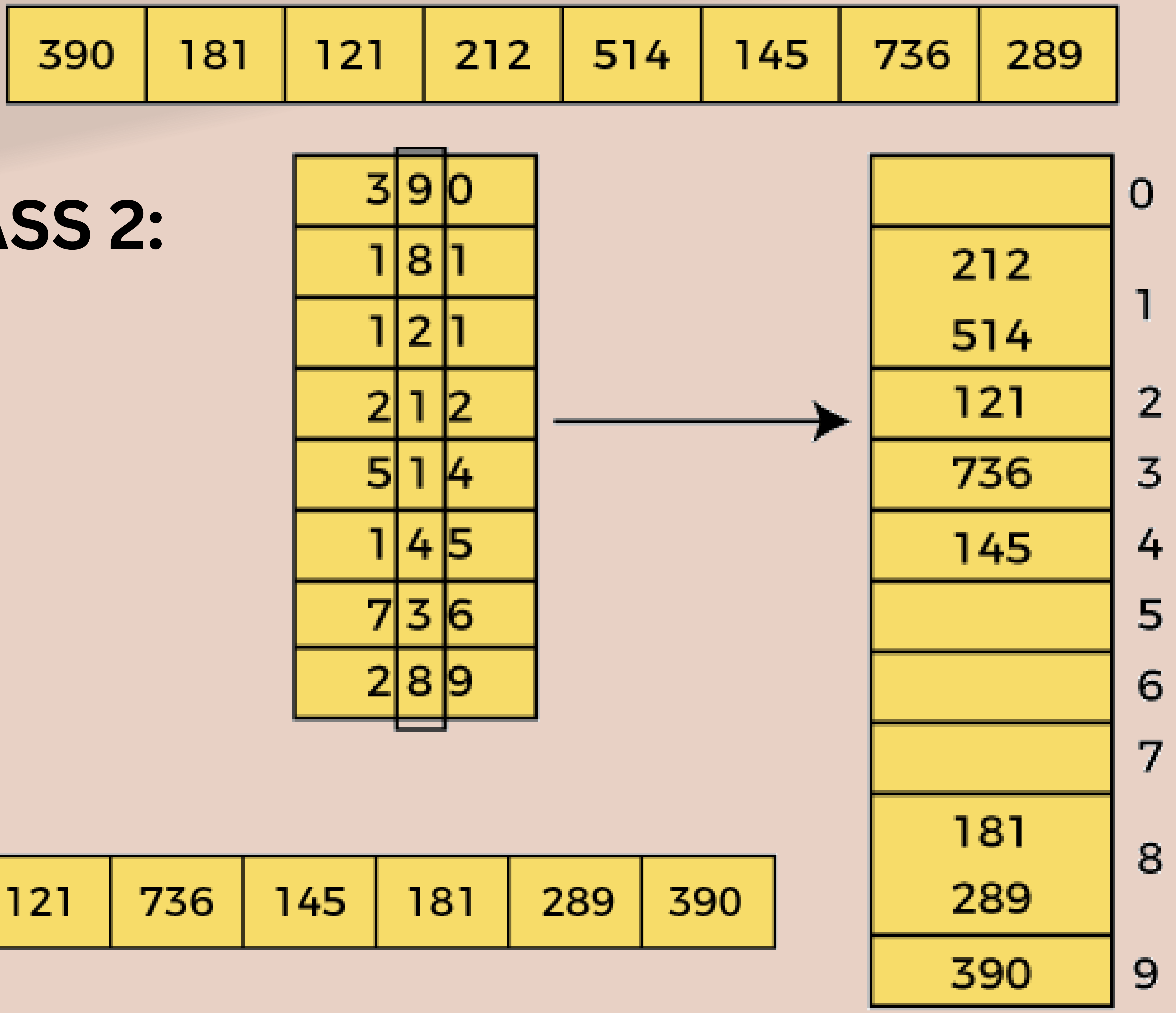
How Radix Sort Works?

PASS 1:



How Radix Sort Works?

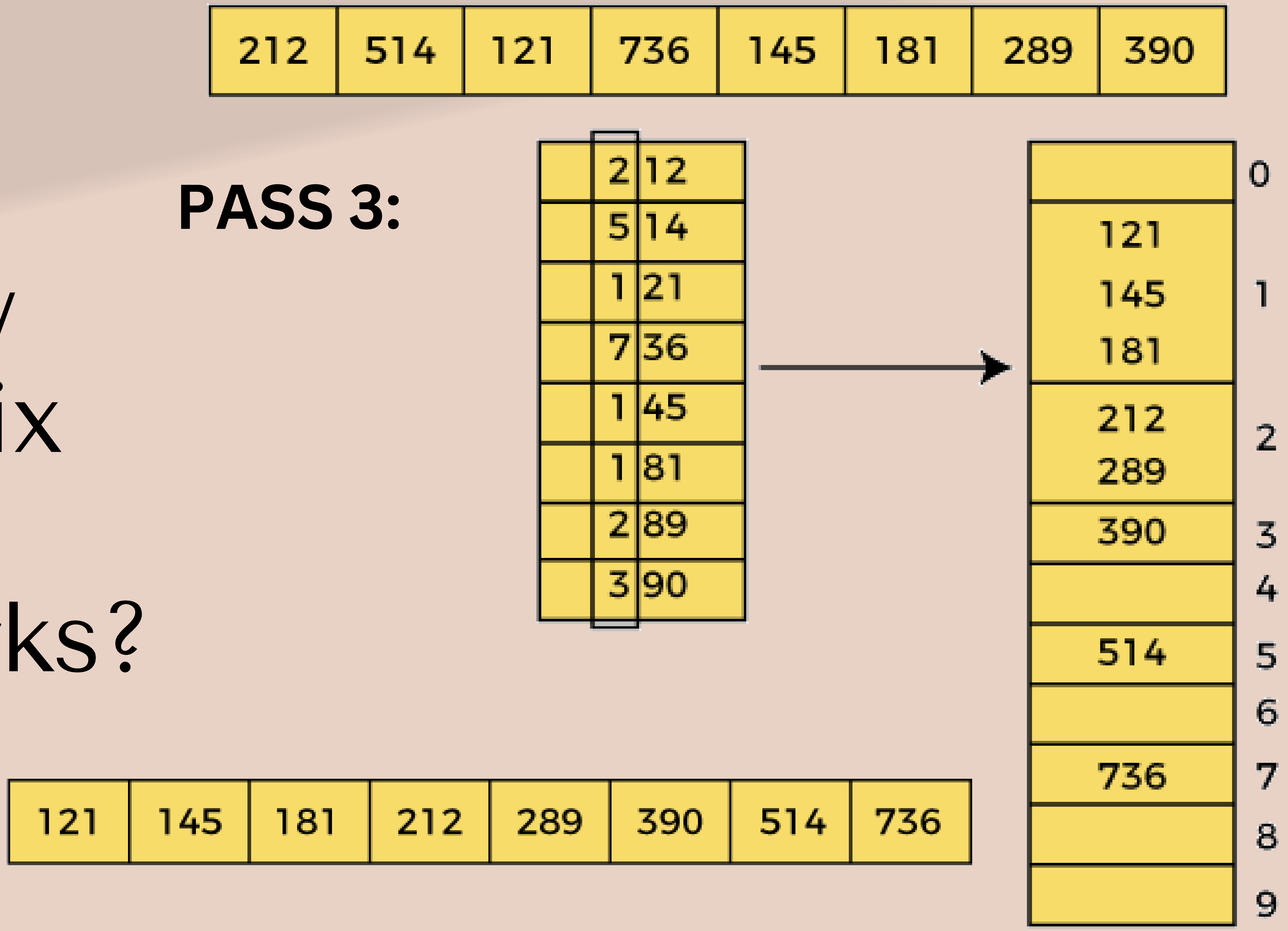
PASS 2:



212	514	121	736	145	181	289	390

How Radix Sort Works?

PASS 3:

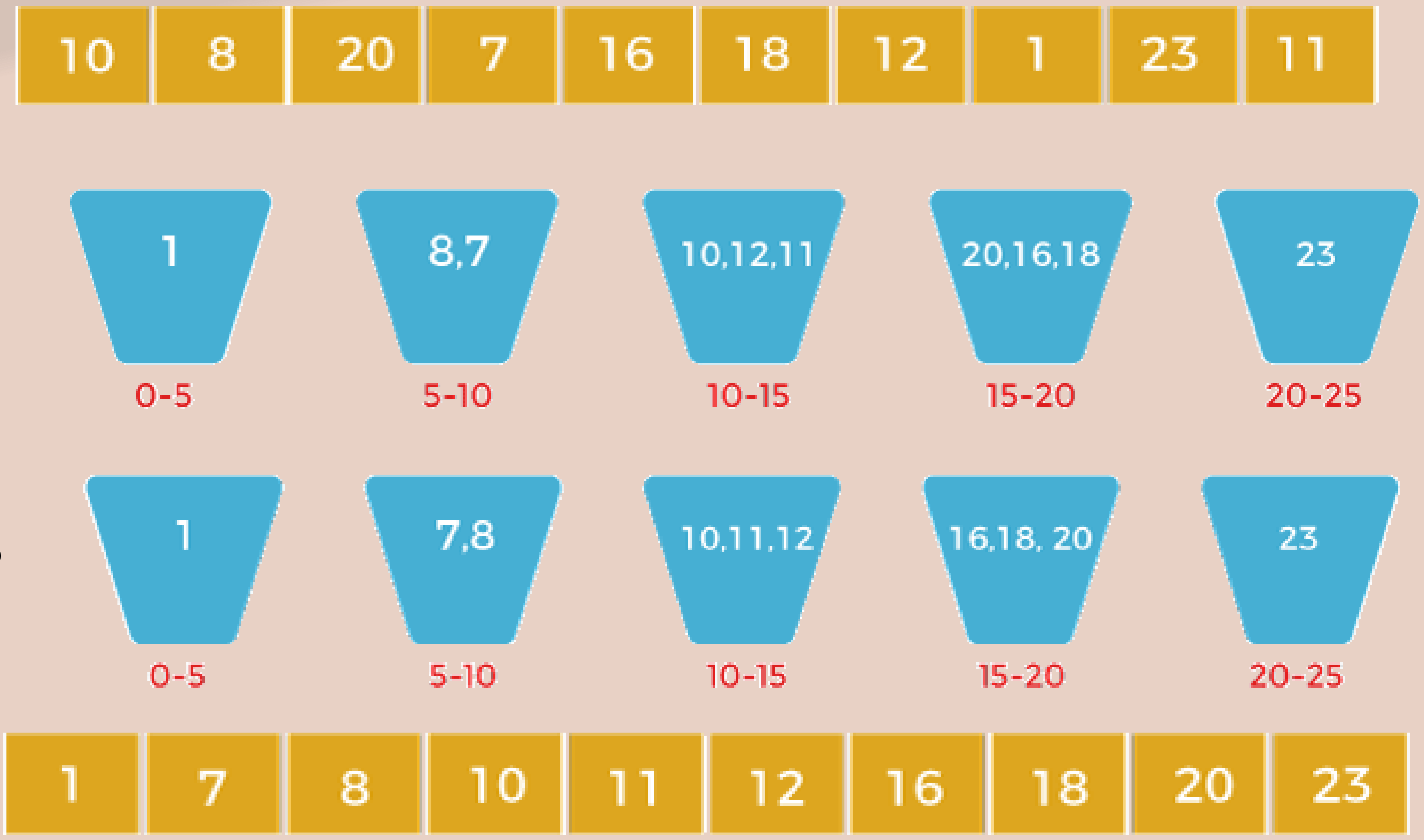


Bucket Sort

Bucket Sort is a sorting algorithm that divides the unsorted array elements into several groups called buckets. Each bucket is then sorted by using any of the suitable sorting algorithms or recursively applying the same bucket algorithm.

Let the Elements of array are:

How Radix Sort Works?



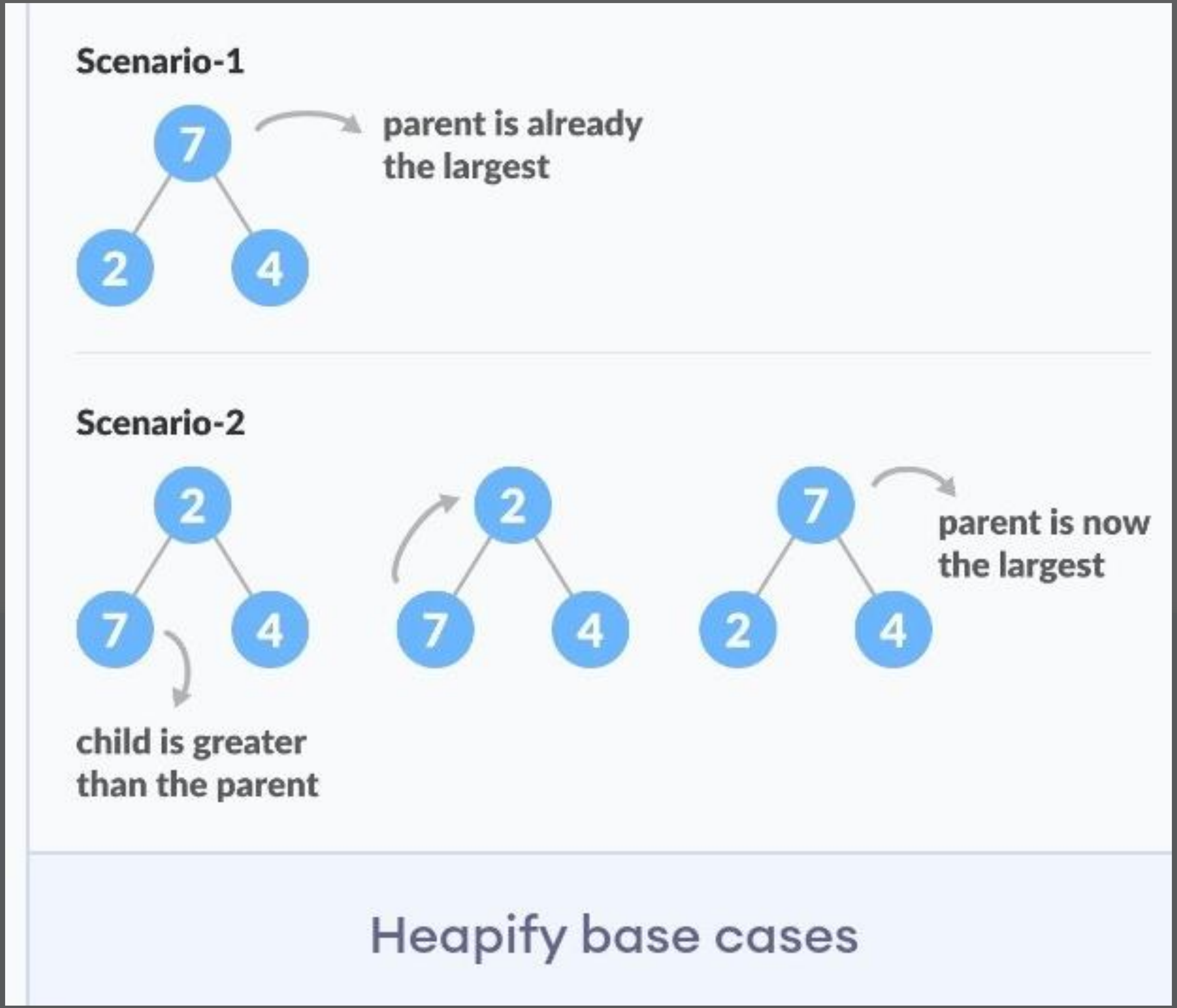
Heap Sort

Heap Sort is a popular and efficient sorting algorithm in computer programming.

Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called a heap.

How to "heapify" a tree

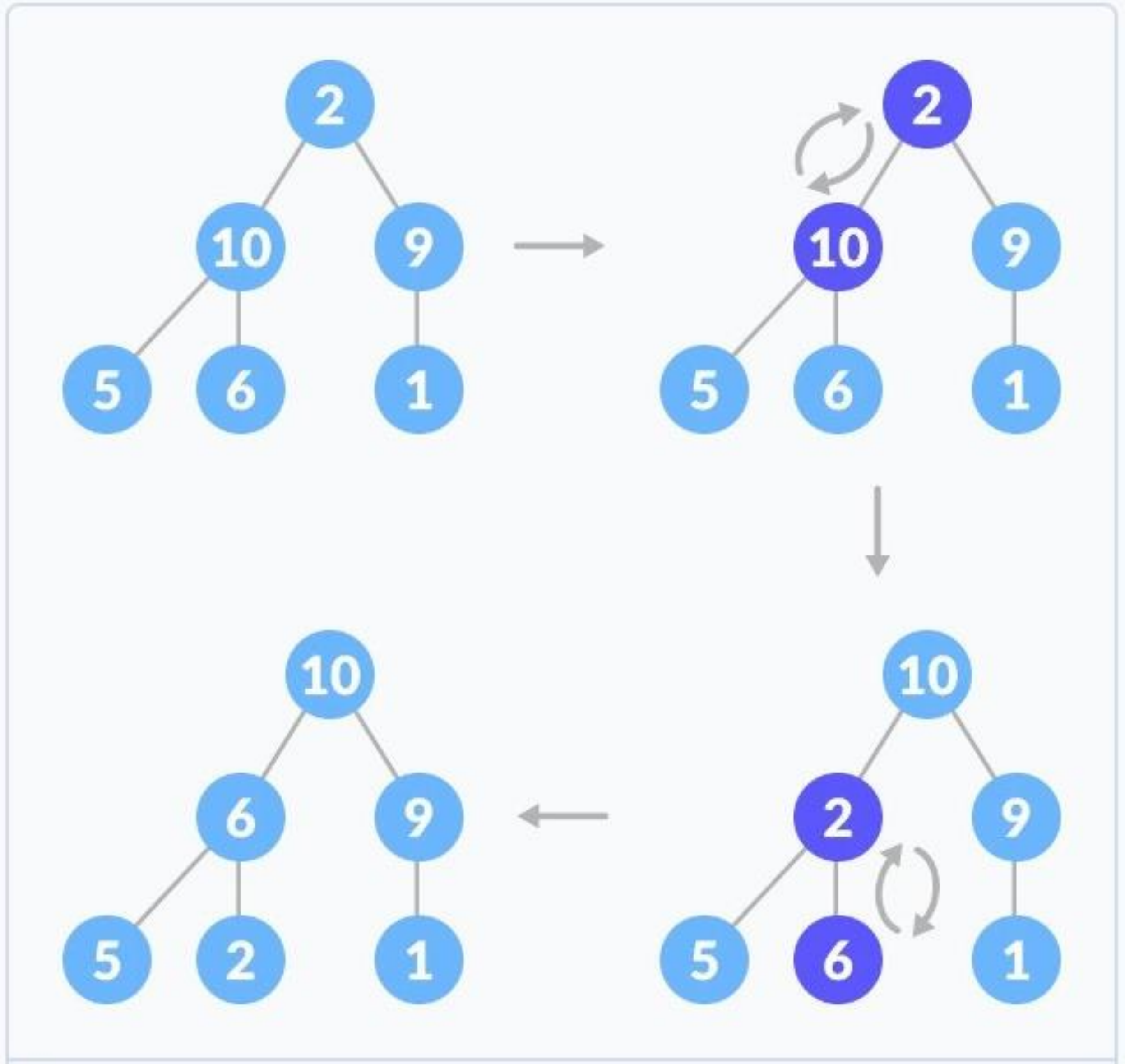
Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called heapify on all the non-leaf elements of the heap.



Scenario-1
parent is already the largest
Scenario-2
child is greater than the parent
parent is now the largest

Heapify base cases

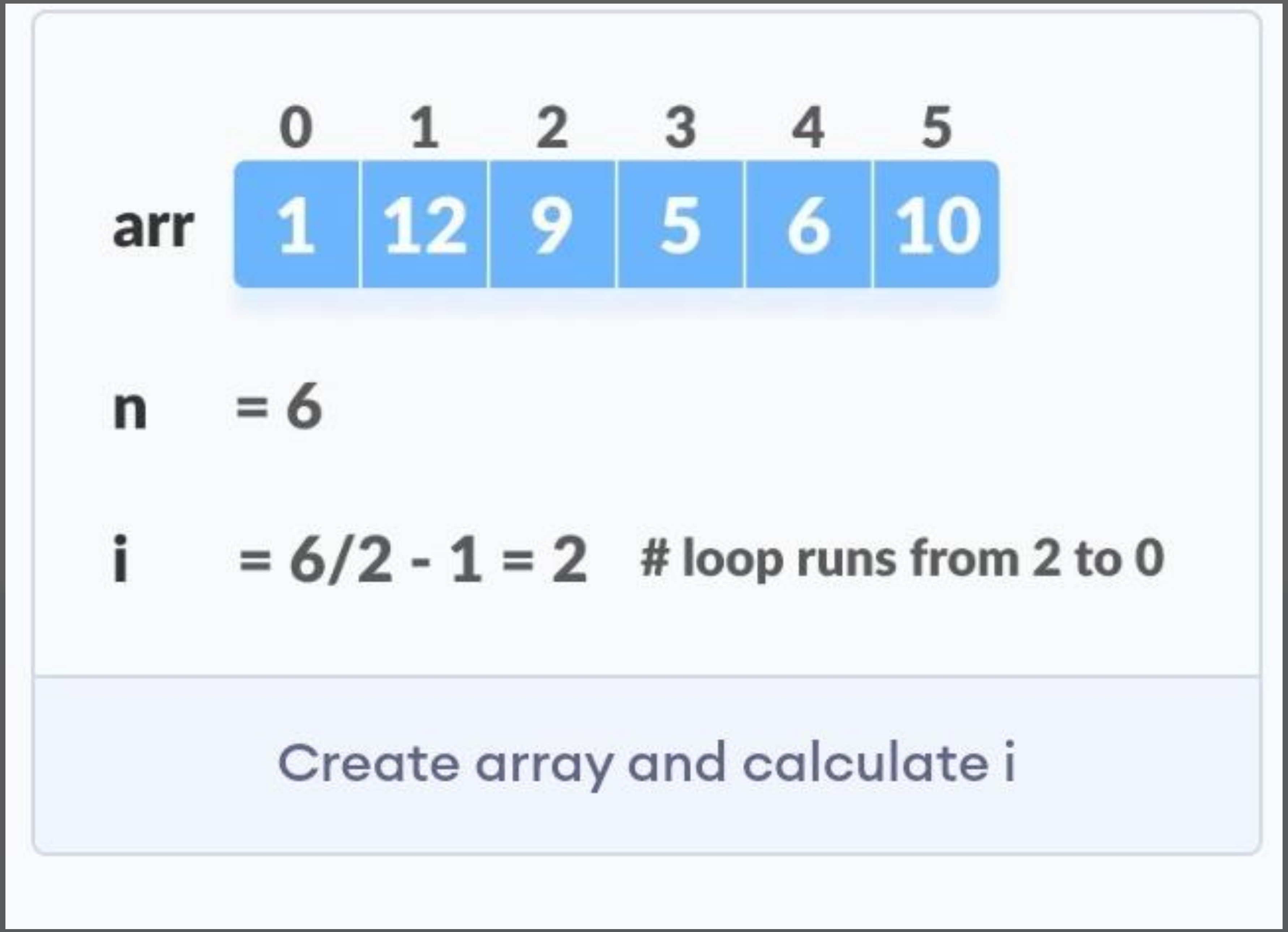
How to "heapify" a tree



How to heapify root element when its subtrees are max-heaps

Build max-heap

To build a max-heap from any tree, we can thus start heapifying each sub-tree from the bottom up and end up with a max-heap after the function is applied to all the elements including the root element.

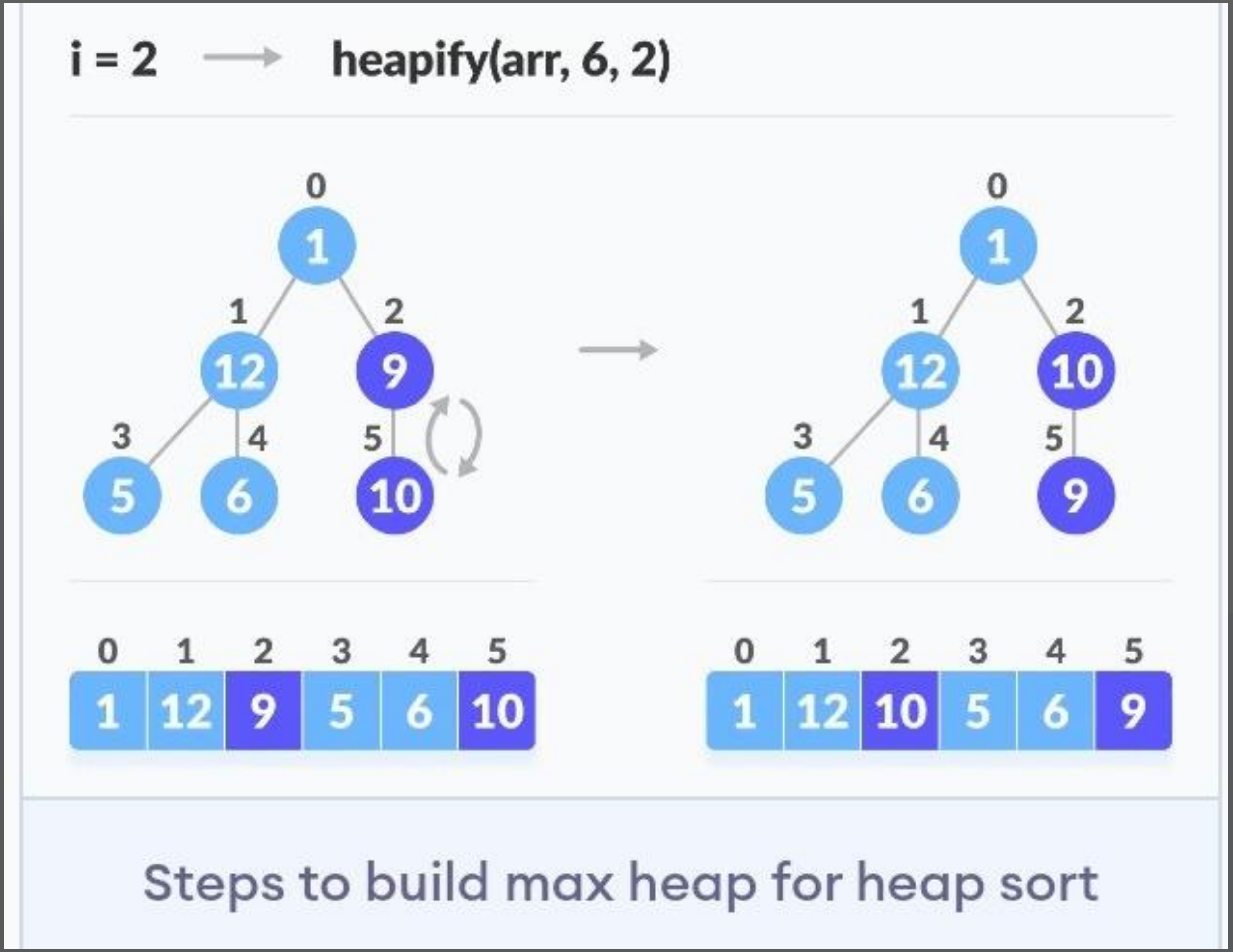


	0	1	2	3	4	5
arr	1	12	9	5	6	10

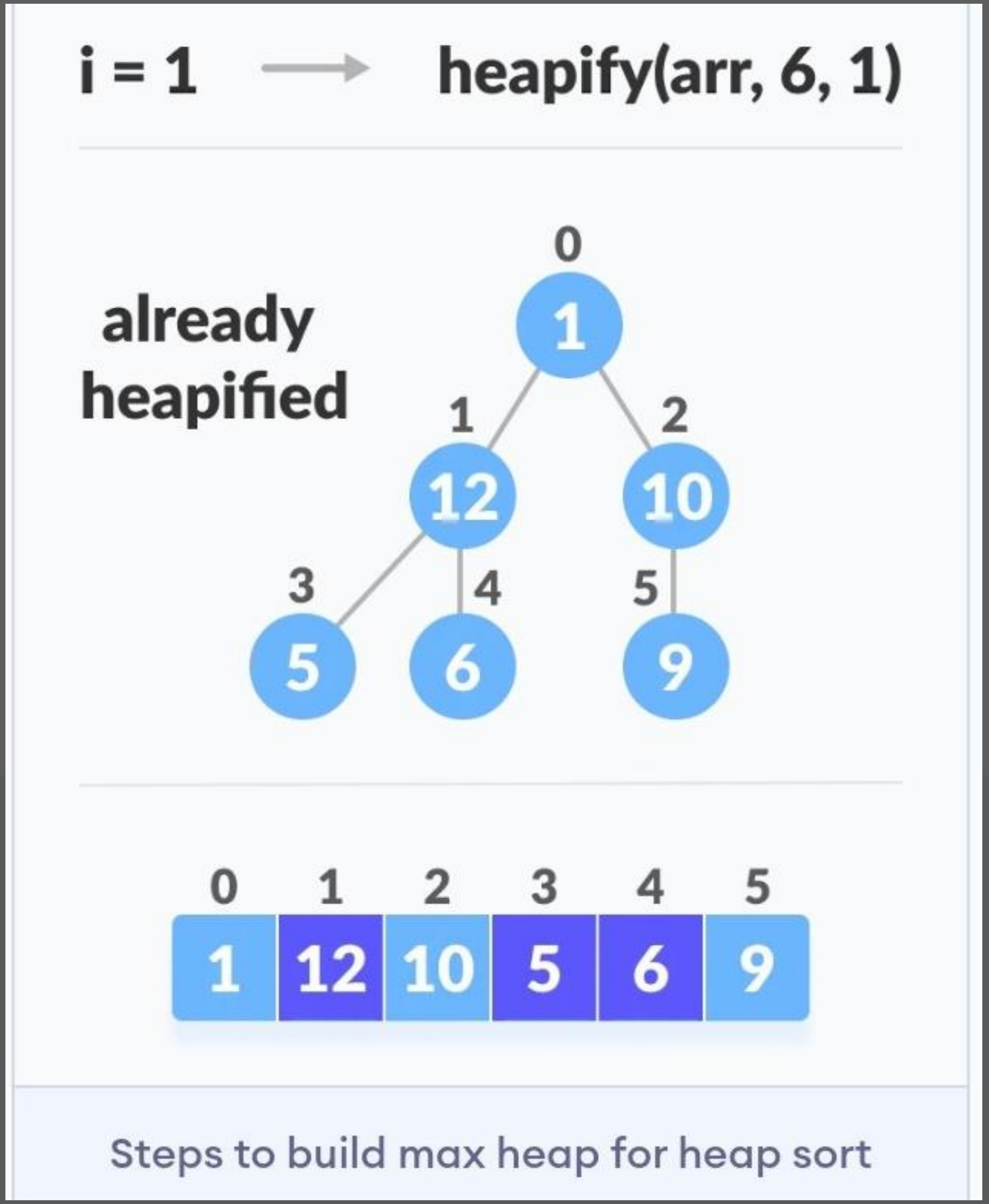
```
n    = 6

i    = 6/2 - 1 = 2   # loop runs from 2 to 0
```

Create array and calculate i

Build max-heap

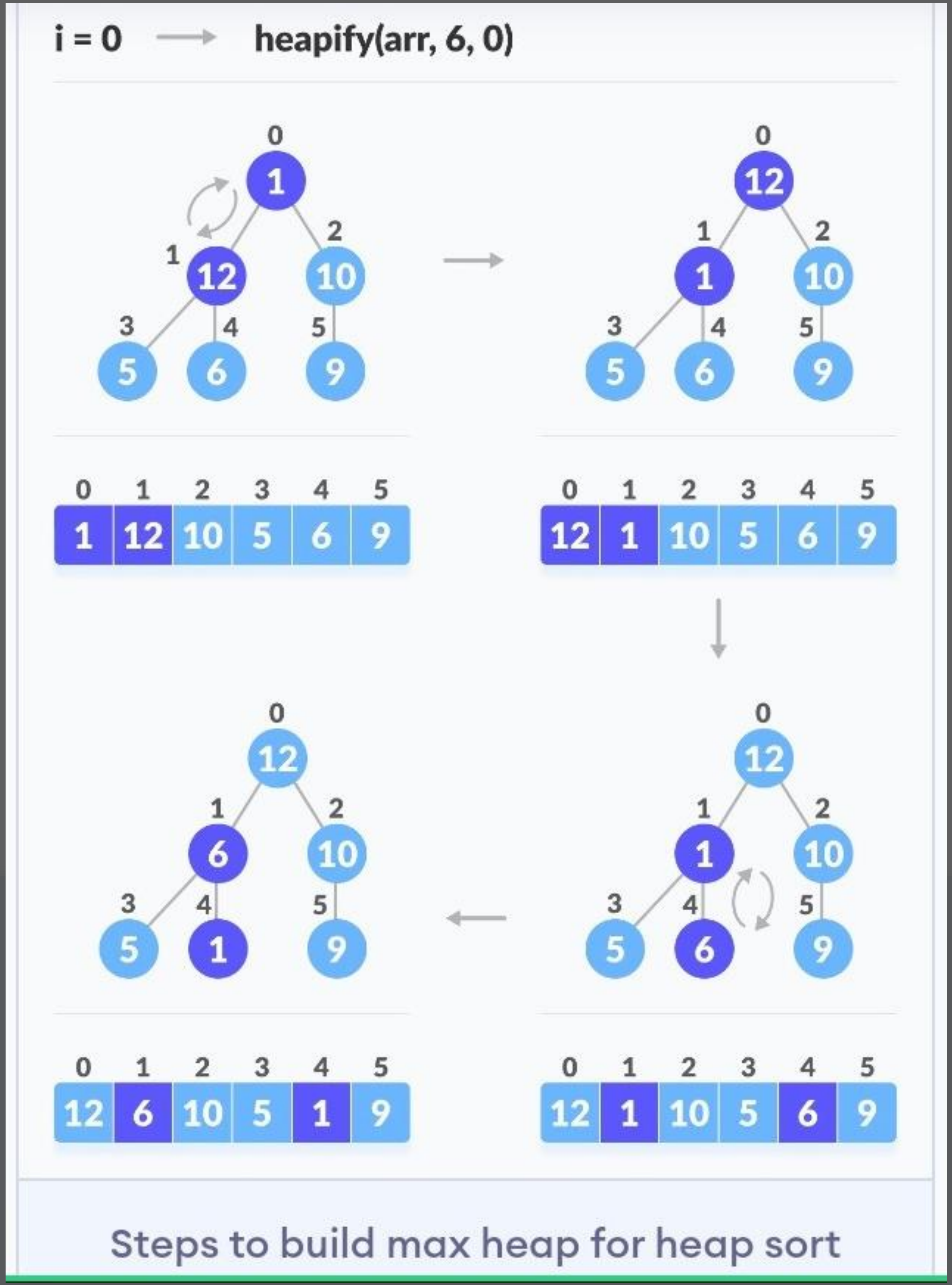


Steps to build max heap for heap sort



Steps to build max heap for heap sort

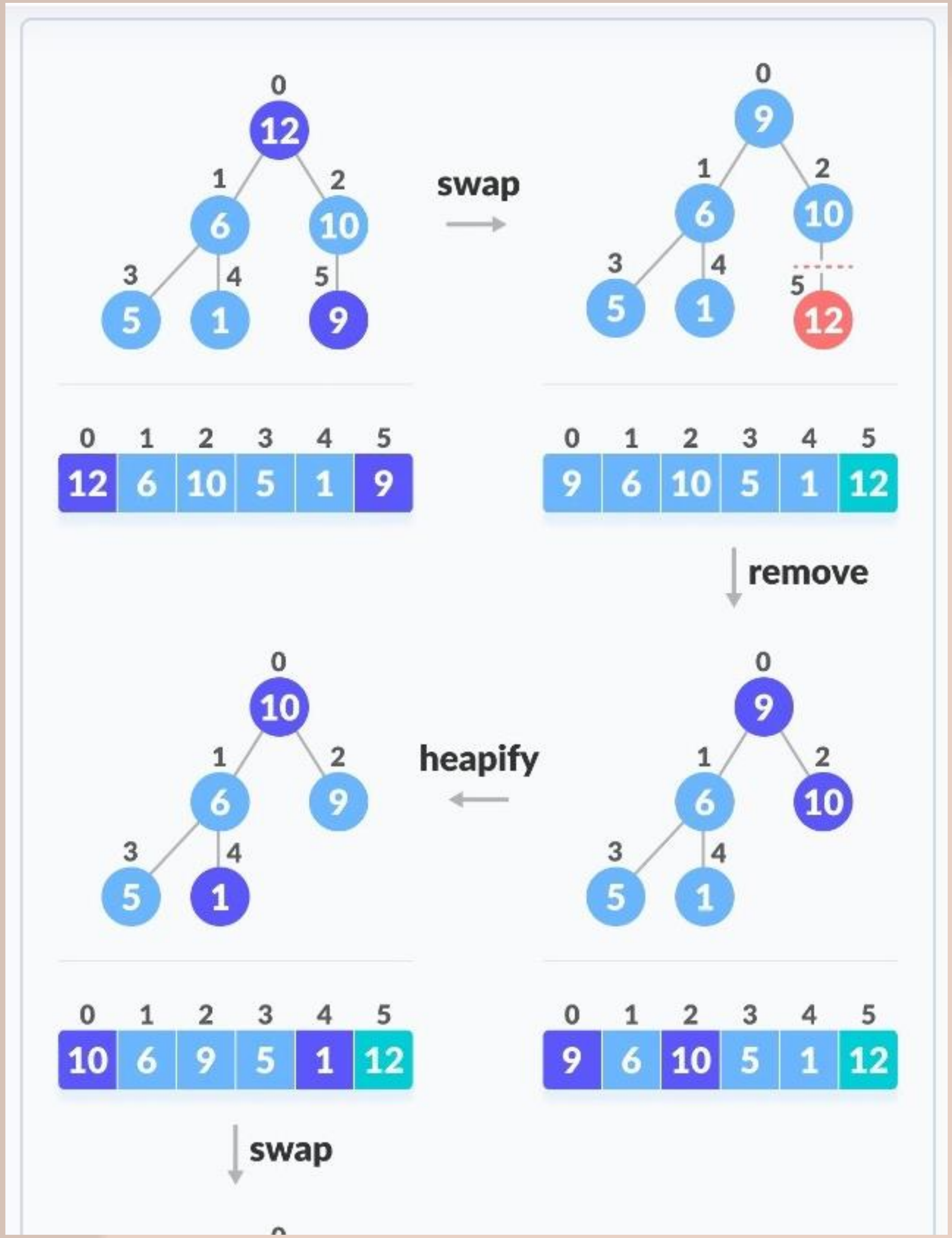
Build max-heap



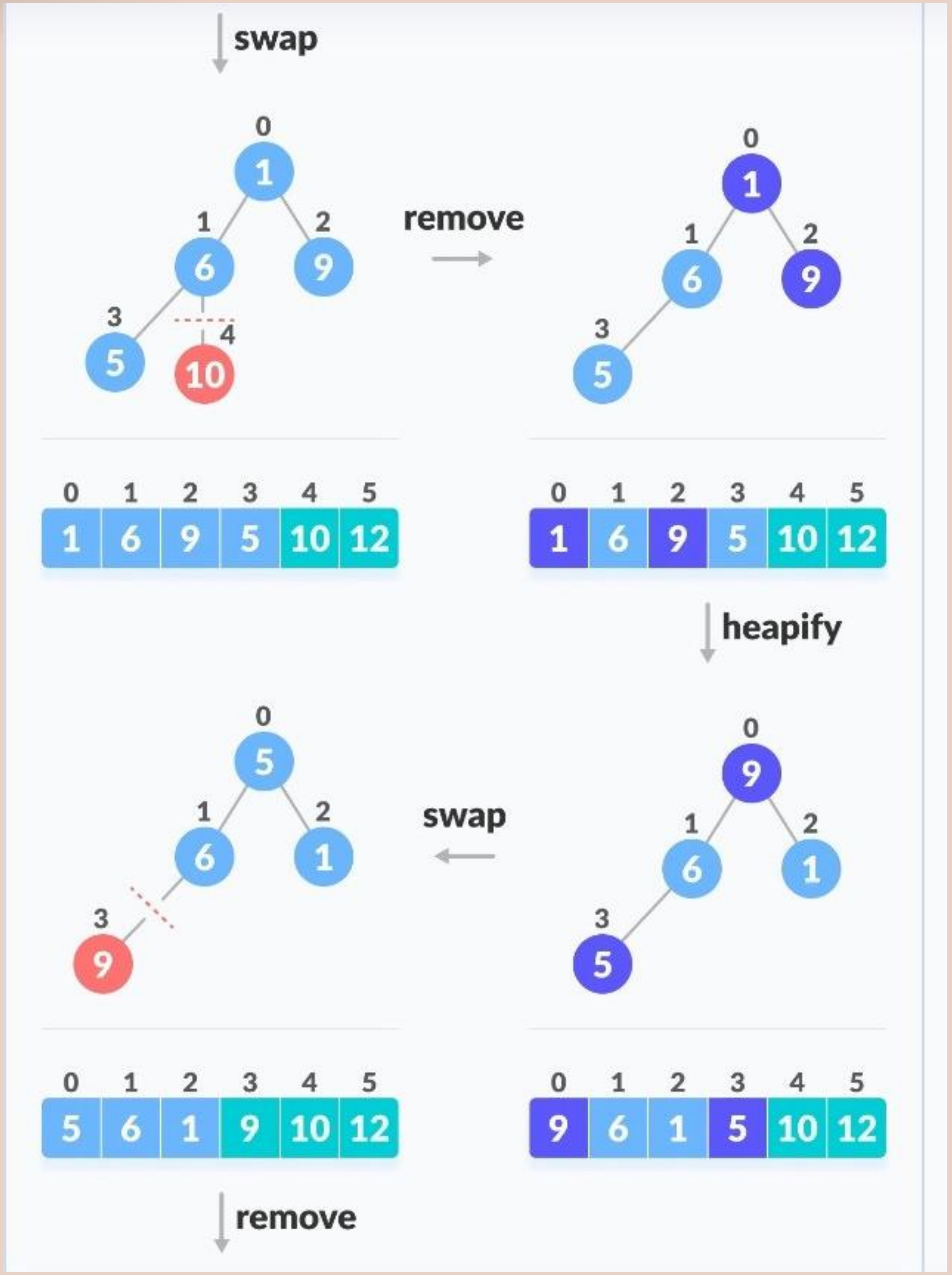
Steps to build max heap for heap sort

Working of Heap Sort

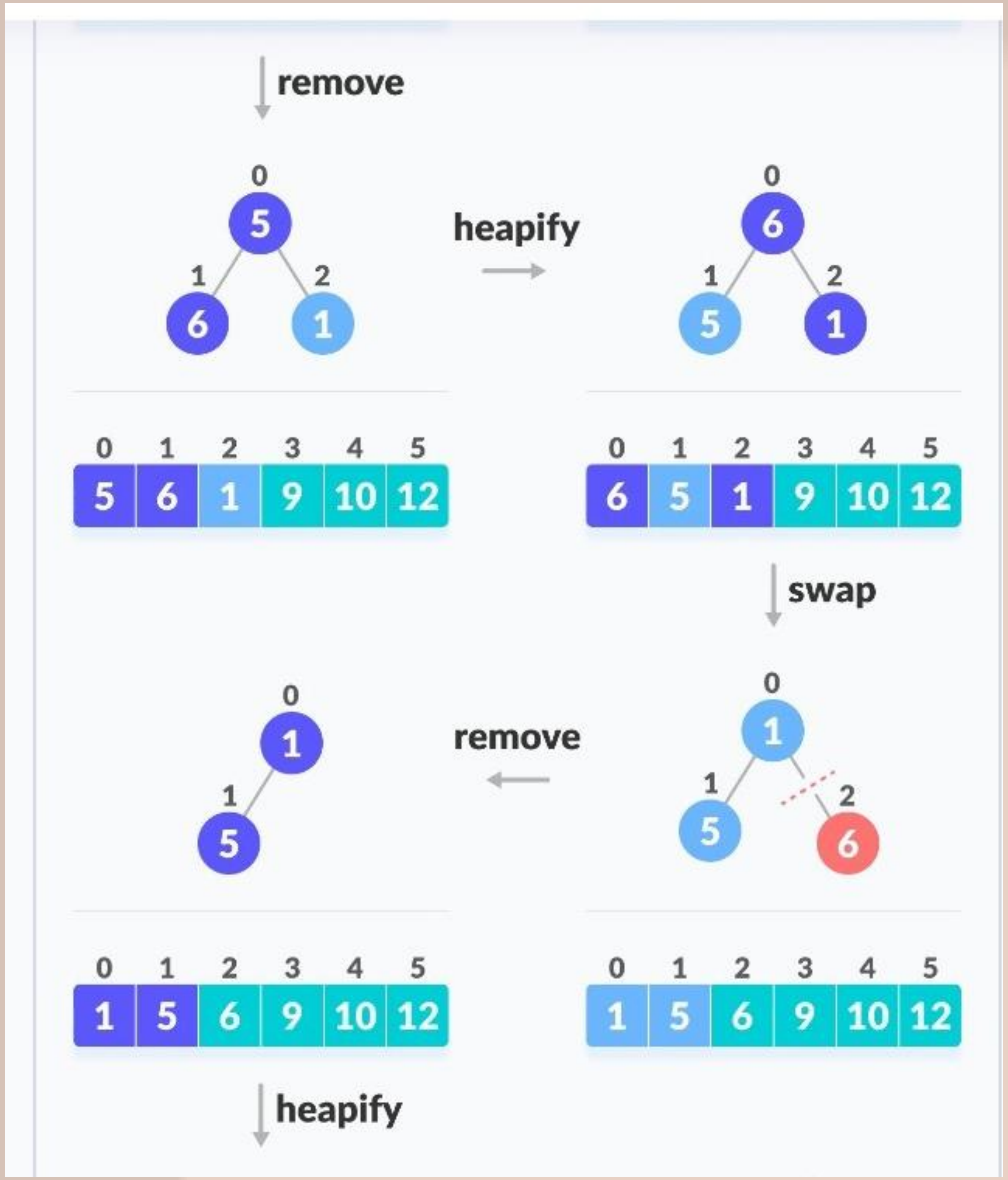
1. Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
2. Swap: Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
3. Remove: Reduce the size of the heap by 1.
4. Heapify: Heapify the root element again so that we have the highest element at root.
5. The process is repeated until all the items of the list are sorted.

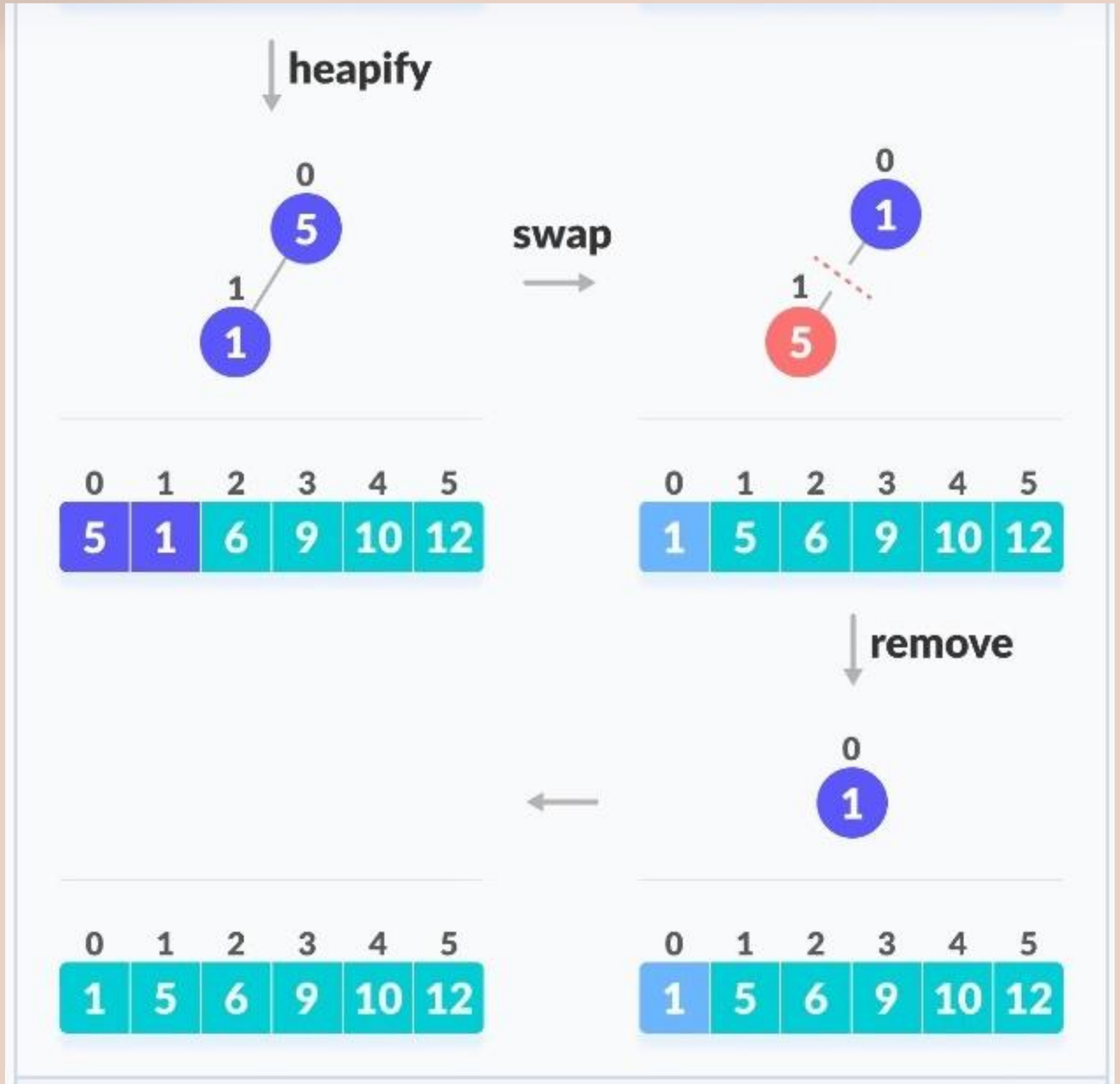


swap
remove
heapify
swap



swap
remove
heapify
swap
remove





Swap, Remove, and Heapify

Shell Sort

Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h’th element are sorted.

How Shell Sort Works?

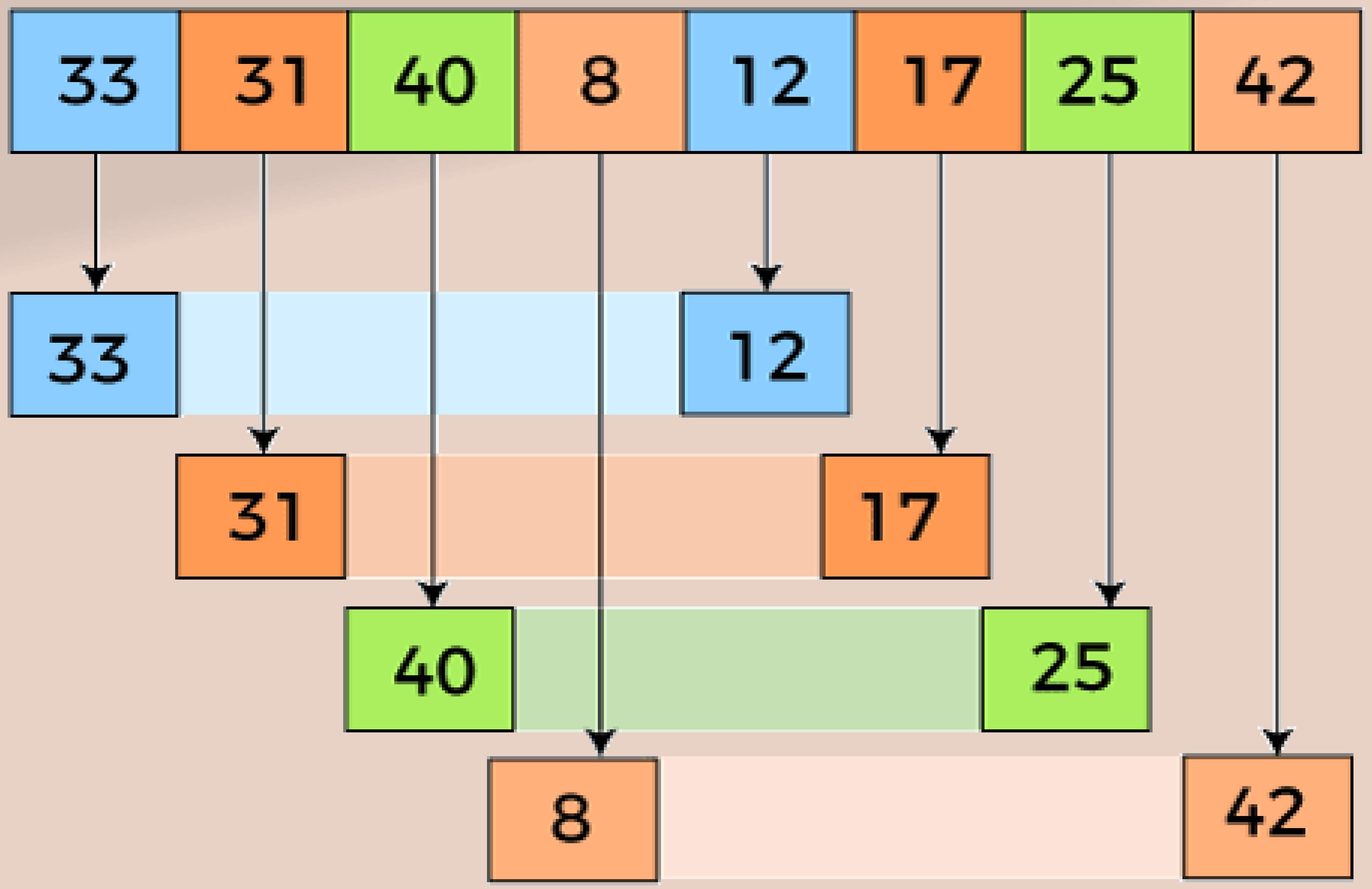
Let the Elements of array are:

33	31	40	8	12	17	25	42

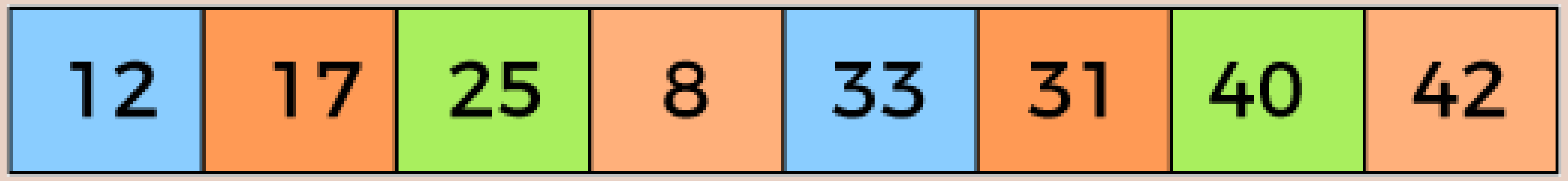
We will use the original sequence of shell sort, i.e., N/2, N/4,....,1 as the intervals.

In the first loop, n is equal to 8 (size of the array), so the elements are lying at the interval of 4 (n/2 = 4). Elements will be compared and swapped if they are not in order.

How Shell Sort Works?



At the interval of 4, the sublists are {33, 12}, {31, 17}, {40, 25}, {8, 42}.

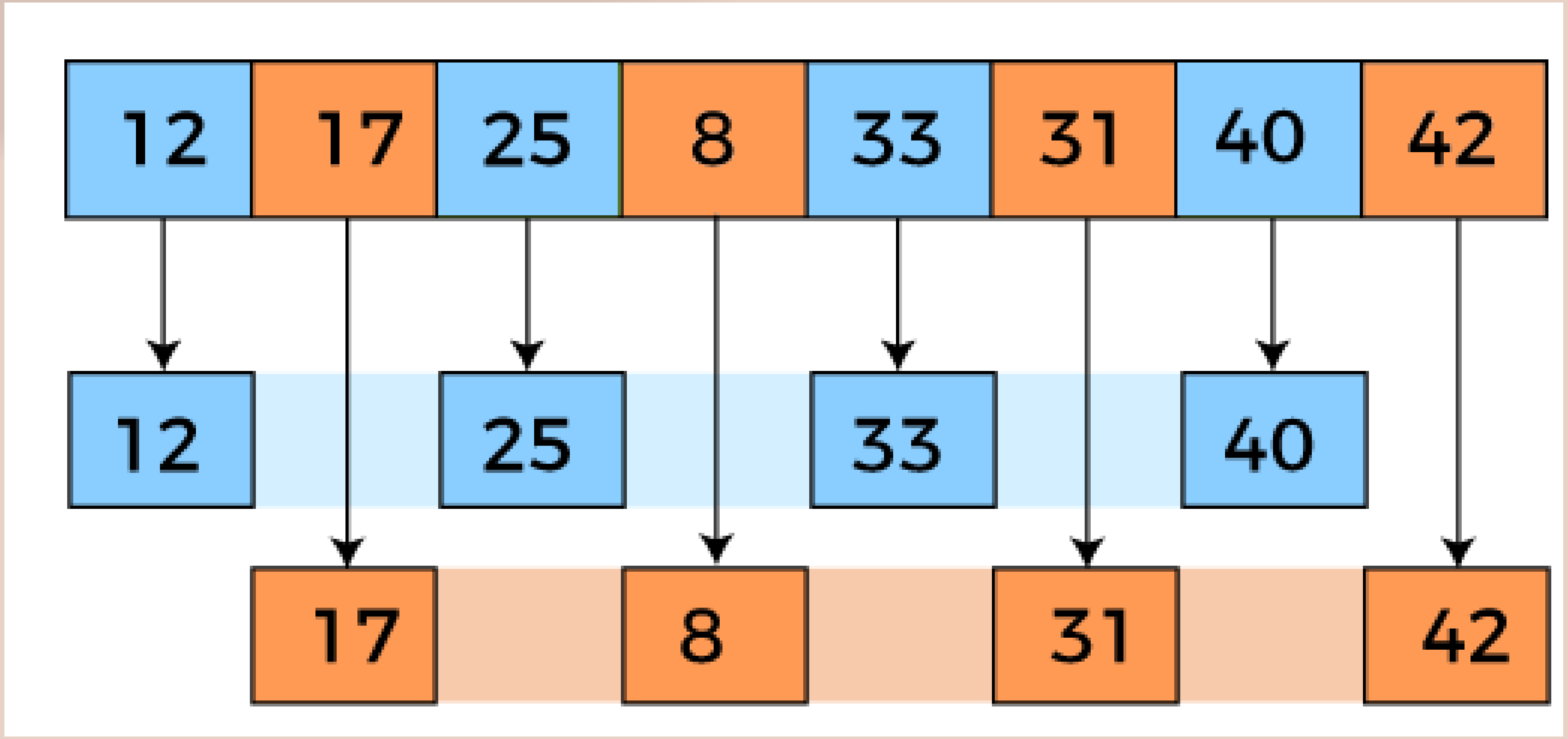


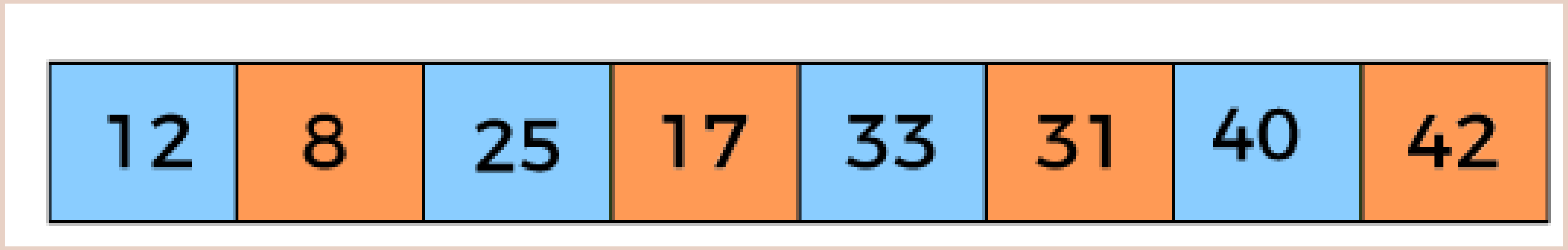
How Shell Sort Works?

In the second loop, elements are lying at the interval of 2 (n/4 = 2), where n = 8

Now, we are taking the interval of 2 to sort the rest of the array. With an interval of 2, two sublists will be generated - {12, 25, 33, 40}, and {17, 8, 31, 42}.

How Shell Sort Works?



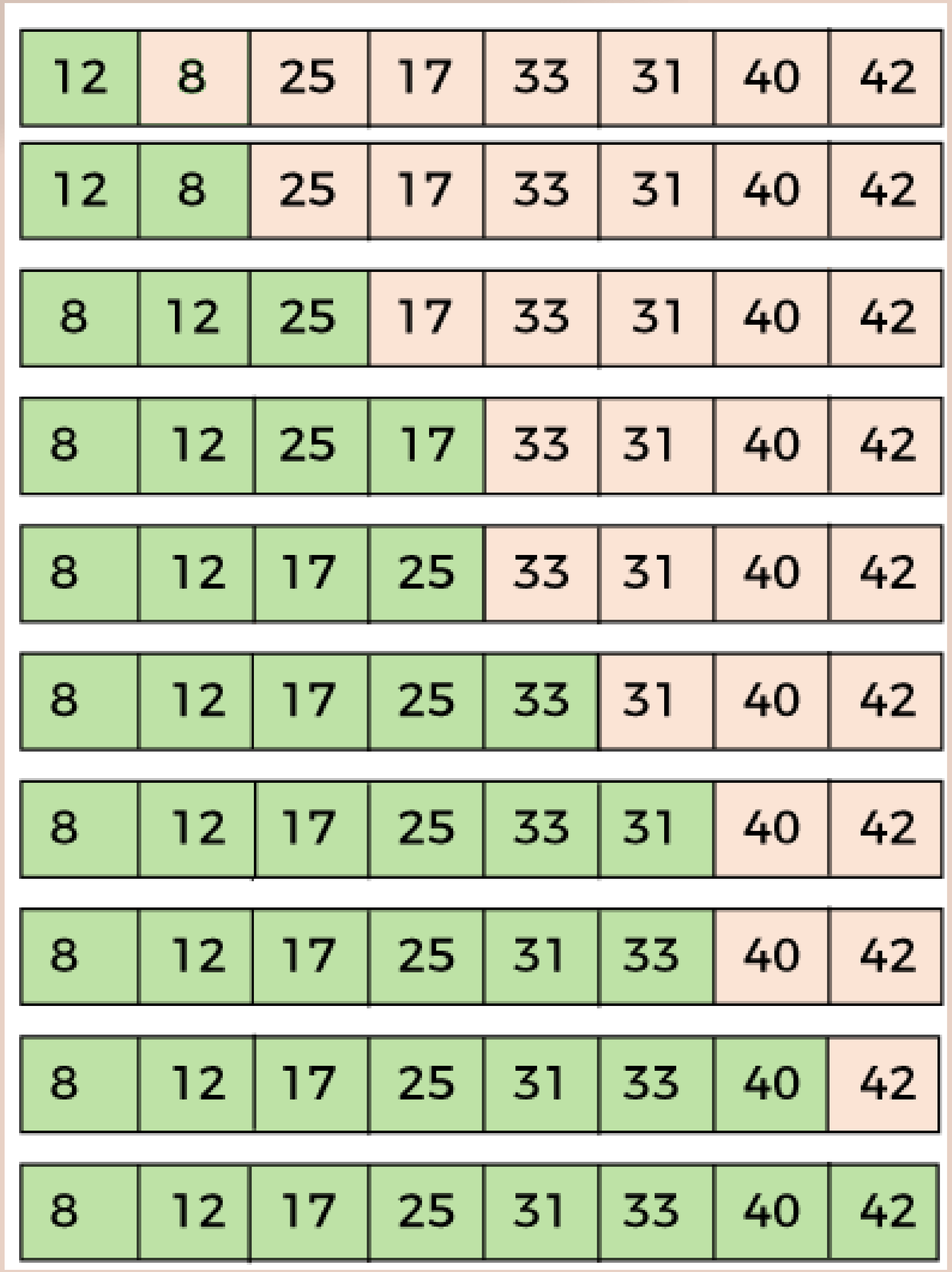


How Shell Sort Works?

In the third loop, elements are lying at the interval of 1 (n/8 = 1), where n = 8. At last, we use the interval of value 1 to sort the rest of the array elements. In this step, shell sort uses insertion sort to sort the array elements.

Now, the array is sorted in ascending order.

How Shell Sort Works?



12	8	25	17	33	31	40	42
12	8	25	17	33	31	40	42
8	12	25	17	33	31	40	42
8	12	25	17	33	31	40	42
8	12	17	25	33	31	40	42
8	12	17	25	33	31	40	42
8	12	17	25	33	31	40	42
8	12	17	25	31	33	40	42
8	12	17	25	31	33	40	42
8	12	17	25	31	33	40	42

Linear Search

Linear search algorithm finds a given element in a list of elements with O(n) time complexity where n is total number of elements in the list. This search process starts comparing search element with the first element in the list. If both are matched then result is element found otherwise search element is compared with the next element in the list. Repeat the same until search element is compared with the last element in the list, if that last element also doesn't match, then the result is "Element not found in the list". That means, the search element is compared with element by element in the list.

Linear search is implemented using following steps...

Step 1 - Read the search element from the user.

Step 2 - Compare the search element with the first element in the list.

Step 3 - If both are matched, then display "Given element is found!!!" and terminate the function

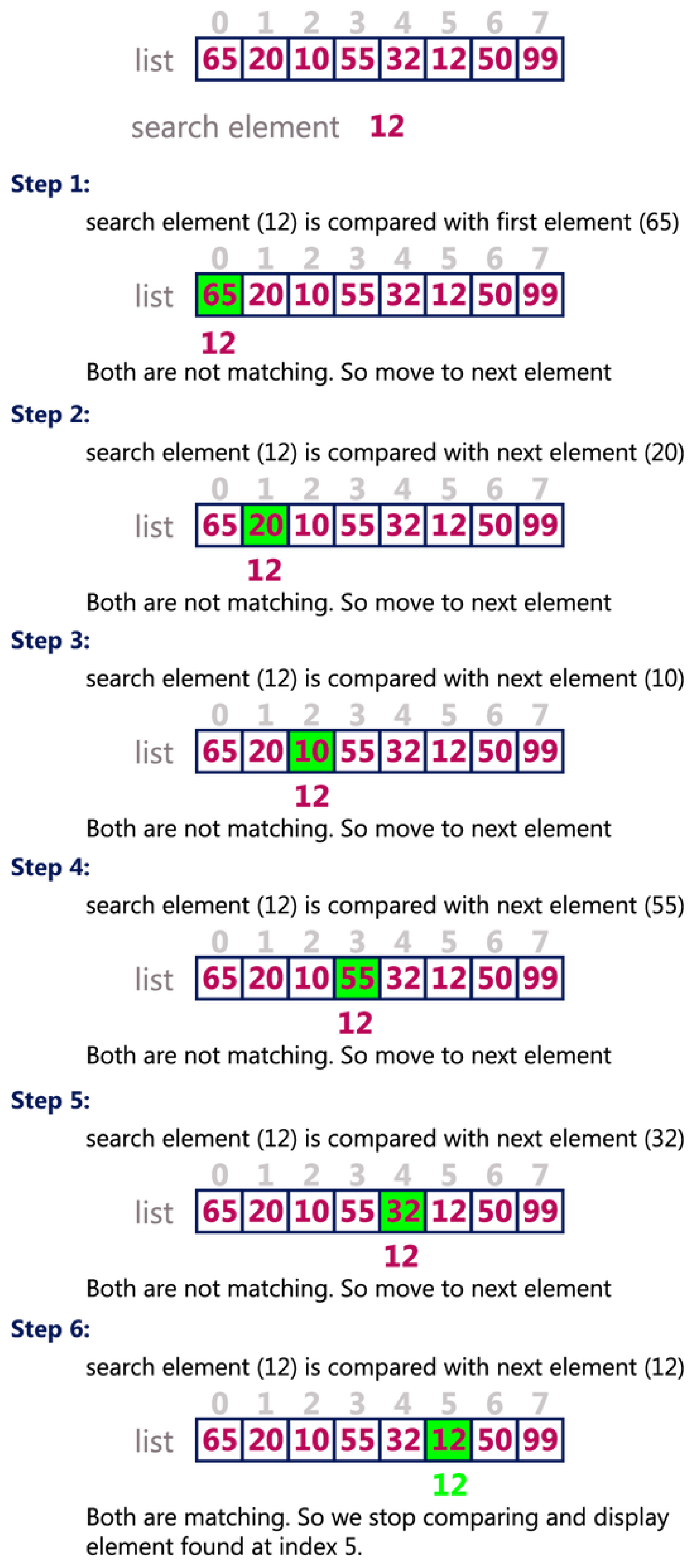
Linear search is implemented using following steps...

Step 4 - If both are not matched, then compare search element with the next element in the list.

Step 5 - Repeat steps 3 and 4 until search element is compared with last element in the list.

Step 6 - If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

How Linear Search Works?



	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

search element 12

Step 1:

search element (12) is compared with first element (65)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
	12							

Both are not matching. So move to next element

Step 2:

search element (12) is compared with next element (20)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
		12						

Both are not matching. So move to next element

Step 3:

search element (12) is compared with next element (10)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
			12					

Both are not matching. So move to next element

Step 4:

search element (12) is compared with next element (55)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
				12				

Both are not matching. So move to next element

Step 5:

search element (12) is compared with next element (32)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
					12			

Both are not matching. So move to next element

Step 6:

search element (12) is compared with next element (12)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
						12		

Both are matching. So we stop comparing and display element found at index 5.

Binary Search

Binary search algorithm finds a given element in a list of elements with O(log n) time complexity where n is total number of elements in the list. The binary search algorithm can be used with only a sorted list of elements. That means the binary search is used only with a list of elements that are already arranged in an order. The binary search can not be used for a list of elements arranged in random order.

Binary search is implemented using following steps...

Step 1 - Read the search element from the user.
Step 2 - Find the middle element in the sorted list.
Step 3 - Compare the search element with the middle element in the sorted list.
Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.
Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.

Binary search is implemented using following steps...

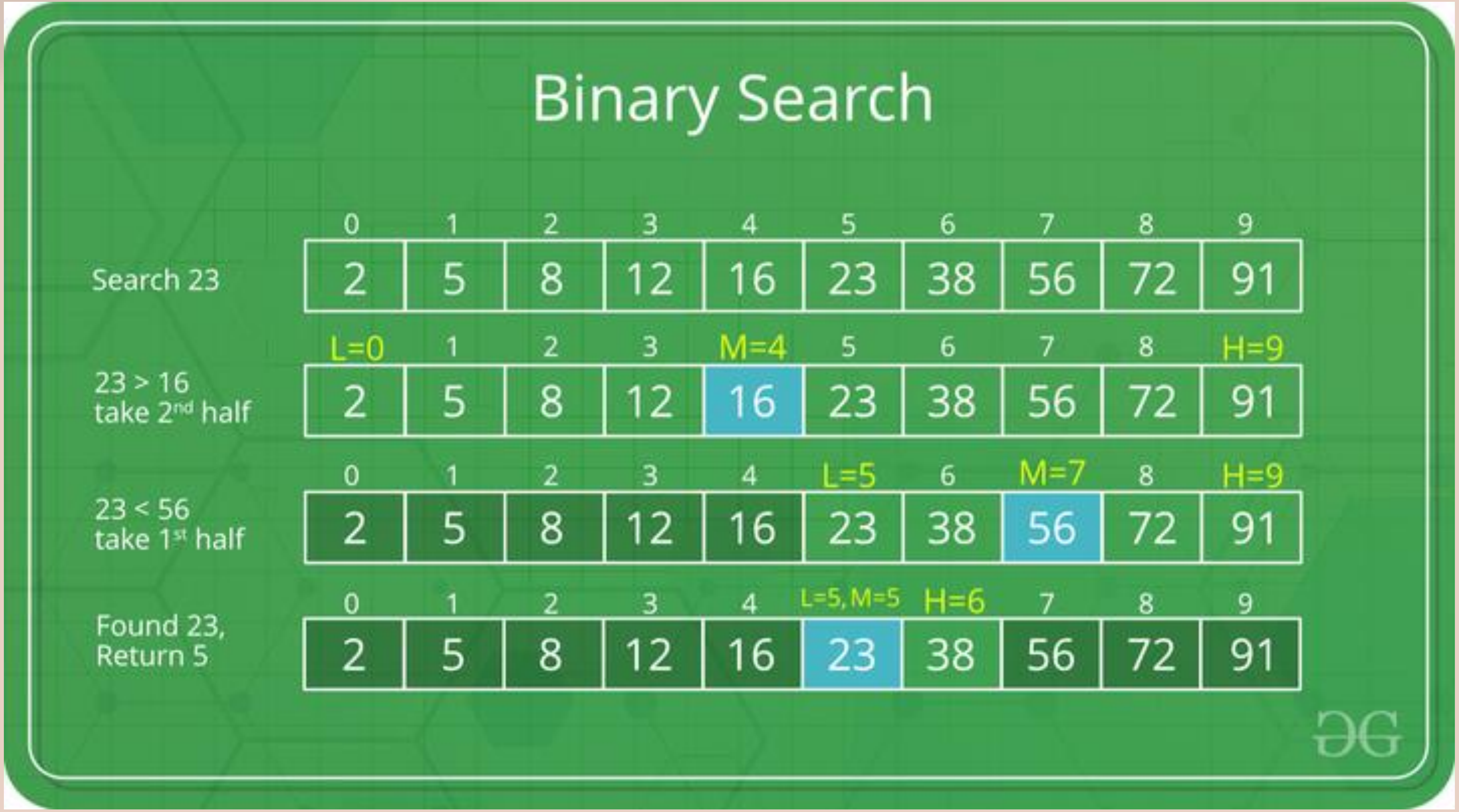
Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element.

Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

How Binary Search Works?



References

https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm#
https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm
https://www.mygreatlearning.com/blog/insertion-sort-with-a-real-world-example/#:~:text=Now%20speaking%20technically%2C%20the%20insertion%20sort%20follows%20the,up%20to%20make%20space%20for%20the%20swapped%20element.
https://www.educba.com/merge-sort-algorithm/
https://en.m.wikipedia.org/wiki/Quicksort
https://www.javatpoint.com/counting-sort
https://www.javatpoint.com/radix-sort
https://www.programiz.com/dsa/heap-sort
https://www.javatpoint.com/shell-sort
http://www.btechsmartclass.com/data_structures/binary-search.html
http://www.btechsmartclass.com/data_structures/linear-search.html
https://en.wikipedia.org/wiki/Linear_search#Analysis