What is Sorting?

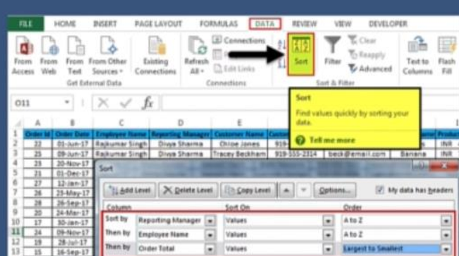
By definition sorting refers to arranging data in a particular format : either ascending or descending.

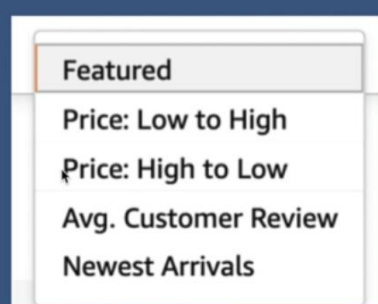
What is Sorting?

Practical Use of Sorting

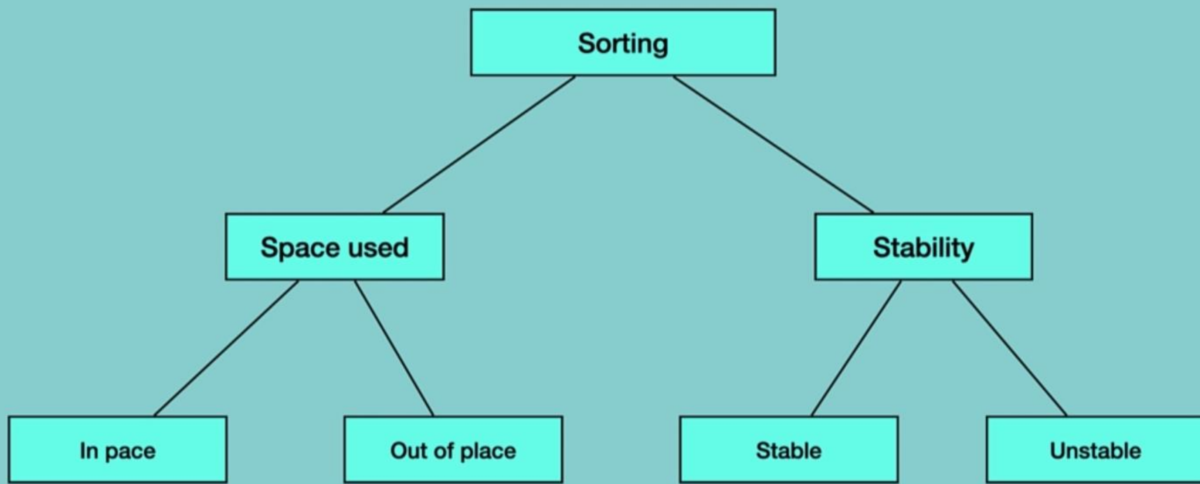
<u>Microsoft Excel :</u> Built in functionality to sort data

<u>Online Stores:</u> Online shopping websites generally have option sor sorting by price, review, ratings..





Types of Sorting?



Space used

In place sorting : Sorting algorithms which does not require any extra space for sorting
Example : Bubble Sort

70	10	80	30	20	40	60	50	90

10	20	30	40	50	60	70	80	90

Out place sorting : Sorting algorithms which requires an extra space for sorting
Example : Merge Sort

70	10	80	30	20	40	60	50	90

10	20	30	40	50	60	70	80	90

Stability

Stable sorting : If a sorting algorithm after sorting the contents does not change the sequence of similar content in which they appear, then this sorting is called stable sorting.

Example : Insertion Sort

70	10	80	40	20	40	60	50	90

10	20	40	40	50	60	70	80	90

UnStable sorting : If a sorting algorithm after sorting the content changes the sequence of similar content in which they appear, then it is called unstable sort.

Example : Quick Sort

70	10	80	40	20	40	60	50	90

10	20	40	40	50	60	70	80	90

Stability

UnStable sorting example

Unsorted data	
Name	**Age**
Renad	7
Nick	6
Richard	6
Parker	7
Sofia	7

Sorted by name	
Name	**Age**
Nick	6
Parker	7
Renad	7
Richard	6
Sofia	7

Sorted by age (stable)	
Name	**Age**
Nick	6
Richard	6
Parker	7
Renad	7
Sofia	7

Sorted by age (unstable)	
Name	**Age**
Nick	6
Richard	6
Renad	7
Parker	7
Sofia	7

GROUP BY command in DBMS is based on this concept.

Sorting Terminology

Increasing Order

- If successive element is greater than the previous one
- Example : 1, 3, 5, 7, 9 ,11

Decreasing Order

- If successive element is less than the previous one
- Example : 11, 9, 7, 5, 3, 1

Non Increasing Order

- If successive element is less than or equal to its previous element in the sequence.
- Example : 11, 9, 7, 5, 5, 3, 1

Non Decreasing Order

- If successive element is greater than or equal to its previous element in the sequence
- Example : 1, 3, 5, 7, 7, 9, 11

When we see "Non" then we should assume that there may be duplicate elements

Sorting Algorithms

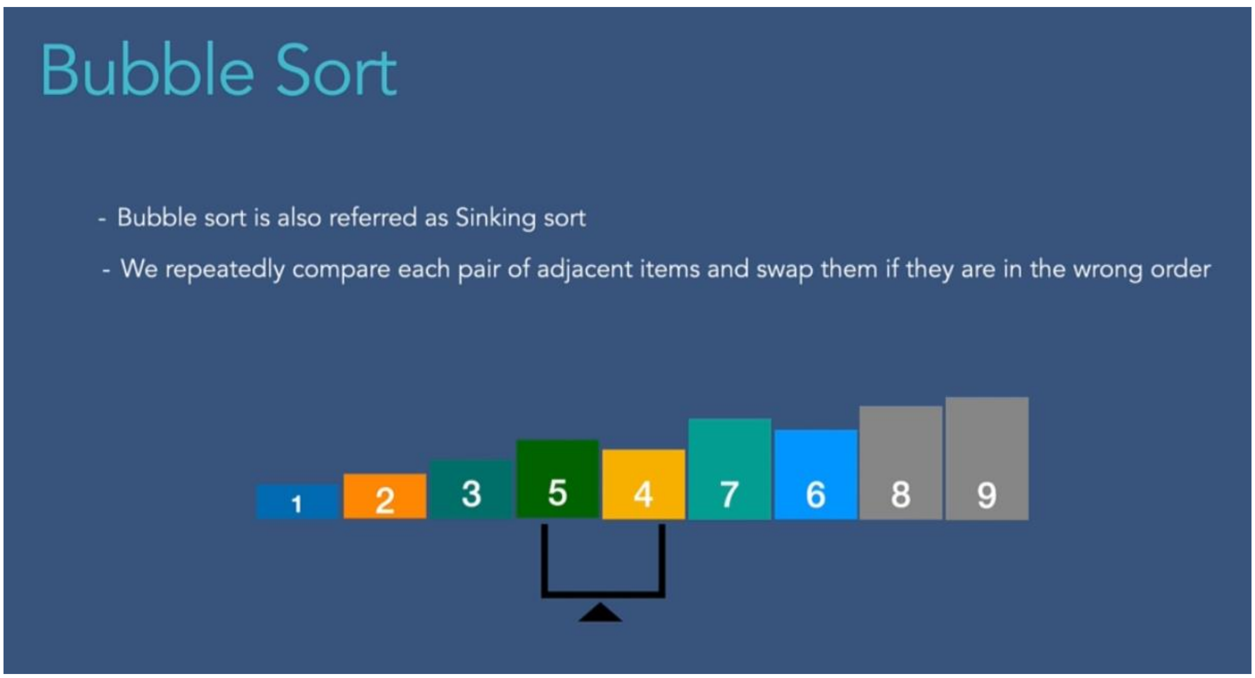
Bubble sort
Selection sort
Insertion sort
Bucket sort
Merge sort
Quick sort
Heap sort

Which one to select?

- Stability
- Space efficient
- Time efficient

Different sorting algorithms are chosen at different instances based on our requirement. We should choose the best algorithm for our need whether it should be space efficient or it should be time efficient, etc.

Initial array



Intermediate State of Bubble Sort

Here the numbers are not sorted. In Bubble sort, we check for the biggest element between two numbers and swap the biggest number to the right side keeping the smallest number to the left. After one iteration the largest element will be at the last position indicating that it is sorted. Similarly proceeding in this way, the numbers will get sorted.

Animation link:

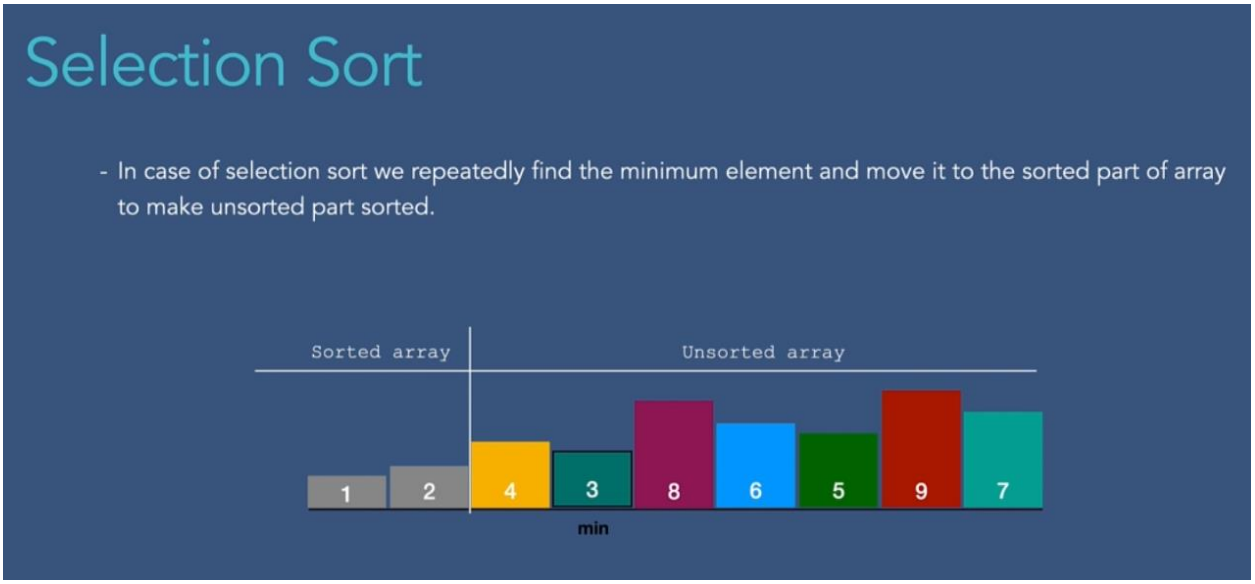
https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif

Visualization of the code:
https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+BubbleSort%7B%0A++++public+static+void+main(String%5B%5D+args)+%7B%0A++++++++int+arr%5B%5D+%3D+%7B5,1,3,9,0%7D%3B%0A++++++++int+n%3Darr.length%3B%0A++++++++for(int+i%3D0%3Bi%3Cn-1%3Bi%2B%2B)%7B%0A++++++++++++for(int+j%3D0%3Bj%3Cn-1%3Bj%2B%2B)%7B%0A++++++++++++++++if(arr%5Bj%5D%3Earr%5Bj%2B1%5D)%7B%0A++++++++++++++++++++int+temp+%3D+arr%5Bj%5D%3B%0A++++++++++++++++++++arr%5Bj%5D%3Darr%5Bj%2B1%5D%3B%0A++++++++++++++++++++arr%5Bj%2B1%5D%3Dtemp%3B%0A++++++++++++++++%7D%0A++++++++++++%7D%0A++++++++%7D%0A%0A++++%7D%0A%7D&mode=display&curInstr= 88

Time complexity: O(N^2)

Space complexity: O(1)

Initial array



Intermediate array in the Selection sort

In Selection Sort, there are two areas in an array one is the sorted area and the other is the unsorted area. We check for the smallest element in the unsorted array and the swap the left most element of the unsorted array which will next be the part of the sorted array.

Animation of Selection Sort:

https://upload.wikimedia.org/wikipedia/commons/9/94/Selection-Sort-Animation.gif

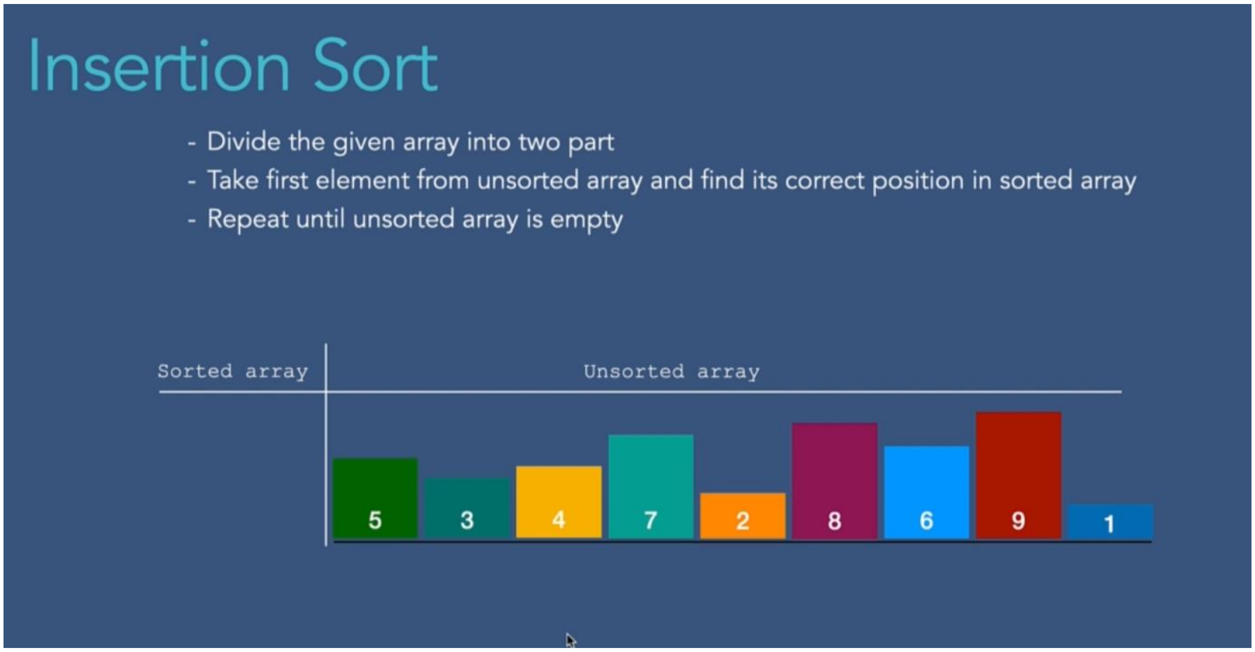
Visualization of the code:

https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+SelectionSort+%7B%0A++++public+static+void+main(String%5B%5D+args)+%7B%0A++++++++int+arr%5B%5D+%3D+%7B10,2,5,8,1,9%7D%3B%0A++++++++int+n%3Darr.length%3B%0A%0A++++++++for(int+i%3D0%3Bi%3Cn%3Bi%2B%2B)%7B

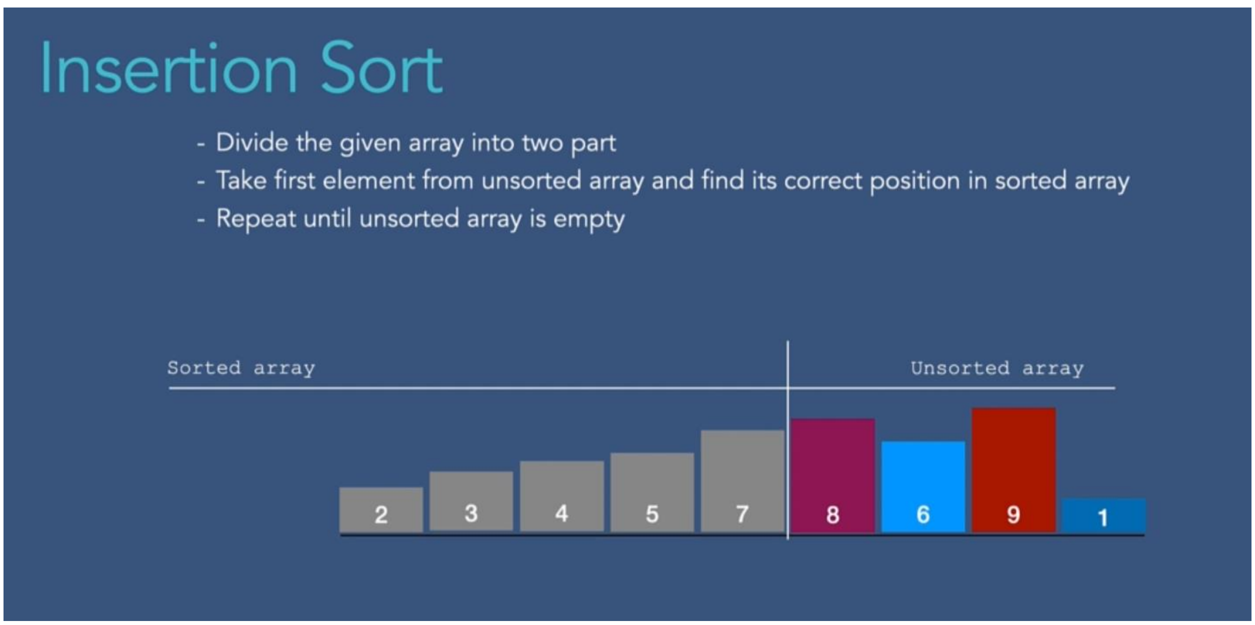
%0A++++++++++++int+minIndex%3Di%3B%0A++++++++++++for(int+j%3Di%2B1%3Bj%3Cn%3Bj%2B%2B)%7B%0A++++++++++++++++if(arr%5Bj%5D%3Carr%5BminIndex%5D)%7B%0A++++++++++++++++++++minIndex%3Dj%3B%0A++++++++++++++++%7D%0A++++++++++++%7D%0A++++++++++++if(minIndex!%3Di)%7B+//optimization%0A++++++++++++++++int+temp%3Darr%5Bi%5D%3B%0A++++++++++++++++arr%5Bi%5D%3Darr%5BminIndex%5D%3B%0A++++++++++++++++arr%5BminIndex%5D%3Dtemp%3B%0A++++++++++++%7D%0A++++++++%7D%0A%0A++++++++for(int+k%3D0%3Bk%3Cn%3Bk%2B%2B)%7B%0A++++++++++++System.out.print(arr%5Bk%5D%2B%22+%22)%3B%0A++++++++%7D%0A++++%7D%0A%7D%0A&mode=display&curInstr=116

Time complexity: O(N^2)

Space complexity: O(1)



Initial array



Intermediate array in Insertion Sort

Insertion sort is quite similar to Selection Sort. In Insertion Sort, we have sorted area and unsorted area in an array. We select the minimum element from the unsorted area of the array and find the right position for the element in the sorted area of the array. By proceeding in this similar way, the elements get sorted in the array.

Animation of Insertion Sort:

https://en.wikipedia.org/wiki/Insertion_sort#/media/File:Insertion-sort-example-300px.gif

Visualization of the code:

https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+InsertionSort+%7B%0A++++public+static+void+main(String%5B%5D+args)+%7B%0A++++++++int+arr%5B%5D+%3D%7B10,3,2,5,8,4,3,1%7D%3B%0A++++++++int+n%3Darr.length%3B%0A++++++++for(int+i%3D1%3Bi%3Cn%3Bi%2B%2B)%7B%0A++++++++++++int+temp%3Darr%5Bi%5D,j%3Di%3B+//temp+important%0A++++++++++++while(j%3E0+%26%26+arr%5Bj-1%5D%3Etemp)%7B%0A++++++++++++++++arr%5Bj%5D%3Darr%5Bj-1%5D%3B%0A++++++++++++++++j--%3B%0A++++++++++++%7D%0A++++++++++++arr%5Bj%5D%3Dtemp%3B%0A++++++++%7D%0A%0A++++++++for(int+i%3D0%3Bi%3Cn%3Bi%2B%2B)%7B%0A++++++++++++System.out.print(arr%5Bi%5D%2B%22+%22)%3B%0A++++++++%7D%0A++++%7D%0A%7D%0A&mode=display&curInstr=0

Time complexity: O(N^2)

Space complexity: O(1)

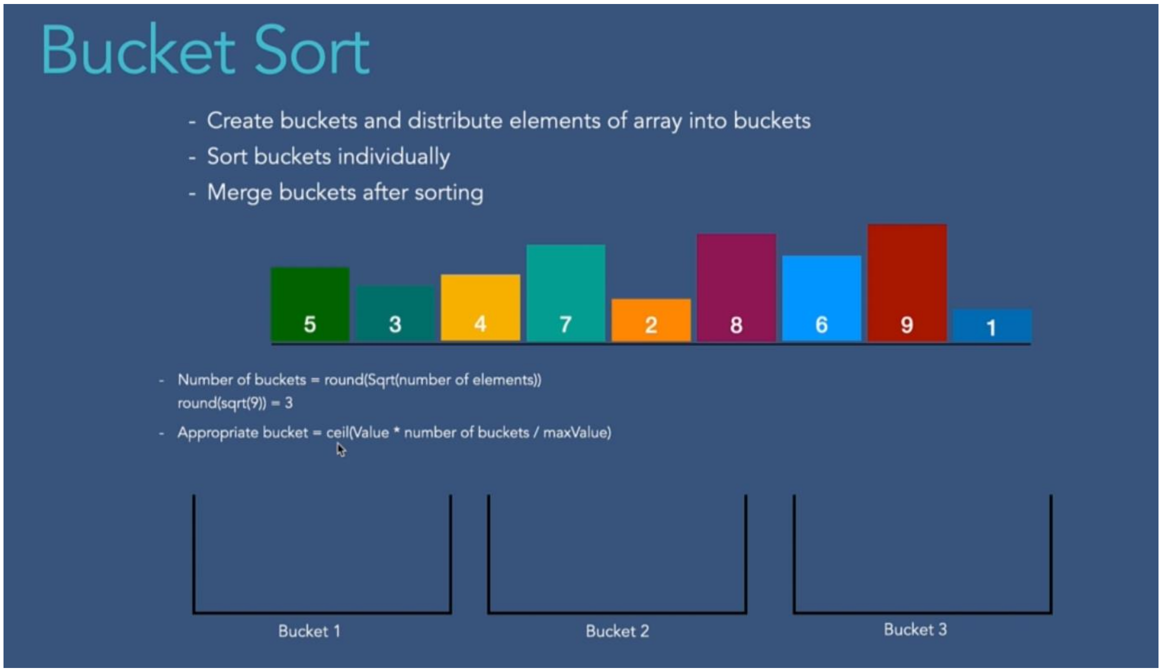
Insertion Sort

When to use Insertion Sort?

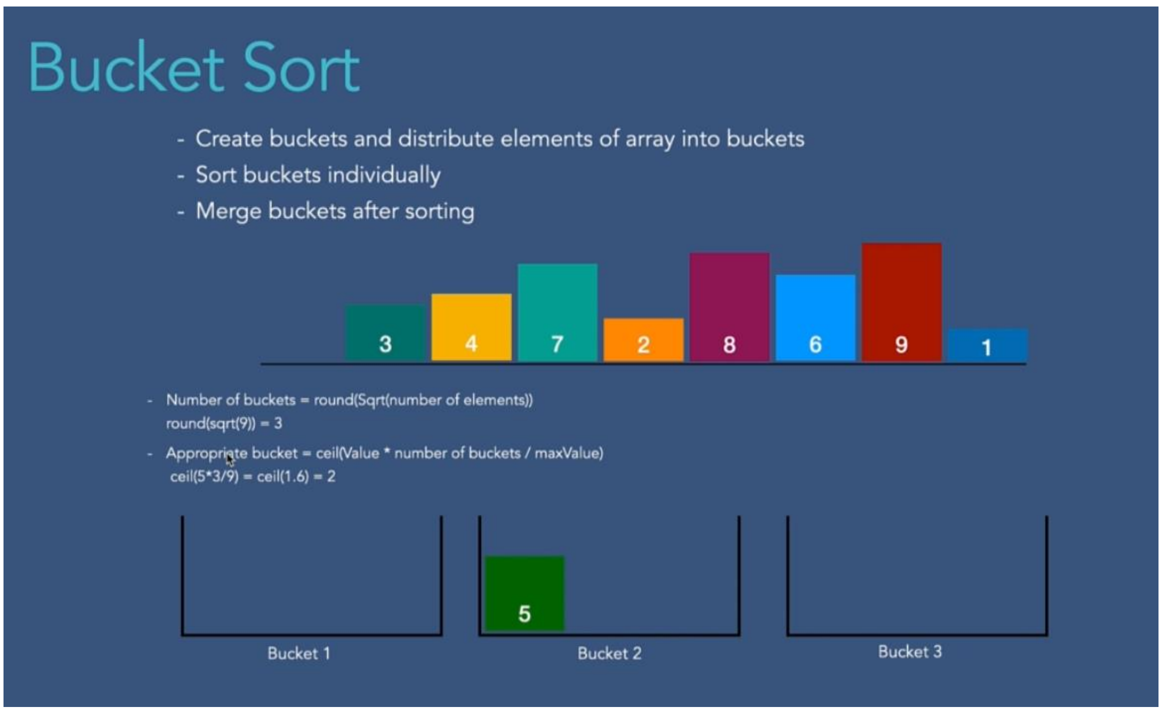
- When we have insufficient memory
- Easy to implement
- When we have continuous inflow of numbers and we want to keep them sorted

When to avoid Insertion Sort?

- When time is a concern

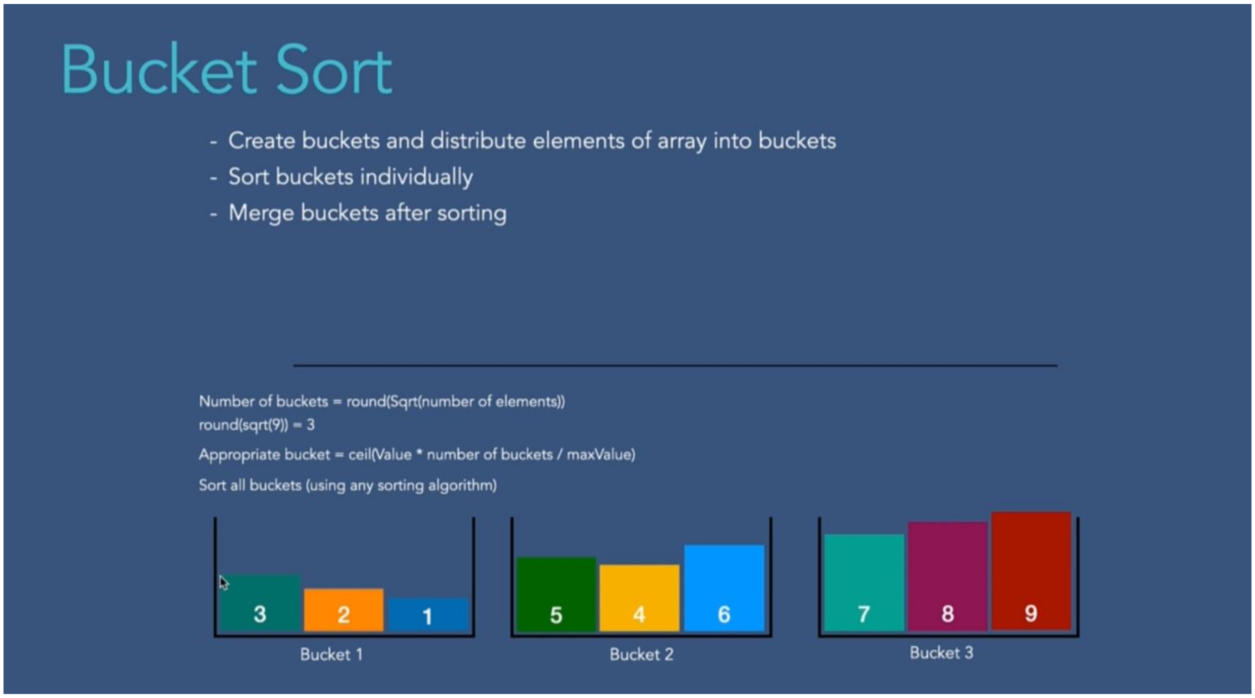


Initial state of the array

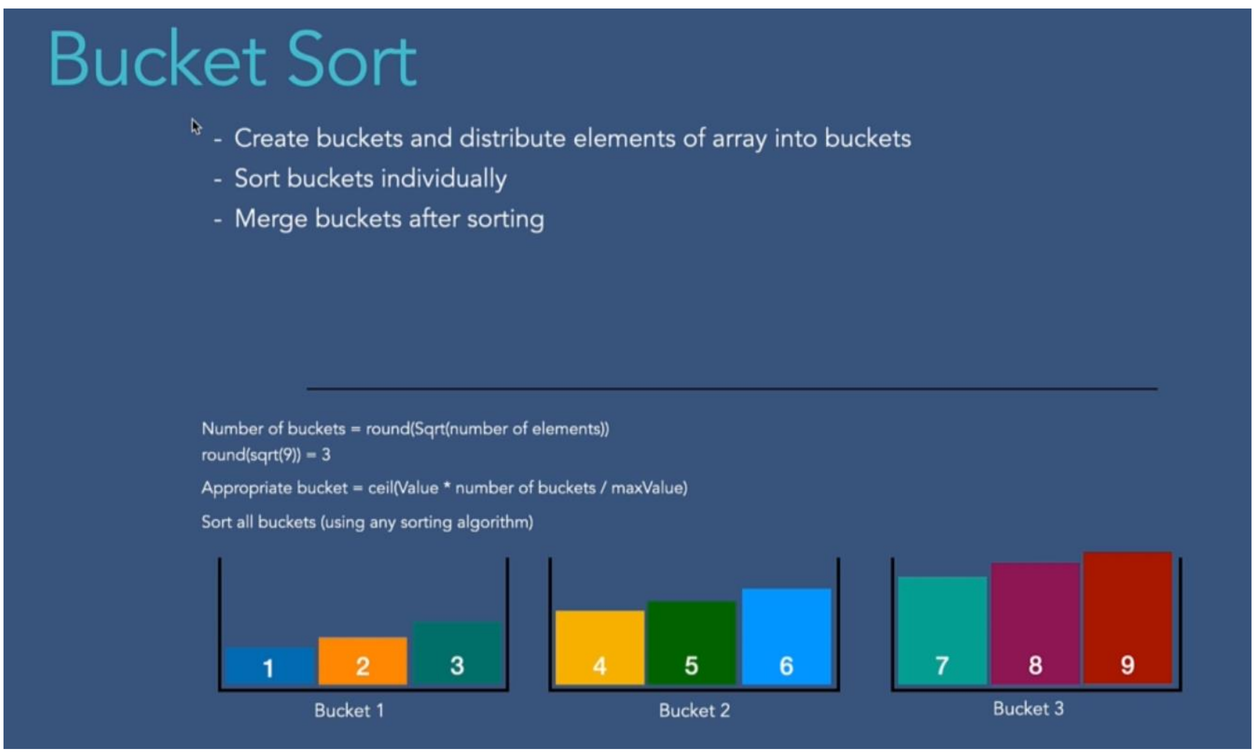


For the first element in the array which is 5

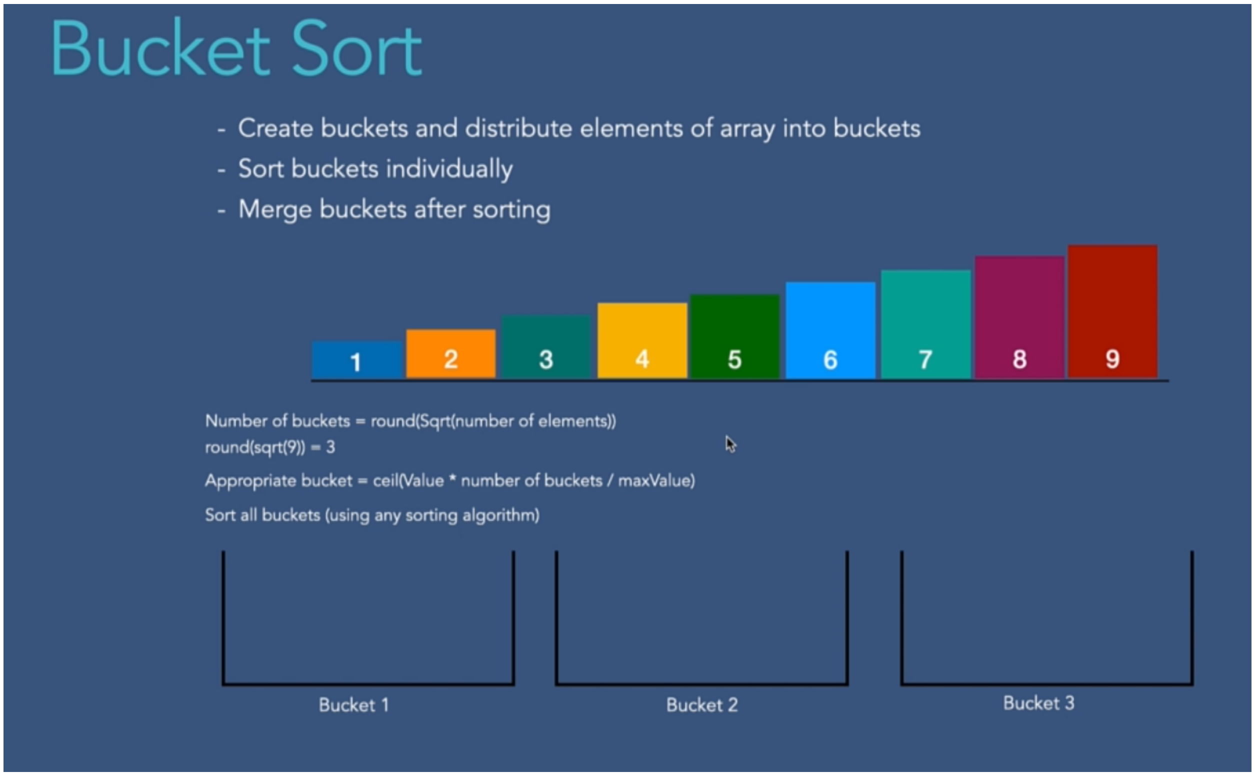
In Bucket Sort, we create buckets for elements of the array. The number of buckets is calculated by the formula in the above image. Appropriate bucket for the selected element will also be calculated by the above formula in the above images. Then, sort the bucket with anyone of the sorting algorithm. Then merging the bucket to get the sorted array as the result.



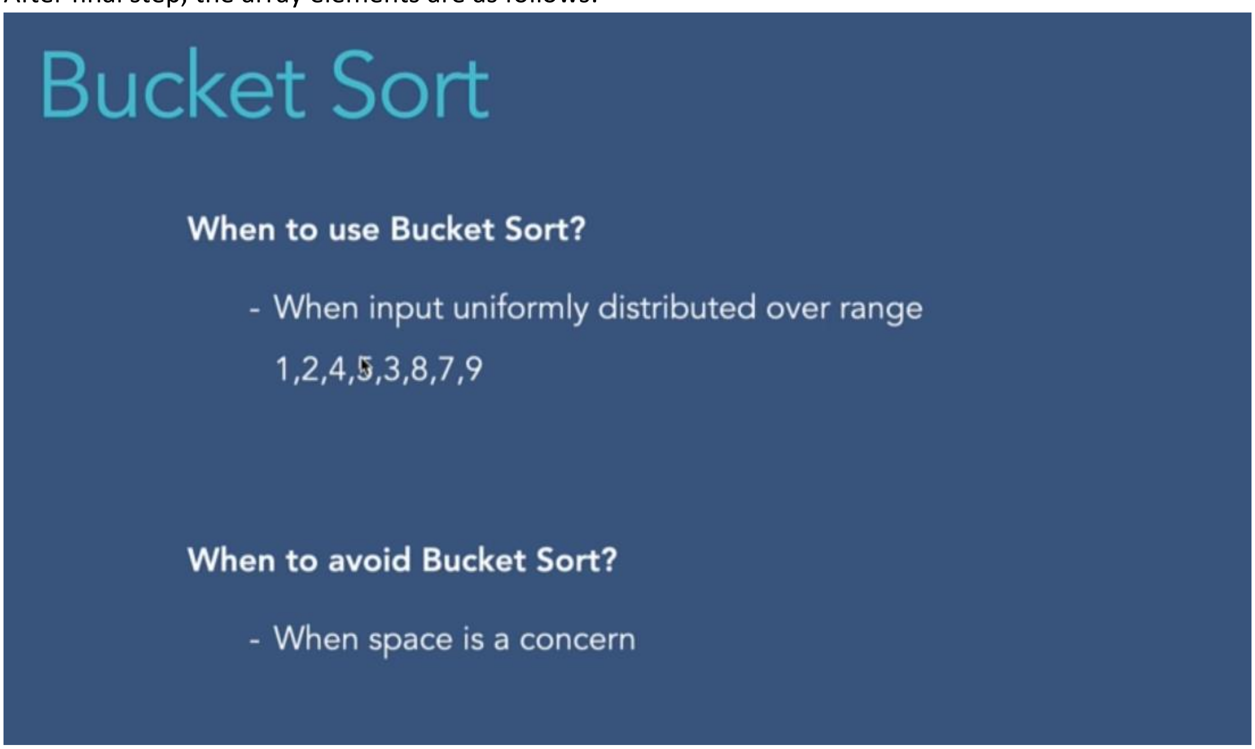
Initial state of the bucket before sorting

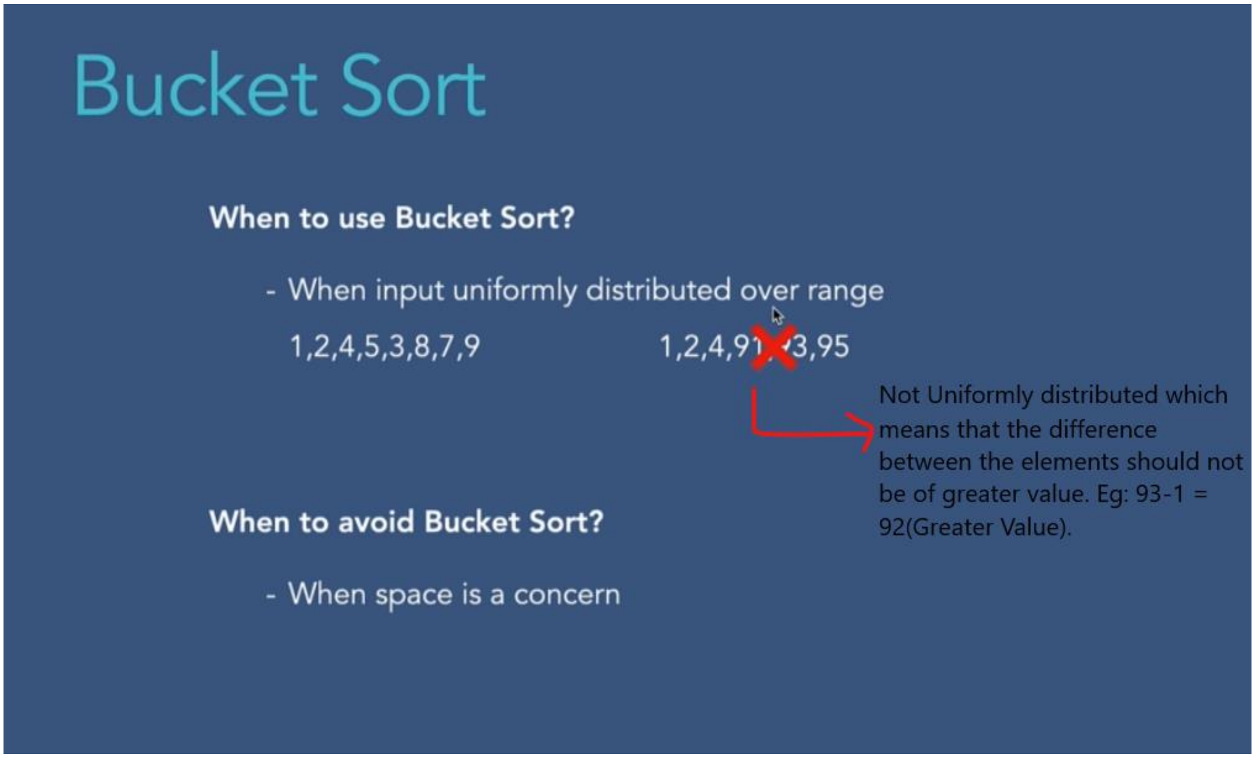


After sorting the individual buckets with any sorting algorithms.

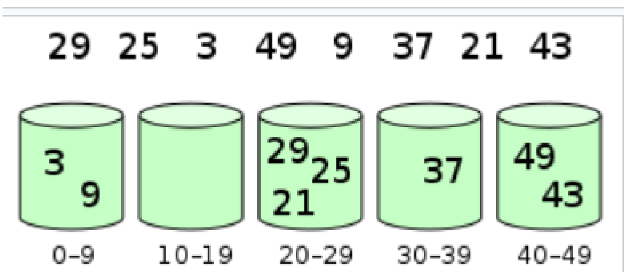


After final step, the array elements are as follows.

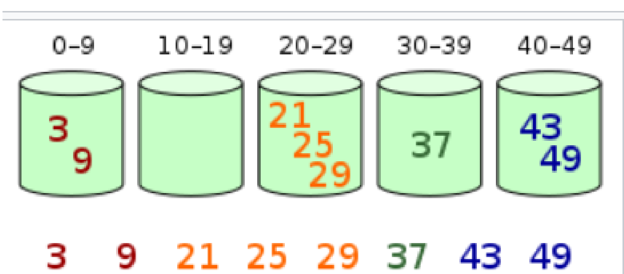




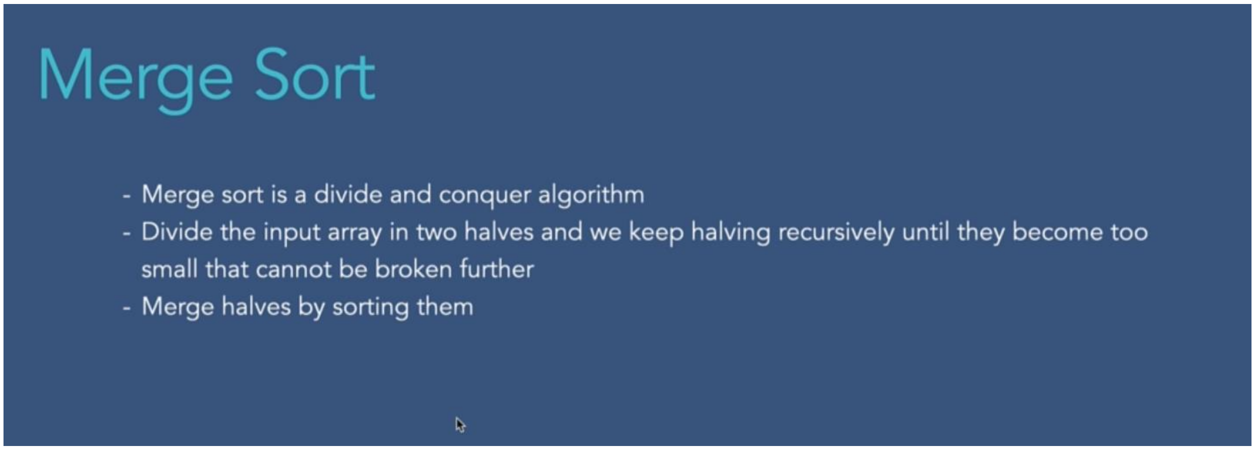
Example Scenario from Wikipedia – Quite Different: so follow the above procedure.

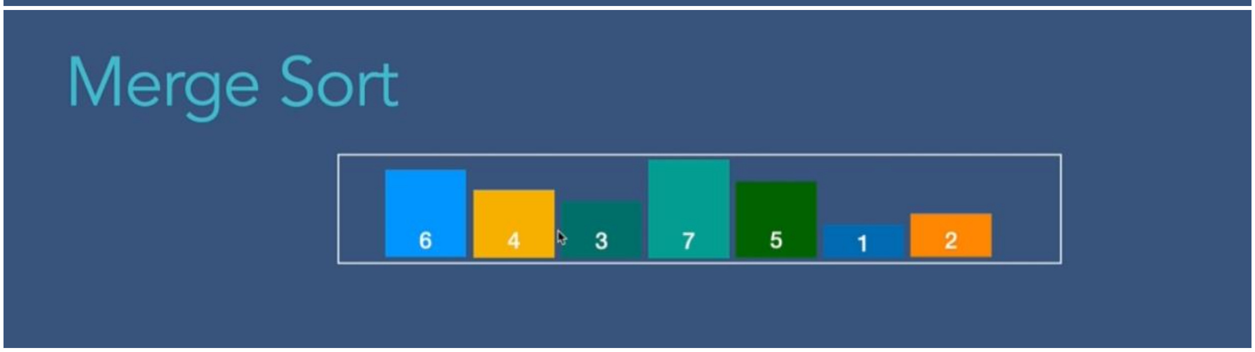


Elements are distributed among bins

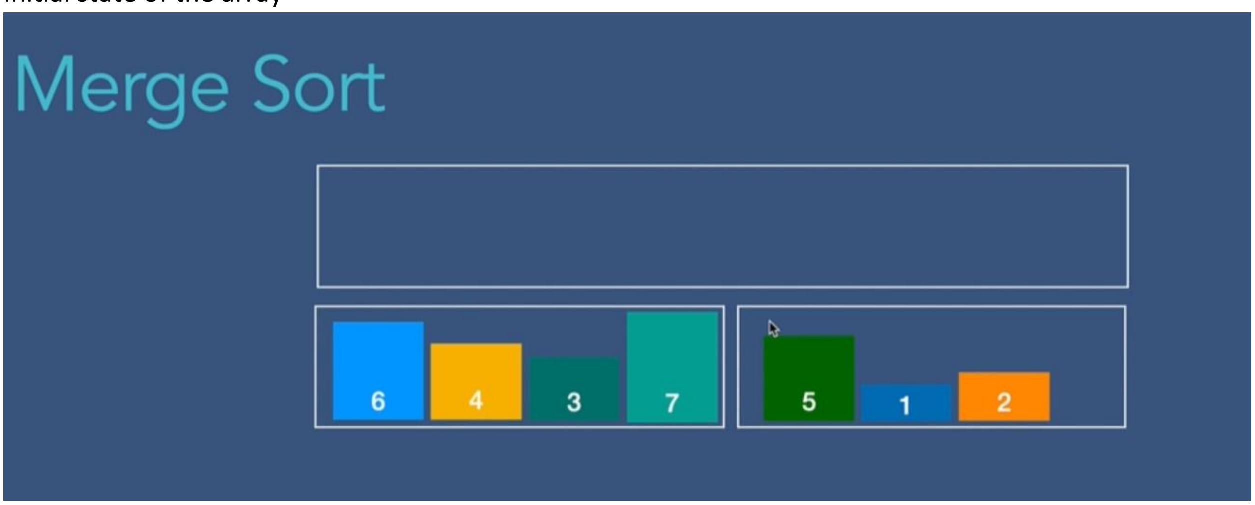


Then, elements are sorted within each bin

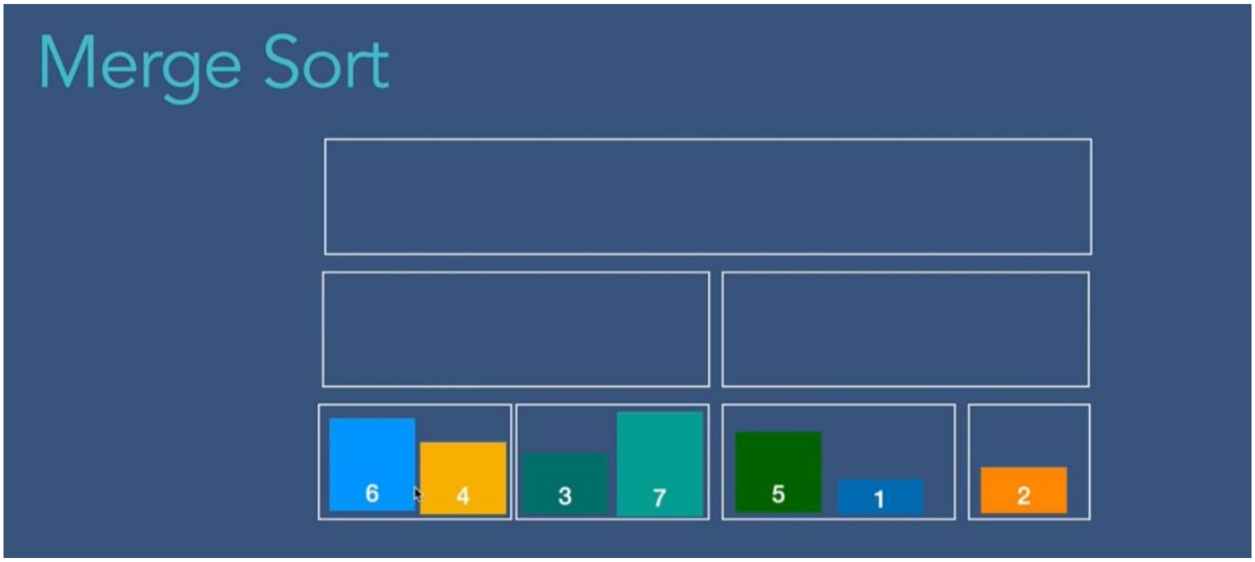


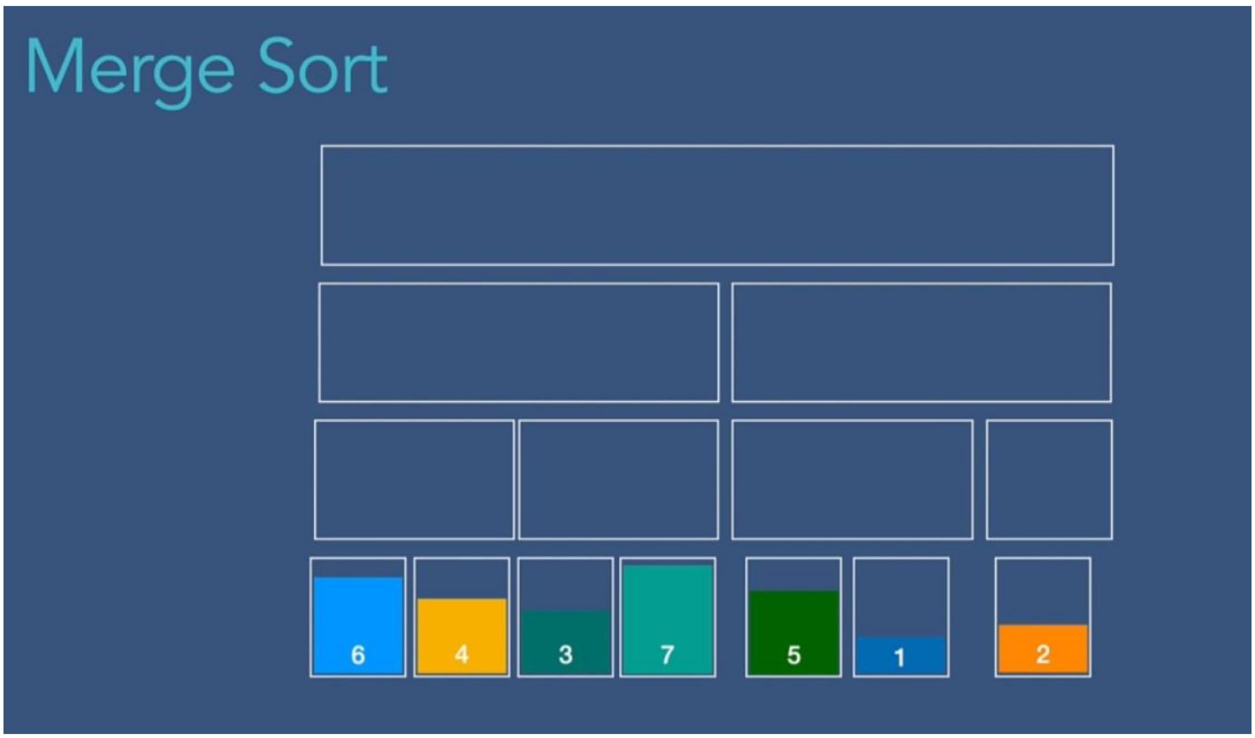


Initial state of the array

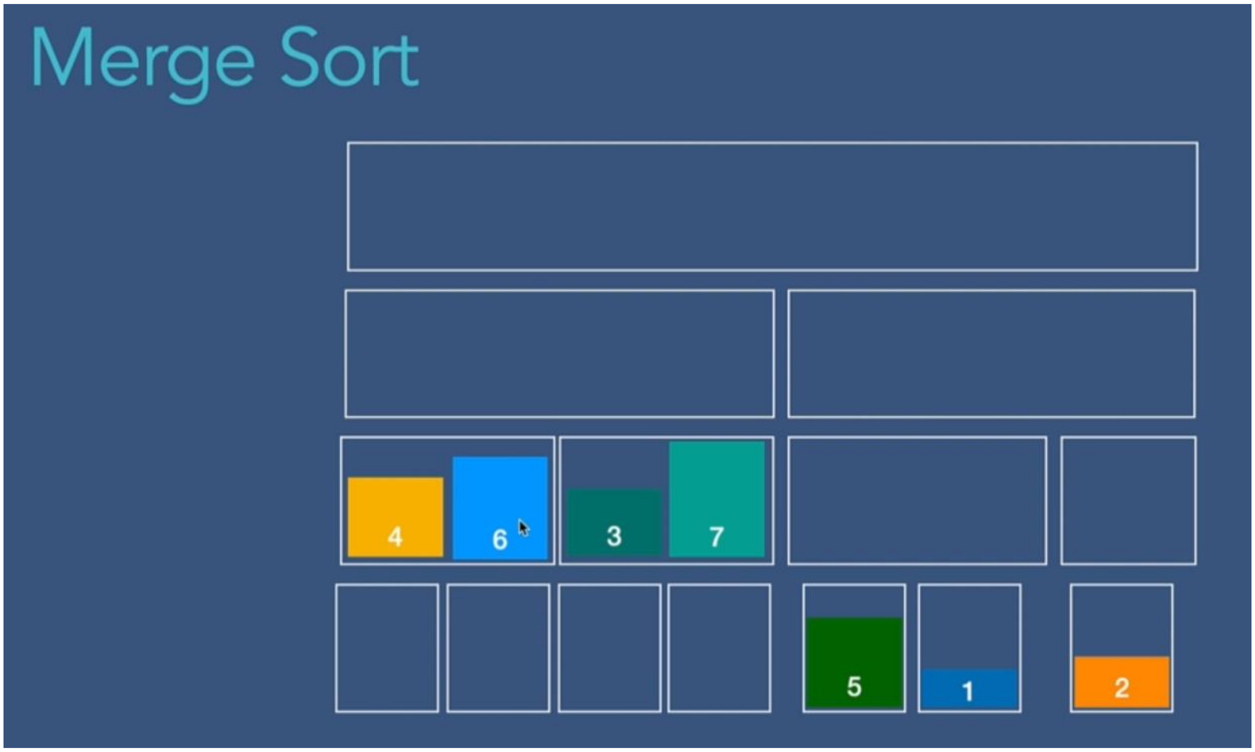


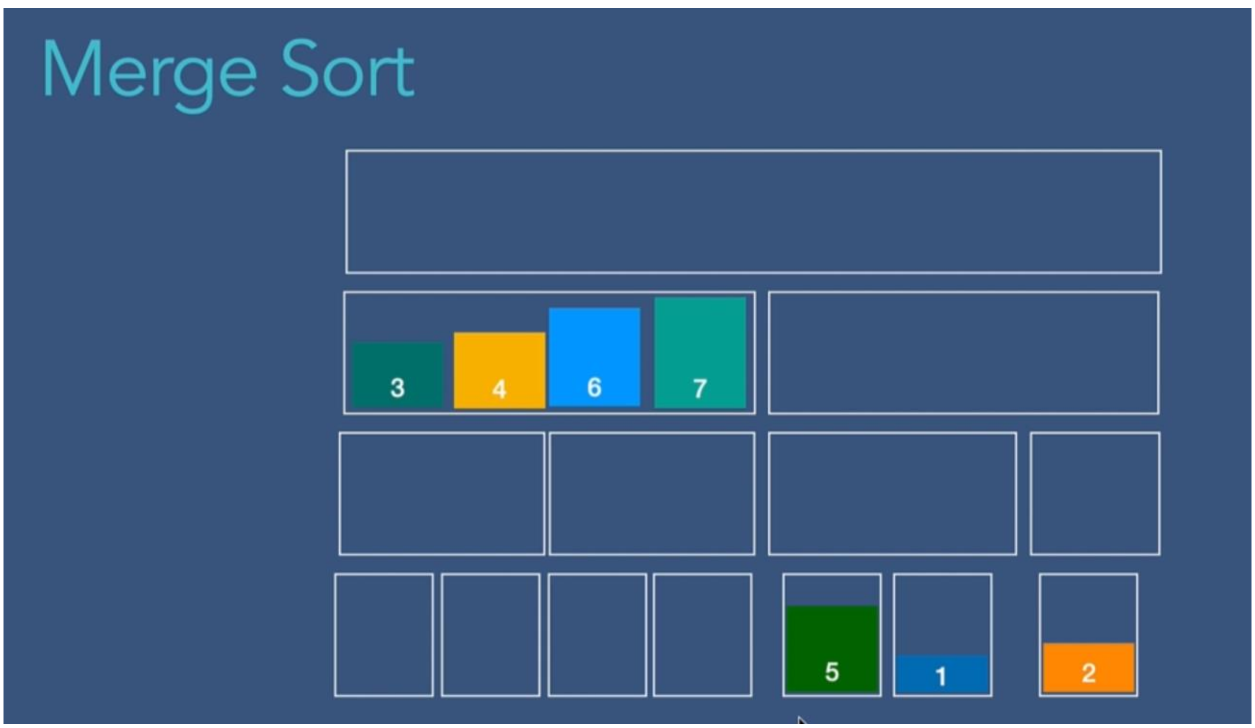
Dividing

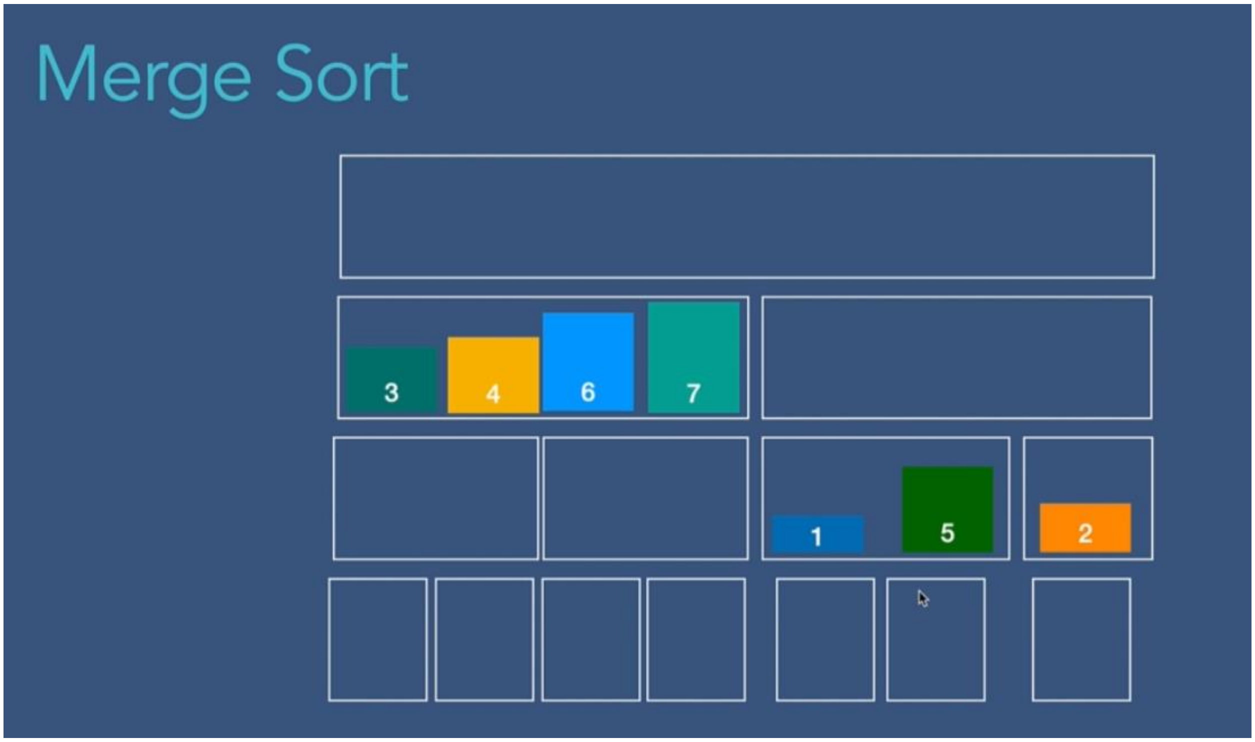

Merge Sort
6
4
3
7
5
1
2

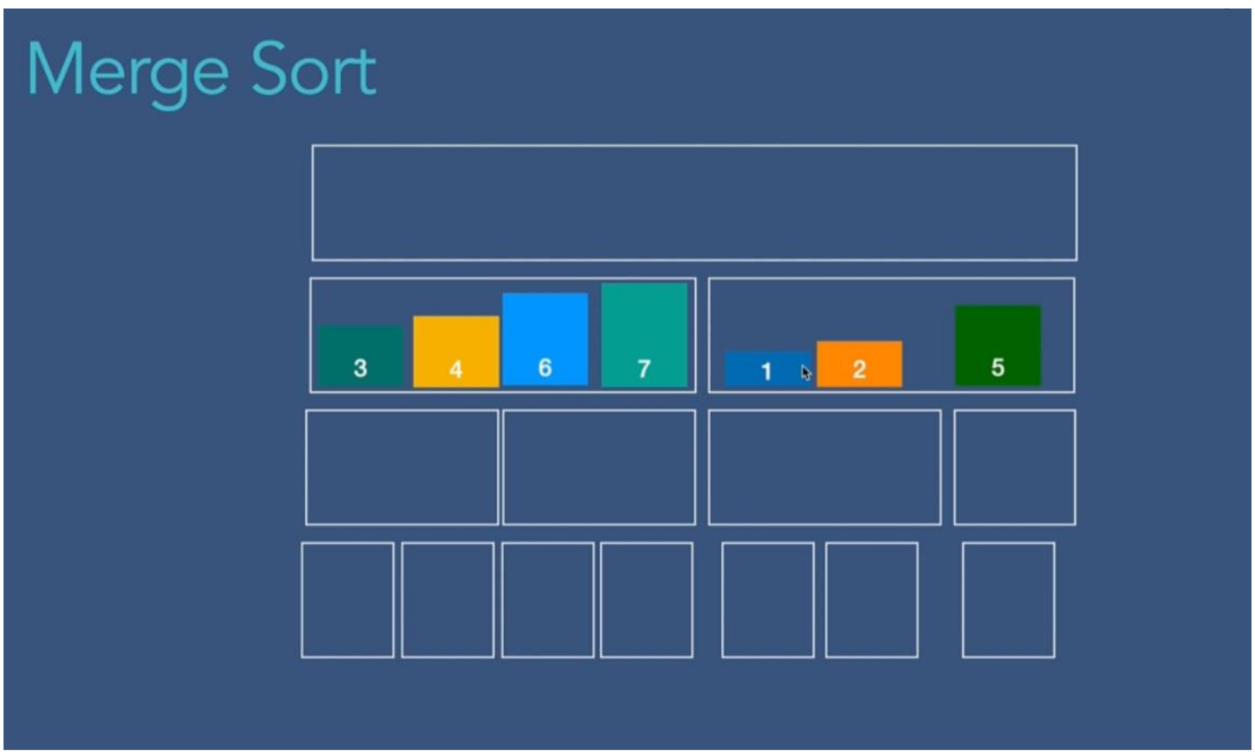

Merge Sort
6
4
3
7
5
1
2

Merging:

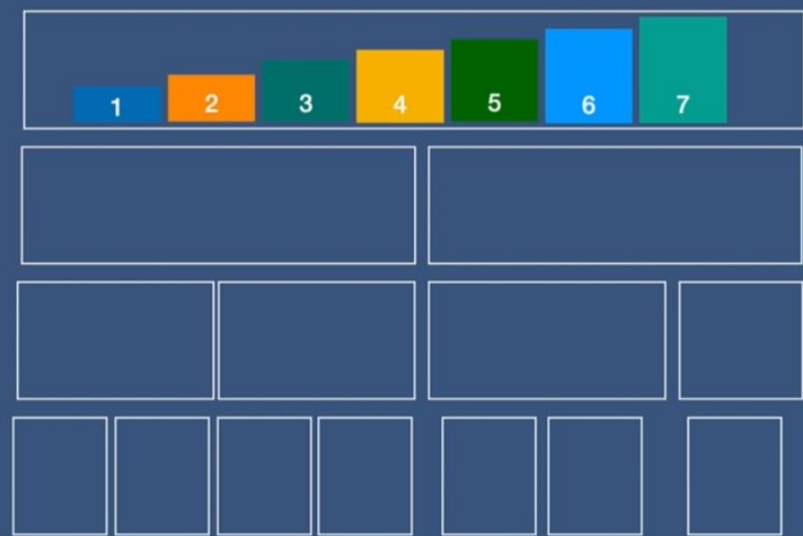





Merge Sort
3
4
6
7
1
5
2


Merge Sort
3
4
6
7
1
2
5

Merge Sort



Merge Sort

When to use Merge Sort?

- When you need stable sort
- When average expected time is O(NlogN)

When to avoid Merge Sort?

- When space is a concern

Time complexity: O(NLogN)

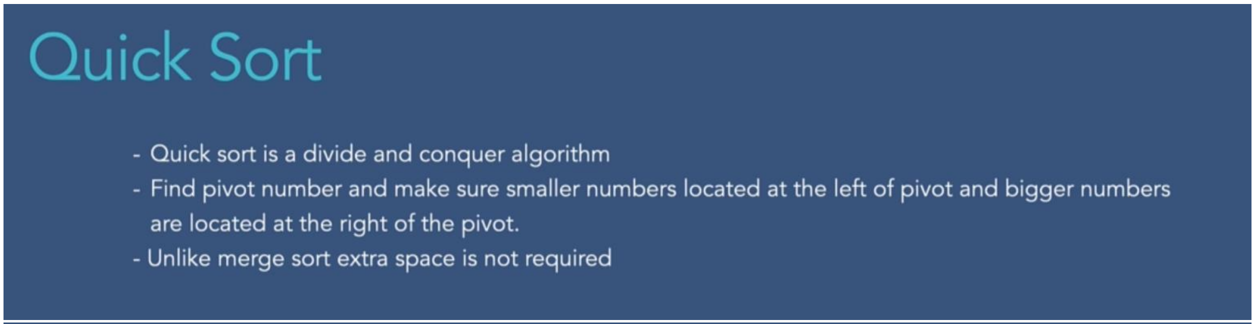
Space complexity: O(N)

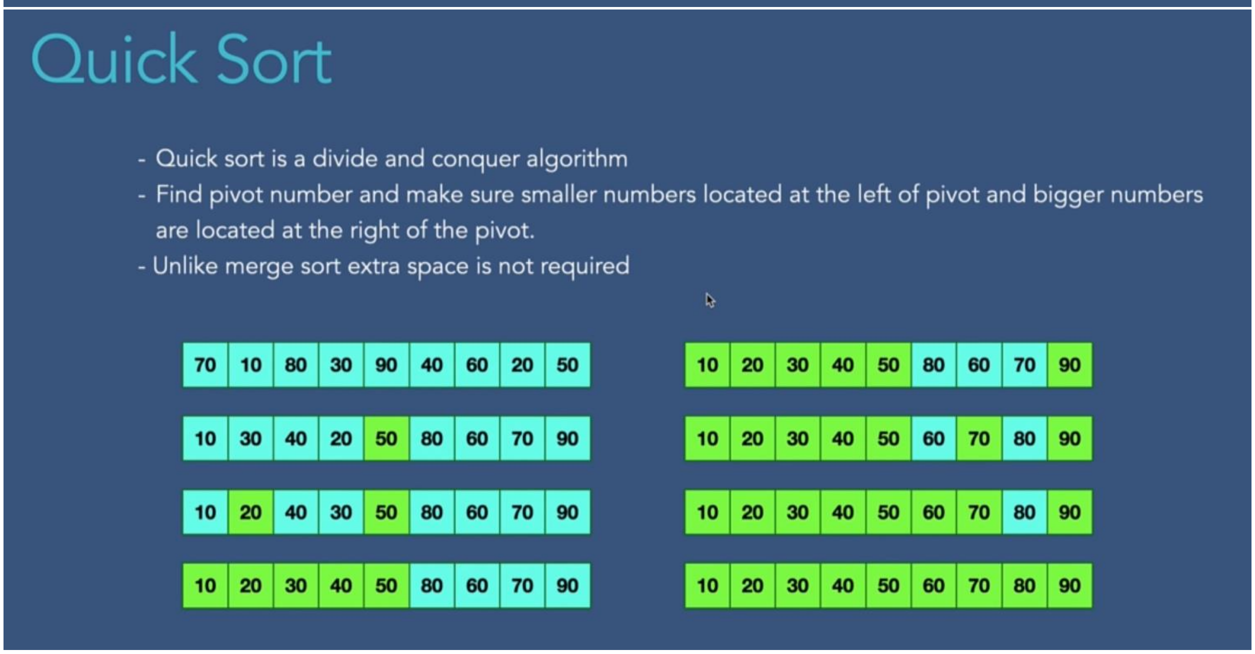
Animation of Merge Sort:

https://en.wikipedia.org/wiki/Merge_sort#/media/File:Merge-sort-example-300px.gif

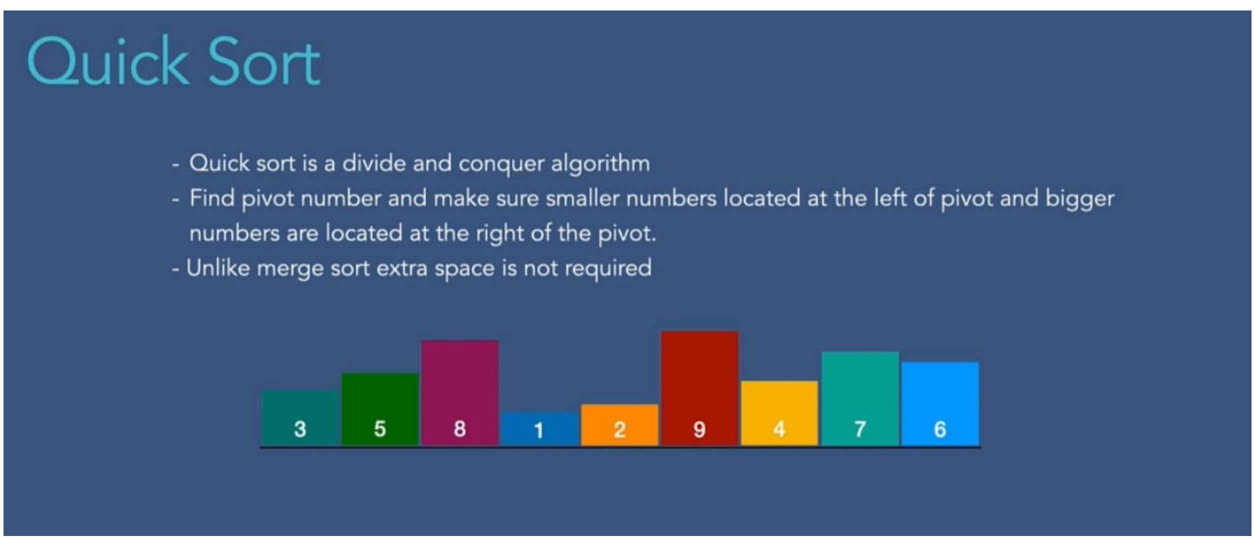
Visualization link of the code:

https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+MergeSort+%7B%0A%0A++static+void+merge(int%5B%5D+A,+int+left,+int+middle,+int+right)+%7B%0A++++int%5B%5D+leftTmpArray+%3D+new+int%5Bmiddle-left%2B2%5D%3B%0A++++int%5B%5D+rightTmpArray+%3D+new+int%5Bright-middle%2B1%5D%3B%0A++++//System.out.println(%22leftarr%22)%3B%0A++++for+(int+i%3D0%3B+i%3C%3Dmiddle-left%3B+i%2B%2B)+%7B%0A++++++leftTmpArray%5Bi%5D+%3D+A%5Bleft%2Bi%5D%3B%0A++++++//System.out.print(leftTmpArray%5Bi%5D%2B%22+%22)%3B%0A++++%7D%0A++++//System.out.println(%22rightarr%22)%3B%0A++++for+(int+i%3D0%3B+i%3Cright-middle%3B+i%2B%2B)+%7B%0A++++++rightTmpArray%5Bi%5D+%3D+A%5Bmiddle%2B1%2Bi%5D%3B%0A++++++//System.out.print(rightTmpArray%5Bi%5D%2B%22+%22)%3B%0A++++%7D%0A%0A%0A++++leftTmpArray%5Bmiddle-left%2B1%5D+%3D+Integer.MAX_VALUE%3B%0A++++rightTmpArray%5Bright-middle%5D+%3D+Integer.MAX_VALUE%3B%0A%0A++++int+i+%3D+0,+j+%3D+0%3B%0A++++for+(int+k+%3D+left%3B+k%3C%3Dright%3B+k%2B%2B)+%7B%0A++++++if+(leftTmpArray%5Bi%5D+%3C+rightTmpArray%5Bj%5D)+%7B%0A++++++++A%5Bk%5D+%3D+leftTmpArray%5Bi%5D%3B%0A++++++++i%2B%2B%3B%0A++++++%7D+else+%7B%0A++++++++A%5Bk%5D+%3D+rightTmpArray%5Bj%5D%3B%0A++++++++j%2B%2B%3B%0A++++++%7D%0A%0A++++%7D%0A%0A++%7D%0A%0A++public+void+mergeSort(int%5B%5D+Array,+int+left,+int+right)+%7B%0A++++if+(right+%3E+left)+%7B%0A++++++int+m+%3D+(left%2Bright)/2%3B%0A++++++mergeSort(Array,+left,+m)%3B%0A++++++mergeSort(Array,+m%2B1,+right)%3B%0A++++++merge(Array,+left,+m,+right)%3B%0A++++%7D%0A++%7D%0A%0A%0A%0A%09public+void+printArray(int+%5B%5Darray)+%7B%0A%09%09for+(int+i+%3D+0%3B+i+%3C+array.length%3B+i%2B%2B)+%7B%0A%09%09%09System.out.print(array%5Bi%5D%2B%22++%22)%3B%0A%09%09%7D%0A%09%7D%0A%0A++public+static+void+main(String%5B%5D+args)+%7B%0A++++int+array%5B%5D+%3D+%7B10,3,2,5%7D%3B%0A++++MergeSort+ms+%3D+new+MergeSort()%3B%0A++++ms.mergeSort(array,+0,+array.length-1)%3B%0A++++ms.printArray(array)%3B%0A++%7D%0A%7D%0A&mode=display&curInstr=0

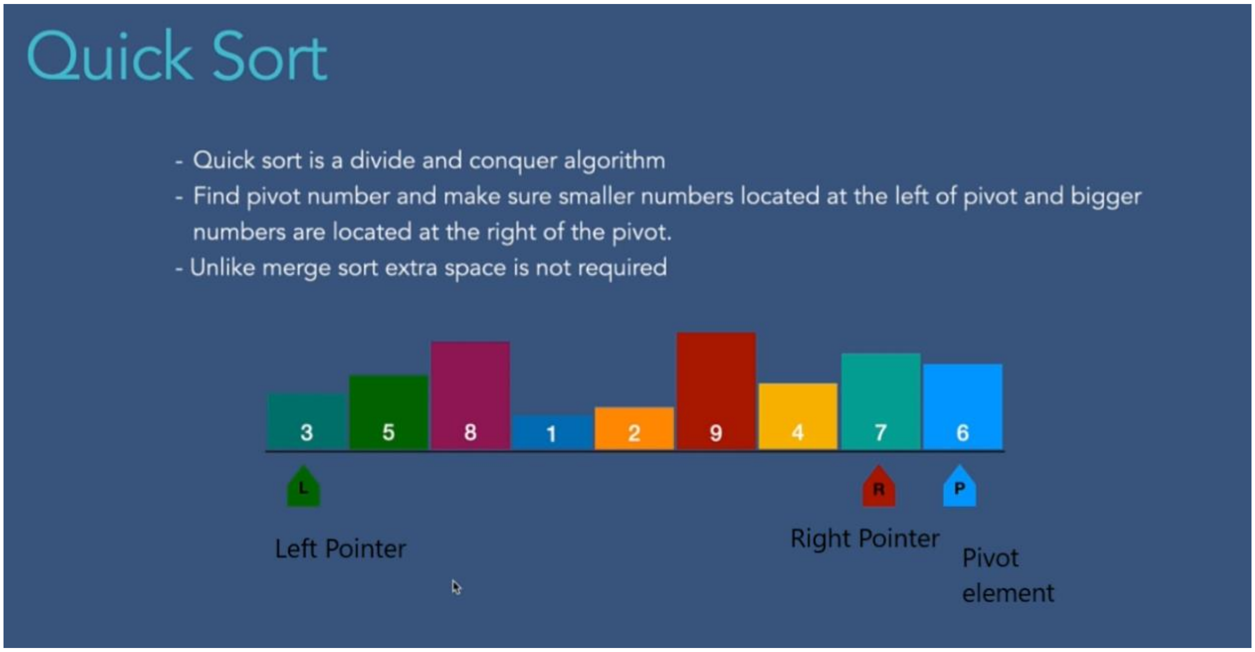


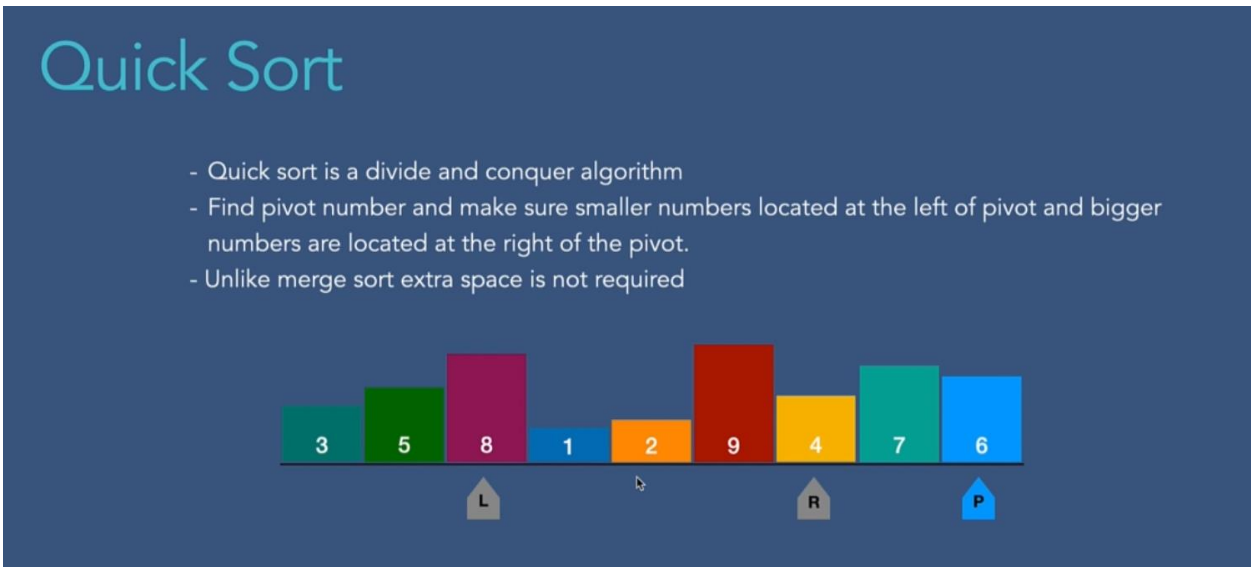


Taking the right most element as the pivot element. The single green colored elements are the pivot elements and the continuous green coloured elements indicates that the array is sorted (partially or completely).

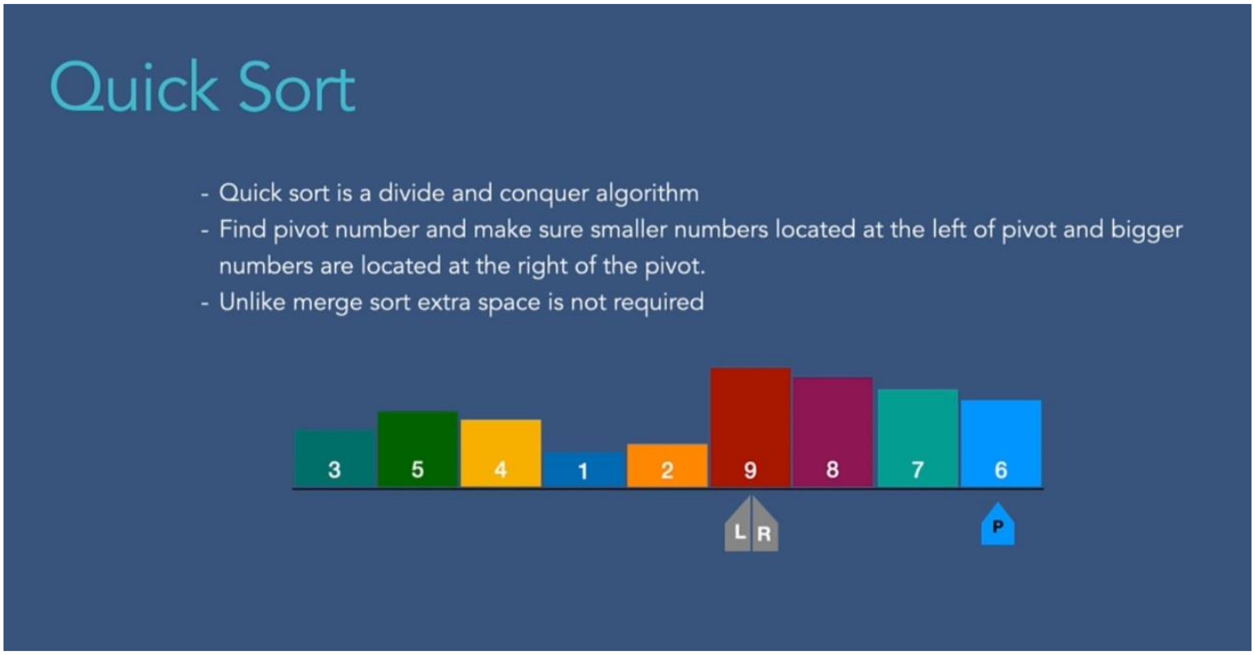


Initial state of the array

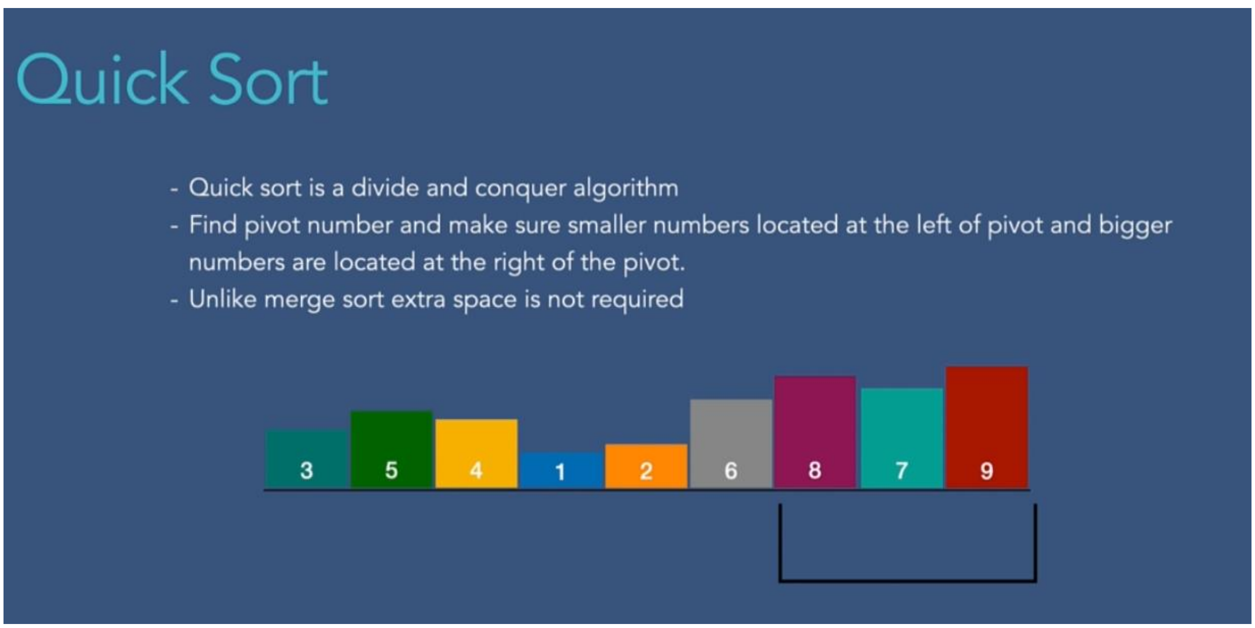




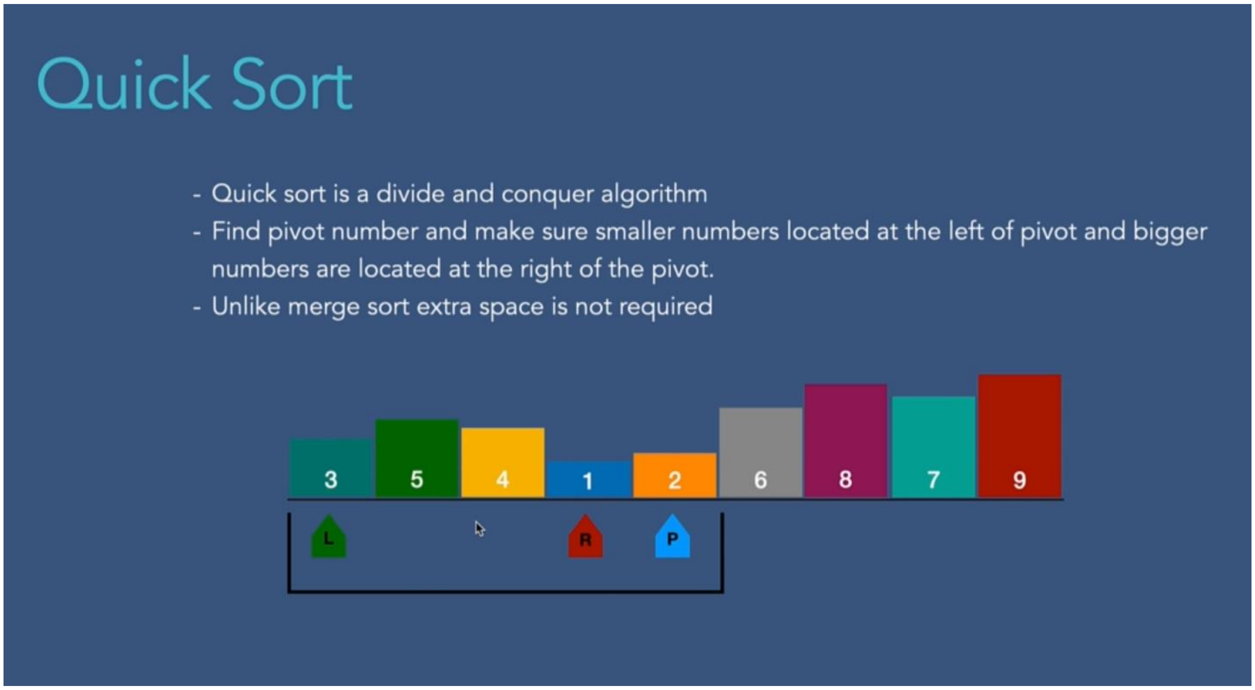
Pointers stop up here. The left pointer moves until it find the element greater than the pivot element and the right pointer moves until it find the element less than the pivot element.Then, the right and the left pointer get swapped.



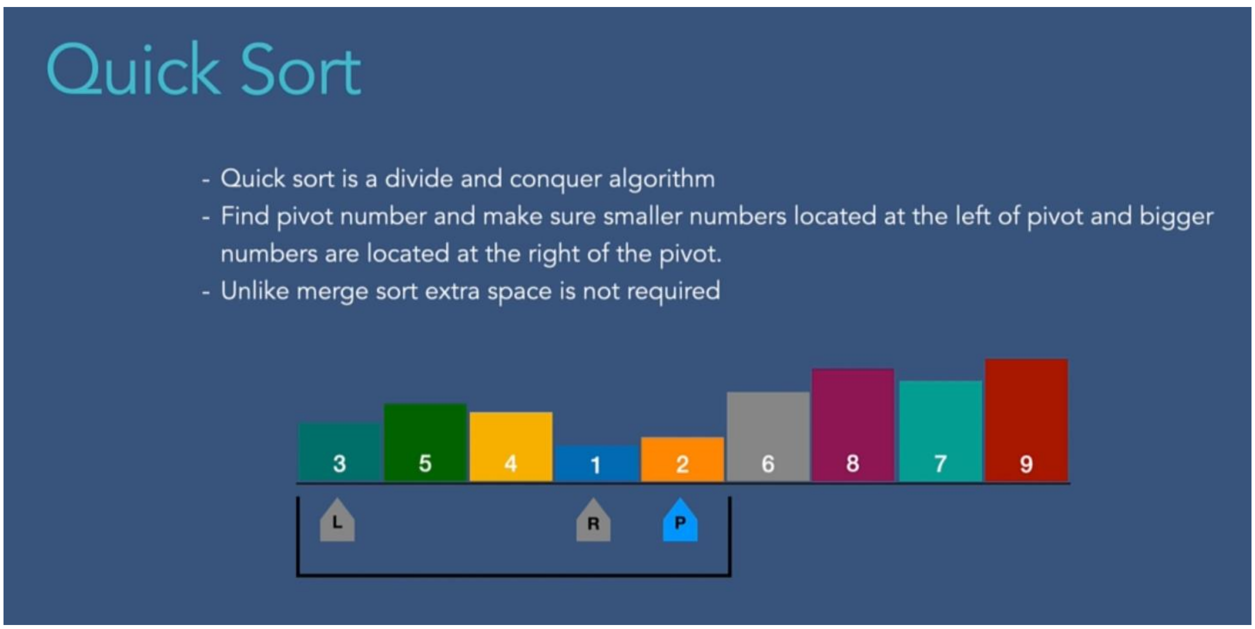
We stop when the right and the left pointer meets up(9→ pointers meet here). Then, we swap the meeting element (9) of the left and the right pointer to the pivot element (6). So, the meeting point of the left and the right pointer is considered to be fully sorted.



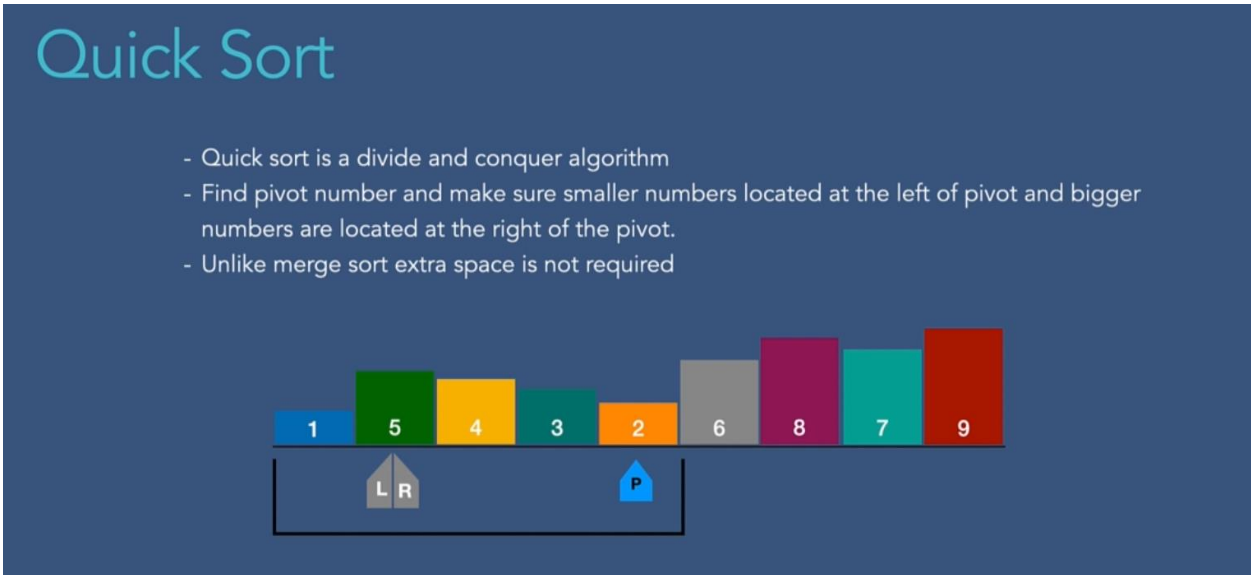
And now we can observe that the left of the sorted element (6) is smaller than the sorted element (6) and to the right are the larger element than the sorted element (6).



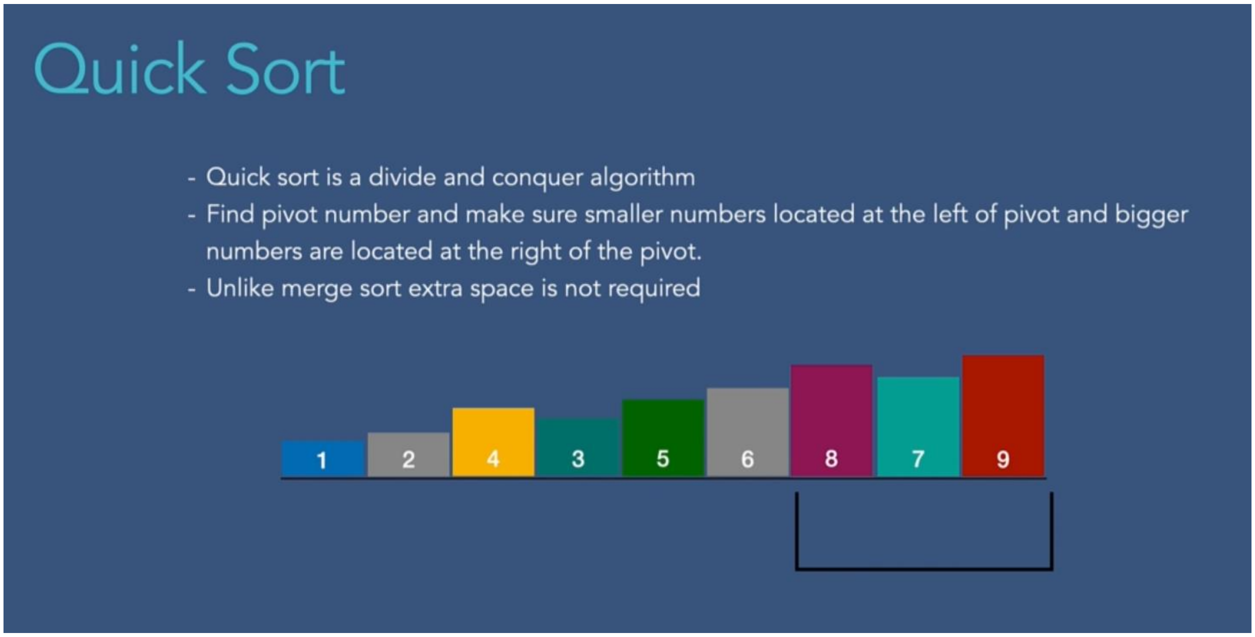
We then continue the same process with the left part of the sorted element (6). Refer the above image.



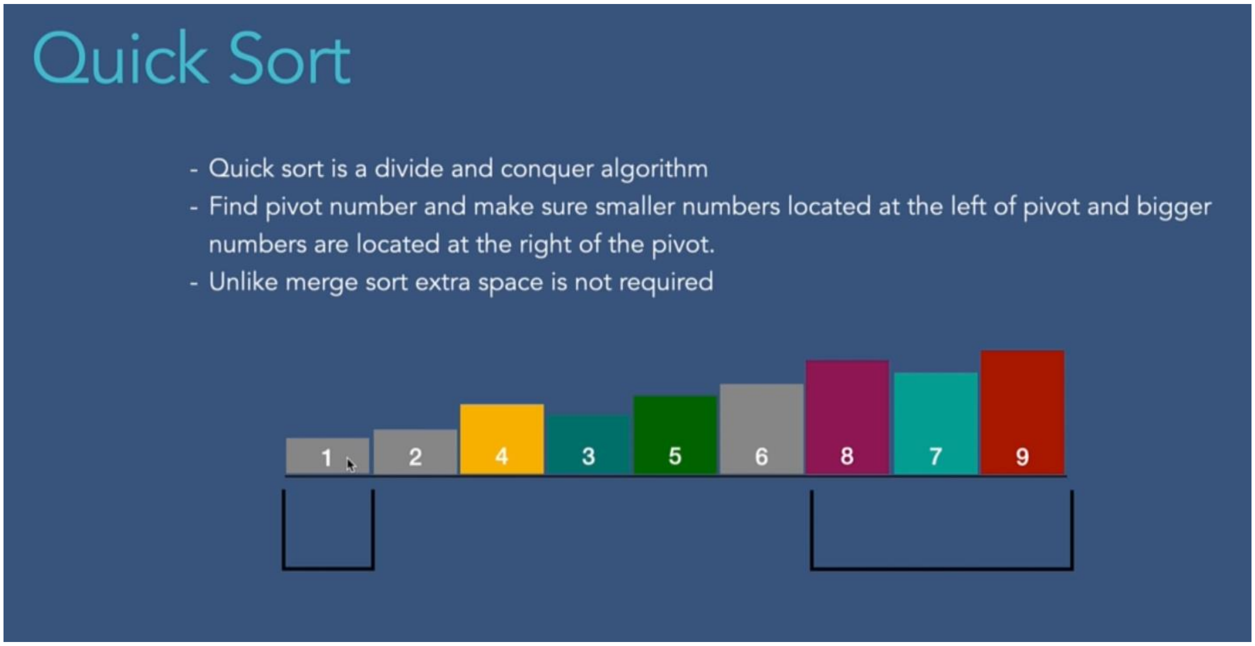
Swapping the element 1 and 3.



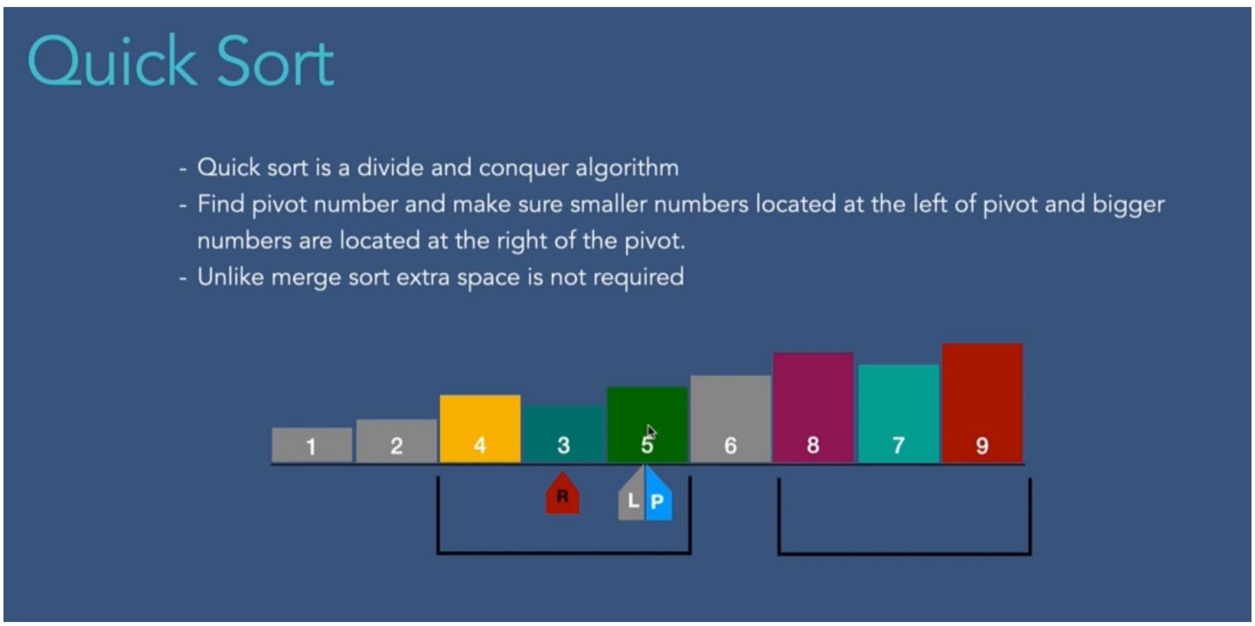
Now swap the pivot number(2) with the meeting element of the left pointer and the right pointer.



Now the element 2 becomes fully sorted.

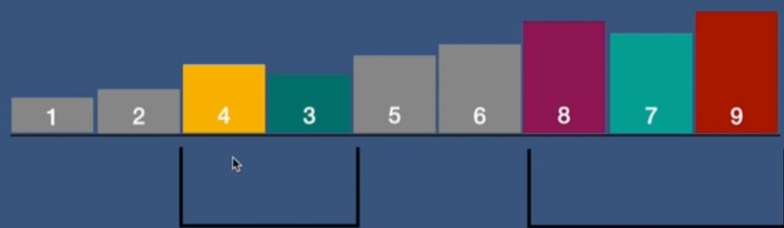


To the left of the element 2 we have only one element so now it becomes fully sorted.



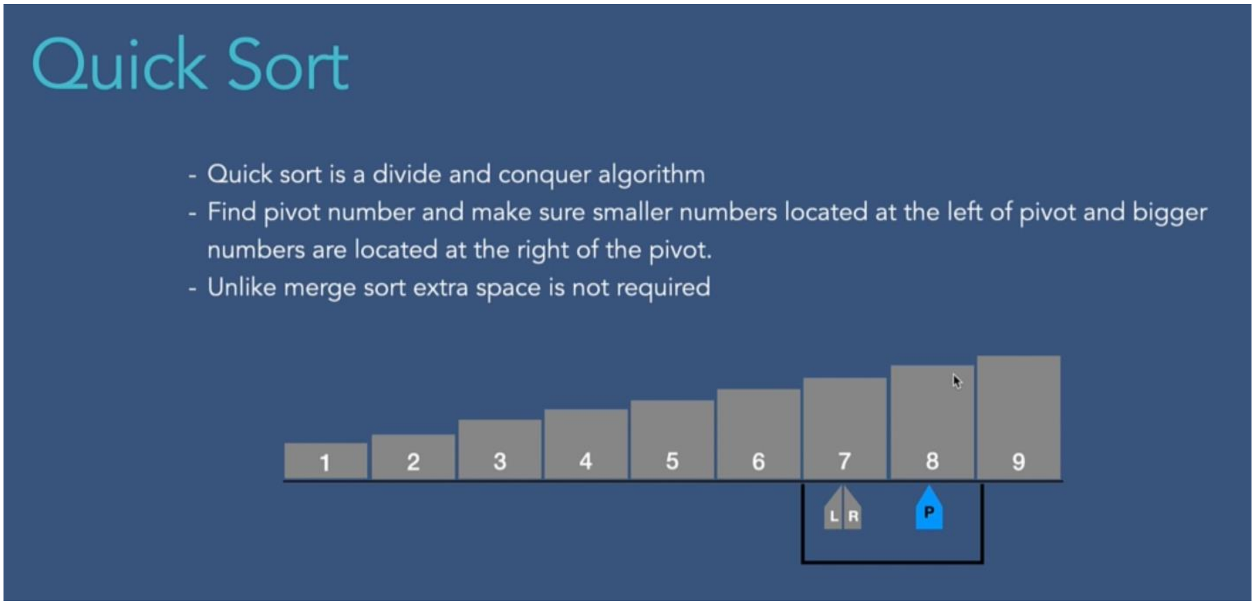
Quick Sort

- Quick sort is a divide and conquer algorithm
- Find pivot number and make sure smaller numbers located at the left of pivot and bigger numbers are located at the right of the pivot.
- Unlike merge sort extra space is not required



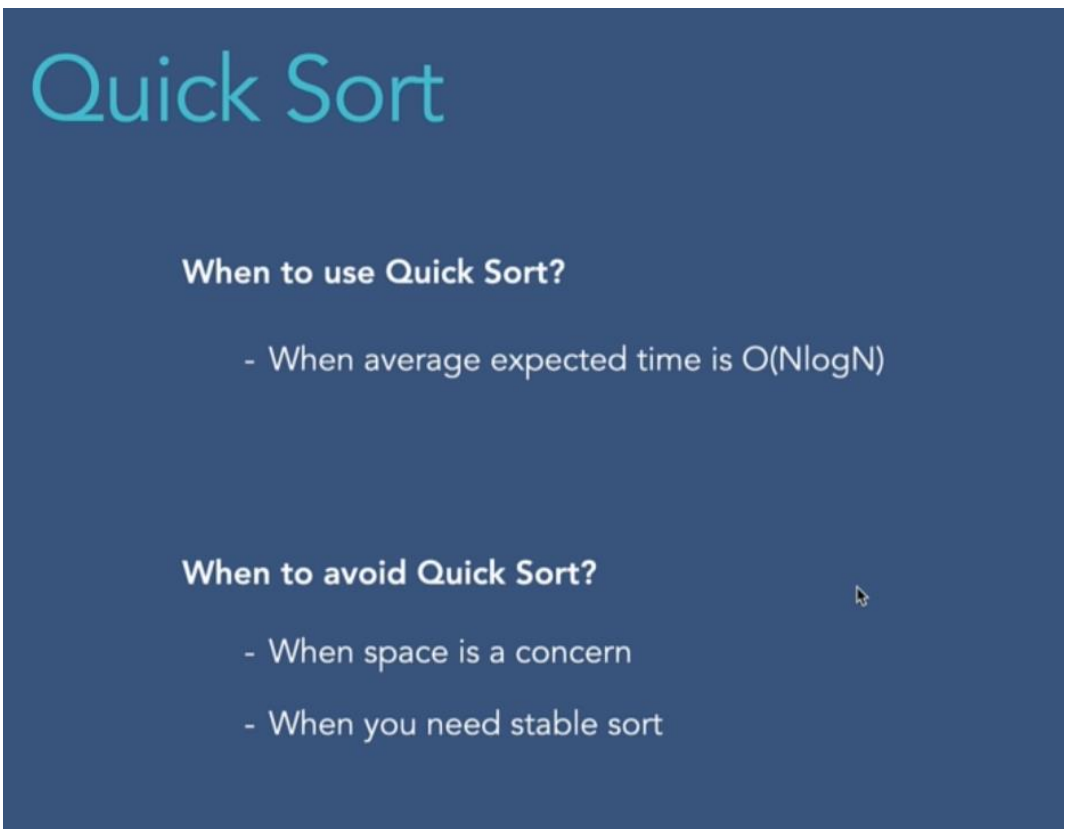
Quick Sort

- Quick sort is a divide and conquer algorithm
- Find pivot number and make sure smaller numbers located at the left of pivot and bigger numbers are located at the right of the pivot.
- Unlike merge sort extra space is not required



Now the array is fully sorted

Quick sort is the most effective sorting algorithm and it is the best even when the size of the array is large.



Animation for Quick Sort:

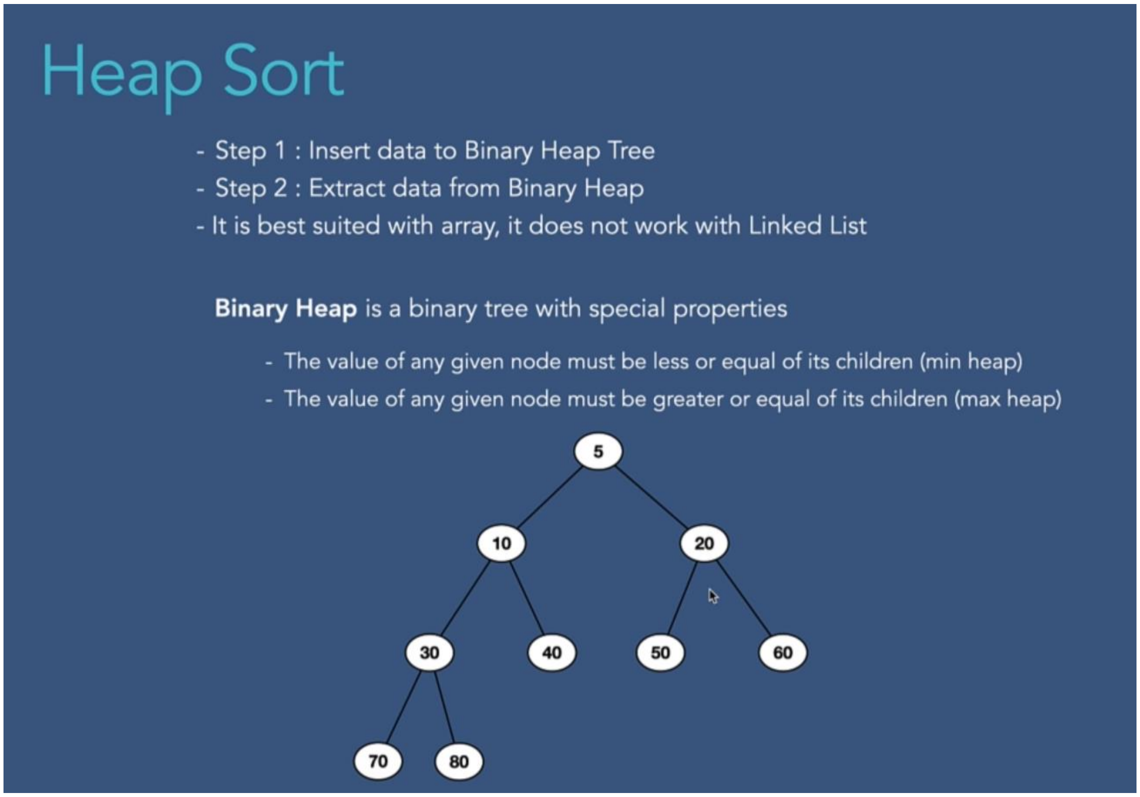
https://en.wikipedia.org/wiki/Quicksort#/media/File:Quicksort-example.gif

Visualization of the code:

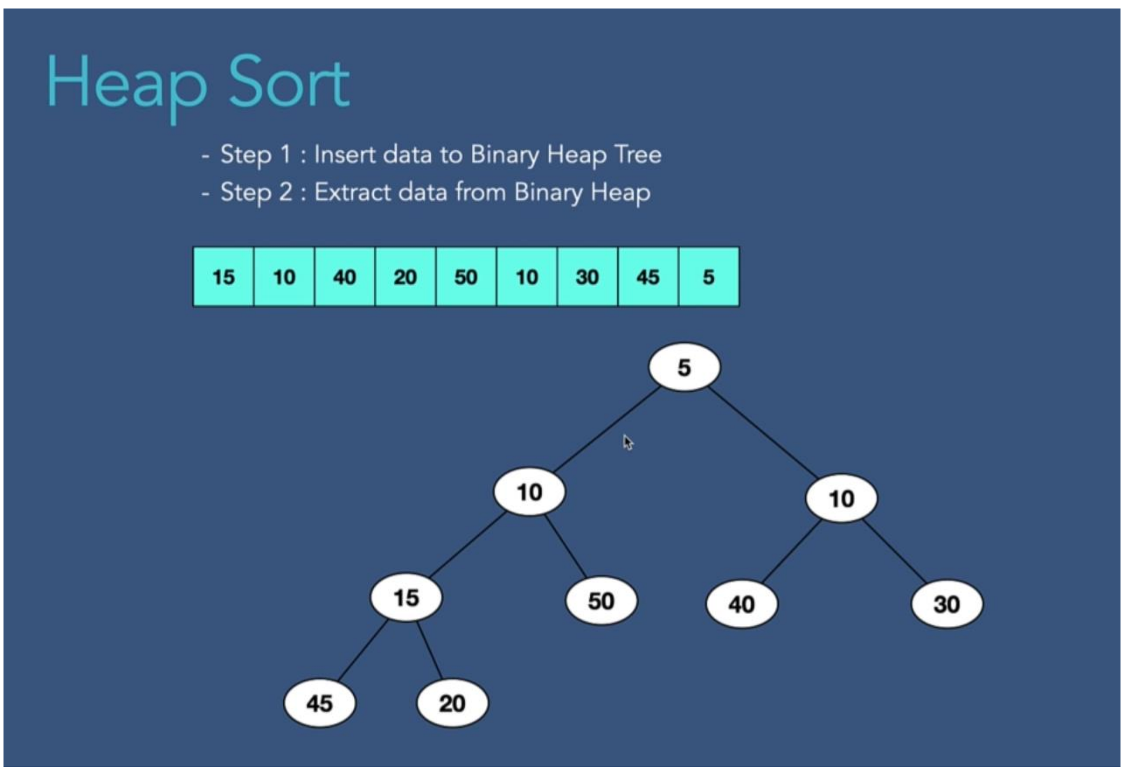
https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+QuickSort+%7B%0A%0A++static
+int+partition(int%5B%5D+array,+int+start,+int+end)+%7B%0A++++int+pivot+%3D+end%3B%0A++++int
+i+%3D+start+-
+1%3B%0A++++for+(int+j%3D+start%3B+j%3C%3Dend%3B+j%2B%2B)+%7B+%0A++++++if+(array%5Bj%
5D+%3C%3D+array%5Bpivot%5D)+%7B%0A++++++++i%2B%2B%3B%0A++++++++int+temp+%3D+array
%5Bi%5D%3B%0A++++++++array%5Bi%5D+%3D+array%5Bj%5D%3B%0A++++++++array%5Bj%5D+%3D+
temp%3B%0A++++++%7D%0A++++%7D%0A++++return+i%3B%0A++%7D%0A%0A++public+static+void+
quickSort(int%5B%5D+array,+int+start,+int+end)+%7B%0A++++if+(start+%3C+end)+%7B%0A++++++int+
pivot+%3D+partition(array,+start,+end)%3B%0A++++++quickSort(array,+start,+pivot+-
1)%3B%0A++++++quickSort(array,+pivot+%2B+1,+end)%3B%0A++++%7D%0A++%7D%0A%0A%09public
+static+void+printArray(int+%5B%5Darray)+%7B%0A%09%09for+(int+i+%3D+0%3B+i+%3C+array.length
%3B+i%2B%2B)+%7B%0A%09%09%09System.out.print(array%5Bi%5D%2B%22++%22)%3B%0A%09%09
%7D%0A%09%7D%0A++public+static+void+main(String%5B%5D+args)+%7B%0A++++int+array%5B%5D+
%3D+%7B10,3,2,7,8%7D%3B%0A++++quickSort(array,+0,+array.length-
1)%3B%0A++++printArray(array)%3B%0A++%7D%0A%7D%0A&mode=display&curInstr=0

Time complexity: O(NLogN)

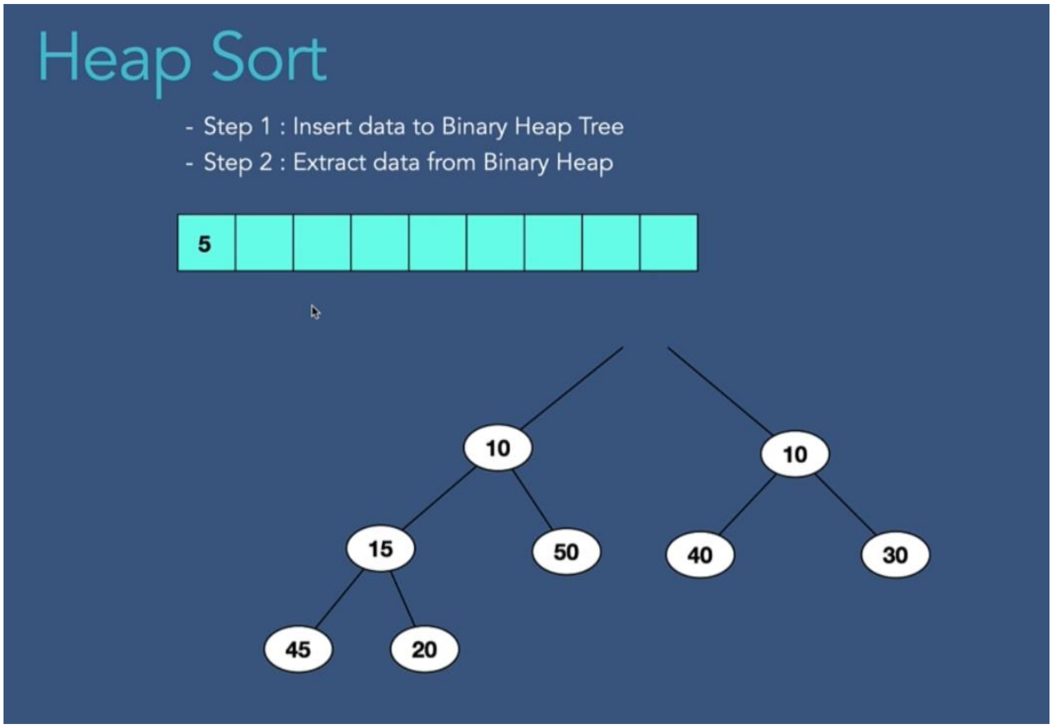
Space complexity: O(N)



The Binary Heap Tree is an example of Min Heap.

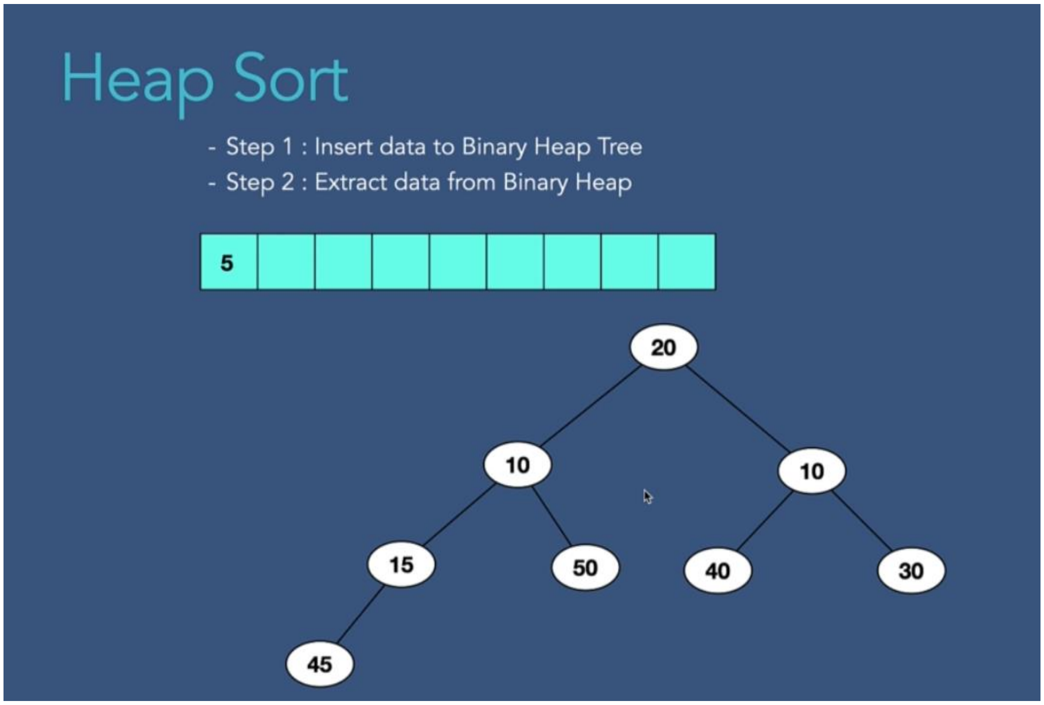


Step 1: Insertion of the data to the Binary Heap Tree.

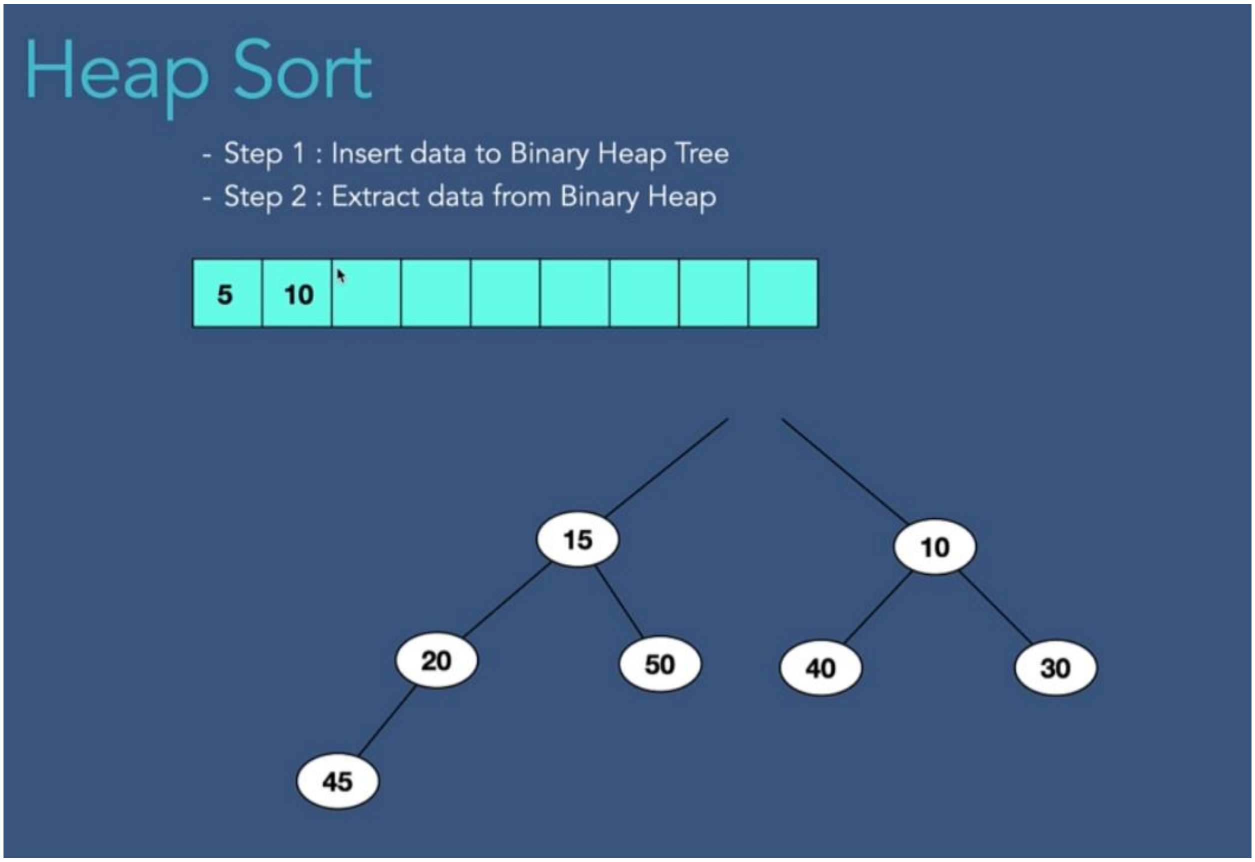


Step 2: Extracting data from the Binary Heap Tree

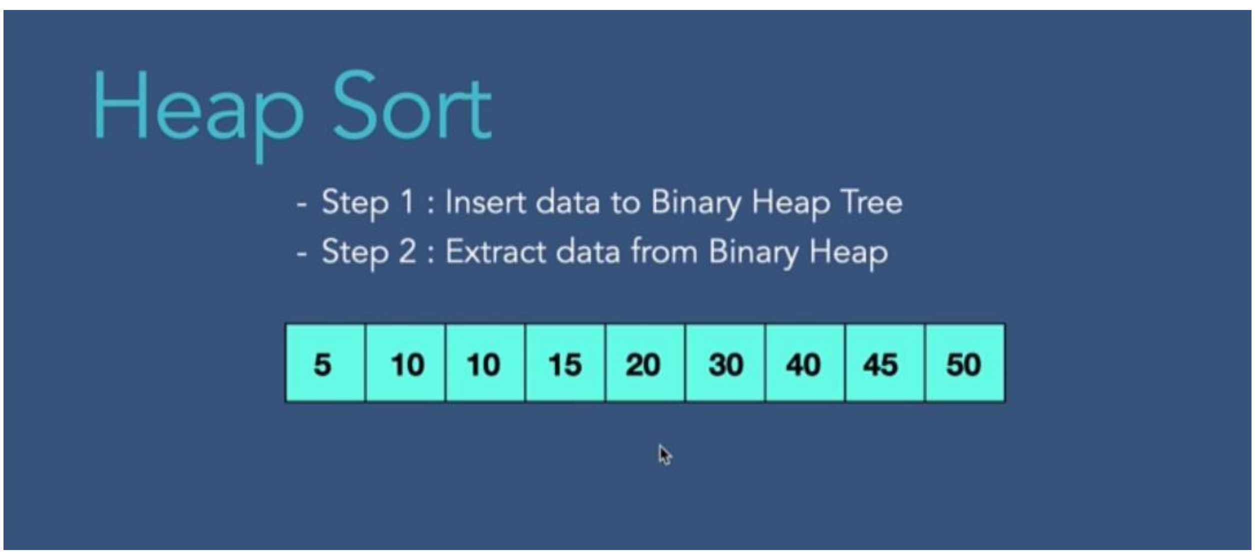
First, we insert the root element of the Binary Heap Tree to the array and the try to insert the left most element of the Binary Heap tree to the array



The left most element of the Binary Heap gets to the root node and then we check for the Binary Heap property. If the Binary Heap property is not satisfied then we call Heapify method.



After Heapifying the Binary Heap Tree, we then extract the root node element and then put that element to the array. This process is continued until the Binary Heap becomes null.



After the complete extraction process from the Binary Heap Tree.

Time complexity: O(NLogN)

Space complexity: O(1)

Sorting Algorithms

Name	Time Complexity	Space Complexity	Stable
Bubble Sort	$O(n^2)$	O(1)	Yes
Selection Sort	$O(n^2)$	O(1)	No
Insertion Sort	$O(n^2)$	O(1)	Yes
Bucket Sort	O(n logn)	O(n)	Yes
Merge Sort	O(n logn)	O(n)	Yes
Quick Sort	O(n logn)	O(n)	No
Heap Sort	O(n logn)	O(1)	No