DATA STRUCTURES (ITPC-203)

Sorting Techniques – Part II

Mrs. Sanga G. Chaki

Department of Information Technology

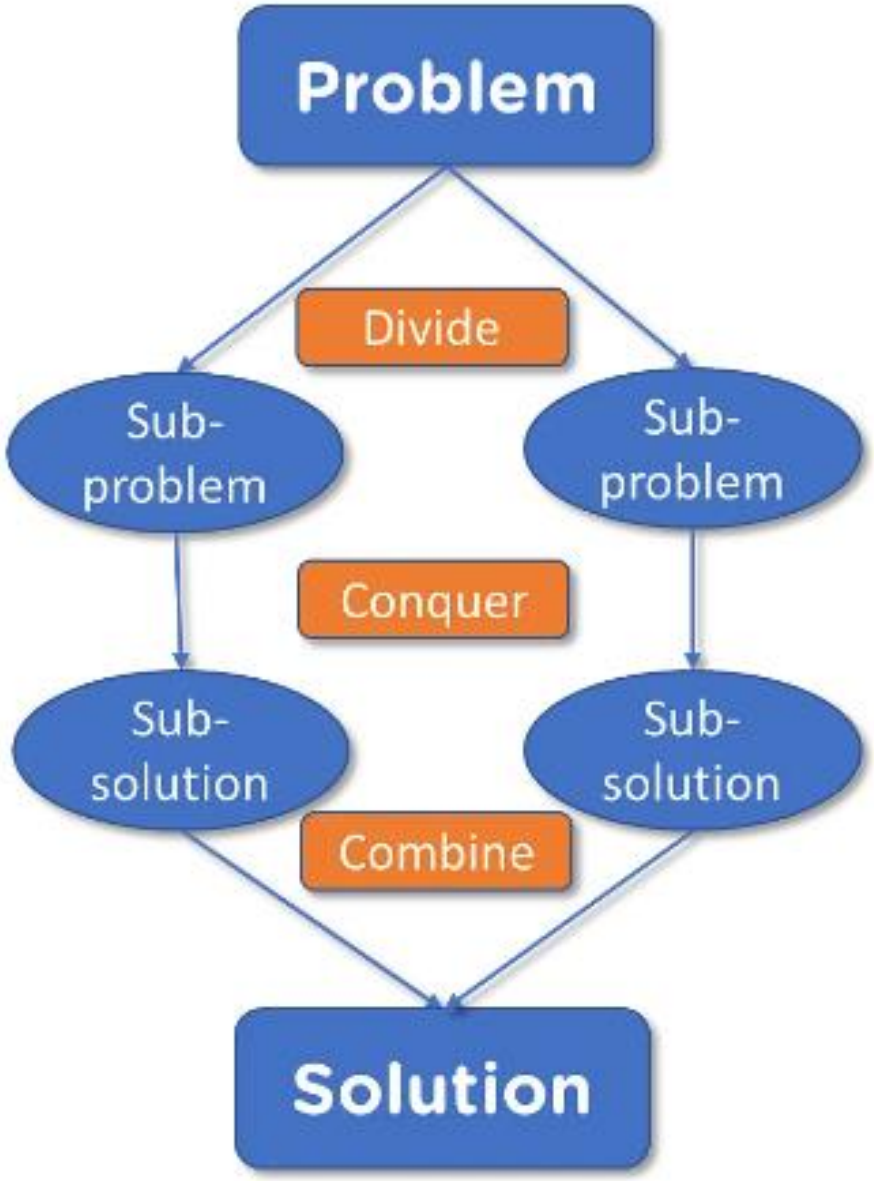
Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Contents

1. Two way Merge Sort
2. Counting Sort
3. Radix Sort
4. Heap Sort – To be done after introduction to graphs and trees

Divide and Conquer Algorithms

1. Divide and conquer is an algorithm design paradigm which
   - recursively breaks down a problem into two or more sub-problems of the same or related type,
   - until these become simple enough to be solved directly.
   - The solutions to the sub-problems are then combined to give a solution to the original problem.
2. Divide-and-conquer recursively solves subproblems;
   - each subproblem must be smaller than the original problem, and
   - each must have a base case.



Divide and Conquer Algorithms

1. This technique can be divided into the following three parts:
   - **Divide**: This involves dividing the problem into smaller sub-problems.
   - **Conquer**: Solve sub-problems recursively until solved.
   - **Combine**: Combine the sub-problems to get the final solution of the whole problem.
2. Example: Merge sort, quick sort, Strassen's Algorithm to multiply two matrices etc

Merge Sort – General Idea

1. Suppose we have to sort an array A.
2. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p….r].
3. **Divide Step**:
   - If q is the half-way point between p and r, then we can split the subarray A[p...r] into two arrays A[p...q] and A[q+1 … r].

Merge Sort – General Idea

1. **Conquer Step**:
   - In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1.. r].
   - If we haven't yet reached the base case (single element in each array), we again divide both these subarrays and try to sort them.
2. **Combine Step**:
   - When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1..r] for array A[p..r],
   - we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1..r].

Merge Sort - Example

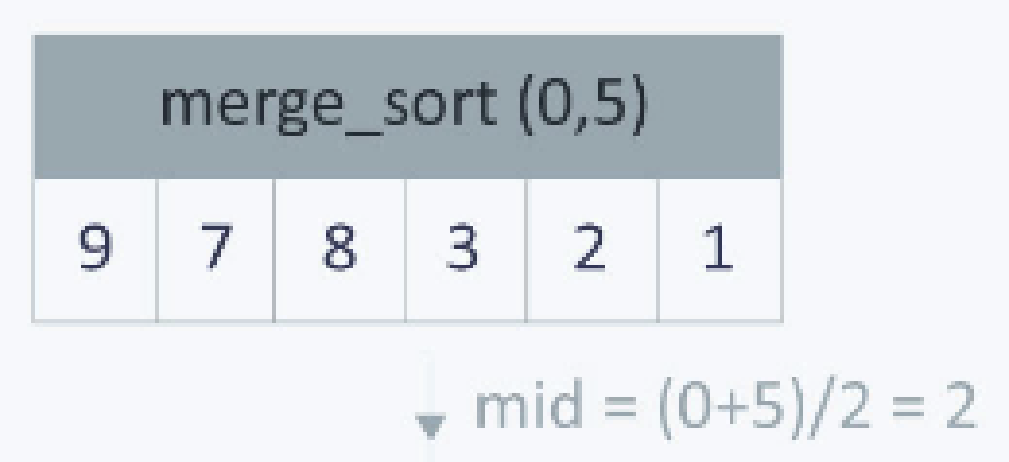
1. Divide the unsorted list into N sublists, each containing 1 element.
2. Take adjacent pairs of two singleton lists and merge them to form a sorted list of 2 elements. N will now convert into N/2 lists of size 2.
3. Repeat the process till a single sorted list of obtained.

Merge Sort - Example

1. How to merge the sublists?
   - ➢ While comparing two sublists for merging, the first element of both lists is taken into consideration.
   - ➢ While sorting in ascending order, the element that is of a lesser value becomes a new element of the sorted list.
   - ➢ This procedure is repeated until both the smaller sublists are empty,
   - ➢ and the new combined sublist comprises all the elements of both the sublists.

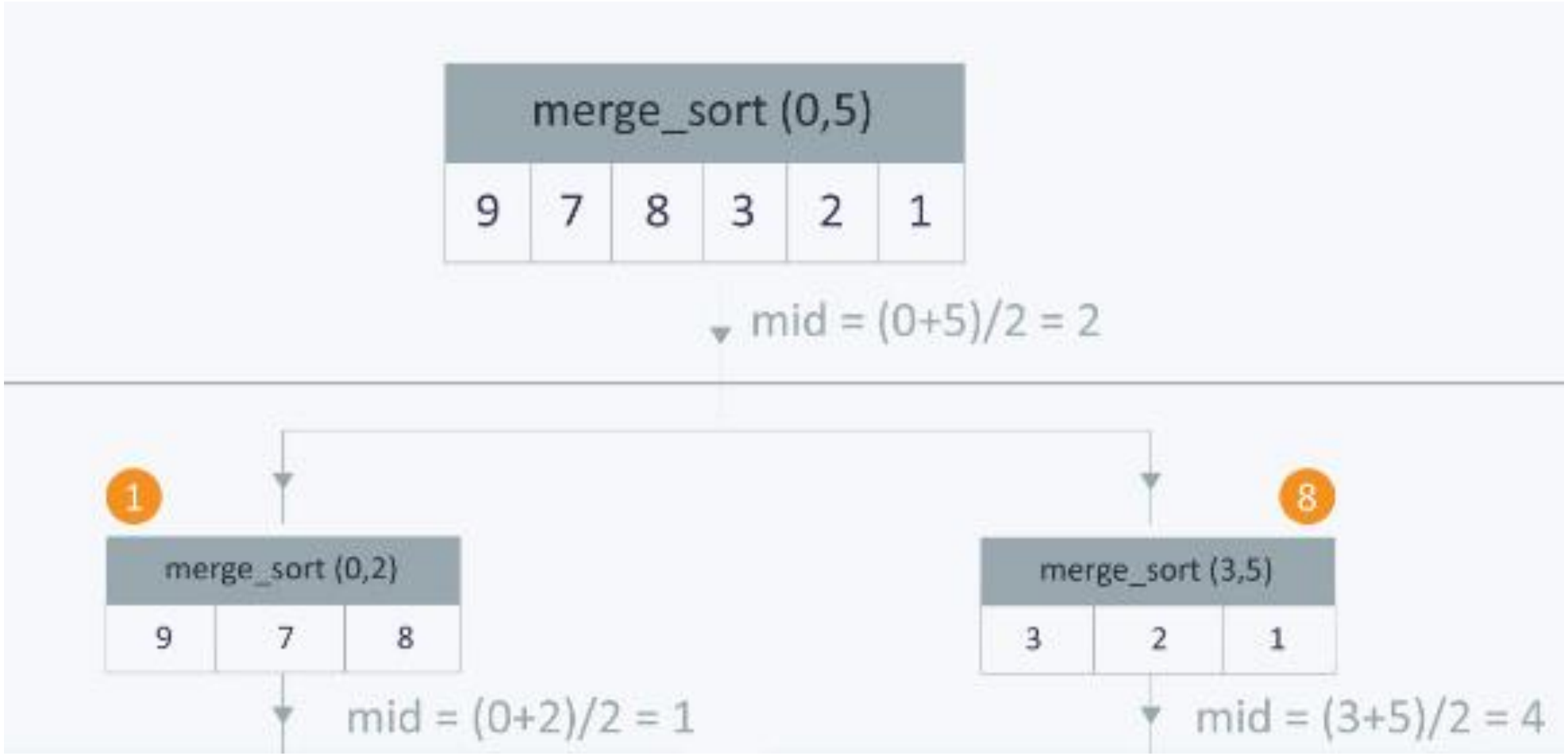
Merge Sort - Example

1. Start with the given list



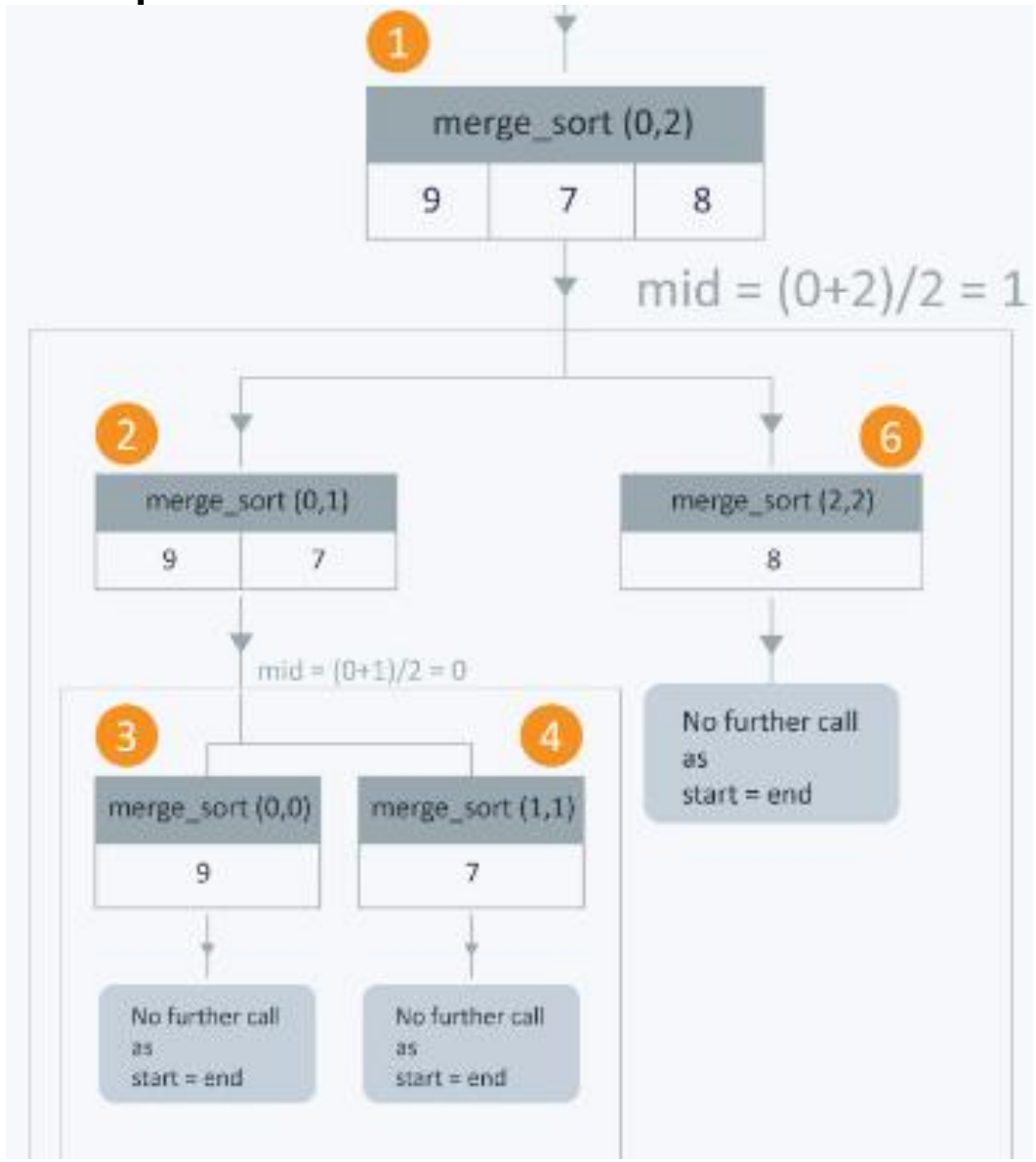
Merge Sort - Example

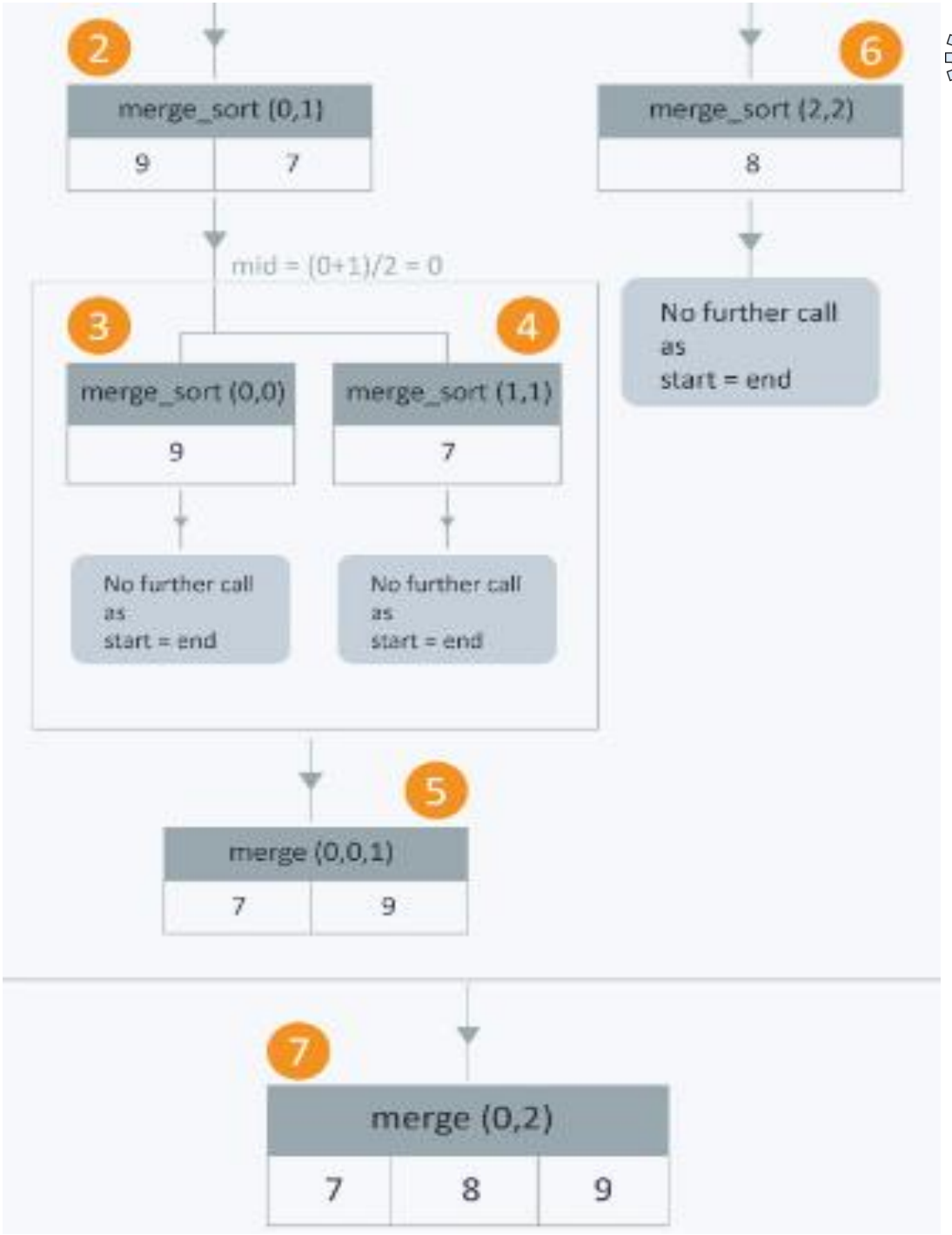
1. Divide:



Merge Sort - Example

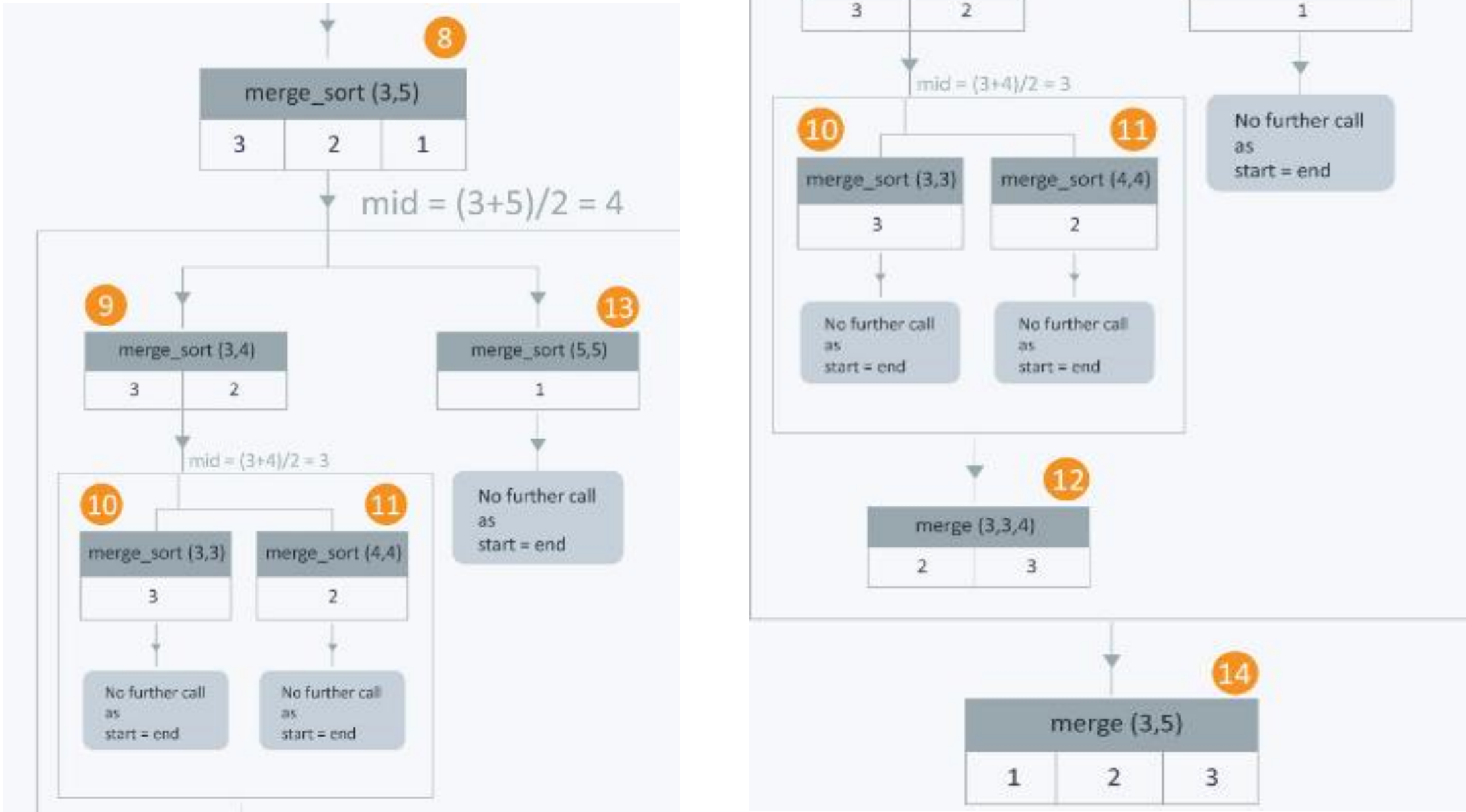
1. Conquer:





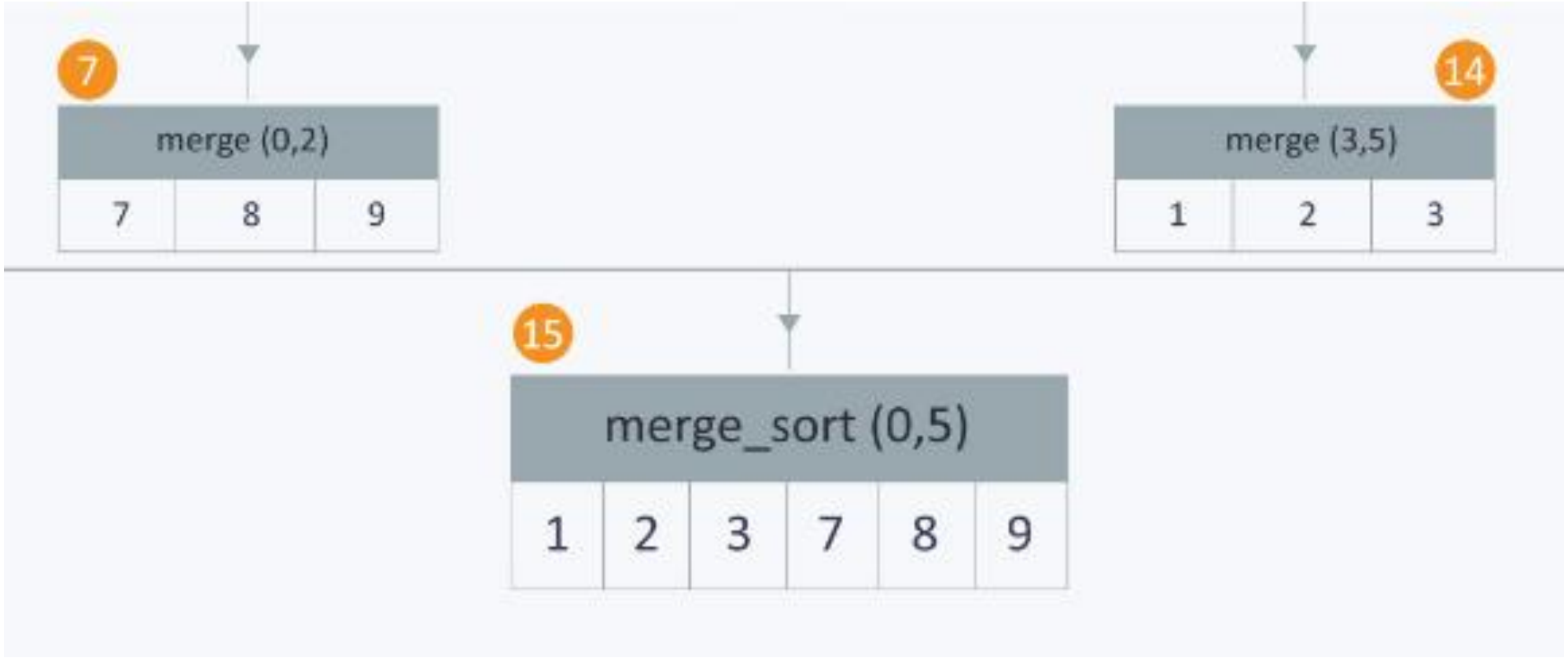
Merge Sort - Example

1. Conquer:



Merge Sort - Example

1. Combine:



Merge Sort - Example

1. At each step a list of size M is being divided into 2 sublists of size M/2, until no further division can be done.
2. To understand better, consider a smaller array A containing the elements (9,7,8).
3. At the first step this list of size 3 is divided into 2 sublists, the first consisting of elements (9,7) and the second one being (8).
4. Now, the first list consisting of elements (9, 7) is further divided into 2 sublists consisting of elements (9) and (7) respectively.

Merge Sort - Example

1. As no further breakdown of this list can be done, as each sublist consists of a maximum of 1 element, we now start to merge these lists.
2. The 2 sub-lists formed in the last step are then merged together in sorted order using the procedure mentioned above leading to a new list (7, 9).
3. Backtracking further, we then need to merge the list consisting of element (8) too with this list, leading to the new sorted list (7, 8, 9).

Merge Sort - Code

```
    void merge_sort (int A[ ] , int start , int end )
    {
            if( start < end ) {
            int mid = (start + end ) / 2 ;                  // defines the current
array in 2 parts .
            merge_sort (A, start , mid ) ;                      // sort the 1st part
of array .
            merge_sort (A,mid+1 , end ) ;                    // sort the 2nd part of
array.

          // merge the both parts by comparing elements of both the parts.
           merge(A,start , mid , end );
    }
}
```

Merge Sort - Code

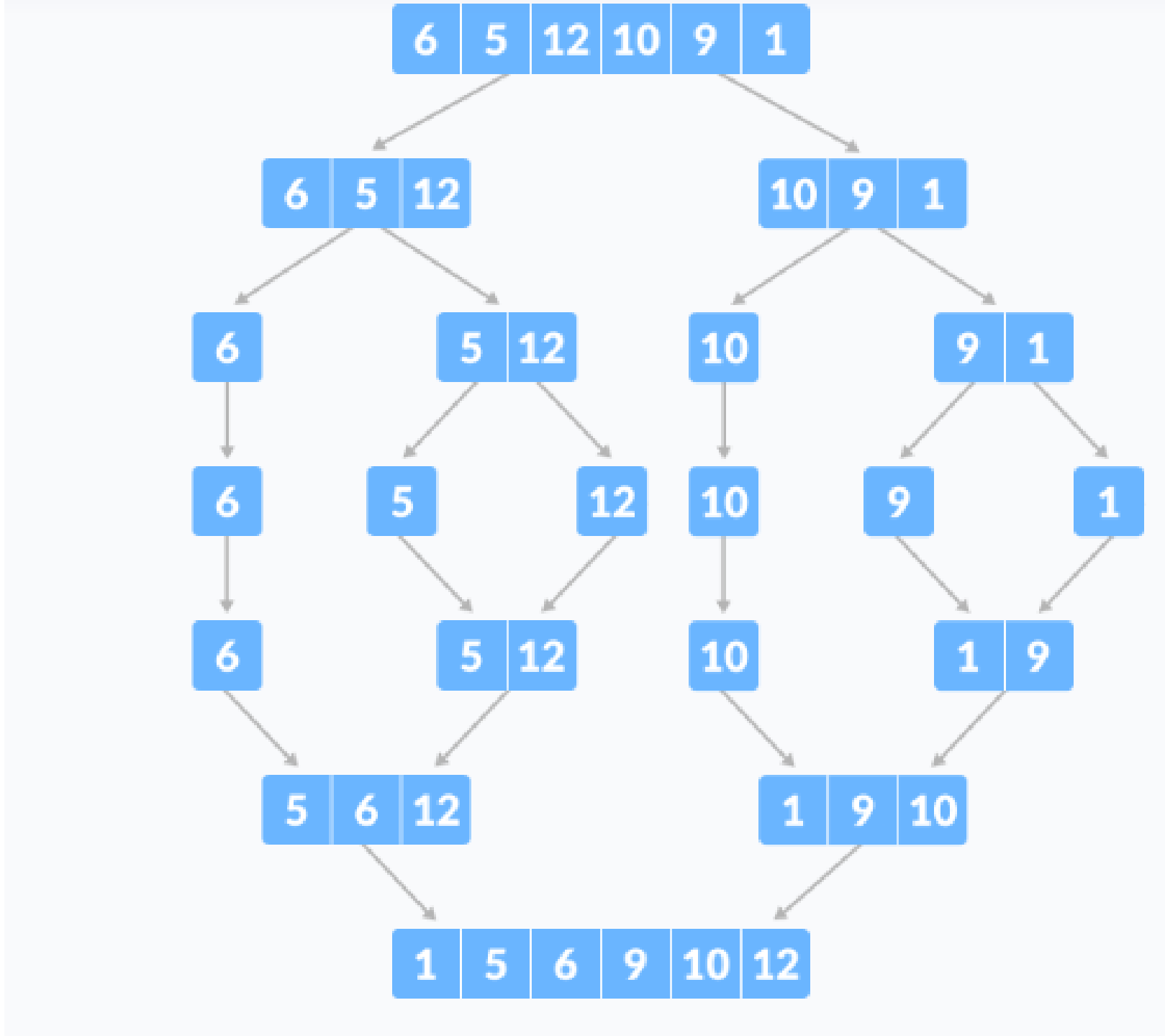
```
 void merge(int A[ ] , int start, int mid, int end) {
 //stores the starting position of both parts in temporary variables.
int p = start ,q = mid+1;

int Arr[end-start+1] , k=0;

for(int i = start ;i <= end ;i++) {
    if(p > mid)       //checks if first part comes to an end or not .
       Arr[ k++ ] = A[ q++] ;

   else if ( q > end)   //checks if second part comes to an end or not
       Arr[ k++ ] = A[ p++ ];

   else if( A[ p ] < A[ q ])     //checks which part has smaller element.
      Arr[ k++ ] = A[ p++ ];

   else
      Arr[ k++ ] = A[ q++];
 }
  for (int p=0 ; p< k ;p ++) {
   /* Now the real array has elements in sorted manner including both
         parts.*/
     A[ start++ ] = Arr[ p ] ;
  }
}
```

Merge Sort - Complexity

Time Complexity:

The list of size N is divided into a max of $logN$ parts, and the merging of all sublists into a single list takes $O(N)$ time, the worst case run time of this algorithm is $O(NLogN)$

Merge Sort – Example



Counting Sort

1. Prerequisite for radix sort
2. In Counting sort, the frequencies of distinct elements of the array to be sorted is counted and stored in an auxiliary array, by mapping its value as an index of the auxiliary array.
3. Example:

Say $A = \{5, 2, 9, 5, 2, 3, 5\}$.

Aux will be of the size $9 + 1$ i.e. 10

Aux = {0, 0, 2, 1, 0, 3, 0, 0, 0, 1}

Counting Sort

Notice that $Aux[2] = 2$ which represents the number of occurrences of 2 in $A[]$. Similarly $Aux[5] = 3$ which represents the number occurrences of 5 in $A[]$.

After applying the counting sort algorithm, $sorted A[]$ will be $\{2, 2, 3, 5, 5, 5, 9\}$

Time Complexity:

The array A is traversed in $O(N)$ time and the resulting sorted array is also computed in $O(N)$ time. $Aux[]$ is traversed in $O(K)$ time. Therefore, the overall time complexity of counting sort algorithm is $O(N + K)$.

Radix Sort

1. QuickSort, and MergeSort are comparison based sorting algorithms.
2. CountSort is not comparison based algorithm.
3. It has the complexity of O(n+k), where k is the maximum element of the input array.
4. So, if k is O(n), CountSort becomes linear sorting.
5. Which is better than comparison based sorting algorithms that have O(n log n) time complexity.
6. The idea is to extend the CountSort algorithm to get a better time complexity when k becomes $O(n^2)$.

Radix Sort

1. Algorithm:
2. For each digit **i** where **i** varies from the least significant digit to the most significant digit of a number
   - Sort input array using countsort() algorithm according to i^{th} digit.

3. We used count sort because it is a stable sort.
   - What is stable and unstable sorting algorithm? Give examples.

Radix Sort - Example

1. Example: Assume the input array is: 10, 21, 17, 34, 44, 11, 654, 123
2. Based on the algorithm, we will sort the input array according to the one's digit (least significant digit). 10, 21, 17, 34, 44, 11, 654, 123
3. The buckets are as follows:

0: 10
1: 21 11
2:
3: 123
4: 34 44 654
5:
6:
7: 17
8:
9:

So, the array becomes

10, 21, 11, 123, 24, 44, 654, 17

Radix Sort - Example

1. In the 2nd step, the input array is: 10, 21, 11, 123, 24, 44, 654, 17
2. Based on the algorithm, we will sort the input array according to the ten's digit. 10, 21, 11, 123, 24, 44, 654, 17
3. The buckets are as follows:

0:
1: 10 11 17
2: 21 123
3: 34
4: 44
5: 654
6:
7:
8:
9:

So, the array becomes

10, 11, 17, 21, 123, 34, 44, 654

Radix Sort - Example

1. In the 3rd step, the input array is: 10, 11, 17, 21, 123, 34, 44, 654
2. Based on the algorithm, we will sort the input array according to the hundred's digit (MSD). 010, 011, 017, 021, 123, 034, 044, 654
3. The buckets are as follows:

0: 010 011 017 021 034 044
1: 123
2:
3:
4:
5:
6: 654
7:
8:
9:

So, the array becomes
10, 11, 17, 21, 34, 44, 123, 654
which is sorted

Radix Sort

1. What is the complexity?


DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR

Thanks