Programming, Data Structures & Algorithms
Sorting

By

Samira Dayan Jayasekara

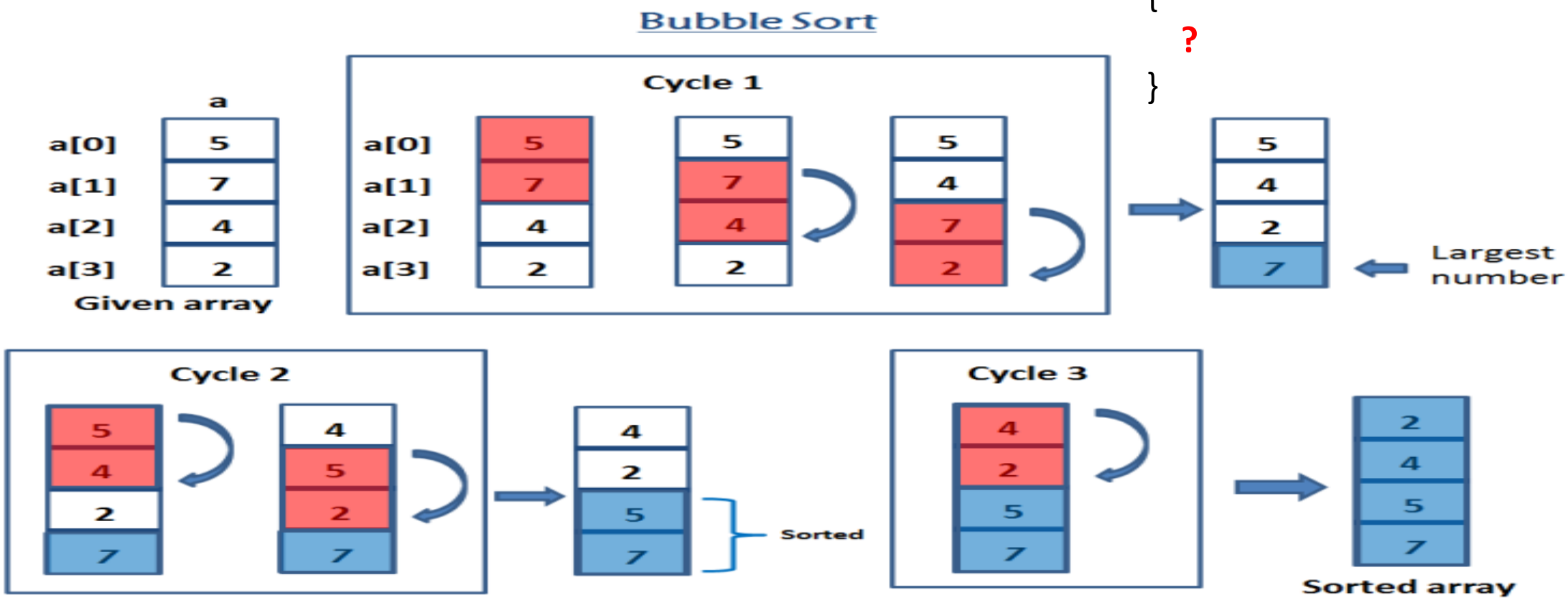
Bubble sort

- **bubble sort**:
  - orders a list of values by repetitively comparing neighboring elements and swapping their positions if necessary

- more specifically:
  - scan the list, exchanging adjacent elements if they are not in relative order;
  - this bubbles the highest value to the bottom
  - scan the list again, bubbling up the second highest value
  - repeat until all elements have been placed in their proper order

```
int n = arr.length;
for (int i = 0; i < ? ; i++)

    for (int j = 0; j <  ? ; j++)
        if (arr[j] > arr[j+1])
        {
            ?
        }
```



A[0]	
A[2]	

- Complete the Following BubbleSort Logic by filling the correct Logic/value for the ? Mark
- Assume Size of the Array is n

```
for (int i = 0; i < ? ; i++)

    for (int j = 0; j <  ? ; j++)
            if (arr[j] > arr[j+1])
            {
                ?
            }
```

Bubble sort code

```
for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-1-i; j++)
            if (arr[j] > arr[j+1])
            {
                // swap temp and arr[i]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
```

```java
// Java program for implementation of Bubble Sort
public class TestBubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }
```

```java
/* Prints the array */
    void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method to test above
    public static void main(String args[])
    {
        TestBubbleSort ob = new TestBubbleSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        ob.bubbleSort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
```

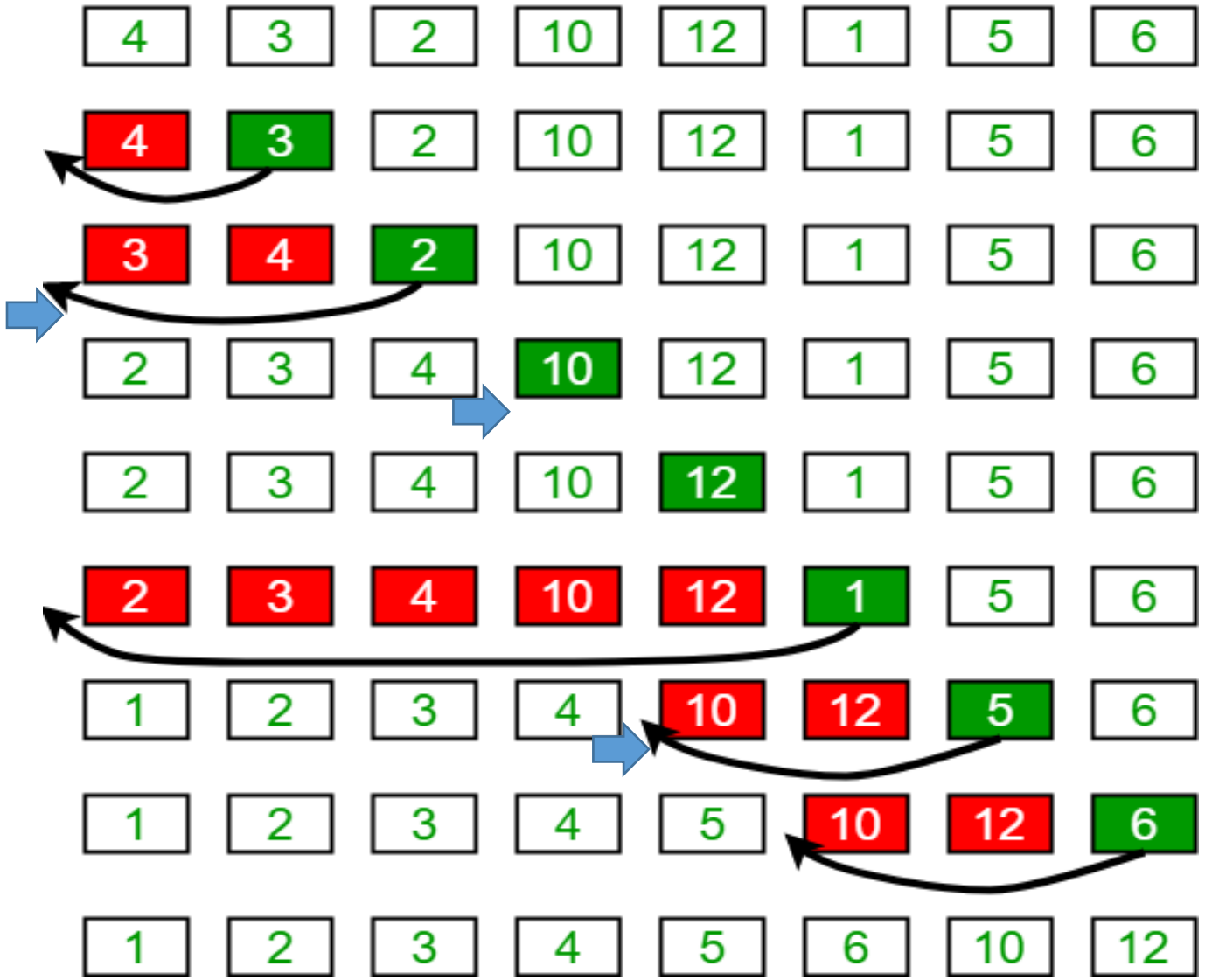
Insertion Sort

- Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

The insertion sort algorithm is performed using following steps...

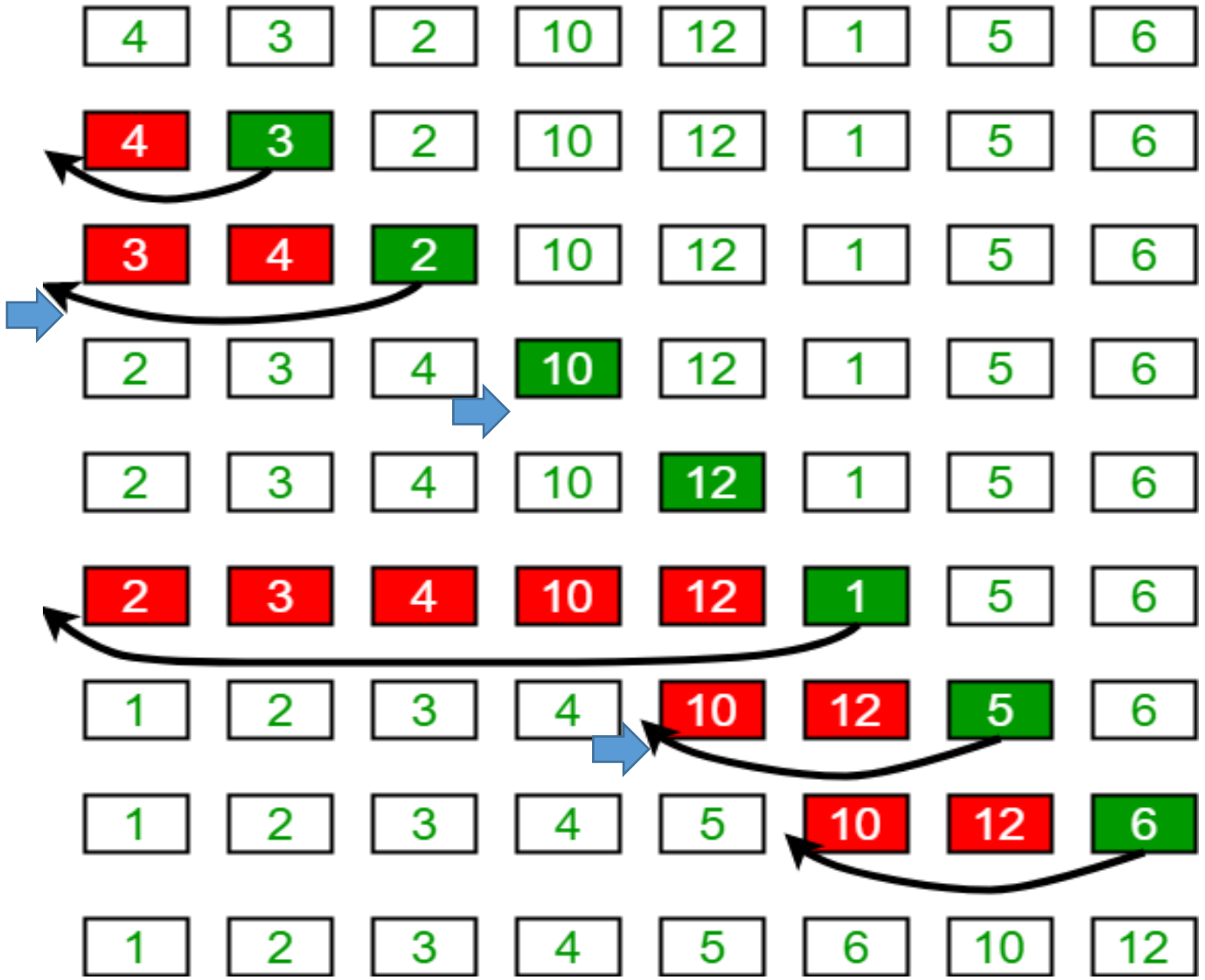
Step 1: Asume that first element in the list is in sorted portion of the list and remaining all elements are in unsorted portion.
Step 2: Consider first element from the unsorted list and insert that element into the sorted list in order specified.
Step 3: Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

Insertion Sort Execution Example




```
void sort(int arr[])
{
    [redacted]
}
```

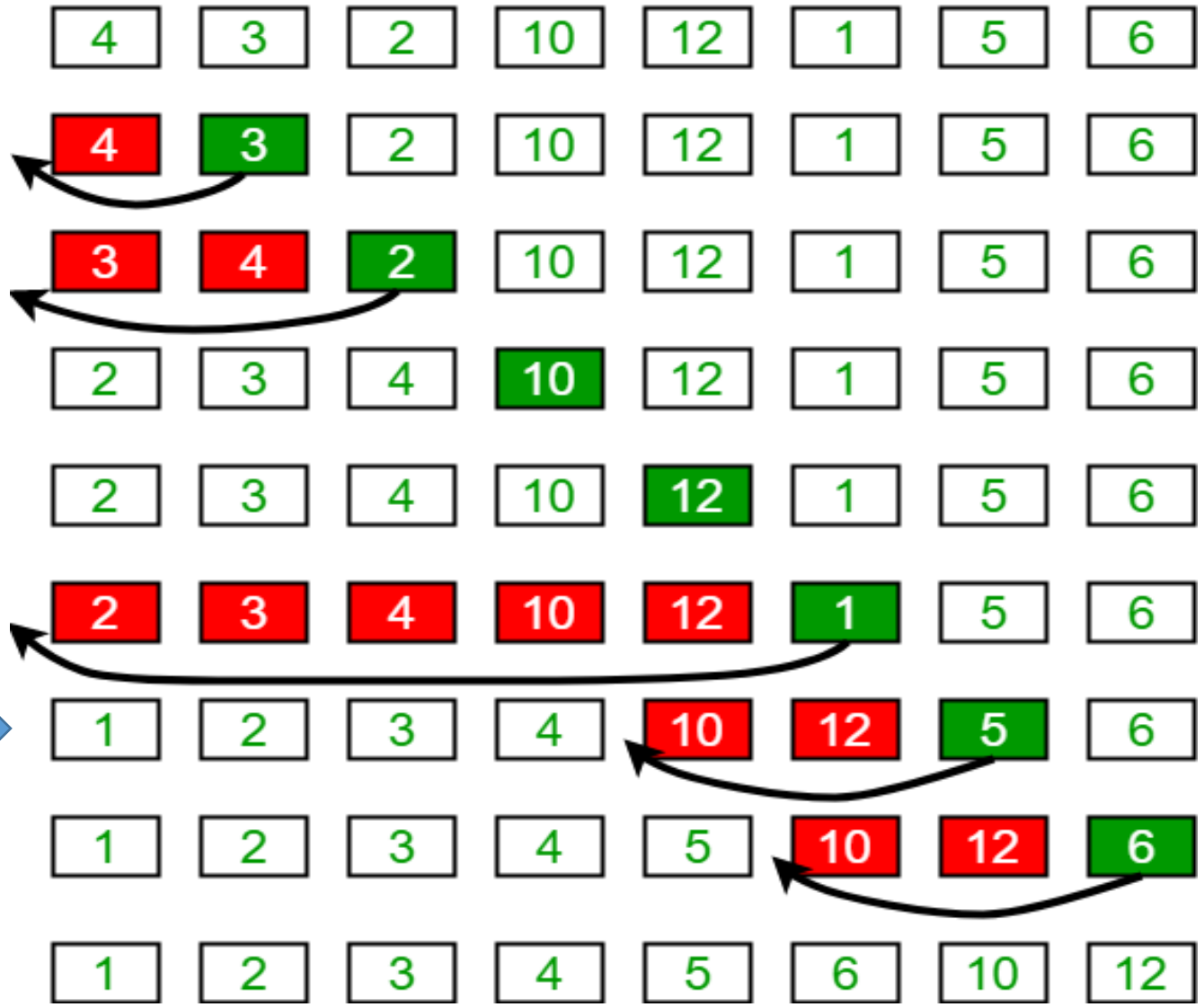
Insertion Sort Execution Example




```
void sort(int arr[])
{
   int n = arr.length;
   for (int i=1; i<n; ++i)
   {
            int key = arr[i];
            int j = i-1;

        /* Move elements of arr[0..i-1], that are greater than
    key,
            to one position ahead of their current position */
            while (j>=0 && arr[j] > key)
            {
               arr[j+1] = arr[j];
               j = j-1;
            }
            arr[j+1] = key;
   }
}
```

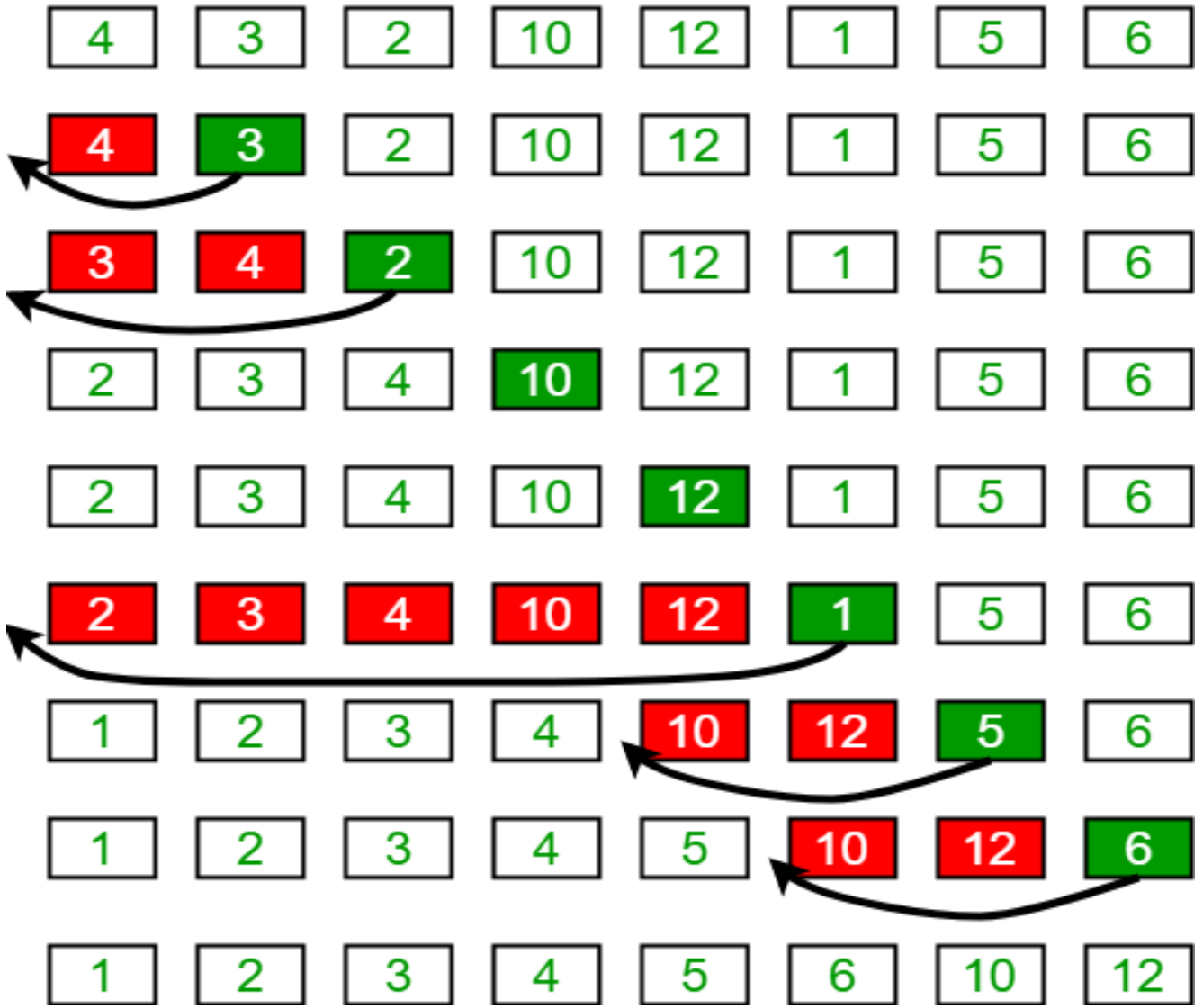
Insertion Sort Execution Example




```
void sort(int arr[])
{
   int n = arr.length;
   for (int i=1; i<n; ++i)
   {
           int key = arr[i];
           int j = i-1;

       /* Move elements of arr[0..i-1], that are greater th
    key,
          to one position ahead of their current position *
           while (j>=0 && arr[j] > key)
           {
              arr[j+1] = arr[j];
              j = j-1;
           }
           arr[j+1] = key;
   }
}
```

Insertion Sort Execution Example




```
void sort(int arr[])
{
  int n = arr.length;
  for (int i=1; i<n; ++i)
  {
          int key = arr[i];
          int j = i-1;

      /* Move elements of arr[0..i-1], that are greater than
 key,
         to one position ahead of their current position */
          while (j>=0 && arr[j] > key)
          {
             arr[j+1] = arr[j];
             j = j-1;
          }
          arr[j+1] = key;
  }
}
```

```java
// Java program for implementation of Insertion Sort
public class TestInsertionSort
{
   /*Function to sort array using insertion sort*/
   void sort(int arr[])
   {
      int n = arr.length;
      for (int i=1; i<n; ++i)
      {
               int key = arr[i];
               int j = i-1;

          /* Move elements of arr[0..i-1], that are greater than
       key,
               to one position ahead of their current position */
               while (j>=0 && arr[j] > key)
               {
                  arr[j+1] = arr[j];
                  j = j-1;
               }
               arr[j+1] = key;
      }
   }
```

```java
   /* A utility function to print array of size n*/
   static void printArray(int arr[])
   {
      int n = arr.length;
      for (int i=0; i<n; ++i)
         System.out.print(arr[i] + " ");

      System.out.println();
   }


   public static void main(String args[])
   {
      int arr[] = {12, 11, 13, 5, 6};

      TestInsertionSort ob = new TestInsertionSort();
      ob.sort(arr);

      printArray(arr);
   }
}
```

Divide and Conquer

Very important technique in algorithm design

1. Divide problem into smaller parts
2. Independently solve the simpler parts
    - Think recursion
    - Or potential parallelism
3. Combine solution of parts to produce overall solution

Divide-and-Conquer Sorting

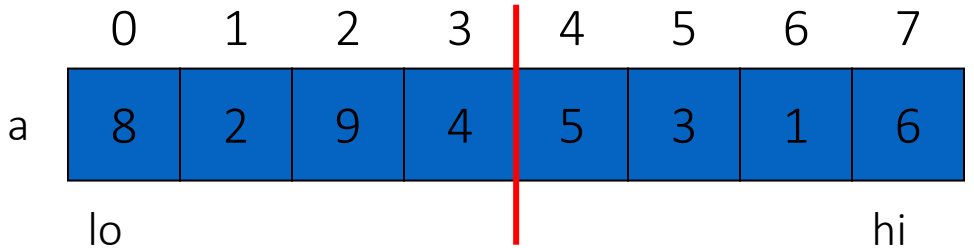
Two great sorting methods are fundamentally divide-and-conquer

Mergesort: Recursively sort the left half

Recursively sort the right half

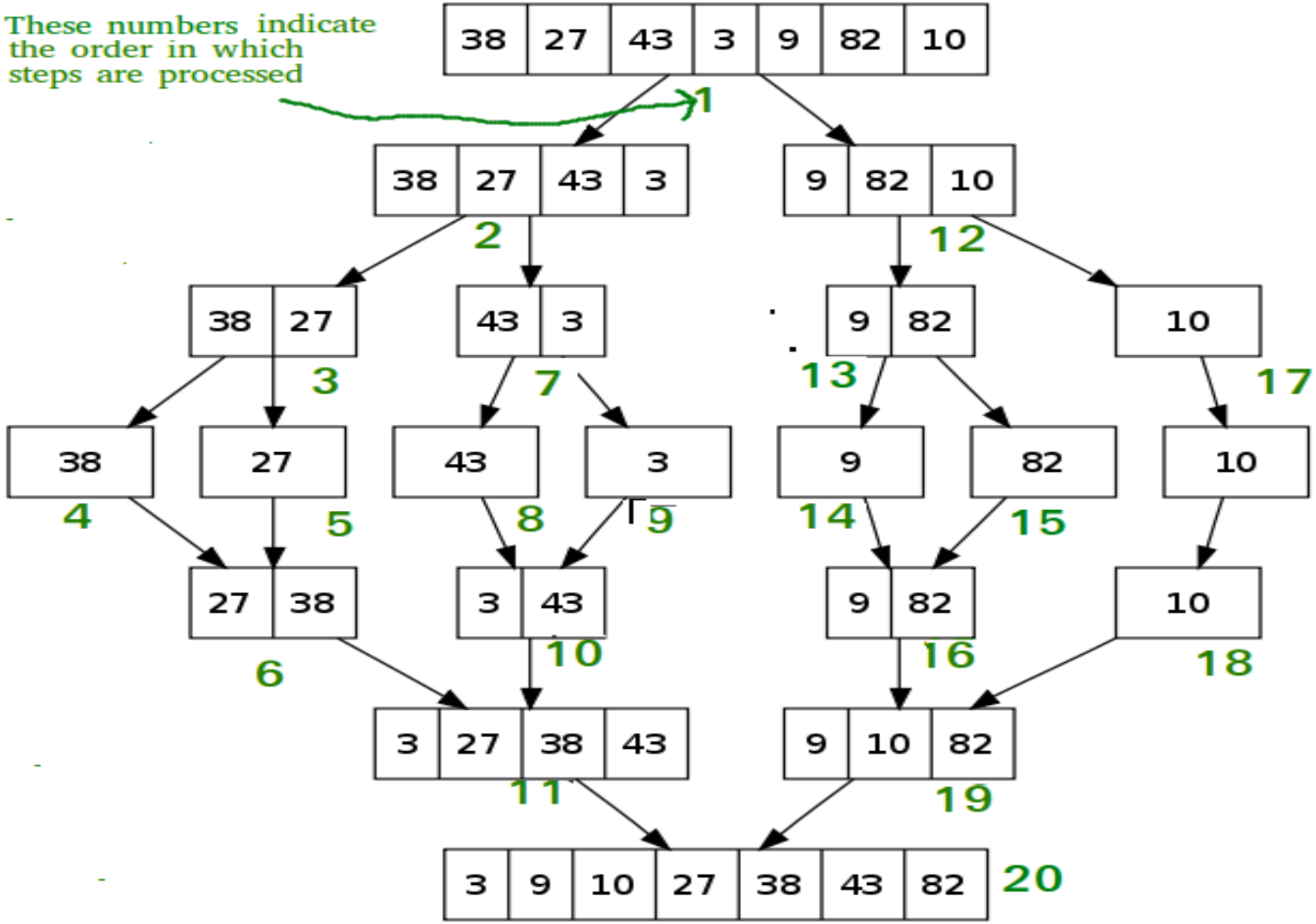
Merge the two sorted halves

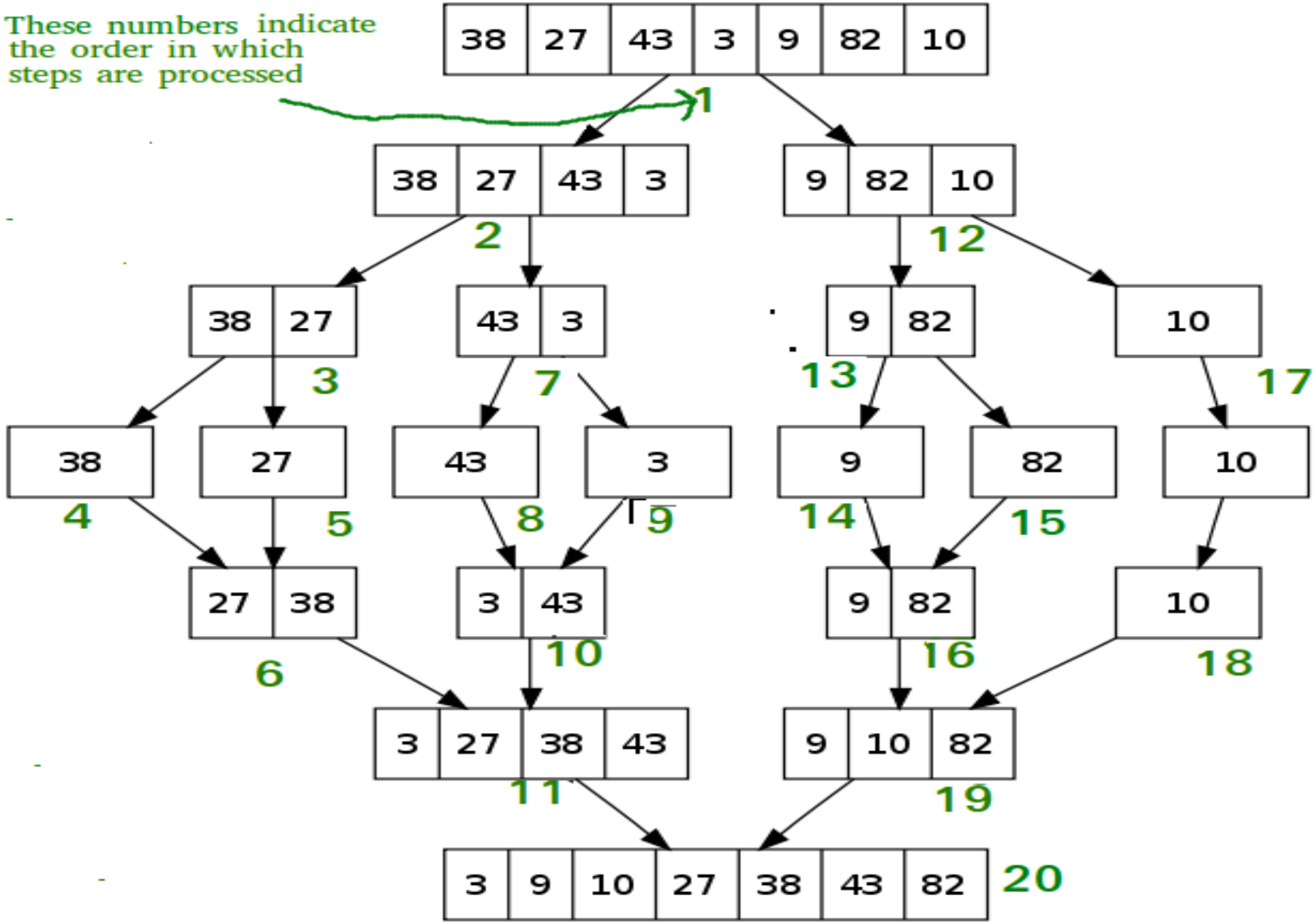
Mergesort



To sort array from position **lo** to position **hi**:

- If range is 1 element long, it is already sorted! (our base case)
- Else, split into two halves:
  - Sort from **lo** to **(hi+lo)/2**
  - Sort from **(hi+lo)/2** to **hi**
  - Merge the two halves together


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


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

```
public class MergeSort
{
 void merge(int arr[], int l, int m, int r)
  {  System.out.println("::Merge L="+l +"M="+m+ "to  R="+r);
     int n1 = m - l + 1; // Find sizes of two subarrays to be merged
     int n2 = r - m;
     /* Create temp arrays */
     int L[] = new int [n1];
     int R[] = new int [n2];
     /*Copy data to temp arrays*/
     for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
     for (int j=0; j<n2 ; ++j)
        R[j] = arr[m + 1+ j];
     /* Merge the temp arrays */
     // Initial indexes of first and second subarrays
     int i = 0, j = 0;
     // Initial index of merged subarry array
     int k = l;
     while (i < n1 && j < n2)
     {
        if (L[i] <= R[j])
        {
           arr[k] = L[i];          i++;
        }
        else
        {
           arr[k] = R[j];        j++;
        }
        k++;
     }
     /* Copy remaining elements of L[] if any */
     while (i < n1)
     {
        arr[k] = L[i];             i++;             k++;
     }
     /* Copy remaining elements of R[] if any */
     while (j < n2)
     {
        arr[k] = R[j];             j++;             k++;
     }
  }
```

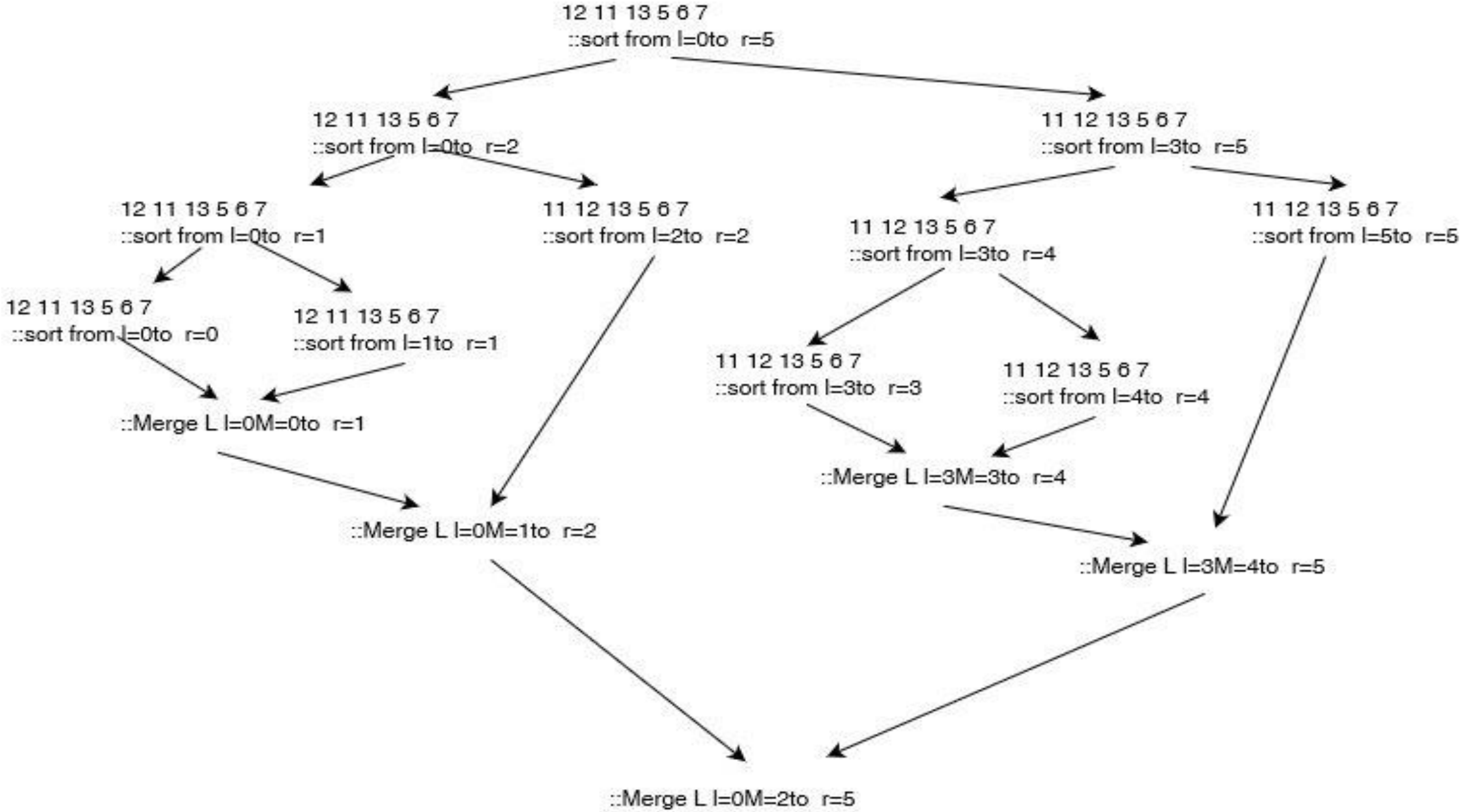
```
  // Main function that sorts arr[l..r] using merge()
   void sort(int arr[], int l, int r)
   {
      if (l < r)
      {
         // Find the middle point
         int m = (l+r)/2;

         // Sort first and second halves
         sort(arr, l, m);
         sort(arr , m+1, r);
         merge(arr, l, m, r);
      }
   }

  static void printArray(int arr[])
   {
      int n = arr.length;
      for (int i=0; i<n; ++i)
         System.out.print(arr[i] + " ");
      System.out.println();
   }

// Driver method
   public static void main(String args[])
   {
      int arr[] = {12, 11, 13, 5, 6, 7};
       System.out.println("Given Array");
      printArray(arr);
       MergeSort ob = new MergeSort();
      ob.sort(arr, 0, arr.length-1);

      System.out.println("\nSorted array");
      printArray(arr);
   }
}
```


12 11 13 5 6 7
::sort from l=0to r=5
12 11 13 5 6 7
::sort from l=0to r=2
11 12 13 5 6 7
::sort from l=3to r=5
12 11 13 5 6 7
::sort from l=0to r=1
11 12 13 5 6 7
::sort from l=2to r=2
11 12 13 5 6 7
::sort from l=3to r=4
11 12 13 5 6 7
::sort from l=5to r=5
12 11 13 5 6 7
::sort from l=0to r=0
12 11 13 5 6 7
::sort from l=1to r=1
11 12 13 5 6 7
::sort from l=3to r=3
11 12 13 5 6 7
::sort from l=4to r=4
::Merge L l=0M=0to r=1
::Merge L l=3M=3to r=4
::Merge L l=0M=1to r=2
::Merge L l=3M=4to r=5
::Merge L l=0M=2to r=5

```
public class MergeSort
{
 void merge(int arr[], int l, int m, int r)
  {  System.out.println("::Merge L="+l +"M="+m+ "to  R="+r);
     int n1 = m - l + 1; // Find sizes of two subarrays to be merged
     int n2 = r - m;
     /* Create temp arrays */
     int L[] = new int [n1];
     int R[] = new int [n2];
     /*Copy data to temp arrays*/
     for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
     for (int j=0; j<n2 ; ++j)
        R[j] = arr[m + 1+ j];
     /* Merge the temp arrays */
     // Initial indexes of first and second subarrays
     int i = 0, j = 0;
     // Initial index of merged subarry array
     int k = l;
     while (i < n1 && j < n2)
     {
        if (L[i] <= R[j])
        {
           arr[k] = L[i];          i++;
        }
        else
        {
           arr[k] = R[j];        j++;
        }
        k++;
     }
     /* Copy remaining elements of L[] if any */
     while (i < n1)
     {
        arr[k] = L[i];             i++;             k++;
     }
     /* Copy remaining elements of R[] if any */
     while (j < n2)
     {
        arr[k] = R[j];             j++;             k++;
     }
  }
```

```
  // Main function that sorts arr[l..r] using merge()
  void sort(int arr[], int l, int r)
  {
     if (l < r)
     {
        // Find the middle point
        int m = (l+r)/2;
        System.out.println("sort from:"+l + "to:"+r + " m:"+m);
        // Sort first and second halves
       System.out.println("\n START Left Array ="+l+":“+m); printSubArray(arr,l,m);
        sort(arr, l, m);
       System.out.println("\n END Left Array ="+l+":“+m); printSubArray(arr,l,m);

        System.out.println("\n START Right Array ="+(m+1)+":”+r); printSubArray(arr,m+1,r);
        sort(arr , m+1, r);
        System.out.println("\n END Right Array ="+(m+1)+":“+r); printSubArray(arr,m+1,r);
        // Merge the sorted halves
        merge(arr, l, m, r);
     }
  }

 static void printArray(int arr[])
  {
     int n = arr.length;
     for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
     System.out.println();
  }

static void printSubArray(int arr[] ,int l,int r)
  {
     int n = arr.length;
     for (int i=l; i<=r; ++i)
        System.out.print(arr[i] + " ");
     System.out.println();
  }
  // Driver method
  public static void main(String args[])
  {
     int arr[] = {12, 11, 13, 5, 6, 7};
      System.out.println("Given Array");
     printArray(arr);
      MergeSort ob = new MergeSort();
     ob.sort(arr, 0, arr.length-1);

     System.out.println("\nSorted array");
     printArray(arr);
  }
}
```

Given Array
12 11 13 5 6 7
sort from:0to:5 m:2

START Left Array =0:2
12 11 13
sort from:0to:2 m:1

START Left Array =0:1
12 11
sort from:0to:1 m:0

START Left Array =0:0
12
END Left Array =0:0
12
START Right Array =1:1
11
END Right Array =1:1
11

::Merge L=0M=0to R=1
11 12

END Left Array =0:1
11 12
START Right Array =2:2
13
END Right Array =2:2
13
::Merge L=0M=1to R=2
11 12 13

END Left Array =0:2
11 12 13
START Right Array =3:5
5 6 7
sort from:3to:5 m:4

START Left Array =3:4
5 6
sort from:3to:4 m:3

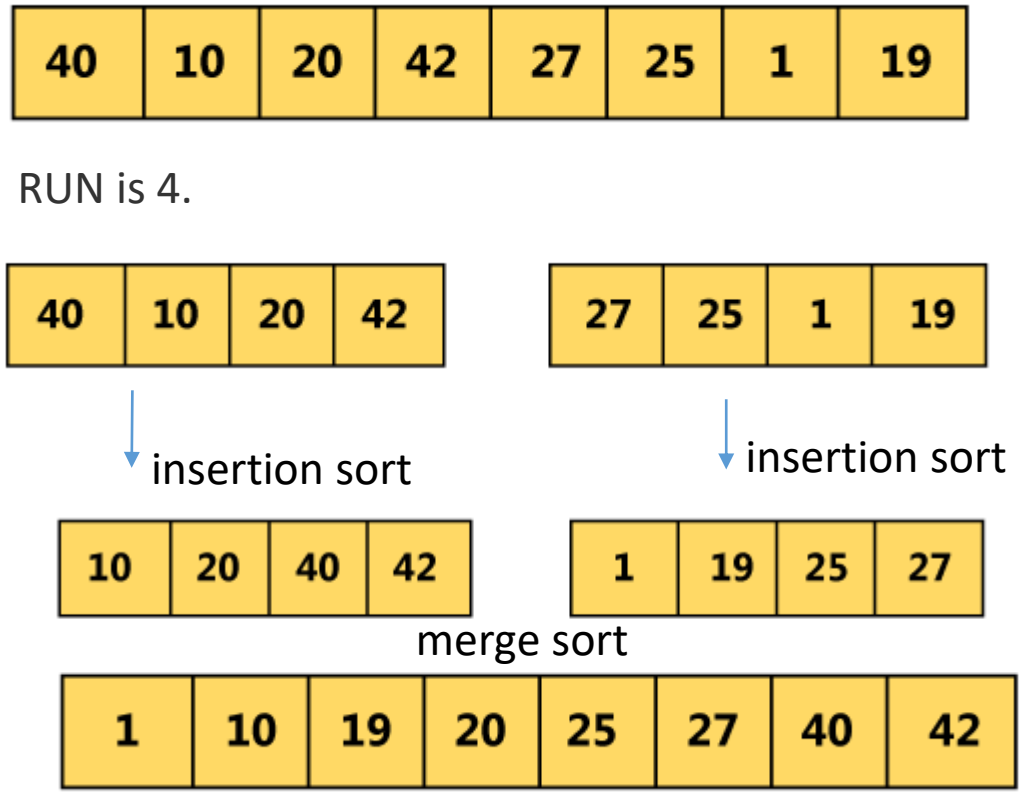
START Left Array =3:3
5
END Left Array =3:3
5
START Right Array =4:4
6
END Right Array =4:4
6
::Merge L=3M=3to R=4
5 6

END Left Array =3:4
5 6
START Right Array =5:5
7
END Right Array =5:5
7
::Merge L=3M=4to R=5
5 6 7

END Right Array =3:5
5 6 7
::Merge L=0M=2to R=5
5 6 7 11 12 13

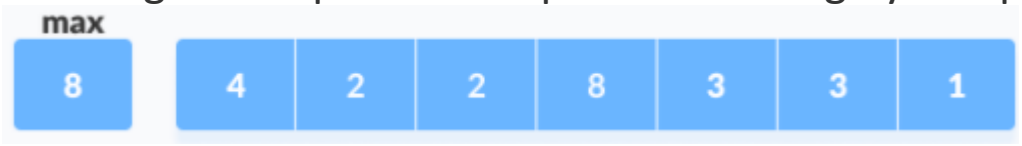
Tim Sort

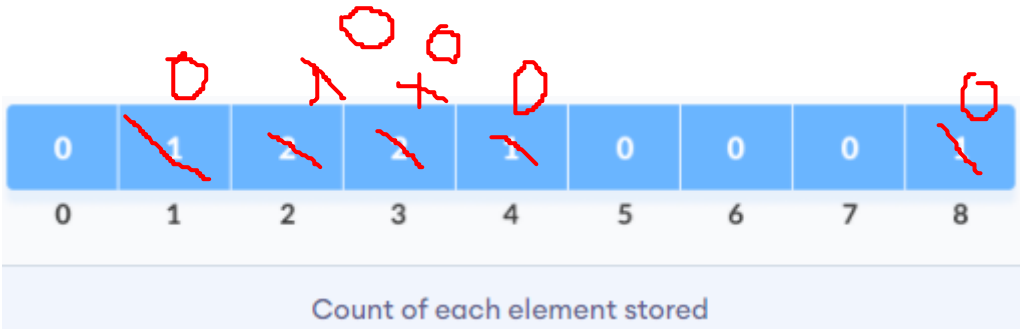
1. Step 1 - Divide the array into the number of blocks known as run.
2. Step 2 - Consider the size of run, either 32 or 64.
3. Step 3 - Sort the individual elements of every run one by one using insertion sort.
4. Step 4 - Merge the sorted runs one by one using the merge function of merge sort.
5. Step 5 - Double the size of merged sub-arrays after every iteration.

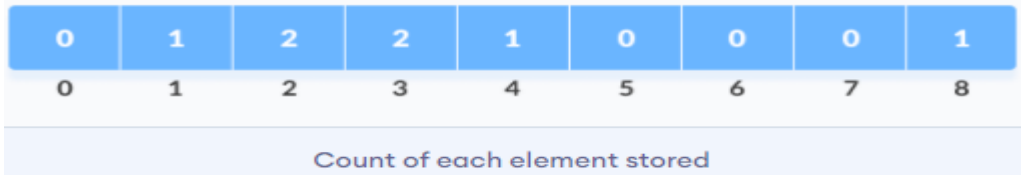


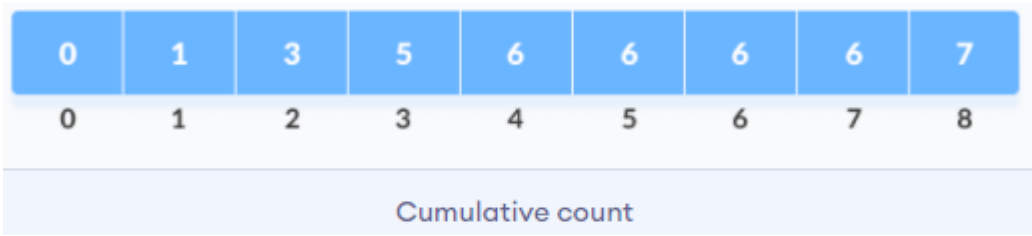
counting sort Algorithm

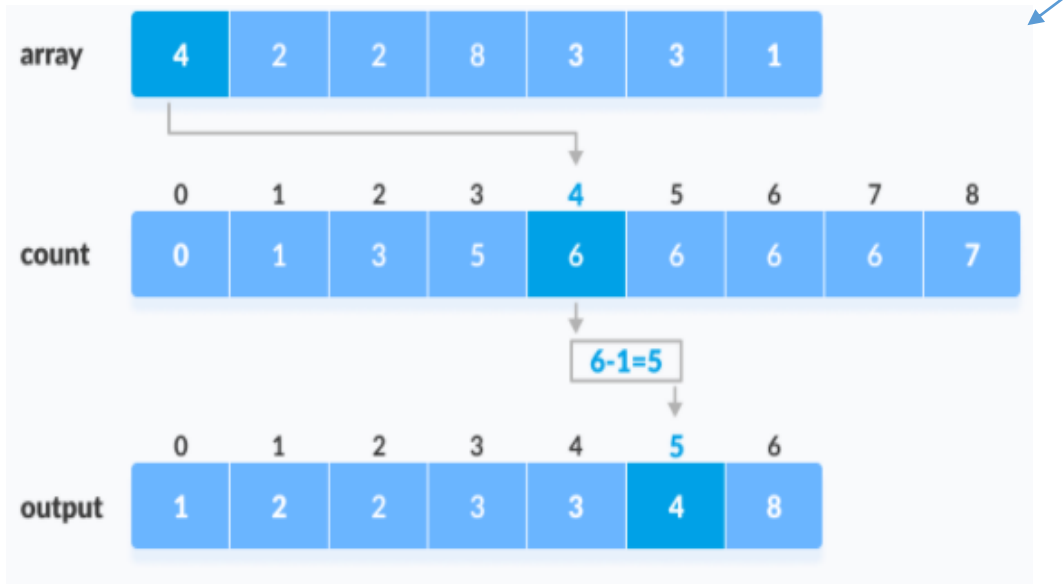
sorting technique doesn't perform sorting by comparing elements. It performs sorting by counting objects







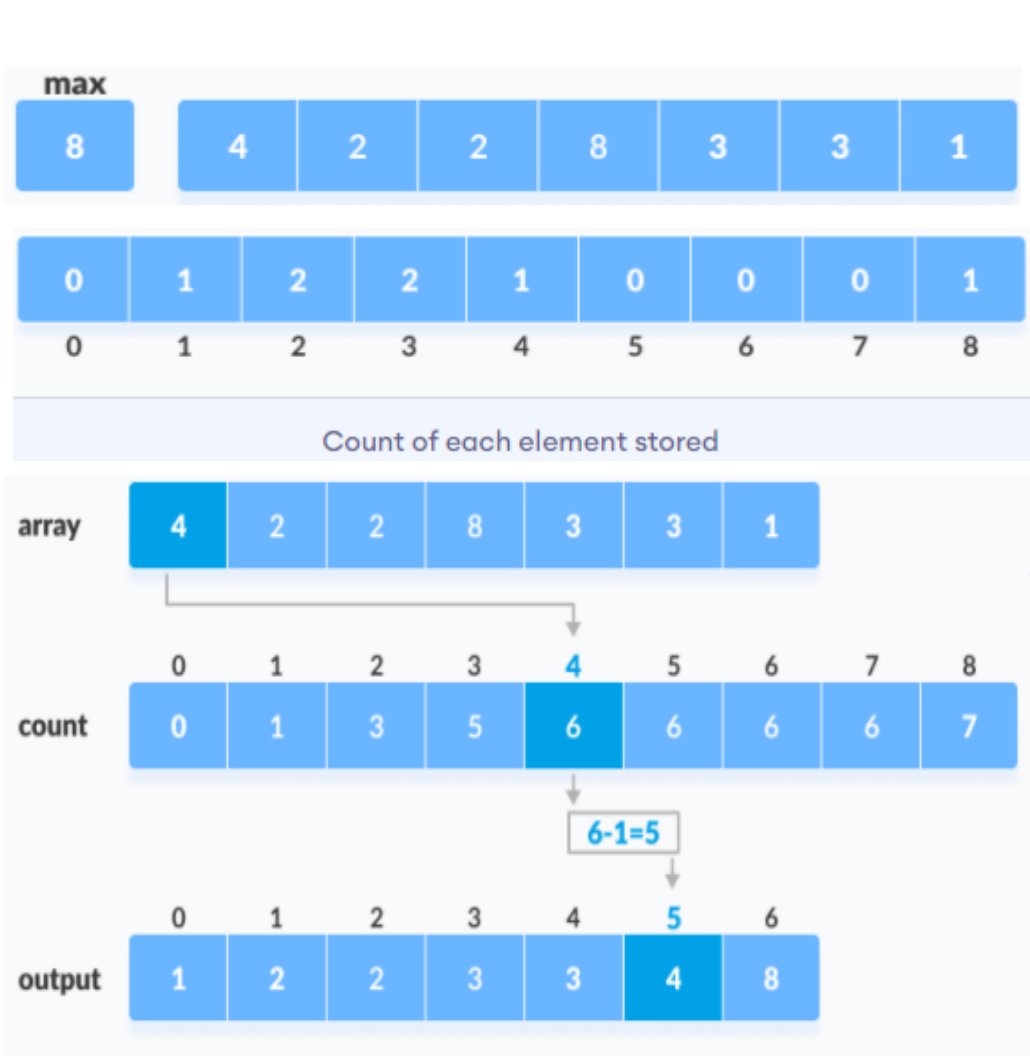


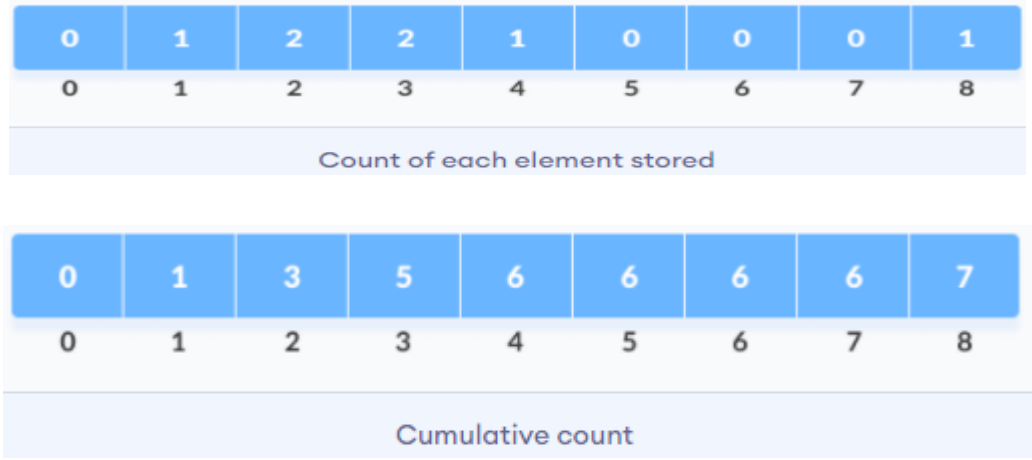


After placing each element at its correct position, decrease its count by one.

counting sort Algorithm

sorting technique doesn't perform sorting by comparing elements. It performs sorting by counting objects





After placing each element at its correct position, decrease its count by one.

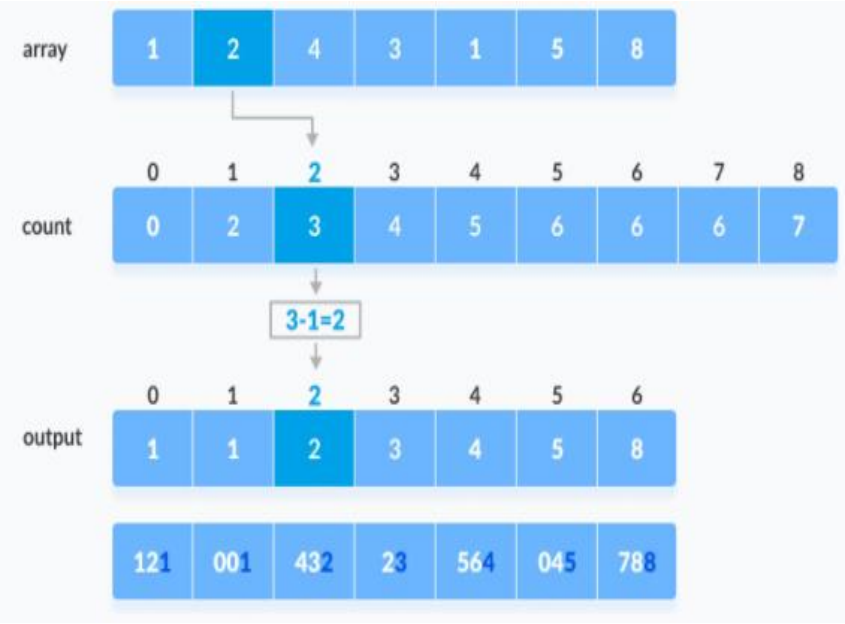
Radix Sort Algorithm

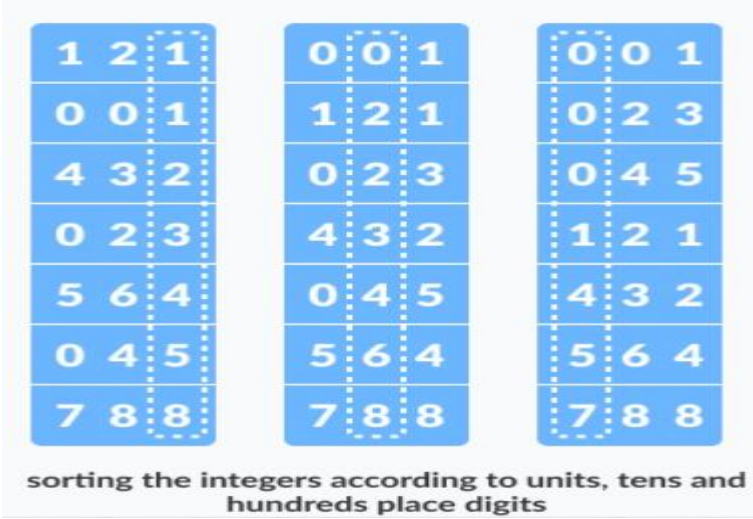
initial array be [121, 432, 564, 23, 1, 45, 788].

- Find the largest element in the array→ 788

It has 3 digits. Therefore, the loop should go up to hundreds place (3 times).

- go through each significant place one by one . Apply counting sort for each





sorting the integers according to units, tens and hundreds place digits

Org Val	121	432	564	023	001	045	788
1st D Val	1	2	4	3	1	5	8
Max=8							

Digit	0	1	2	3	4	5	6	7	8
Count	0	2	1	1	1	1	0	0	1
C Count	0	2	3	4	5	6	6	6	7

Org Val	121	432	564	23	1	45	788
1st D Val	1	2	4	3	1	5	8
Index Calc	2-2=0	3-1=2	5-1=4	4-1=3	2-1=1	6-1=5	7-1=6

index	0	1	2	3	4	5	6
S. 1st D Val	1	1	2	3	4	5	8
S. Org Val	121	001	432	023	564	045	788

Org Val	121	001	432	23	564	45	788
2nd D Val	2	0	3	2	6	4	8
Max=8							

Digit	0	1	2	3	4	5	6	7	8
Count	1	0	2	1	1	0	1	0	1
C Count	1	1	3	4	5	5	6	6	7

Org Val	121	001	432	23	564	45	788
2nd D Val	2	0	3	2	6	4	8
Index Calc	3-2=1	1-1=0	4-1=3	3-1=2	6-1=5	5-1=4	7-1=6

index	0	1	2	3	4	5	6
S. 2nd D Val	0	2	2	3	4	6	8
S. Org Val	001	121	023	432	045	564	788

Org Val	001	121	023	432	045	564	788
3rd D Val	0	1	0	4	0	5	7
max=7							

Digit	0	1	2	3	4	5	6	7
Count	3	1	0	0	1	1	0	1
C Count	3	4	4	4	5	6	6	7

Org Val	001	121	023	432	045	564	788
3rd D Val	0	1	0	4	0	5	7
Index Calc	3-3=0	4-1=3	3-2=1	5-1=4	3-1=2	6-1=5	7-1=6

index	0	1	2	3	4	5	6
S. 3rd D Val	0	0	0	1	4	5	7
S. Org Val	001	023	045	121	432	564	788

Sorting in Collection

- Used to sort the elements of List.List elements must be of Comparable type

```
import java.util.*;
class TestSort{
public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();
al.add("Viru");
al.add("Saurav");
al.add("Mukesh");
al.add("Tahir");

Collections.sort(al);
Iterator itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
 }
}
}
```