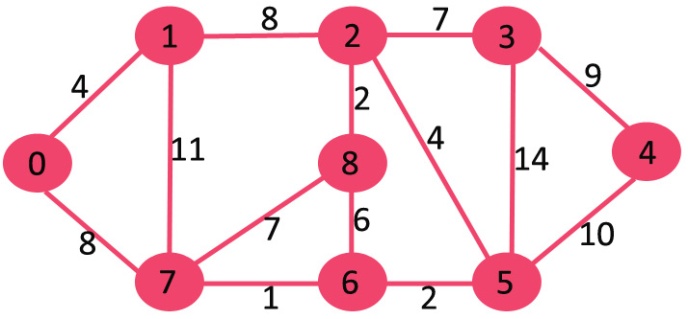
**Dijkstra’s shortest path algorithm |**

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Like Prim’s MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.  
Algorithm  
**1)** Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.  
**2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.  
**3)** While sptSet doesn’t include all vertices  
….**a)** Pick a vertex u which is not there in sptSet and has minimum distance value.  
….**b)** Include u to sptSet.  
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Let us understand with the following example:



The set sptSet is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST1.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ2.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).  
[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ3.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ4.jpg)

We repeat the above steps until sptSet does include all vertices of given graph. Finally, we get the following Shortest Path Tree (SP



**Sorting**

Sorting is a technique through which we arrange the data in such a manner so that the searching of the data becomes easy. A lot of sorting techniques has been implemented till now to cope up the faster execution of the result and to manage the data comfortably . Sorting and Searching are fundamental operations in computer science. Sorting refers to the operation of arranging data in some given order. Searching refers to the operation of searching the particular record from the existing information. Normally, the information retrieval involves searching, sorting and merging.

Let get to know about two sorting techniques and analyze their performance. The two techniques are:

1. Internal Sorting
2. External Sorting

* Internal Sorting takes place in the main memory of a computer. The internal sorting methods are applied to small collection of data. It means that, the entire collection of data to be sorted in small enough that the sorting can take place within main memory. We will study the following methods of internal sorting
* **External sorting** is a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually [RAM](https://en.wikipedia.org/wiki/RAM)) and instead they must reside in the slower external memory usually a hard disk drive Thus, external sorting algorithms are external memory algorithms and thus applicable in the external memory model of computation

Some common internal sorting algorithms include:

1. Bubble Sort
2. [Insertion Sort](https://en.wikipedia.org/wiki/Insertion_Sort)
3. [Quick Sort](https://en.wikipedia.org/wiki/Quick_Sort)
4. [Heap Sort](https://en.wikipedia.org/wiki/Heap_Sort)
5. [Radix Sort](https://en.wikipedia.org/wiki/Radix_Sort)
6. [Selection sort](https://en.wikipedia.org/wiki/Selection_sort)

## External sorting

External sorting is a technique in which the data is stored on the secondary memory, in which part by part data is loaded into the main memory and then sorting can be done over there. Then this sorted data will be stored in the intermediate files. Finally, these files will be merged to get a sorted data. Thus by using the external sorting technique, a huge amount of data can be sorted easily. In case of external sorting, all the data cannot be accommodated on the single memory, in this case, some amount of memory needs to be kept on a memory such as hard disk, compact disk and so on.

The requirement of external sorting is there, where the data we have to store in the main memory does not fit into it. Basically, it consists of two phases that are:

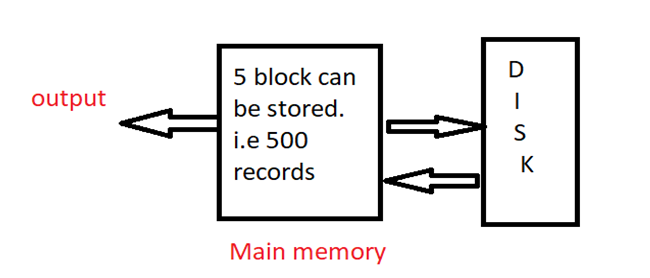
1. Sorting phase: This is a phase in which a large amount of data is sorted in an intermediate file.
2. Merge phase: In this phase, the sorted files are combined into a single larger file.

One of the best examples of external sorting is external merge sort.

## External merge sort

The **external merge sort** is a technique in which the data is stored in intermediate files and then each intermediate files are sorted independently and then combined or merged to get a sorted data.

**For example:** Let us consider there are 10,000 records which have to be sorted. For this, we need to apply the external merge sort method. Suppose the main memory has a capacity to store 500 records in a block, with having each block size of 100 records.



in this example, we can see 5 blocks will be sorted in intermediate files. This process will be repeated 20 times to get all the records. Then by this, we start merging a pair of intermediate files in the main memory to get a sorted output.

### Two-Way Merge Sort

**Two-way merge sort** is a technique which works in two stages which are as follows here:

**Stage 1**: Firstly break the records into the blocks and then sort the individual record with the help of two input tapes.

**Stage 2**: In this merge the sorted blocks and then create a single sorted file with the help of two output tapes.

By this, it can be said that **two-way merge sort** uses the two input tapes and two output tapes for sorting the data.

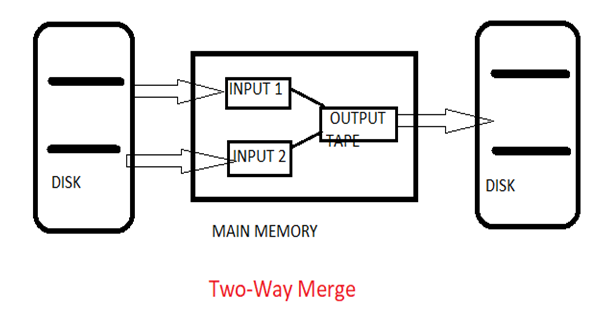
### Algorithm for Two-Way Merge Sort:

**Step 1)** Divide the elements into the blocks of size M. Sort each block and then write on disk.

**Step 2)** Merge two runs

1. Read first value on every two runs.
2. Then compare it and sort it.
3. Write the sorted record on the output tape.

**Step 3)** Repeat the step 2 and get longer and longer runs on alternates tapes. Finally, at last, we will get a single sorted list.



**Radix Sort Algorithm**

Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order. In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers. Sorting is performed from least significant digit to the most significant digit.

Radix sort algorithm requires the number of passes which are equal to the number of digits present in the largest number among the list of numbers. For example, if the largest number is a 3 digit number then that list is sorted with 3 passes.

**Step by Step Process**

The Radix sort algorithm is performed using the following steps...

Step 1 - Define 10 queues each representing a bucket for each digit from 0 to 9.

Step 2 - Consider the least significant digit of each number in the list which is to be sorted.

Step 3 - Insert each number into their respective queue based on the least significant digit.

Step 4 - Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.

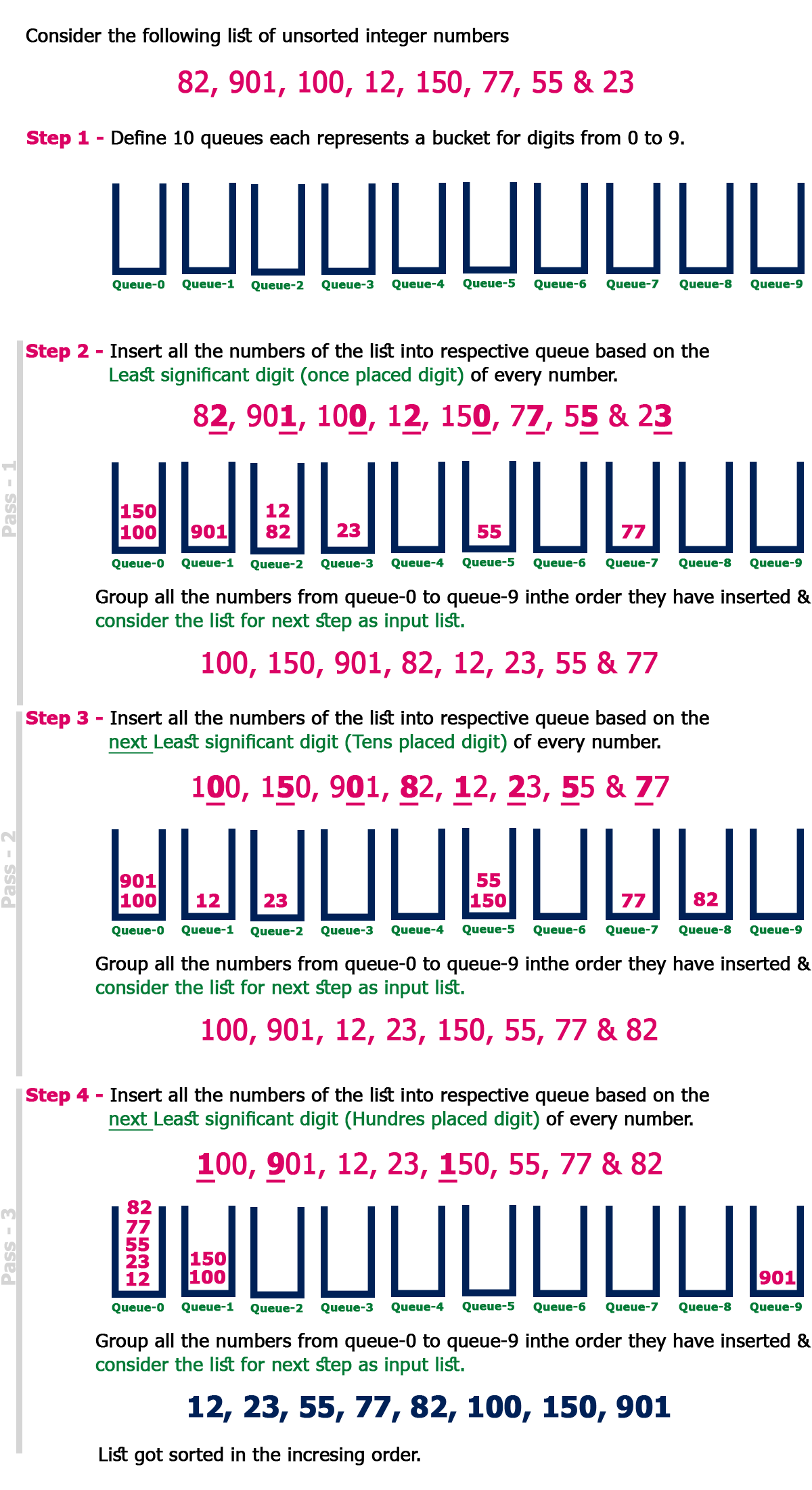
Step 5 - Repeat from step 3 based on the next least significant digit.

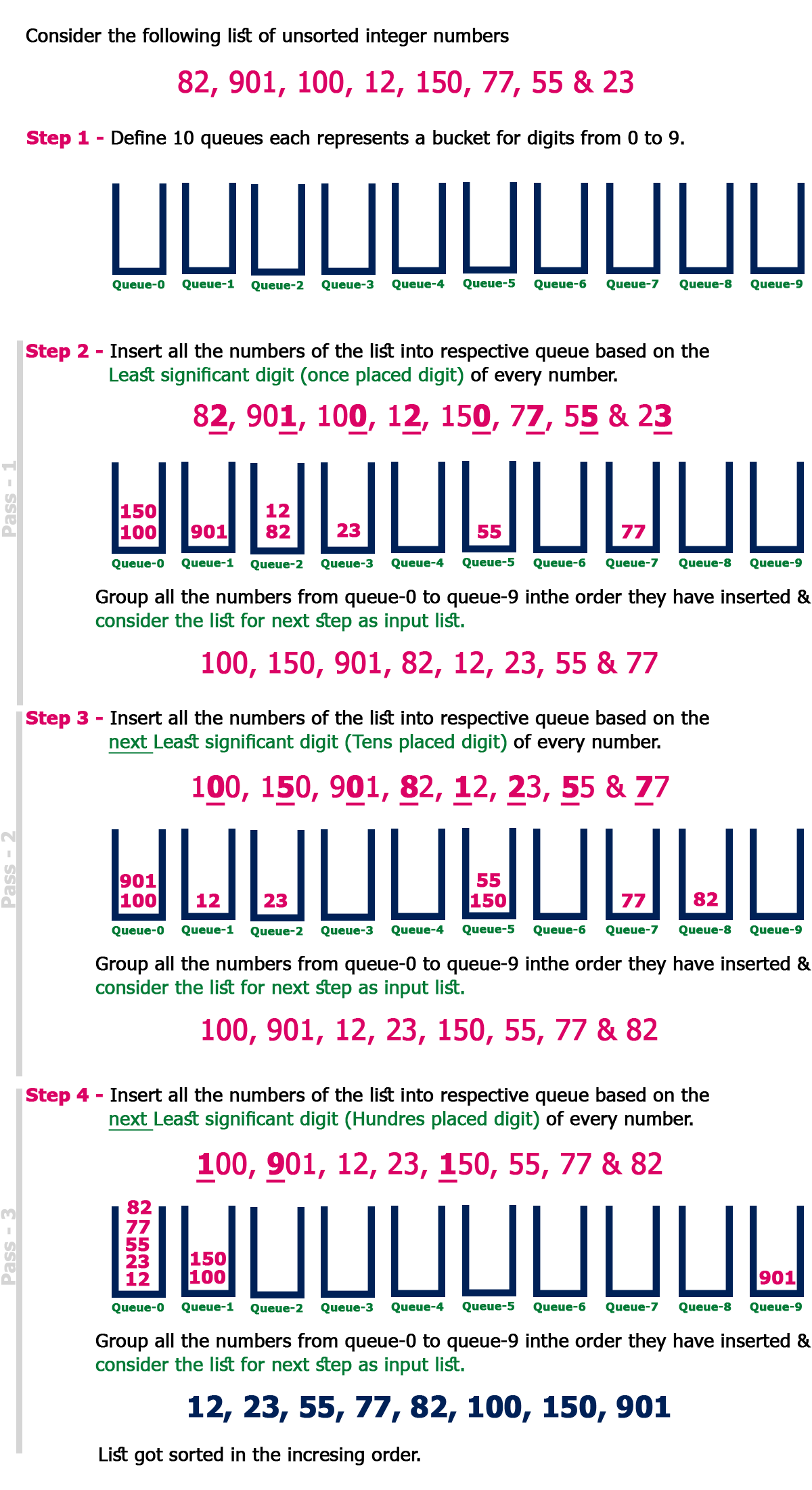
Step 6 - Repeat from step 2 until all the numbers are grouped based on the most significant digit.

**Complexity of the Radix Sort Algorithm**

To sort an unsorted list with **'n'** number of elements, Radix sort algorithm needs the following complexities...

**Worst Case : O(n)**  
**Best Case : O(n)**  
**Average Case : O(n)**





**Quick Sort Algorithm**

Quick sort is a fast sorting algorithm used to sort a list of elements. Quick sort algorithm is invented by C. A. R. Hoare.

The quick sort algorithm attempts to separate the list of elements into two parts and then sort each part recursively. That means it use divide and conquer strategy. In quick sort, the partition of the list is performed based on the element called pivot. Here pivot element is one of the elements in the list.

The list is divided into two partitions such that "all elements to the left of pivot are smaller than the pivot and all elements to the right of pivot are greater than or equal to the pivot".

**Step by Step Process**

In Quick sort algorithm, partitioning of the list is performed using following steps..

Step 1 - Consider the first element of the list as pivot (i.e., Element at first position in the list).

Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.

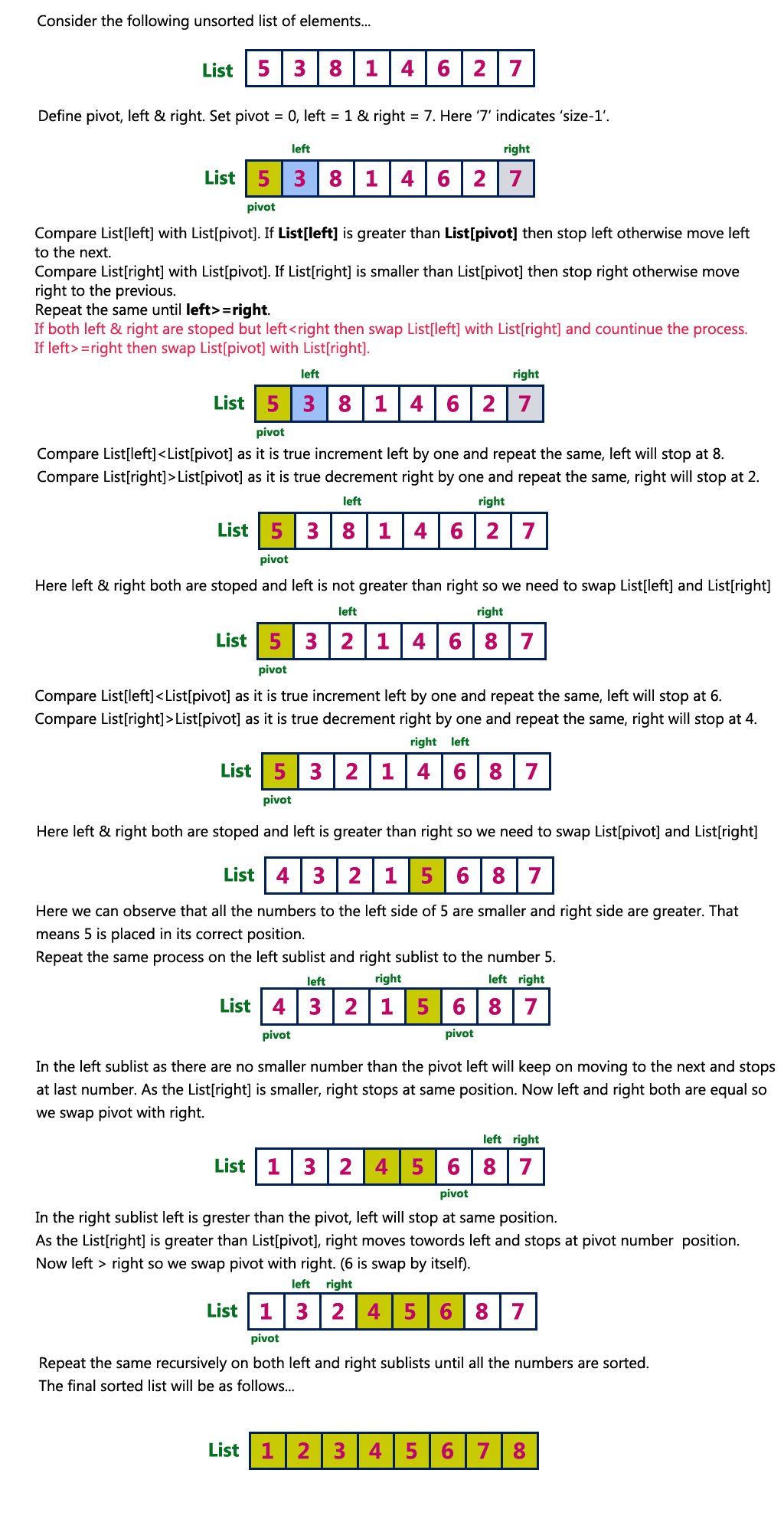
Step 3 - Increment i until list[i] > pivot then stop.

Step 4 - Decrement j until list[j] < pivot then stop.

Step 5 - If i < j then exchange list[i] and list[j].

Step 6 - Repeat steps 3,4 & 5 until i > j.

Step 7 - Exchange the pivot element with list[j] element.



**Complexity of the Quick Sort Algorithm**

To sort an unsorted list with **'n'** number of elements, we need to make **((n-1)+(n-2)+(n-3)+......+1) = (n (n-1))/2** number of comparisions in the worst case. If the list is already sorted, then it requires **'n'** number of comparisions.

**Worst Case : O(n2)**  
**Best Case : O (n log n)**  
**Average Case : O (n log n)**

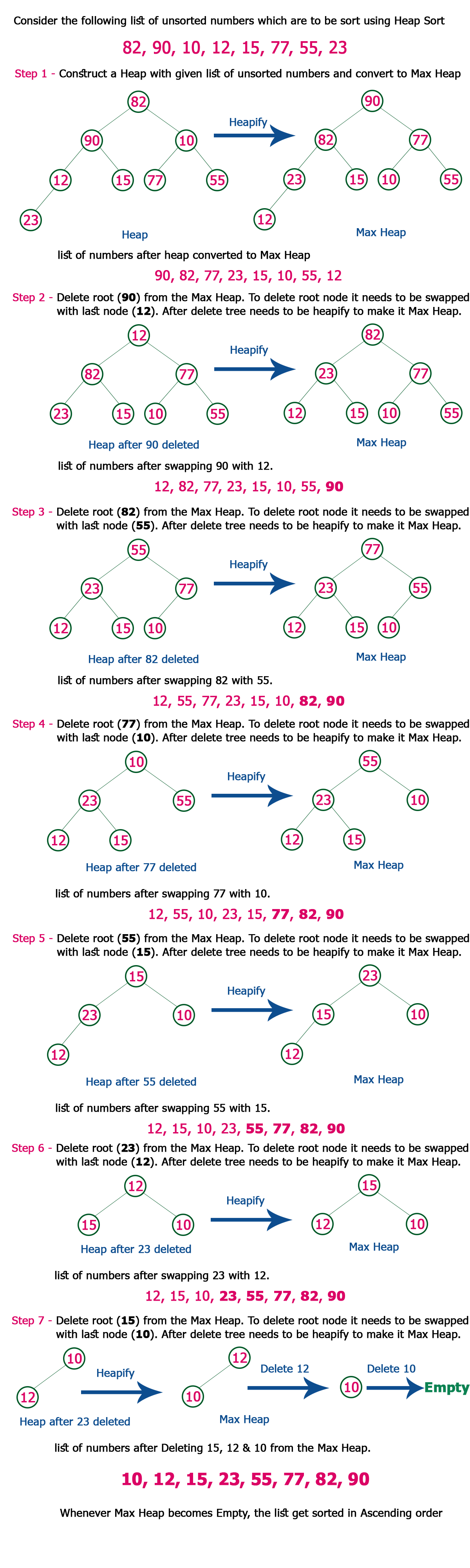
**Heap Sort Algorithm**

Heap sort is one of the sorting algorithms used to arrange a list of elements in order. Heapsort algorithm uses one of the tree concepts called **Heap Tree**. In this sorting algorithm, we use **Max Heap** to arrange list of elements in Descending order and **Min Heap** to arrange list elements in Ascending order.

Step by Step Process

The Heap sort algorithm to arrange a list of elements in ascending order is performed using following steps...

* Step 1 - Construct a **Binary Tree** with given list of Elements.
* Step 2 - Transform the Binary Tree into **Min Heap.**
* Step 3 - Delete the root element from Min Heap using **Heapify** method.
* Step 4 - Put the deleted element into the Sorted list.
* Step 5 - Repeat the same until Min Heap becomes empty.
* Step 6 - Display the sorted list.



**Complexity of the Heap Sort Algorithm**

To sort an unsorted list with 'n' number of elements, following are the complexities...

Worst Case : O(n log n)

Best Case : O(n log n)

Average Case : O(n log n)

## What is Searching?

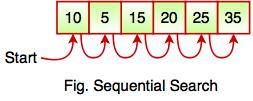
* Searching is the process of finding a given value position in a list of values.
* It decides whether a search key is present in the data or not.
* It is the algorithmic process of finding a particular item in a collection of items.
* It can be done on internal data structure or on external data structure.

## Searching Techniques

**To search an element in a given array, it can be done in following ways:**  
  
1. Sequential Search  
2. Binary Search

#### 1. Sequential Search

* Sequential search is also called as Linear Search.
* Sequential search starts at the beginning of the list and checks every element of the list.
* It is a basic and simple search algorithm.
* Sequential search compares the element with all the other elements given in the list. If the element is matched, it returns the value index, else it returns -1.

  
  
The above figure shows how sequential search works. It searches an element or value from an array till the desired element or value is not found. If we search the element 25, it will go step by step in a sequence order. It searches in a sequence order. Sequential search is applied on the unsorted or unordered list when there are fewer elements in a list.  
  
**The following code snippet shows the sequential search operation:**

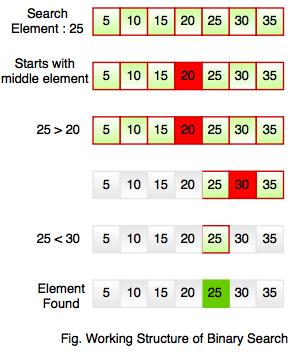
function searchValue(value, target)  
{  
      for (var i = 0; i < value.length; i++)  
      {  
             if (value[i] == target)  
             {  
                     return i;  
             }  
      }  
      return -1;  
}  
searchValue([10, 5, 15, 20, 25, 35] , 25);   // Call the function with array and number to be searched

#### Binary Search

* Binary Search is used for searching an element in a sorted array.
* It is a fast search algorithm with run-time complexity of O(log n).
* Binary search works on the principle of divide and conquer.
* This searching technique looks for a particular element by comparing the middle most element of the collection.
* It is useful when there are large number of elements in an array.

binary array

* The above array is sorted in ascending order. As we know binary search is applied on sorted lists only for fast searching.

**For example,** if searching an element 25 in the 7-element array, following figure shows how binary search works:  
  
  
  
Binary searching starts with middle element. If the element is equal to the element that we are searching then return true. If the element is less than then move to the right of the list or if the element is greater than then move to the left of the list. Repeat this, till you find an element



# Analysis of different sorting techniques

In this article, we will discuss important properties of different sorting techniques including their complexity, stability and memory constraints. Before understanding this article, you should understand basics of different sorting techniques (See : [Sorting Techniques](https://www.geeksforgeeks.org/fundamentals-of-algorithms/#SearchingandSorting)).

**Time complexity Analysis –**  
We have discussed the best, average and worst case complexity of different sorting techniques with possible scenarios.

**Comparison based sorting –**  
In comparison based sorting, elements of an array are compared with each other to find the sorted array.

* **Bubble sort and Insertion sort –**  
  Average and worst case time complexity: n^2  
  Best case time complexity: n when array is already sorted.  
  Worst case: when the array is reverse sorted.
* **Selection sort –**  
  Best, average and worst case time complexity: n^2 which is independent of distribution of data.
* **Merge sort –**  
  Best, average and worst case time complexity: nlogn which is independent of distribution of data.
* **Heap sort –**  
  Best, average and worst case time complexity: nlogn which is independent of distribution of data.
* **Quick sort –**  
  It is a divide and conquer approach with recurrence relation:
* T(n) = T(k) + T(n-k-1) + cn

Worst case: when the array is sorted or reverse sorted, the partition algorithm divides the array in two subarrays with 0 and n-1 elements. Therefore,

T(n) = T(0) + T(n-1) + cn

Solving this we get, T(n) = O(n^2)

Best case and Average case: On an average, the partition algorithm divides the array in two subarrays with equal size. Therefore,

T(n) = 2T(n/2) + cn

Solving this we get, T(n) = O(nlogn)

**Non-comparison based sorting –**  
In non-comparison based sorting, elements of array are not compared with each other to find the sorted array.

* **Radix sort –**  
  Best, average and worst case time complexity: nk where k is the maximum number of digits in elements of array.
* **Count sort –**  
  Best, average and worst case time complexity: n+k where k is the size of count array.
* **Bucket sort –**  
  Best and average time complexity: n+k where k is the number of buckets.  
  Worst case time complexity: n^2 if all elements belong to same bucket.