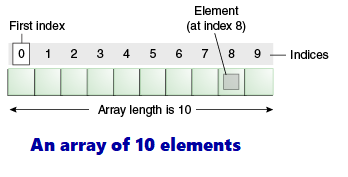
Array

* An array is a container object that holds a fixed number of values of a single type.
* The length of an array is established when the array is created. After creation, its length is fixed.
* As we know Array is a data structure where we store similar elements and Array a starts from index 0.
* Each item in an array is called an element, and each element is accessed by its numerical index.
* Since arrays are objects in Java, we can find their length using member length.
* A Java array variable can also be declared like other variables with [] after the data type.
* The variables in the array are ordered and each has an index beginning from 0.
* Java array can be also be used as a static field, a local variable or a method parameter.
* The size of an array must be specified by an int value and not long or short.

**[](https://3.bp.blogspot.com/-9f26wMChr3A/W3GkEF8RJ9I/AAAAAAAADGQ/IGnhpWIeuYgtjDEcTHrhY3Thh2iiOU2BACLcBGAs/s1600/objects-tenElementArray.png)**

### Stack Data Structure

**What is a Stack?**

A stack is an ordered list in which insertion and deletion are done at one end, called a top. The last element inserted is the first one to be deleted. Hence, it is called the **Last in First out (LIFO) or First in Last out (FILO)** list.

### Stack Concepts

### **When an element is inserted in a stack, the concept is called a**push**.**

### **When an element is removed from the stack, the concept is called**pop**.**

### **Trying to pop out an empty stack is called**underflow **(treat as Exception).**

### **Trying to push an element in a full stack is called**overflow **(treat as Exception).**

### Main stack operations

#### **Push Operation**

The process of putting a new data element onto the stack is known as a *Push*Operation.

Push operation involves a series of steps −

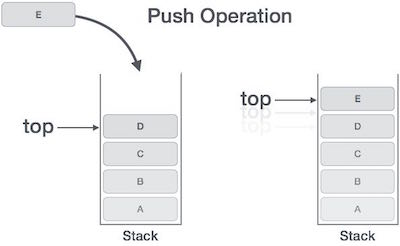
Step 1 − Checks if the stack is full.

Step 2 − If the stack is full, produces an error and exit.

Step 3 − If the stack is not full, increments top to a point next empty space.

Step 4 − Adds data element to the stack location, where the top is pointing.

Step 5 − Returns success.

**[](https://2.bp.blogspot.com/-FyrsJQMkg60/W4-fcoQ3U5I/AAAAAAAADjI/L7vj0_LaOswwKeKR0Rckm1KapoD8gXQiACLcBGAs/s1600/stack_push_operation.jpg)**

If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.

#### **Pop Operation**

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

A Pop operation may involve the following steps −

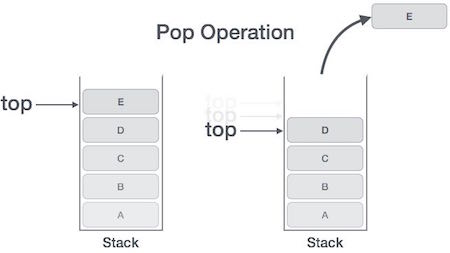
Step 1 − Checks if the stack is empty.

Step 2 − If the stack is empty, produces an error and exit.

Step 3 − If the stack is not empty, accesses the data element at which top is pointing.

Step 4 − Decreases the value of top by 1.

Step 5 − Returns success.

**[](https://4.bp.blogspot.com/-WgJI9z02W1Y/W4-gAr7A5qI/AAAAAAAADjU/dej4TpJ4zXA1MVxlLH6Yjnc_1vOfw8OaQCLcBGAs/s1600/stack_pop_operation.jpg)**

### **Auxiliary stack operations**

### ***int top()*: Returns the last inserted element without removing it.**

### ***int size()*: Returns the number of elements stored in the stack.**

### ***int isEmptyStack()*: Indicates whether any elements are stored in the stack or not.**

### ***int isFullStack()*: Indicates whether the stack is full or not.**

### Queue Data Structure

## What is a Queue?

A queue is an ordered list in which insertions are done at one end (rear) and deletions are done at another end (front). The first element to be inserted is the first one to be deleted. Hence, it is called **First in First out (FIFO) or Last in Last out (LILO)** list.

## Queue Concepts

* When an element is inserted in a queue, the concept is called EnQueue.
* When an element is removed from the queue, the concept is called DeQueue.
* DeQueueing an empty queue is called underflow (treat as Exception).
* *EnQueuing*an element in a full queue is called overflow (treat as Exception).

## Queue Operations

Let's list out basic operations associated with queues −

* *enqueue()* − add (store) an item to the queue.
* *dequeue()* − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are −

* *peek()* − Gets the element at the front of the queue without removing it.
* *isFull()* − Checks if the queue is full.
* *isEmpty()*− Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by a front pointer and while enqueuing (or storing) data in the queue we take help of rear pointer.

### **Enqueue Operation**

Queues maintain two data pointers, *front*, and *rear*. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

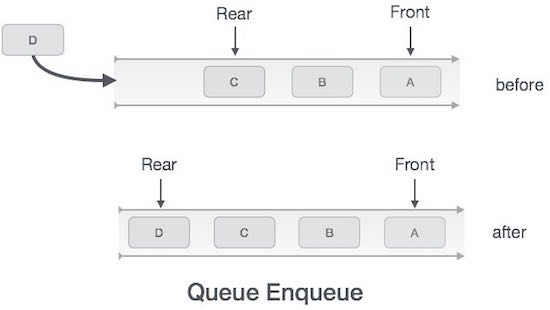
Step 1 − Check if the queue is full.

Step 2 − If the queue is full, produce overflow error and exit.

Step 3 − If the queue is not full, increment a rear pointer to point the next empty space.

Step 4 − Add data element to the queue location, where the rear is pointing.

Step 5 − return success.

**[](https://1.bp.blogspot.com/-oGqe8JYlPhI/W4-3me9SNQI/AAAAAAAADjo/Kr4AGdWGtn8MNxrMcvELnBZ1llrqR3uggCLcBGAs/s1600/queue_enqueue_diagram.jpg)**

### **Dequeue Operation**

Accessing data from the queue is a process of two tasks − access the data where the *front*is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

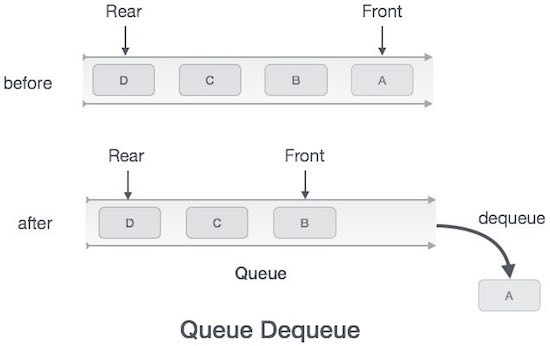
Step 1 − Check if the queue is empty.

Step 2 − If the queue is empty, produce underflow error and exit.

Step 3 − If the queue is not empty, access the data where the front is pointing.

Step 4 − Increment front pointer to point to the next available data element.

Step 5 − Return success.

**[](https://3.bp.blogspot.com/-EcnYz6iTj0s/W4-3vU08lbI/AAAAAAAADjs/32VRIi4SxdYgbjo4O0WncfcfEj3-p--9wCLcBGAs/s1600/queue_dequeue_diagram.jpg)**

### Linked List Data Structure

A linked list is a data structure used for storing collections of data. A linked list has the following properties.

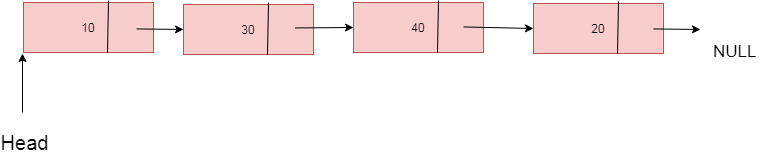
* Successive elements are connected by pointers
* The last element points to *NULL*
* Can grow or shrink in size during execution of a program
* Can be made just as long as required (until systems memory exhausts)
* Does not waste memory space (but takes some extra memory for pointers). It allocates memory as the list grows.

### **Nodes and Node Chaining**

Each link in the chain is called a node, and a node contains two things. First, it contains a single value. This is the thing being stored in your container. Second, it contains a reference that refers to the next item in the list. This pattern of value and reference to the next node is repeated over and over again through the entirety of the list. Nodes can have a pretty straightforward implementation

**Linked List Representation**

A linked list can be visualized as a chain of nodes, where every node points to the next node.

**[](https://2.bp.blogspot.com/-2mz-SofblO8/W5xz9Md9lxI/AAAAAAAADzo/x30u8OVFt3kTotuuCP1-B5IM72bR0RLCwCLcBGAs/s1600/linkedlist.png)**

* Linked List contains a link element called *first*.
* Each link carries a data field(s) and a link field called *next*.
* Each link is linked with its next link using its next link.
* The last link carries a link as null to mark the end of the list.

**Basic Operations**

Following are the basic operations supported by a list.

1. *Insertion* − Adds an element at the beginning of the list.
2. *Deletion* − Deletes an element at the beginning of the list.
3. *Display* − Displays the complete list.
4. *Search* − Searches an element using the given key.
5. *Delete* − Deletes an element using the given key.

**Types of Linked List**

Following are the various types of linked list.

1. Singly Linked List - This list consists of a number of nodes in which each node has a next pointer to the following element. The link of the last node in the list is NULL, which indicates the end of the list.
2. Doubly Linked List - Items can be navigated forward and backward.
3. Circular Linked List - Last item contains link of the first element as next and the first element has a link to the last element as previous.

[Binary Tree](https://www.mygreatlearning.com/academy/learn-for-free/courses/binary-trees?gl_blog_id=17069)

* Hierarchical  Data Structure
* Topmost element is known as the root of the tree
* Every Node can have at most 2 children in the binary tree
* Can access elements randomly using index
* Eg: File system hierarchy
* Common traversal methods:
  + preorder(root) : print-left-right
  + postorder(root) : left-right-print
  + inorder(root) : left-print-right

Binary Search Tree

* Binary tree with the additional restriction
* Restriction:
  + The left child must always be less than the root node
  + The right child must always be greater than the root node
* Insertion, Deletion, Search is much more efficient than a binary tree

### Binary Search Algorithm

## What is Searching?

In computer science, searching is the process of finding an item with specified properties from a collection of items. The items may be stored as records in a database, simple data elements in arrays, text in files, nodes in trees, vertices and edges in graphs, or they may be elements of other search spaces.

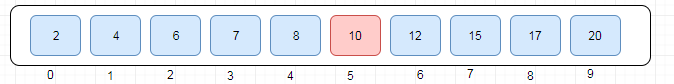
## Why do we need Searching?

*Searching*is one of the core computer science algorithms. We know that today’s computers store a lot of information. To retrieve this information proficiently we need very efficient searching algorithms.

## How Binary Search Works?

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example.

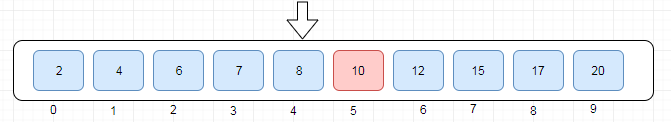
The following is our sorted array and let us assume that we need to search the location of value 10 using binary search.

**[](https://1.bp.blogspot.com/-SeQrYsk-0WA/W5Dc2sqRcsI/AAAAAAAADko/FxtBpBEH4qYdeLor3voBVX3D3s4CNJrVACLcBGAs/s1600/binary_search_1.PNG)**

First, we shall determine half of the array by using this formula −

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0 ) / 2 = 4 (integer value of 4.5). So, 4 is mid of the array.

**[](https://4.bp.blogspot.com/-k1fxzgQHMs4/W5Dc9kLsIfI/AAAAAAAADks/3eaDejFPqJ885ZVmstVCT0akKRjJeF3XgCLcBGAs/s1600/binary_search_2.PNG)**

Now we compare the value stored at location 4, with the value being searched, i.e. 10. We find that the value at location 4 is 8, which is not a match. As the value is greater than 8 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

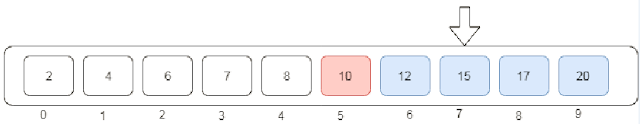
**[](https://3.bp.blogspot.com/-q__Ah0cZtXM/W5DdI8UogMI/AAAAAAAADk0/p6v0_r0tTTAO47eF2GrDYR_UaM5A3bnUACLcBGAs/s1600/binary_search_3.PNG)**

We change our low to mid + 1 and find the new mid value again.

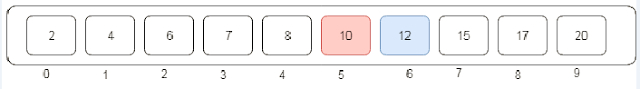
low = mid + 1

mid = low + (high - low) / 2

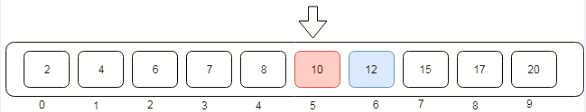
Our new mid is 7 now. We compare the value stored at location 7 with our target value 10.

**[](https://2.bp.blogspot.com/-GFEcpoPUs8I/W5DdOhEtr_I/AAAAAAAADk4/TjVWzQqgOUcm_-zoeshtnTGOCeGLDULqwCLcBGAs/s1600/binary_search_4.PNG)**

The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.

**[](https://4.bp.blogspot.com/-noNc1aYjtdI/W5DdToU3leI/AAAAAAAADk8/TVhYDR7_fHElKVsZnSfIkPASXTuBwrJXACLcBGAs/s1600/binary_search_5.PNG)**

Hence, we calculate the mid again. This time it is index 5.

**[](https://1.bp.blogspot.com/-4aJah1ZooyM/W5DdXnUUZ-I/AAAAAAAADlA/z1Lqxz0MzD4I3m6o0AyKTyG-pOTJAMi8wCLcBGAs/s1600/binary_search_6.PNG)**

We compare the value stored at location 5 with our target value. We find that it is a match.

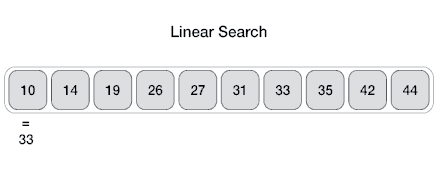
**[](https://3.bp.blogspot.com/-WZ_LZ4Hwt_A/W5De86oHBNI/AAAAAAAADlY/hH0Rc-dVap44iWpsqG1b49HgYA_kp0SgwCLcBGAs/s1600/binary_search_7.PNG)**

I conclude that the target value 10 is stored at location 5.

### Linear Search Algorithm

## What is Linear Search?

In computer science, linear search or sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

**[](https://4.bp.blogspot.com/-Oe596xjTKlM/W5DgdquPiZI/AAAAAAAADlk/dGruMjP_JLI-gdXo2DRxVJeYbvirr8ZOACLcBGAs/s1600/linear_search.gif)**

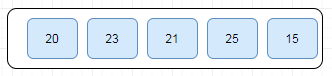
### **DS Sorting Algorithms**

### Bubble Sort Algorithm

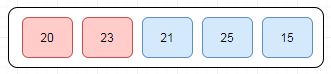
Bubble sort is the simplest sorting algorithm. It works by iterating the input array from the first element to the last, comparing each pair of elements and swapping them if needed. Bubble sort continues its iterations until no more swaps are needed. This algorithm is not suitable for large datasets as its average and worst case complexity is of*Ο(n2)* where n is the number of items.

## How Bubble Sort Works?

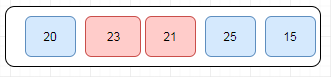
We take an unsorted array for our example. Bubble sort takes Ο(n2) time so we're keeping it short and precise.

**[](https://2.bp.blogspot.com/-oXxPk-y9-UI/W5DvuSRosSI/AAAAAAAADms/TDxs93WbUAUZyQMmmXUbYpSzvYLM6sKcwCLcBGAs/s1600/bubble-sort-1.PNG)**

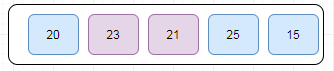
Bubble sort starts with very first two elements, comparing them to check which one is greater.

**[](https://2.bp.blogspot.com/-JUm-vXMlOn8/W5DvxpuyWLI/AAAAAAAADmw/6xTIBLcjT2cQDJXQvOyiUplPUXoz9n8vgCLcBGAs/s1600/bubble-sort-2.PNG)**

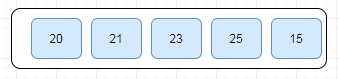
In this case, value 23 is greater than 20, so it is already in sorted locations. Next, we compare 23 with 21.

**[](https://2.bp.blogspot.com/-J5yuPLqVTZQ/W5DvJZNlPoI/AAAAAAAADmQ/ylL3VeVg1VwXlhzoJruxQIg28V-TP1BlgCLcBGAs/s1600/bubble-sort-3.PNG)**

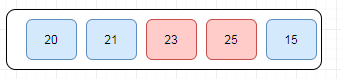
We find that 21 is smaller than 23 and these two values must be swapped.

**[](https://1.bp.blogspot.com/-12aJNwEi3ms/W5DvRjarCDI/AAAAAAAADmY/jefLR5s6uCoV-ntNYXBkl0ucsCfTQn8swCLcBGAs/s1600/bubble-sort-4.PNG)**

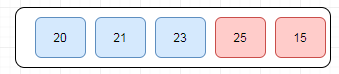
The new array should look like this −

**[](https://3.bp.blogspot.com/-bdPwttGw_lQ/W5Dv1XboMvI/AAAAAAAADm0/VZ5PznYe5f09Z8qV6Hj2W-RZyiq-i5n9ACLcBGAs/s1600/bubble-sort-5.PNG)**

Next, we compare the 23 and 25. We find that both are in already sorted positions.

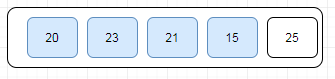
**[](https://4.bp.blogspot.com/-YUuAgTBmkHo/W5DxYbrtoHI/AAAAAAAADnQ/jmckve7QJZc8F4oYZl3nlsFhUgUp4OSEQCLcBGAs/s1600/bubble-sort-6.PNG)**

Then we move to the next two values, 25 and 15 -

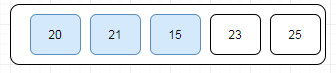
**[](https://1.bp.blogspot.com/-2N4ZSKRkqWo/W5DxtBzqAXI/AAAAAAAADnY/WzXW0gFNfKQmuV6SRYxw9BFQL5yRfKemQCLcBGAs/s1600/bubble-sort-7.PNG)**

We know then that 15 is smaller 25 and these values are not sorted. Let's swap these values and we find that we have reached the end of the array.

After the **first iteration**, the array should look like this −

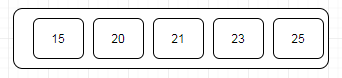
**[](https://1.bp.blogspot.com/-jOFtfk4BeA0/W5DyNHoHW9I/AAAAAAAADng/BAhOGnijTLwvoJ6jxYZI6hhVfK_32-_zQCLcBGAs/s1600/bubble-sort-9.PNG)**

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

**[](https://2.bp.blogspot.com/-YT5wjjaoeTw/W5DypqAgf4I/AAAAAAAADns/mXMSU1bfWDoKNPCb5M2AkCp8fKzX3FfiACLcBGAs/s1600/bubble-sort-10.PNG)**

Notice that after each iteration, at least one value moves at the end.

And when there's no swap required, bubble sorts learns that an array is completely sorted.

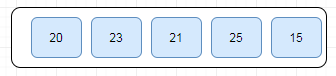
**[](https://4.bp.blogspot.com/-rOMTBCSi_bE/W5DzC0UUJpI/AAAAAAAADn4/ypIlx0Gblx8dgdMPuA_ITHNzw0OblbPTgCLcBGAs/s1600/bubble-sort-12.PNG)**

### Selection Sort Algorithm

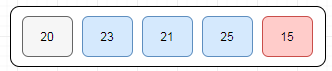
Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end.

## How Selection Sort Works?

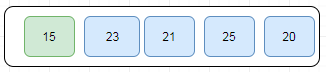
Consider the following depicted integer array as an example.

**[](https://3.bp.blogspot.com/-7OAm9AtOziY/W5EHVPt6uzI/AAAAAAAADoE/lieRPQjlVGcYpC2pHrzjWOdl2fg-vCZSwCLcBGAs/s1600/selection-sort-1.PNG)**

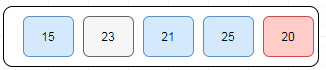
For the first position in the sorted list, the whole list is scanned sequentially. The first position where 20 is stored presently, we search the whole list and find that 15 is the lowest value.

**[](https://3.bp.blogspot.com/-RipWIYYV5_c/W5EHjNSGviI/AAAAAAAADoI/R4096MDYD6IWAf17ssYS7v0Gfqwp_ol1QCLcBGAs/s1600/selection-sort-2.PNG)**

So we replace 20 with 15. After one iteration 15, which happens to be the minimum value in the list, appears in the first position of the sorted list.

**[](https://3.bp.blogspot.com/-D387SxFFbPo/W5EHrxs6kEI/AAAAAAAADoQ/s4teKFEXD8w_U4CB6rCTPQt-3PBynMgOACLcBGAs/s1600/selection-sort-3.PNG)**

For the second position, where 23 is residing, we start scanning the rest of the list in a linear manner.

**[](https://1.bp.blogspot.com/-sVZULxMMD-Q/W5EH-QQbRDI/AAAAAAAADoc/GvnCas2B51U_j3Iop9_xwoEJigrjLPYhgCLcBGAs/s1600/selection-sort-4.PNG)**

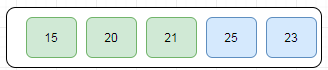
We find that 20 is the second lowest value in the list and it should appear in the second place. We swap these values.

**[](https://3.bp.blogspot.com/-ZcSDU_1av3E/W5EIKDGQSOI/AAAAAAAADog/rb33zzkm7Yccfz227XbtNsCIAK0JA6xfACLcBGAs/s1600/selection-sort-5.PNG)**

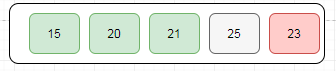
After two iterations, the two least values are positioned at the beginning in a sorted manner.

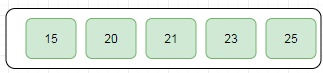
**[](https://2.bp.blogspot.com/-daGXxIHp80Y/W5EIZq3oojI/AAAAAAAADok/gtm_BpuqbZ8UYGL_rSOaH3kbuwnkChbZwCLcBGAs/s1600/selection-sort-6.PNG)**

Now, scan the rest of the list in a linear manner and check third smallest value in the list 21 and it stayed in the same position.

**[](https://1.bp.blogspot.com/-P4HGPmoSYRk/W5EIkE1VvfI/AAAAAAAADos/B1l9sOPe3OUjvRIt8g5VnOGbko91R1wVQCLcBGAs/s1600/selection-sort-7.PNG)**

Let's scan the rest of the list in a linear manner and find fourth smallest value in the list that 23 and replace 25 with 23.

**[](https://3.bp.blogspot.com/-h3CZJrr8Zcw/W5EI6TfxocI/AAAAAAAADo4/Csg_eMigIzsl6ARTBt8f0q9XzaDB6NfpgCLcBGAs/s1600/selection-sort-8.PNG)**

This is an end of an array, after sorting array the final output looks as below:**[](https://2.bp.blogspot.com/-ZQqG0tAofgI/W5EJD9qtZ0I/AAAAAAAADo8/39jmBy7rywYdijfY32RtTqxg9j-IsyP2gCLcBGAs/s1600/selection-sort-9.PNG)**

### Insertion Sort Algorithm

## What is Insertion Sort Algorithm?

Insertion sort is a simple and efficient comparison sort. In this algorithm, each iteration removes an element from the input data and inserts it into the correct position in the list is sorted. The choice of the element is removed from the input is random and this process is repeated until all input elements have gone through.

### How Insertion Sort Works?

Let's understand how insertion sort works with a simple example.

Step 1: Let's we have input as integer array:

**12**, 11, 13, 5, 6

Let us loop for i = 1 (second element of the array) to 5 (Size of input array)

Step 2: For i = 1, since 11 is smaller than 12, move 12 and insert 11 before 12

**11, 12**, 13, 5, 6

Step 3: For i = 2, 13 will remain at its position as all elements in A[0..i-1] are smaller than 13

**11, 12, 13**, 5, 6

Step 4: For i = 3, 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

5, 11, 12, 13, 6

Step 5: For i = 4, 6 will move to a position after 5, and elements from 11 to 13 will move one position ahead of their current position.

**5, 6, 11, 12, 13**

### Merge Sort Algorithm

Merge sort is an example of the divide and conquer strategy. With worst-case time complexity being *Ο(n log n)*, it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

## How Merge Sort Works?

To understand merge sort, we take an unsorted array as the following −

**[https://3.bp.blogspot.com/-pELmhxO8JgU/W5N7wVnYnUI/AAAAAAAADqc/CZTxDL-YPlACuTrM8CQKqHZg5MaCf9w-gCLcBGAs/s1600/merge-sort-1.PNG](https://3.bp.blogspot.com/-pELmhxO8JgU/W5N7wVnYnUI/AAAAAAAADqc/CZTxDL-YPlACuTrM8CQKqHZg5MaCf9w-gCLcBGAs/s1600/merge-sort-1.PNG)**

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

**[https://3.bp.blogspot.com/-oKhWdoxZIfI/W5N70rws3oI/AAAAAAAADqg/xRh8XRtYJ3AFvaynueuHAdLWfsaofRezQCLcBGAs/s1600/merge-sort-2.PNG](https://3.bp.blogspot.com/-oKhWdoxZIfI/W5N70rws3oI/AAAAAAAADqg/xRh8XRtYJ3AFvaynueuHAdLWfsaofRezQCLcBGAs/s1600/merge-sort-2.PNG)**

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

**[https://1.bp.blogspot.com/-4S6o3fKvouw/W5N745VMtyI/AAAAAAAADqk/chSffs5an0gmEZlSJl1tYWYvEE-56YfOgCLcBGAs/s1600/merge-sort-3.PNG](https://1.bp.blogspot.com/-4S6o3fKvouw/W5N745VMtyI/AAAAAAAADqk/chSffs5an0gmEZlSJl1tYWYvEE-56YfOgCLcBGAs/s1600/merge-sort-3.PNG)**

We further divide these arrays and we achieve an atomic value which can no more be divided.

**[https://1.bp.blogspot.com/-G8nCRXQnWUA/W5N7-DJg1PI/AAAAAAAADqo/W1fPByk0u9QMZF-CWVWEjoJp-thtyF3tgCLcBGAs/s1600/merge-sort-4.PNG](https://1.bp.blogspot.com/-G8nCRXQnWUA/W5N7-DJg1PI/AAAAAAAADqo/W1fPByk0u9QMZF-CWVWEjoJp-thtyF3tgCLcBGAs/s1600/merge-sort-4.PNG)**

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 20 and 23 are in sorted positions. We compare 25 and 21 and in the target list of 2 values, we put 21 first, followed by 25. Similarly, compare 30 and 10 these are not sorted so let's sort it and compare 65 and 15 these are also not sorted so sort it out. The final outcome is:

**[https://2.bp.blogspot.com/-_K-AvnVGFsA/W5N8LN2fCOI/AAAAAAAADq0/pKZaf8aQWzMl7H3KalYi2jox01LguKDvQCLcBGAs/s1600/merge-sort-5.PNG](https://2.bp.blogspot.com/-_K-AvnVGFsA/W5N8LN2fCOI/AAAAAAAADq0/pKZaf8aQWzMl7H3KalYi2jox01LguKDvQCLcBGAs/s1600/merge-sort-5.PNG)**

In the next iteration of the combining phase, we compare lists of two data values and merge them into a list of found data values placing all in a sorted order. Note that the items should be sorted while combining these halves.

**[https://4.bp.blogspot.com/-8NypUwp3NxU/W5N8RHigL3I/AAAAAAAADq8/JMr1TIU3mEMy2GEyuwScWXe9KZVq4qBJQCLcBGAs/s1600/merge-sort-7.PNG](https://4.bp.blogspot.com/-8NypUwp3NxU/W5N8RHigL3I/AAAAAAAADq8/JMr1TIU3mEMy2GEyuwScWXe9KZVq4qBJQCLcBGAs/s1600/merge-sort-7.PNG)**

After the final merging, the list should look like this −

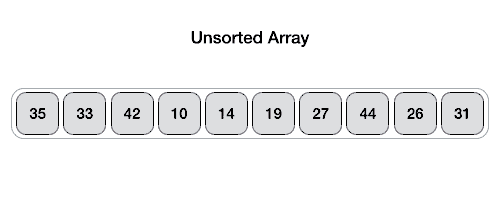
### 10 15 20 21 23 25 30 65

### Quick Sort Algorithm

Quick Sort is an example of a divide-and-conquer algorithmic technique. It is also called partition exchange sort. It uses recursive calls for sorting the elements, and it is one of the famous algorithms among comparison-based sorting algorithms.

## Partition in Quick Sort

Following animated representation explains how to find the pivot value in an array.

**[](https://3.bp.blogspot.com/-6e1hp4vtW2A/W5OGy25HrgI/AAAAAAAADrU/aaM5W--ufugeDxMvLW3iGCIS5HjXkDcDwCLcBGAs/s1600/quick_sort_partition_animation.gif)**

The pivot value divides the list into two parts. And recursively, we find the pivot for each sub-lists until all lists contain only one element.

**Algorithm**

The recursive algorithm consists of four steps:

1. If there are one or no elements in the array to be sorted, return.
2. Pick an element in the array to serve as the “pivot” point. (Usually, the left-most element in the array is used.)
3. Split the array into two parts – one with elements larger than the pivot and the other with elements smaller than the pivot.
4. Recursively repeat the algorithm for both halves of the original array.