**Data Structures**

* **Difference between Array List and Linked List**

Linked List

* Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
* A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
* The last node of the list contains pointer to the null.

DS Linked List

Uses of Linked List

* The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
* list size is limited to the memory size and doesn't need to be declared in advance.
* Empty node can not be present in the linked list.
* We can store values of primitive types or objects in the singly linked list.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Time Complexity** | | | | | | | | **Space Compleity** |
|  | **Average** | | | | **Worst** | | | | **Worst** |
|  | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion |  |
| Singly Linked List | θ(n) | θ(n) | θ(1) | θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |

### Node Creation

1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. };
6. struct node \*head, \*ptr;
7. ptr = (struct node \*)malloc(sizeof(struct node \*));

[**next →**](https://www.javatpoint.com/data-structure-2d-array)[**← prev**](https://www.javatpoint.com/data-structure-structure)

Array List :

* Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
* Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
* Array is the simplest data structure where each data element can be randomly accessed by using its index number.
* For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

The array **marks[10]** defines the marks of the student in 10 different subjects where each subject marks are located at a particular subscript in the array i.e. **marks[0]** denotes the marks in first subject, **marks[1]** denotes the marks in 2nd subject and so on.

Stack

1. Stack is an ordered list in which, insertion and deletion can be performed only at one end that is called **top**.
2. Stack is a recursive data structure having pointer to its top element.
3. Stacks are sometimes called as Last-In-First-Out (LIFO) lists i.e. the element which is inserted first in the stack, will be deleted last from the stack.

Applications of Stack

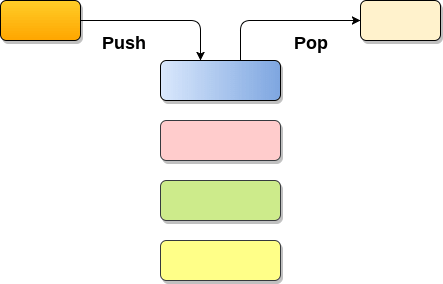
1. Recursion
2. Expression evaluations and conversions
3. Parsing
4. Browsers
5. Editors
6. Tree Traversals

**Algorithm:**

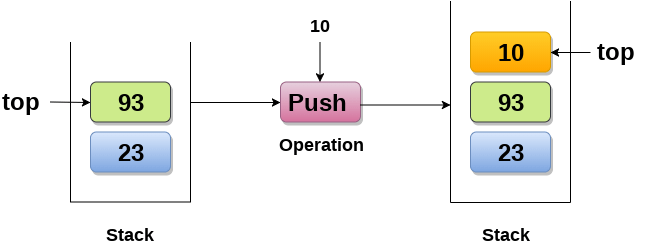
1. begin
2. **if** top = n then stack full
3. top = top + 1
4. stack (top) : = item;
5. end

Operations on Stack

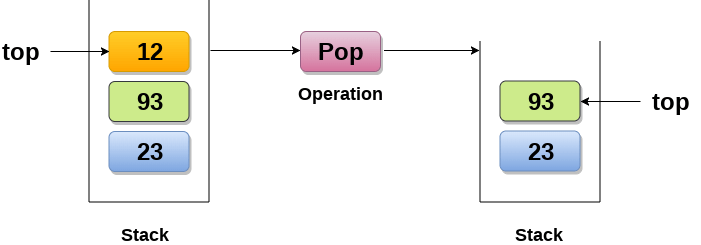
There are various operations which can be performed on stack.



**1. Push :** Adding an element onto the stack



**2. Pop :** Removing an element from the stack



**3. Peek :** Look all the elements of stack without removing them.

# Queue

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



Applications of Queue

Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
3. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
4. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
5. Queues are used in operating systems for handling interrupts.

Algorithm to insert any element in a queue

Check if the queue is already full by comparing rear to max - 1. if so, then return an overflow error.

If the item is to be inserted as the first element in the list, in that case set the value of front and rear to 0 and insert the element at the rear end.

Otherwise keep increasing the value of rear and insert each element one by one having rear as the index.

Algorithm

* **Step 1:** IF REAR = MAX - 1  
  Write OVERFLOW  
  Go to step  
  [END OF IF]
* **Step 2:** IF FRONT = -1 and REAR = -1  
  SET FRONT = REAR = 0  
  ELSE  
  SET REAR = REAR + 1  
  [END OF IF]
* **Step 3:** Set QUEUE[REAR] = NUM
* **Step 4:** EXIT

Searching

Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful and the process returns the location of that element, otherwise the search is called unsuccessful.

There are two popular search methods that are widely used in order to search some item into the list. However, choice of the algorithm depends upon the arrangement of the list.

* Linear Search
* Binary Search

Linear Search

Linear search is the simplest search algorithm and often called sequential search. In this type of searching, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match found then location of the item is returned otherwise the algorithm return NULL.

Linear search is mostly used to search an unordered list in which the items are not sorted. The algorithm of linear search is given as follows.

Algorithm

* LINEAR\_SEARCH(A, N, VAL)
* **Step 1:** [INITIALIZE] SET POS = -1
* **Step 2:** [INITIALIZE] SET I = 1
* **Step 3:** Repeat Step 4 while I<=N
* **Step 4:** IF A[I] = VAL  
  SET POS = I  
  PRINT POS  
  Go to Step 6  
  [END OF IF]  
  SET I = I + 1  
  [END OF LOOP]
* **Step 5:** IF POS = -1  
  PRINT " VALUE IS NOT PRESENTIN THE ARRAY "  
  [END OF IF]
* **Step 6:** EXIT

# Binary Search

Binary search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search technique, we must ensure that the list is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

Binary search algorithm is given below.

[**next →**](https://www.javatpoint.com/bubble-sort)[**← prev**](https://www.javatpoint.com/linear-search)

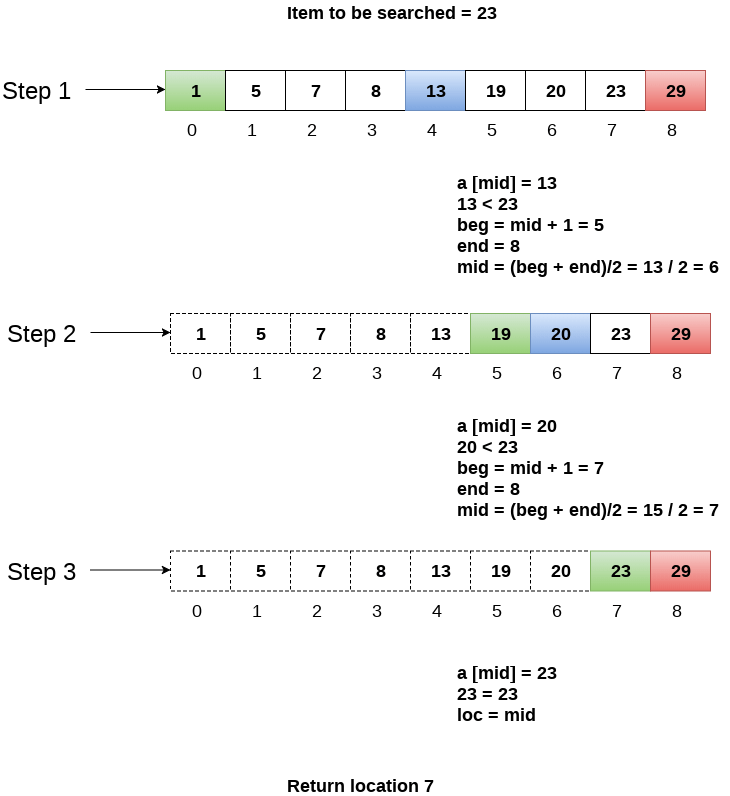
Binary Search

Binary search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search technique, we must ensure that the list is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

Binary search algorithm is given below.

BINARY\_SEARCH(A, lower\_bound, upper\_bound, VAL)

* **Step 1:** [INITIALIZE] SET BEG = lower\_bound  
  END = upper\_bound, POS = - 1
* **Step 2:** Repeat Steps 3 and 4 while BEG <=END
* **Step 3:** SET MID = (BEG + END)/2
* **Step 4:** IF A[MID] = VAL  
  SET POS = MID  
  PRINT POS  
  Go to Step 6  
  ELSE IF A[MID] > VAL  
  SET END = MID - 1  
  ELSE  
  SET BEG = MID + 1  
  [END OF IF]  
  [END OF LOOP]
* **Step 5:** IF POS = -1  
  PRINT "VALUE IS NOT PRESENT IN THE ARRAY"  
  [END OF IF]
* **Step 6:** EXIT
* 

**Factorial of a Number Using Recursion**

1. public class Factorial {
2. public static void main(String[] args) {
3. int num = 6;
4. long factorial = multiplyNumbers(num);
5. System.out.println("Factorial of " + num + " = " + factorial);
6. }
7. public static long multiplyNumbers(int num)
8. {
9. if (num >= 1)
10. return num \* multiplyNumbers(num - 1);
11. else
12. return 1;
13. }
14. }

**What is Recursion?**  
The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder / Preorder/ Postorder Tree Traversals, DFS of Graph, etc.