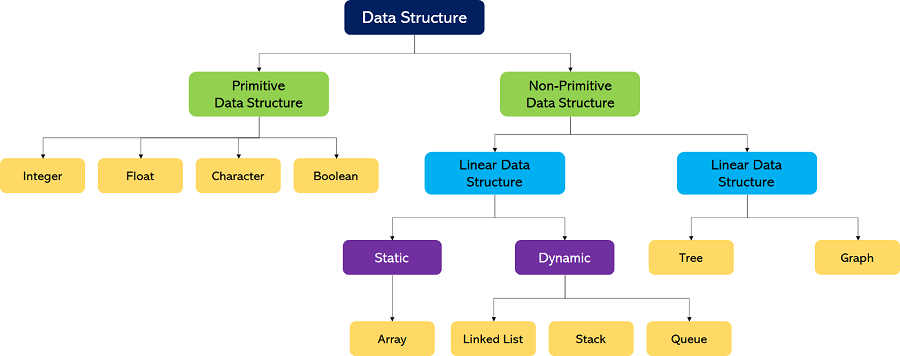
**What is a data structure? What are some common data structures?**

Data Structure is a particular way of storing and organizing data in the memory of the computer so that these data can easily be retrieved and efficiently utilized in the future when required. The data can be managed in various ways, like the logical or mathematical model for a specific organization of data is known as a data structure.

We can classify Data Structures into two categories:

1. Primitive Data Structure
2. Non-Primitive Data Structure

The following figure shows the different classifications of Data Structures.



**What is an algorithm? What are some common algorithm design techniques?**

An algorithm is a process or a set of rules required to perform calculations or some other problem-solving operations especially by a computer. The formal definition of an algorithm is that it contains the finite set of instructions which are being carried in a specific order to perform the specific task. It is not the complete program or code; it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudocode.

The algorithms can be classified in various ways. They are:

1. Implementation Method
2. Design Method
3. Design Approaches
4. Other Classifications

**Classification by Implementation Method:** There are primarily three main categories into which an algorithm can be named in this type of classification. They are:

1. Recursion or Iteration
2. Exact or Approximate
3. Serial or Parallel or Distributed Algorithms

**Classification by Design Method:** There are primarily three main categories into which an algorithm can be named in this type of classification. They are:

1. Greedy Method
2. Divide and Conquer
3. Dynamic Programming
4. Linear Programming
5. Reduction(Transform and Conquer)
6. Backtracking
7. Branch and Bound

**Classification by Design Approaches :** There are two approaches for designing an algorithm. these approaches include

1. Top-Down Approach
2. Bottom-up approach

**Other Classifications:** Apart from classifying the algorithms into the above broad categories, the algorithm can be classified into other broad categories like:

1. Randomized Algorithms
2. **Classification by complexity**
3. Classification by Research Area
4. Branch and Bound Enumeration and Backtracking

**What is the time complexity of an algorithm? How is it calculated?**

Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input. It measures the time taken to execute each statement of code in an algorithm. It is not going to examine the total execution time of an algorithm. Rather, it is going to give information about the variation (increase or decrease) in execution time when the number of operations (increase or decrease) in an algorithm. Yes, as the definition says, the amount of time taken is a function of the length of input only.

**Let us illustrate how to evaluate the time complexity of an algorithm with an example:**

**The algorithm is defined as:**

1. Given 2 input matrix, which is a square matrix with order n

2. The values of each element in both the matrices are selected randomly using np.random function

3. Initially assigned a result matrix with 0 values of order equal to the order of the input matrix

4. Each element of X is multiplied by every element of Y and the resultant value is stored in the result matrix

5. The resulting matrix is then converted to list type

6. For every element in the result list, is added together to give the final answer

Let us assume cost function C as per unit time taken to run a function while ‘n’ represents the number of times the statement is defined to run in an algorithm.

For example, if the time taken to run print function is say 1 microseconds (C) and if the algorithm is defined to run PRINT function for 1000 times (n),

then total run time = (C \* *n) = 1 microsec \** 1000 = 1 millisec

Run time for each line is given by:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Line 1 = C1 \* 1  Line 2 = C2 \* 1  Line 3,4,5 = (C3 \* 1) + (C3 \* 1) + (C3 \* 1)  Line 6,7,8 = (C4\*[n+1]) \* (C4\*[n+1]) \* (C4\*[n+1])  Line 9 = C4\*[n]  Line 10 = C5 \* 1  Line 11 = C2 \* 1  Line 12 = C4\*[n+1]  Line 13 = C4\*[n]  Line 14 = C2 \* 1  Line 15 = C6 \* 1 |

Total run time = (C1\*1) + 3(C2\*1) + 3(C3\*1) + (C4\*[n+1]) \* (C4\*[n+1]) \* (C4\*[n+1]) + (C4\*[n]) + (C5\*1) + (C4\*[n+1]) + (C4\*[n]) + (C6\*1)

Replacing all cost with C to estimate the Order of notation,

Total Run Time

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | = C + 3C + 3C + ([n+1]C \* [n+1]C \* [n+1]C) + nC + C + [n+1]C + nC + C                                 = 7C + ((n^3) C + 3(n^2) C + 3nC + C + 3nC + 3C              = 12C + (n^3) C + 3(n^2) C + 6nC                = C(n^3) + C(n^2) + C(n) + C              = O(n^3) + O(n^2) + O(n) + O (1) |

By replacing all cost functions with C, we can get the degree of input size as 3, which tells the order of time complexity of this algorithm. Here, from the final equation, it is evident that the run time varies with the polynomial function of input size ‘n’ as it relates to the cubic, quadratic and linear forms of input size.

This is how the order is evaluated for any given algorithm and to estimate how it spans out in terms of runtime if the input size is increased or decreased. Also note, for simplicity, all cost values like C1, C2, C3, etc. are replaced with C, to know the order of notation. In real-time, we need to know the value for every C, which can give the exact run time of an algorithm given the input value ‘n’.

**What is the space complexity of an algorithm? How is it calculated?**

The space complexity is the measurement of total space required by an algorithm to execute properly. It also includes memory required by input variables. Basically, it's the sum of auxiliary space and the memory used by input variables.

Note: Space complexity == Auxiliary space ++ Memory used by input variables

Space complexity is the parallel concept to [time complexity](https://www.educative.io/answers/time-complexity-vs-space-complexity). For example, if we want to initialize an array of *n* integers, then its space complexity would be *O*(*n*).

**Calculate space complexity**

include <iostream>

using namespace std;

int sum(int a , int b)   *// Inter a and b*

{

  return a + b;        *// returning sum is integer too*

}

int main() {

*// your code goes here*

  int x = 10;

  int y = 20;

  cout << "Sum of a & b =" << sum(x,y);

  return 0;

}

In this program, there are five variables allocated in the memory:

* The integer variables in the main function x , y .
* The integer variables in the sum function a , b..
* The third integer, returning variable sum.

A single variable uses 44bytes of memory so the total memory for this program is 2020 bytes (5∗4=205∗4=20 bytes). The space complexity is of constant time, it can be expressed in big-O notation as *O*(1).

**What is the difference between an array and a linked list? When would you use one over the other?**

| **Array** | **Linked List** |
| --- | --- |
| An array is a collection of elements of a similar data type. | A Linked list is a group of objects called nodes, which consists of two fields: data and address to the next node |
| An array stores elements in a contiguous memory location. | Linked lists store elements randomly at any address in the memory. |
| In an array, memory size is fixed while declaration and cannot be altered at run time. | Linked lists utilize dynamic memory, i.e. memory size can be altered at run time. |
| Elements in an array are not dependent on each other. | Elements in a linked list are dependent on each other, as each node stores the address of its next node. |
| Operation like insertion, deletion, etc., takes more time in an array. | Operations like insertion, deletion, etc., take less time than arrays. |
| Memory is allocated at compile time. | Memory is allocated at run time. |
| It is easier and faster to access the element in an array with the help of Indices. | Accessing an element in a linked list is time-consuming as we have to start traversing from the first element. |
| Memory utilization is ineffective in arrays. For example, if the array size is 5 and contains only 2 elements, the rest of the space will be wasted. | In linked lists, memory utilization is effective, as it can be allocated or deallocated at the run time according to our requirements. |
| Arrays support multi-dimensional structures. | Linked lists are typically used for one-dimensional structures. |
| Arrays are commonly used in low-level programming and for implementing data structures. | Linked lists are often used for specific data management needs like task scheduling and memory allocation. |

**What is a stack? How is it implemented? What are some common use cases?**

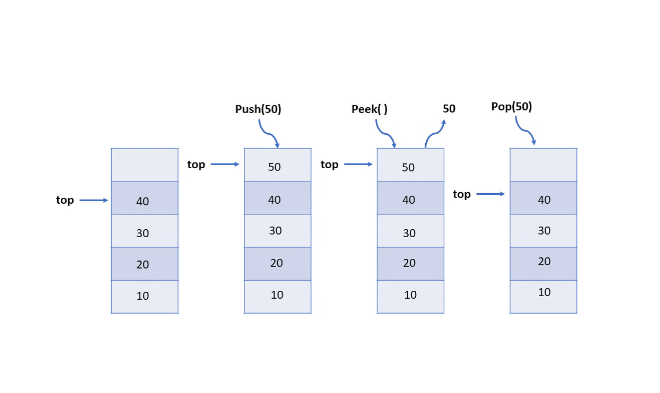
A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**). It contains only one pointer **top pointer** pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a ***stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.***

Now, assume that you have a stack of books.

You can only see the top, i.e., the top-most book, namely 40, which is kept top of the stack.

If you want to insert a new book first, namely 50, you must update the top and then insert a new text.

And if you want to access any other book other than the topmost book that is 40, you first remove the topmost book from the stack, and then the top will point to the next topmost book.



Here are the top 7 applications of the stack in data structure:

* Expression Evaluation and Conversion
* Backtracking
* Function Call
* Parentheses Checking
* String Reversal
* Syntax Parsing
* Memory Management

**What is a queue? How is it implemented? What are some common use cases?**

A queue is a linear data structure that follows the “first-in, first-out” (FIFO) principle. It is a collection of elements that supports two primary operations – enqueue and dequeue. In the enqueue operation, an element is added to the back of the queue, while in the dequeue operation, an element is removed from the front of the queue.