**Data Structures In Python**

**Data structures are code structures for storing and organizing data that make it easier to modify, navigate, and access information.**

## ****Built-in Data Structures****

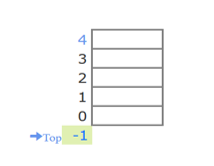
1. List
2. Tuple
3. Dictionary

## ****User-Defined Data Structures****

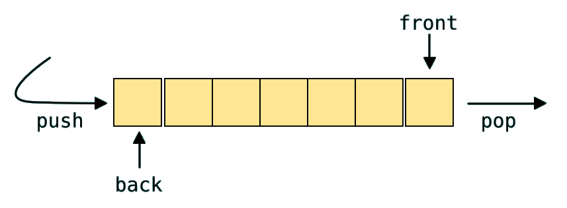
Linear & Non-linear

* Linear : Stack & Queue , linked list
* Non-linear : Tree, graph

## ****Stack :**** Stacks are linear Data Structures which are based on the principle of Last-In-First-Out (LIFO) where data which is entered last will be the first to get accessed.

****

### ****Queue :**** A queue is also a linear data structure which is based on the principle of First-In-First-Out (FIFO) where the data entered first will be accessed first.



### Linked list : A linked list is a sequence of data elements, which are connected together via links.



* Types of Lists:

1. Singly-linked list
2. Circular linked list
3. Doubly linked list

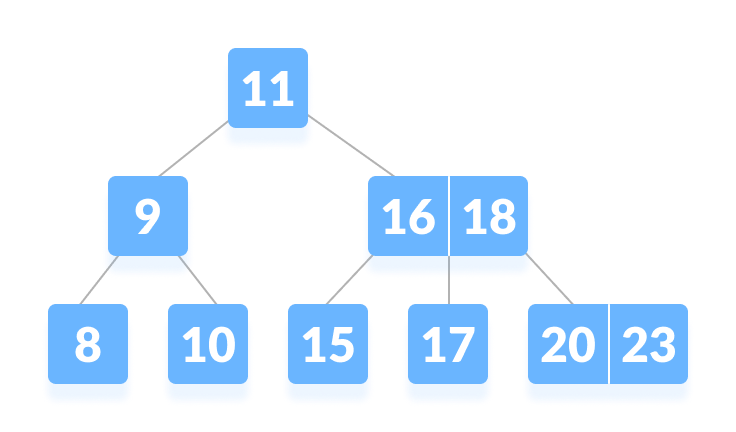
* Tree : A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



A Binary Tree node contains following parts.

1. Data
2. Pointer to left child
3. Pointer to right child

* B Tree
* B+ tree
* Red black tree
* B Tree
* each node can contain more than one key and can have more than two children.
* It is also known as a height-balanced m-way tree.



Properties

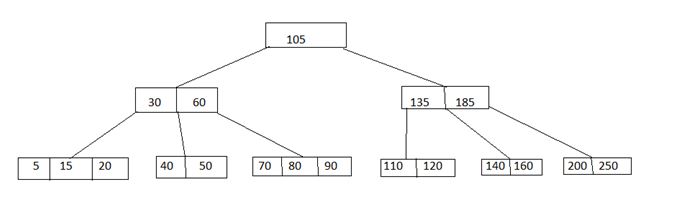
* Every node has max ‘m’ children
* Minimum children : leaf – 0

Root – 2

Internal nodes – [m/2]

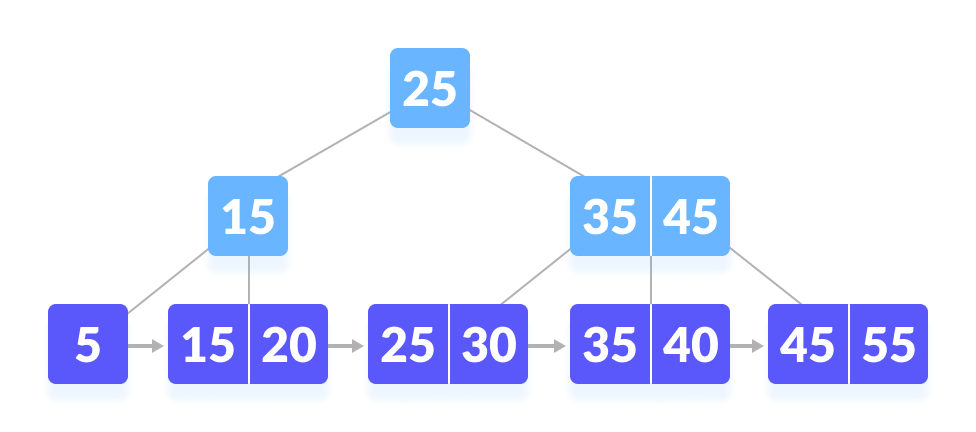
* Every node has max(m-1) keys
* Minimum keys : root node =1

Other nodes = [m/2]-1



* B+ tree

A B+ tree is an advanced form of a self-balancing tree in which all the values are present in the leaf level.



## Properties of a B+ Tree

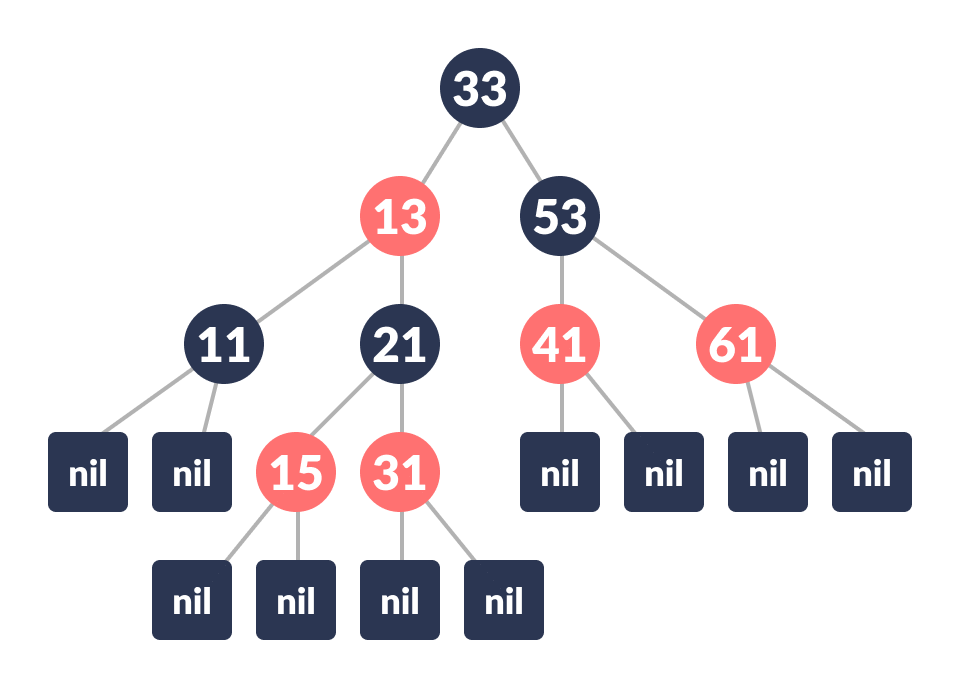
1. All leaves are at the same level.
2. The root has at least two children.
3. Each node except root can have a maximum of m children and at least m/2 children.
4. Each node can contain a maximum of m - 1 keys and a minimum of  ⌈m/2⌉ - 1 keys.

* Red black tree

Red-Black tree is a self-balancing binary search tree in which each node contains an extra bit for denoting the color of the node, either red or black.

Properties:

1. **Red/Black Property:** Every node is colored, either red or black.
2. **Root Property:** The root is black.
3. **Leaf Property:** Every leaf (NIL) is black.
4. **Red Property:** If a red node has children then, the children are always black.
5. **Depth Property:** For each node, any simple path from this node to any of its descendant leaf has the same black-depth (the number of black nodes).



Root=black

No two adjacent red nodes

Count number of black nodes in each path

* If tree is empty, create new node as root node with color black
* If tree is not empty, create new node as leaf node with color red
* If parent of new node is black then exit
* If parent of new node is red then check the color of parent’s sibling of new node:
  1. If color is black or null then do suitable rotation & recolor
  2. If color is red then recolor & also check if parent’s parent of new node is not root node then recolor it & recheck

Sorting

**Selection sort**

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Eg 64 25 12 22 11

**Bubble sort**

**Bubble sort** is a sorting algorithm that compares two adjacent elements and swaps them until they are not in the intended order.

EG: -2 45 0 11 -9

**Merge Sort**

Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.

Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.



**Quicksort**

Quicksort is a sorting algorithm based on the **divide and conquer approach** where

1. An array is divided into subarrays by selecting a **pivot element** (element selected from the array).  
     
   While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.
2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
3. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

