Data structure

It is a way or style of storing data in our program, is called as data structure

1. Array
    a. There is a limit on size of array
    b. Memory is always allocated in consecutive memory locations
    c. Array name always points to the starting location of the array, It is a constant pointer, it means its value cannot be changed
    d. In array data can be accessed by using index position
    e. Because we use index position to access data, data can be accessed randomly
    f. Adding data at the end is faster, than adding data in between
2. Linked List
    a. There is no limit on size
    b. Memory is not allocated consecutively
    c. Data cannot be accessed by using index, and hence data cannot be accessed randomly, if you want to read value at nth position then, you need to read first n-1 values
    d. Adding data in between is faster than adding it at the end.
    e. In linked list we need more memory to store data

3. Tree
    a. If you want to arrange data in parent and child format then we use trees
    b. In trees data cannot be accessed randomly
4. Hash table
    a. It uses hash function to store data in hash table
    b. We cannot find the order in which data is added
    c. Searching is very fast in hashing technique

To perform any task, if we write step by step procedure, then it is called as algorithm.

Every task needs some time to complete the task, this time usually we calculate in terms of size of input, it is called as time complexity

a. Time complexity is represented using big O notation example O(n), O(log n)

To perform any task, we need space to store data. The extra space required to perform the task is considered as space complexity.

Searching techniques

1. Sequential Search
    a. If data is not sorted then we sequential search
    b. Time complexity of sequential search is O(n)

2. Binary search
    a. If the data is in sorted order then we use binary search
    b. Time complexity of binary search is O(log n)

Algorithm for binary search

1. Find mid=(low+high)/2
2. Then check if the number is at mid position, if found then return mid
3. If num<arr[mid] , then search in the array (low,mid-1)
4. Else id num>arr[mid] , then search in the array (mid+1,high)
5. Repeat steps 1 to 4, until low<=high, otherwise return -1

Time complexity --- (log n)

Binary search is faster than sequential search