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

* A data structure is a way of organizing and storing data in a computer to be accessed and used efficiently. It refers to the logical or mathematical representation of data and its implementation in a computer program.

Following are some common data structures:

1. Array
2. Hash Table
3. Linked List
4. Stacks
5. Queues
6. Heaps
7. Graphs
8. Tree
9. **What is an algorithm? What are some common algorithm design techniques?**

* The word [Algorithm](https://www.geeksforgeeks.org/fundamentals-of-algorithms/) means “ A set of finite rules or instructions to be followed in calculations or other problem-solving operations.”

Or

“A procedure for solving a mathematical problem in a finite number of steps frequently involving recursive operations”.

Following are some common algorithm design techniques:

1. Implementation Method

* Recursion or Iteration
* Exact or Approximate
* Serial or Parallel or Distributed Algorithm

1. Design Method

* Greedy Method
* Divide and Conquer
* Dynamic Programming
* Linear Programming
* Back Tracking
* Branch and Bound

1. Design Approaches

* Top-down Approach
* Bottom-up Approach

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

* It is time taken by the algorithm to solve the problem. It is measured by calculating the iteration of loops, number of comparisons, etc. Time complexity is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. “Time” can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real-time the algorithm will take.

Most People use Big O notation to calculate Time Complexity as the binary search almost takes four operations, equivalent to log2 (10), where 10 is the size of the input.

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

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

Most developers use big-O notation because it's relatively easy to estimate the maximum amount of space required.

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

|  |  |
| --- | --- |
| **Array** | **Linked List** |
| It is a collection of elements of similar data types. | It is a collection of entities called nodes. |
| They use contiguous memory locations. | They use non-contiguous memory locations. |
| Works with static memory. | Works with dynamic memory. |
| Random access is possible. | Only sequential access is possible. |
| Insertion and deletion take more time. | Insertion and deletion take less time. |
| Memory allocated at compile time. | Memory allocated at run time. |

We use an array when we need speed for iterating through the list of items. And if we know how many elements will be in the array ahead of time.

We use a linked list when we need constant time for performing insertion or deletion. And if we have no idea how many items will be there in the list.

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

* A stack is an ordered list or we can say a container in which insertion and deletion can be done from the one end known as the top of the stack. The last inserted element is available first and is the first one to be deleted. Hence, it is known as Last In-First Out (LIFO).

Stack can be used in systems where memory is allocated in a static way to manage memory allocation or track which memory blocks are in use. It can also be used to keep track of function calls in a program and ensure that the program returns to the correct location when a function is complete.

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

* A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket. Queue follows the First In-First out (FIFO) rule - the item that goes in first is the item that comes out first.

Queues have numerous applications, including job scheduling, printer spooling, breadth-first search, call center management, CPU task management, and buffering. Queues are essential tools for efficiently managing and organizing data in a way that ensures tasks are executed in the correct order.

1. **What is a binary tree? How is it implemented? What are some common traversal algorithms?**

* A binary tree is a tree-type non-linear data structure with a maximum of two children for each parent. Every node in a binary tree has a left and right reference along with the data element. The node at the top of the hierarchy of a tree is called the root node.

Binary trees are commonly implemented using nodes, where each node contains data and references (pointers) to its left and right children. The implementation can be achieved using various programming languages such as C++, Java, Python, etc.

Common Traversal Algorithms are as follows:-

1. In-order Traversal
2. Pre-Order Traversal
3. Post-Order Traversal
4. **What is a hash table? How is it implemented? What are some common use cases?**

* A hash table, also known as a hash map, is a data structure that stores key-value pairs. It uses a hash function to compute an index (often called a hash code) into an array of buckets or slots, from which the desired value can be found.

Hashing is implemented in two steps:

An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.

The element is stored in the hash table where it can be quickly retrieved using hashed key.

hash = hashfunc(key)

index = hash % array size

In this method, the hash is independent of the array size and it is then reduced to an index (a number between 0 and array size − 1) by using the modulo operator (%).

Following are the common uses of Hash Table:-

1. Hash tables are commonly used to implement many types of in-memory tables. They are used to implement associative arrays (arrays whose indices are arbitrary strings or other complicated objects).
2. Hash tables may also be used as disk-based data structures and database indices (such as in dbm).
3. Hash tables can be used to implement caches i.e. auxiliary data tables that are used to speed up the access to data, which is primarily stored in slower media.
4. Several dynamic languages, such as Perl, Python, JavaScript, and Ruby use hash tables to implement objects.
5. Hash Functions are used in various algorithms to make their computing faster.
6. **What is a graph? How is it implemented? What are some common traversal algorithms?**

* A graph is a non-linear kind of data structure made up of nodes or vertices and edges.

The two most common ways of implementing graphs are using adjacency matrices and using adjacency lists. We tend to prefer adjacency matrices when the graphs are dense, that is when the number of edges is near the maximum possible number, which is n 2 n^2 n2 for a graph of n n n nodes.

There are two traversal algorithms:-

1. Breadth First Search(BFS)

The Breadth First Search (BFS) traversal is an algorithm, which is used to visit all of the nodes of a given graph. In this traversal algorithm, one node is selected and then all of the adjacent nodes are visited one by one. After completing all of the adjacent vertices, it moves further to check another vertex and checks its adjacent vertices again.

1. Depth First Search(DFS)

The Depth First Search (DFS) is a graph traversal algorithm. In this algorithm one starting vertex is given, and when an adjacent vertex is found, it moves to that adjacent vertex first and try to traverse in the same manner.

1. **What is dynamic programming? What are some common problems that can be solved using dynamic programming?**

* Dynamic programming is a technique that breaks the problems into sub-problems and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as the optimal substructure property. The main use of dynamic programming is to solve optimization problems.

The common problems that can be solved using dynamic programming are as follows:-

1. Optimization Problem

* The Longest Common Subsequence
* The max subarray sum problem

1. Counting Problem

* Count ways to make change
* Count all palindromic subsequences in a string

1. **What is recursion? How does it work? What are some common problems that can be solved using recursion?**

* Recursion is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get to a small enough problem that it can be solved trivially. Usually, recursion involves a function calling itself.

The recursion works following certain steps. They are:-

1. Base Case: Every recursive function must have one or more base cases. These are the simplest possible cases that can be solved directly without further recursion. Base cases are crucial because they prevent the function from calling itself indefinitely and causing a stack overflow.
2. Recursive Case: In the recursive case, the function calls itself with modified input parameters to break down the problem into smaller, more manageable subproblems. Each recursive call works towards reaching the base case.
3. Termination Condition: The recursion terminates when the base case is reached. Once the base case is reached, the function starts returning values up the call stack, combining the results of each recursive call to solve the original problem.

The problems that can be solved using recursion are as follows:-

1. Calculate the sum of two numbers.
2. Calculating a factorial of a number.
3. Calculating a Fibonacci sequence.