### 1. What do you mean by Data Structures?

Data structures are specialized formats for organizing, processing, and storing data. They enable efficient access and modification of data and are fundamental in computer science for handling data effectively.

### 2. Explain self-referential structure? Explain its need?

A self-referential structure is a data structure that contains a reference to itself. For example, a linked list node may contain a pointer to another node of the same type. This is essential for creating dynamic data structures like linked lists, trees, and graphs, which can grow and shrink in size.

### 3. List out the areas in which data structures are applied extensively?

* Databases
* Operating Systems
* Compilers
* Networking
* Artificial Intelligence
* Graphics
* Game Development
* File Systems

### 4. Explain the data structures used to perform recursion?

The primary data structure used for recursion is the **call stack**, which keeps track of function calls. Each time a function is called, its context (variables, return address) is pushed onto the stack, and when the function returns, that context is popped off.

### 5. List different types of data structures.

* Arrays
* Linked Lists
* Stacks
* Queues
* Trees (Binary Trees, AVL Trees, etc.)
* Graphs
* Hash Tables

### 6. What is the difference between iterative method and recursive method? Which one is feasible?

* **Iterative Method**: Uses loops to repeat a process; typically more memory-efficient since it doesn't use the call stack.
* **Recursive Method**: A function calls itself to solve a problem; can be more elegant and easier to understand but may use more memory due to call stack overhead.

**Feasibility** depends on the problem. For simple tasks, iteration is often preferable; for complex problems (like traversing trees), recursion can be more intuitive.

### 7. What are the types of Recursion?

* **Direct Recursion**: A function calls itself directly.
* **Indirect Recursion**: A function calls another function that calls the first function.
* **Tail Recursion**: The recursive call is the last operation in the function.
* **Head Recursion**: The recursive call occurs before any other operations.

### 8. How to reverse String in Java using Recursion.

public class ReverseString {

public static String reverse(String str) {

if (str.isEmpty()) {

return str;

}

return reverse(str.substring(1)) + str.charAt(0);

}

public static void main(String[] args) {

String input = "Hello";

System.out.println(reverse(input)); // Output: olleH

}

}

### 9. What is a stack data structure? What are the applications of stack?

A stack is a linear data structure that follows the Last In First Out (LIFO) principle. The last element added is the first to be removed.

**Applications**:

* Expression evaluation (postfix and prefix)
* Backtracking (like maze solving)
* Undo mechanisms in applications
* Function call management in programming languages

### 10. What are different operations available in stack data structure?

* **Push**: Add an element to the top of the stack.
* **Pop**: Remove and return the top element.
* **Peek/Top**: Return the top element without removing it.
* **isEmpty**: Check if the stack is empty.

### 11. Explain tower of hanoi?

The Tower of Hanoi is a mathematical puzzle involving three rods and a number of disks of different sizes. The objective is to move all the disks from the source rod to the destination rod following these rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or an empty rod.
3. No larger disk may be placed on top of a smaller disk.

Show with implementation code & recursion tree.\*\*\*\*\*dry run code too\*\*\*

### 12. Which data structure is ideal to perform Recursion operation and why?

The **stack** is ideal for recursion because it manages the function calls automatically. Each function call's context is stored on the stack until the function completes, making it easy to return to the previous state.

### 13. Briefly write in brief the approaches to develop algorithms.

* **Brute Force**: Solve the problem through exhaustive search.
* **Divide and Conquer**: Break the problem into smaller subproblems, solve them independently, and combine results.
* **Dynamic Programming**: Solve complex problems by breaking them down into simpler overlapping subproblems.
* **Greedy Method**: Build up a solution piece by piece, always choosing the next piece that offers the most immediate benefit.

### 14. What is backtracking?

Backtracking is an algorithmic technique for solving problems incrementally by trying partial solutions and eliminating those that fail to satisfy the conditions of the problem. It explores all possible options until a solution is found or all options are exhausted.

### 15. What is the difference between backtracking and recursion?

* **Backtracking**: A method for solving problems that involves exploring all potential solutions and discarding those that do not meet the requirements.
* **Recursion**: A programming technique where a function calls itself to solve subproblems.

While backtracking often uses recursion as a tool, not all recursive algorithms are backtracking algorithms. Backtracking specifically aims to find solutions by searching through potential candidates and discarding unviable ones.