ADS DAY 1 Questions & Lab work

1) What do you mean by Data Structures?
Ans:- The arrangement and organizing of data in main memory for efficient utilization of data by the applications is called data structure.

2) Define The Goals Of Data Structure?

3) What is the Need of DS ?
Ans:- We need a proper way of organizing the data so that it can accessed efficiently when we need that particular data. DS provides different ways of data organization so we have options to store the data in different data structures based on the requirement.

4) List out the areas in which data structures are applied extensively(real time examples)?
Ans:-

1- To implement back functionality in the internet browser. -Stack
2- To store the possible moves in a chess game. -Tree
3- You need to evaluate an expression (i.e., parse). – stack or tree
4- You need to store undo/redo operations in a word processor. – stack
5- To store the customer order information in a drive-in burger place. -Queue

5) List different types of data structures.
Ans:- There are two types of data structure

- **1- Physical Data Structure:-** This are called physical because they are more related about storing data. Eg. Array and LinkedList.
  - 1- **Array:-** It should be used when we know the fixed size and it can be created in Heap as well as in stack memory.
  - **2- LinkedList:-** It is used when we don't know the fixed size and it is not created in heap memory.
- **2- Logical Data Structure :-** To perform operation on physical data structure such as insertion, deletion, sorting etc. we use Logical data structure.
  - **1- Stack.**
  - **2- Queue.**
  - **3- Trees.**
  - **4- Graphs.**
  - **5- HashTree.**

6) What Does Abstract Data Type Mean?
Ans:- Abstract Data Type(ADT) is a data type, where only behavior is defined but not implementation. Opposite of ADT is Concrete Data Type (CDT), where it contains an implementation of ADT. Examples: Array, List, Map, Queue, Set, Stack, Table, Tree, and Vector are ADTs

7) What is Recursion?

Ans:- Recursion is the technique of making a function call itself to solve some problem. The method that uses this technique is recursive. There should be a end statement to know were should the recursive method should stop.

8) List and Explain types of Recursion

Ans:- Recursion are mainly of two types depending on whether a function calls itself from within itself or more than one function call one another mutually. The first one is called direct recursion and another one is called indirect recursion. Thus, the two types of recursion are:

1. **Direct Recursion**: These can be further categorized into four types**:**

   1- **Tail Recursion**: If a recursive function calling itself and that recursive call is the last statement in the function then it's known as Tail Recursion**.** After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

   2- **Head Recursion**: If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head Recursion. There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

   3- **Tree Recursion**: To understand Tree Recursion let's first understand Linear Recursion. If a recursive function calling itself for one time then it's known as Linear Recursion. Otherwise if a recursive function calling itself for more than one time then it's known as Tree Recursion.

   4- **Nested Recursion**: In this recursion, a recursive function will pass the parameter as a recursive call. That means "recursion inside recursion". Let see the example to understand this recursion.

2. **. Indirect Recursion**: In this recursion, there may be more than one functions and they are calling one another in a circular manner.

9) Explain the data structures used to perform recursion?

Ans:- Stack has the LIFO (Last In First Out) property; it remembers it's 'caller'. Therefore, it knows to whom it should return when the function has to return. On the other hand, recursion makes use of the system stack for storing the return addresses of the function calls.

Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written explicit, stack is to be used.

10)List the examples where recursion is used .

Ans:- Some Computer related examples include: Adding a list of numbers, Computing the Fibonacci sequence, computing a Factorial, and Sudoku.

11)Explain the difference between Recursion and Iteration, justify which to use when,Tail recursion?
Ans-

Recursion	Iteration
Recursion is when a statement in a function calls itself repeatedly	The iteration is when a loop repeatedly executes until the controlling condition becomes false.
recursion is a process, always applied to a function	iteration is applied to the set of instructions which we want to get repeatedly executed**.**
Recursion uses selection structure.	Iteration uses repetition structure.
Recursion terminates when a base case is recognized.	An iteration terminates when the loop condition fails.
Recursion is usually slower than iteration due to the overhead of maintaining the stack.	An iteration does not use the stack so it's faster than recursion.
More Memory	less memory
Recursion makes the code smaller.	Iteration makes the code longer.

The tail recursive functions considered better than non tail recursive functions as tail-recursion can be optimized by the compiler. Compilers usually execute recursive procedures by using a stack**.** This stack consists of all the pertinent information, including the parameter values, for each recursive call. When a procedure is called, its information is pushed onto a stack, and when the function terminates the information is popped out of the stack. Thus for the non-tail-recursive functions, the stack depth (maximum amount of stack space used at any time during compilation) is more. The idea used by compilers to optimize tail-recursive functions is simple, since the recursive call is the last statement, there is nothing left to do in the current function, so saving the current function's stack frame is of no use.

12)Difference between Primitive and Non Primitive DS
Ans:-

Primitive data structure	Non-primitive data structure
Primitive data structure is a kind of data structure that stores the data of only one type	Non-primitive data structure is a type of data structure that can store the data of more than one type.
Examples of primitive data structure are integer, character, float.	Examples of non-primitive data structure are Array, Linked list, stack.
Primitive data structure will contain some value, i.e., it cannot be NULL.	Non-primitive data structure can consist of a NULL value.
The size depends on the type of the data structure.	In case of non-primitive data structure, size is not fixed.
It starts with a lowercase character.	It starts with an uppercase character.
Primitive data structure can be used to call the methods.	Non-primitive data structure cannot be used to call the methods.

13)Difference between Linear and Non Linear DS
Ans:-

Linear Data Structure	Non-linear Data Structure
In a linear data structure, data elements are arranged in a linear order where each and every elements are attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
In linear data structure, single level is involved.	Whereas in non-linear data structure, multiple levels are involved.
Implementation is easy.	Implementation is complex.
In linear data structure, data elements can be traversed in a single run only.	While in non-linear data structure, data elements can't be traversed in a single run only.
In a linear data structure, memory is not utilized in an efficient way.	While in a non-linear data structure, memory is utilized in an efficient way.
Its examples are: array, stack, queue, linked list, etc	While its examples are: trees and graphs.
Applications of linear data structures are mainly in application software development.	Applications of non-linear data structures are in Artificial Intelligence and image processing.

14)What are different characterstics of an Algorithm?types of Algorithm.
Ans:- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics −

- Unambiguous − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- Input − An algorithm should have 0 or more well-defined inputs.
- Output − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- Finiteness − Algorithms must terminate after a finite number of steps.
- Feasibility − Should be feasible with the available resources.
- Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

Types of Algorithm

Brute Force Algorithm:
This is the most basic and simplest type of algorithm. A Brute Force Algorithm is the straightforward approach to a problem i.e., the first approach that comes to our mind on seeing the problem. More technically it is just like iterating every possibility available to solve that problem.

Recursive Algorithm:
This type of algorithm is based on recursion. In recursion, a problem is solved by breaking it into subproblems of the same type and calling own self again and again until the problem is solved with the help of a base condition.

Divide and Conquer Algorithm:
In Divide and Conquer algorithms, the idea is to solve the problem in two sections, the first section divides the problem into subproblems of the same type. The second section is to solve the smaller problem independently and then add the combined result to produce the final answer to the problem.
Some common problem that is solved using Divide and Conquers Algorithms are Binary Search, Merge Sort, Quick Sort, Strassen's Matrix Multiplication, etc.

Dynamic Programming Algorithms:
This type of algorithm is also known as the memoization technique because in this the idea is to store the previously calculated result to avoid calculating it again and again. In Dynamic Programming, divide the complex problem into smaller overlapping subproblems and storing the result for future use.
The following problems can be solved using Dynamic Programming algorithm Knapsack Problem, Weighted Job Scheduling, Floyd Warshall Algorithm, Dijkstra Shortest Path Algorithm, etc.

Greedy Algorithm:
In the Greedy Algorithm, the solution is built part by part. The decision to choose the next part is done on the basis that it gives the immediate benefit. It never considers the choices that had taken previously.
Some common problems that can be solved through the Greedy Algorithm are Prim's Algorithm, Kruskal's Algorithm, Huffman Coding, etc.

Backtracking Algorithm:
In Backtracking Algorithm, the problem is solved in an incremental way i.e. it is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.
Some common problems that can be solved through the Backtracking Algorithm are Hamiltonian Cycle, M-Coloring Problem, N Queen Problem, Rat in Maze Problem, etc.

15)State Advantages and Disadvantages of Recursion.
Ans:-
Advantages of recursion

1. The code may be easier to write.
2. To solve such problems which are naturally recursive such as tower of Hanoi.
3. Reduce unnecessary calling of function.
4. Extremely useful when applying the same solution.
5. Recursion reduce the length of code.
6. It is very useful in solving the data structure problem.
7. Stacks evolutions and infix, prefix, postfix evaluations etc.

Disadvantages of recursion

1. Recursive functions are generally slower than non-recursive function.
2. It may require a lot of memory space to hold intermediate results on the system stacks.
3. Hard to analyze or understand the code.
4. It is not more efficient in terms of space and time complexity.
5. The computer may run out of memory if the recursive calls are not properly checked.