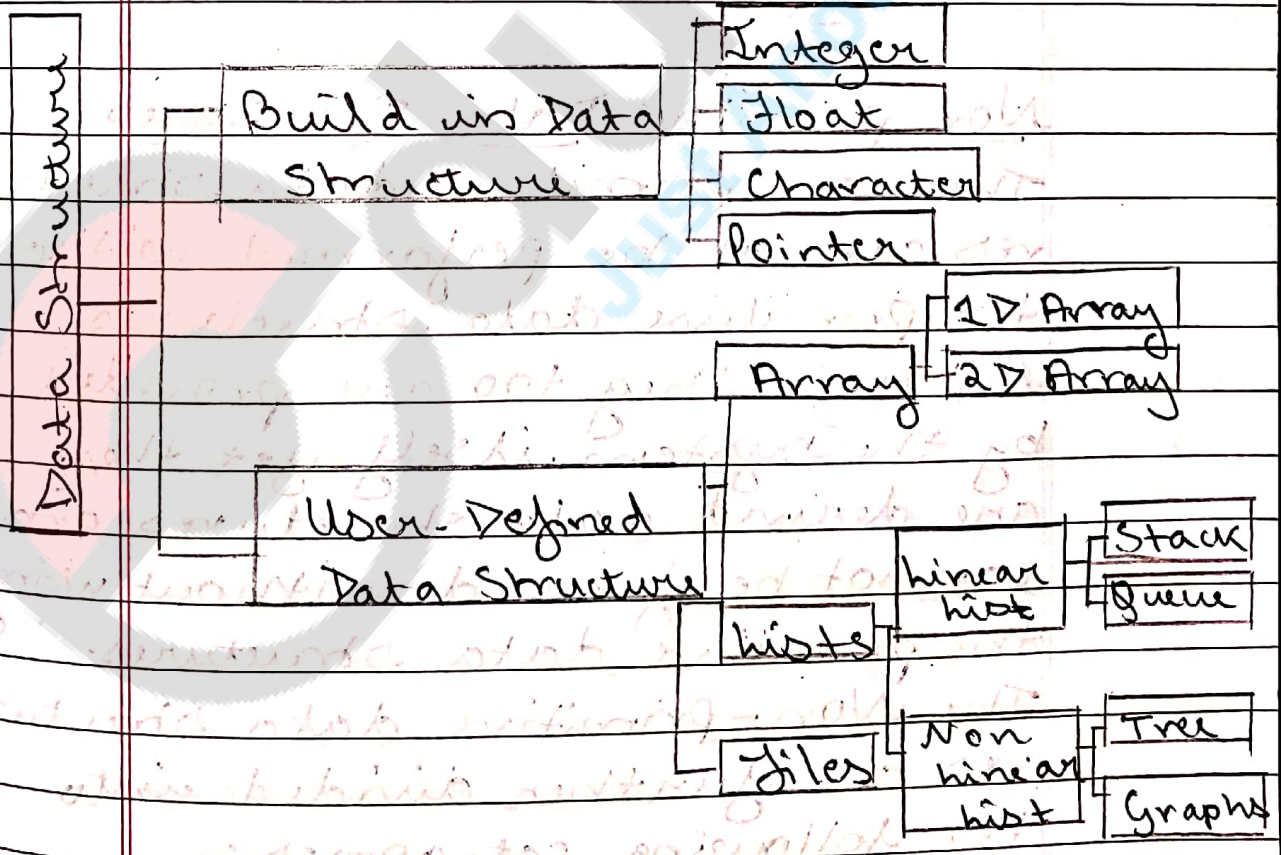
Unit 1

Q] Data Structures and its types

Data structures are a very important programming concept. They provide us with a means to store, organize and retrieve data in an efficient manner. The data structures are used to make working with our data, easier. There are many data structures which help us with this.

Types of Data Structures



Primitive Data Structures

These are the structures which are supported at the machine level, they can be used to make non-primitive data structures. These are integral and are pure in form. They have predefined behavior and specifications

Examples: Integer, float, character, pointers.

The pointers, however don't hold value, instead, they hold memory addresses of the data values.

These are also called the reference data types.

Non-primitive Data Structures

The non-primitive data structures cannot be performed without the primitive data structures. Although, they too are provided by the system itself yet they are derived data structures and cannot be formed without using the primitive data structures.

The Non-primitive data structures are further divided into the following categories:

1) Arrays

Arrays are a homogeneous and contiguous collection of same data types. They have a static memory allocation technique, which means, if memory space is allocated for once, it cannot be changed during runtime. The arrays are used to implement vectors, matrices and also other data structures. If we do not know the memory to be allocated in advance then array can lead to wastage of memory. Also, insertions and deletions are complex in arrays since elements are stored in consecutive memory allocations.

2) Files

A file is a collection of records. The file data structure is primarily used for managing large amounts of data which is not in the primary storage of the system. The files help us to process, manage, access and retrieve or basically work with such data, easily.

3) lists

The lists support dynamic memory allocation. The memory space allocated, can be changed at run time also. The lists are of two types:

a) linear lists :-

The linear lists are those which have the elements stored in a sequential order. The insertions and deletions are easier in the lists. They are divided into two types:

- Stacks : The stack follows a "LIFO" technique for storing and retrieving elements. The element which is stored at the end will be the first one to be retrieved from the stack. The Stack has the following primary functions:

  Push(): To insert an element in the stack.

  Pop(): To remove an element from the stack.

- Queues : The queues follow "FIFO" mechanism for storing and retrieving elements. The

elements which are stored first into the queue will only be the first elements to be removed out from the queue. The "ENQUEUE" operation is used to insert an element into the queue whereas the "DEQUEUE" operation is used to remove an element from the queue.

b) Non linear lists

The non linear lists do not have elements stored in a certain manner. These are:

• Graphs: The graph data structure is used to represent a network. It comprises of vertices and edges (to connect the vertices). The graphs are very useful when it comes to study a network.

• Trees: Tree ~~is~~ data structure comprises of nodes connected in a particular arrangement and they (particular binary trees) make search operations on the data items easy. The tree data structures consists of a root node which is further divided

into various child nodes and so on. The number of levels of the tree is also called height of the tree.

Q) Space and Time Complexity

Space Complexity

Space complexity is an amount of memory used by the algorithm (including the input values of the algorithm), to execute it completely and produce the result.
We know that to execute an algorithm it must be loaded in the main memory. The memory can be used in different forms:

1) Variables (This includes the constant values and temporary values)
2) Program Instruction
3) Execution

Auxiliary Space

Auxiliary Space is extra space or temporary space

used by the algorithms during its execution.

Memory Usage during program execution

1) Instruction Space is used to save compiled instruction in the memory.

2) Environmental Stack is used to storing the addresses while a module calls another module or functions during execution.

3) Data space is used to store data, variables, and constants which are stored by the program and it is updated during execution.

Time Complexity

Time complexity is simply a measure of the time it takes for a function or expression to complete its task, as well as the name of the process to measure that time. It can be applied to almost any algorithm or function but is more useful for recursive functions. There is little point in measuring

time complexity for applications such as fetching the username and password from a database for comparison or simply saving data whether it is 20ms or 5ms; that would be more in the line of access time. It has nothing to do with caring about its execution time, but rather that the difference is negligible. However, if there is a recursive function that may be called multiple times, determining and understanding the source of its time complexity may help shorten the overall processing time from, say 600ms to 100ms.

Time complexity is expressed typically in the "big O notation", but there are other notations. This is a mathematical representation of the upper limit of scaling factor for an algorithm and it is written as O(Nn), with "N" being the number of inputs and "n" being the number

of looping expressions. ~~for example~~

6] Asymptotic Analysis

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/ framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation

running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small. Usually, the time required by an algorithm falls under three types -

1) Best case - Minimum time required for program execution.
2) Average case - Average time required for program execution
3) Worst case - Maximum time required for program execution

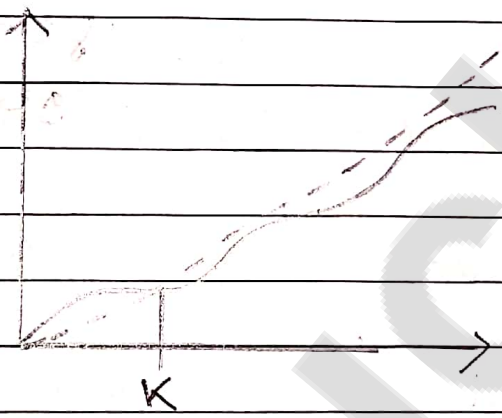
Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

1) O Notation
2) Ω Notation
3) θ Notation

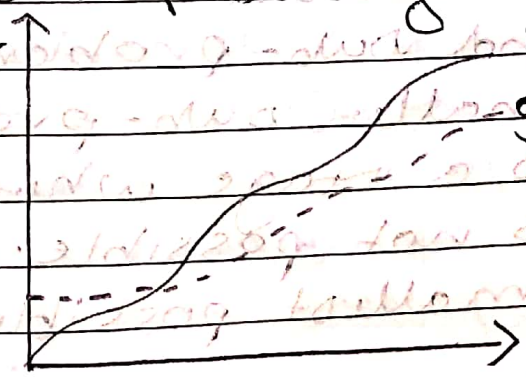
Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete



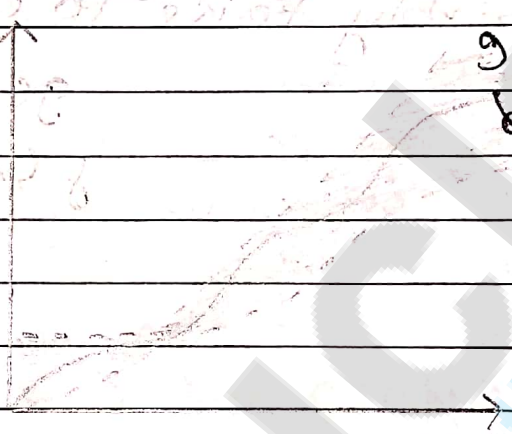
Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



Theta Notation, Θ

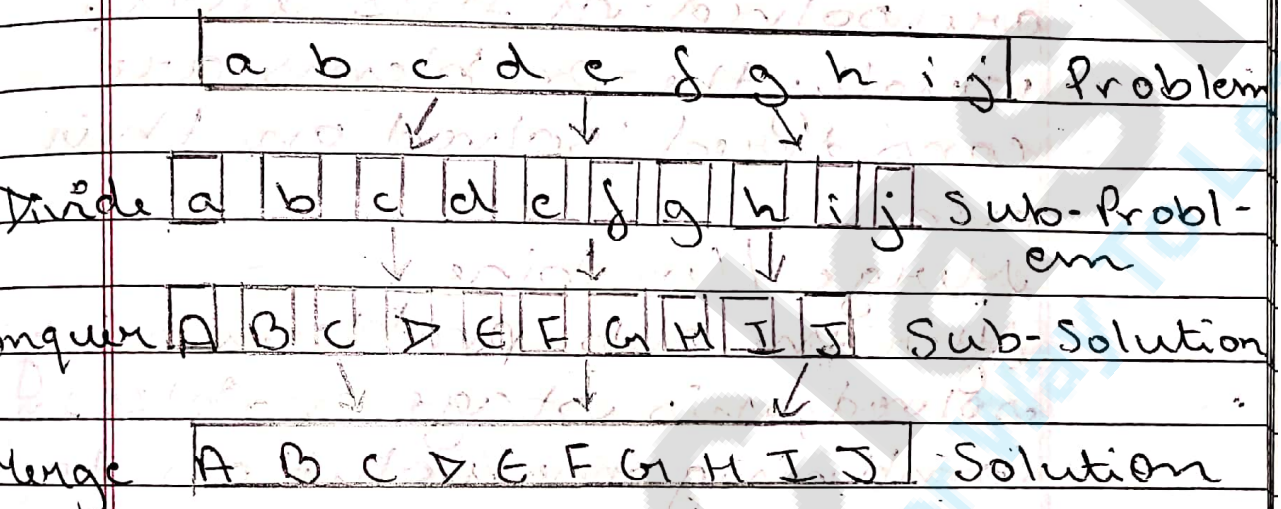
The notation Θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows:-



Q) Divide and Conquer approach of Data Structures

In divide and conquer approach, the problem is divided into sub-problems and each sub-problem is independently solved. The problems can be divided into sub-problems and sub-problems to even smaller sub-problems, but to a stage where division is not possible. Now the smallest possible sub-problems

of the sub-problems are solved. Finally, the solution of all sub-problems is merged to obtain solution of original problem



The concept of divide-and-conquer approach is explained in a three-step process.

Divide/Break

In this step, the problem is broken into smaller sub-problems such that each sub-part should represent a part of the original problem. Recursive approach is used in this step to divide the problem till further sub-division of the problem is not possible. In this approach, the problems

turn to be atomic in nature but even then represent some part of the original problem.

Conquer / Solve

In this step, the sub-problems are solved. At this level, usually the problems are considered 'solved' on their own.

Merge / Combine

After the sub-problems being solved, this stage enables in combining the solution to formulate a solution for the original problem. This algorithmic approach works recursively and conquers & merge steps works so close that they appear as one.

The computer algorithms which are based on divide-and-conquer programming approach are:

- Merge Sort
- Quick Sort
- Binary Search

Q) Dynamic programming approach of Data Structures

The Dynamic programming approach is similar to that of divide and conquer rule. Even in dynamic programming approach, the problems are divided into sub problems and sub-programs into further smaller sub problems. But unlike divide and conquer, the sub problems are not solved independently, but the results of the sub-problems are noted and used further for similar or overlapping sub-problems.

The approach of dynamic programming is used where the problems can be divided into sub-problems such that their results are re-used. Mostly, these algorithms are used for optimization. The dynamic algorithm, before solving the sub-problem, will examine the results of the previously solved sub-problems. To achieve the best solution, these sub-problems are combined.

It is understood that -
- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use memorization

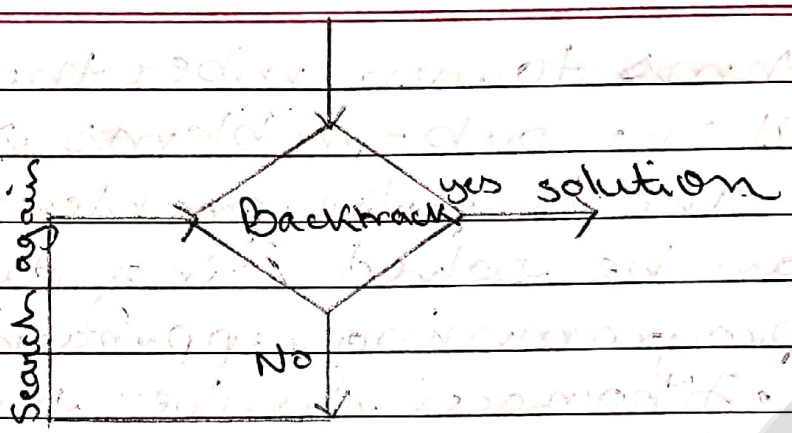
When compared with greedy algorithms which address the local optimization, dynamic algorithms on the other hand are motivated for overall optimization of the problem.
When compared with divide and conquer algorithms where the solutions are combined to achieve the overall solution, dynamic algorithms on the other hand use the output of smaller sub-problem and then bigger sub problems are optimized. Memorization is used by dynamic algori-

rams to remember the output of the sub-problems solved.
The computer problems that can be solved using dynamic programming approach are-
- Fibonacci number series
- Tower of Hanoi
- Shortest path by Dijkstra
- Project scheduling

Dynamic programming can be used in both top-down and bottom-up manner.

8) Backtracking

Backtracking is nothing but the modified process of the brute force approach, where the technique systematically searches for a solution to a problem among all available options. It does so by assuming that the solutions are represented by vectors $(v_1, \ldots, i_n)$ of values and by traversing through the domains of the vectors until the solutions is found.



Backtracking systematically try and search possibilities to find the solution. Also it is an important process for solving constraint satisfaction problem like crossword, Sudoku and many other puzzles. It can be more continent technique for parsing other combinatorial optimization problem. Basically the process is used when the problem has a number of option and just one solution have to be selected. After having a new option set means recursion, the procedure is repeated over and over until final stage.

Advantages

- Comparison with the Dyna-

mic Programming, Backtracking Approach is more effective in some cases.
- Backtracking Algorithm is the best option for solving tactical problem
- Also Backtracking is effective for constraint satisfaction problem
- In greedy Algorithm, getting the global Optimal Solution is a long procedure and depends on used statements but in Backtracking it ~~can~~ is easily getable.
- Backtracking technique is simple to implement and easy to code
- Different states are stored into stack so that the data or info can be usable anytime.
- The accuracy is granted.

Disadvantages
- ~~Backtra~~ Backtracking Approach is not efficient for solving strategic problem.
- The overall runtime of

Backtracking Algorithm is normally slow.
- To solve large problem sometime it needs to take the help of other techniques like branch or bound.
- Need large amount of memory space for storing different state function in the stack for big problem.
- Thrashing is one of the main problem of backtracking.
- The basic approach detects the conflicts too late.

Application of Backtracking
1) Optimization and tactical problems
2) Electrical Engineering
3) Robotics
4) Artificial intelligence
5) Network communication
6) Solving puzzles and path.