Content

- Data Management Concepts
- Data Types
  - Primitive and
  - Non-Primitive
- Types of Data Structures
  - Linear Data Structure
  - Non Linear Data Structure

Data Management Concepts

- A program should give correct results but along with that it should run efficiently.
- Efficient program executes:
  - Minimum time
  - Minimum memory space

- To write efficient program we need to apply Data management concepts.

Data Management Concepts

- Data Management concept includes,
  - Data collection
  - Organization of data in proper structure
  - Developing and maintaining routines for quality assurance

*"A **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently."*

Data Structure

- When selecting DS, must perform below steps:
  1) Analysis of the problem to determine basic operations like insert/delete/search a data in DS
  2) Quantify the resource constraints for each operation
  3) Select DS that best meets these requirements

- First-data and operations that are to be performed on them
- Second –representation of data
- Third –implementation of that representation

Data Types

- A data type is a classification of data, which can store a specific type of information.

Data Type = Basic Data Type = Primitive Data Type

- Primitive data types are predefined, supported by C language.
  - int, char, float, double

Data Types

- Non-Primitive data types are not defined by C language, but are created by the programmer.

- They are created using the basic data types.
- Example:
  1. Linked List
  2. Stacks
  3. Queue
  4. Graph

Types of Data Structure

1. Arrays

- An array is a fixed size, sequence collection of elements of same data type.

- Array Syntax:

  int Age[10];

Age 0	Age 1	Age 2	Age 3	Age 4	Age 5	Age 6	Age 7	Age 8	Age 9
30	32	54	32	26	29	23	43	34	5

Types of Data Structure

Arrays

Age 0	Age 1	Age 2	Age 3	Age 4	Age 5	Age 6	Age 7	Age 8	Age 9
30	32	54	32	26	29	23	43	34	5

- Limitations of Arrays:
  - Fixed size
  - Data elements are stored in continuous memory locations which may not be available, always
  - Adding and removing of elements is tough because of shifting the elements from their positions

Types of Data Structure

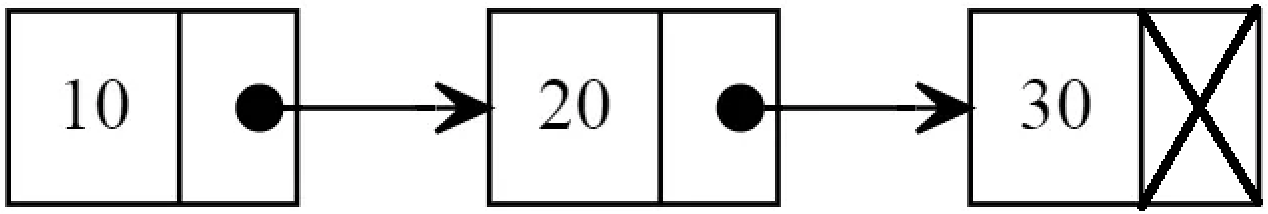
2. Linked List

- Very flexible, dynamic data structure which allows for efficient insertion or deletion of elements from any position in the list

- Each element (is called a node) in the list points to the next node in the list. Therefore, every node contains two information:
  1) The value or data of the node
  2) A pointer or link to the next node in the list

Types of Data Structure

Linked List



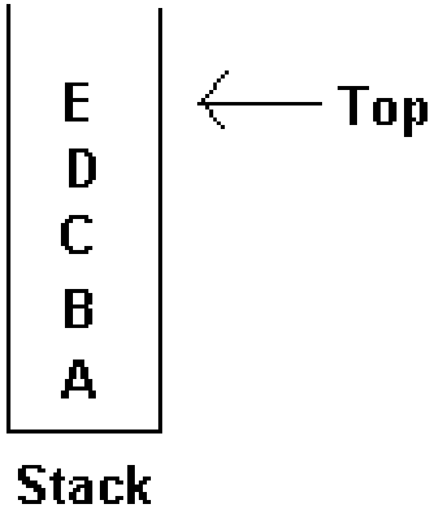
Linked List

- Advantage: Provides quick insert and delete operations
- Disadvantage: Slow search operations and requires more memory space

Types of Data Structure

3. Stack

- Stack can be represented as a linear array.
- Every stack has a variable TOP associated with it, to store the address of the topmost element of the stack.



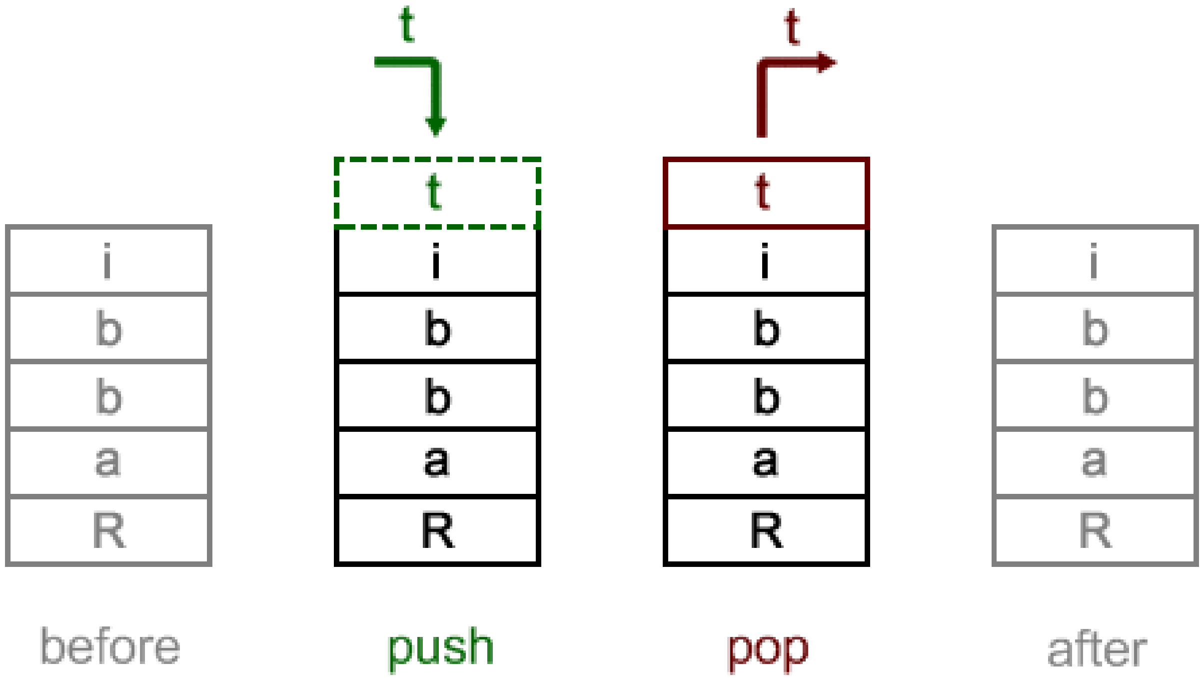
Types of Data Structure

Stack

- A stack is a last in, first out *(LIFO)*data structure.

  If TOP = NULL, then it indicates stack is empty

  If TOP = MAX, then it indicates stack is full.



Types of Data Structure

Stack Example

- Draw the ***stack*** structure in each case when following operations are performed on an empty stack.
  - Add A, B, C, D, E, F *(push operation)*
  - Delete two alphabets *(pop operation)*
  - Add G *(push operation)*
  - Add H *(push operation)*
  - Delete four alphabets *(pop operation)*
  - Add I *(push operation)*

Types of Data Structure

Stack

- Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are those that have been on the stack the longest.

- Advantage: last-in first-out (LIFO) access.
- Disadvantage: Slow access to other elements.

Types of Data Structure

4. Queue

- ***Queue*** is a data structure in which data can be added to one end and retrieved from the other.

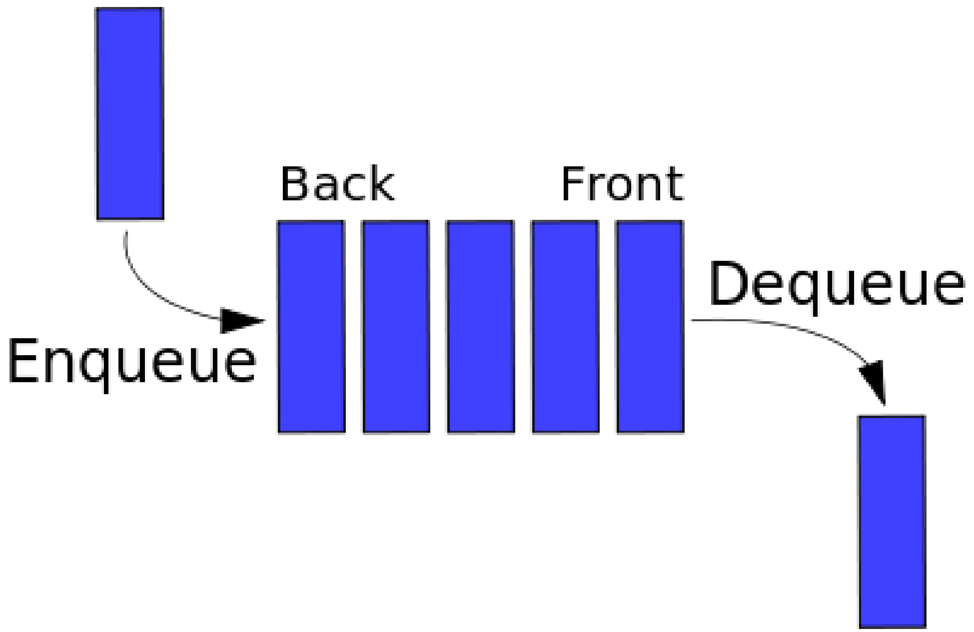
- You can think of it as a line in a grocery store. The first one in the line is the first one to be served. Just like a queue.

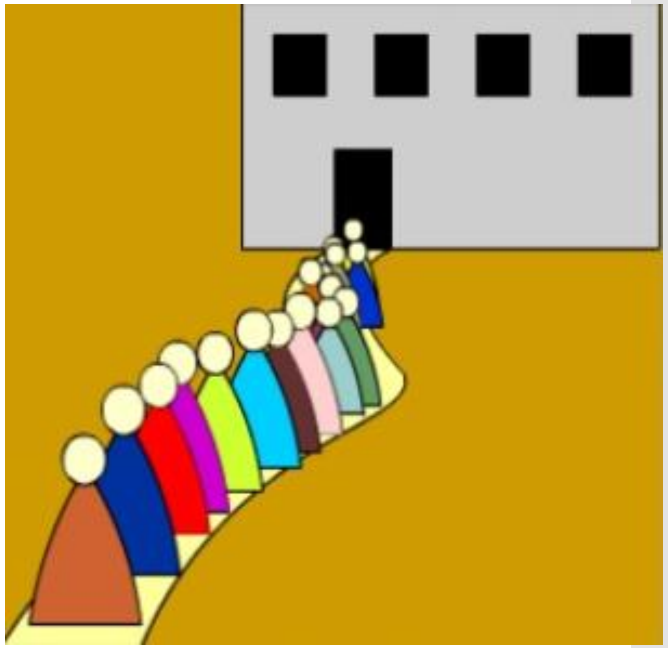
- A queue is also called a ***FIFO*** (First In First Out) to demonstrate the way it accesses data.

Types of Data Structure

Queue

- Advantage: Provides first-in, first-out data access.
- Disadvantage: Slow access to other items.





Types of Data Structure

Queue Example

- Draw the ***queue*** structure in each case when following operations are performed on an empty queue.
  - Add A, B, C, D, E, F *(enqueue operation)*
  - Delete two alphabets *(dequeue operation)*
  - Add G *(enqueue operation)*
  - Add H *(enqueue operation)*
  - Delete four alphabets *(dequeue operation)*
  - Add I *(enqueue operation)*

Types of Data Structure

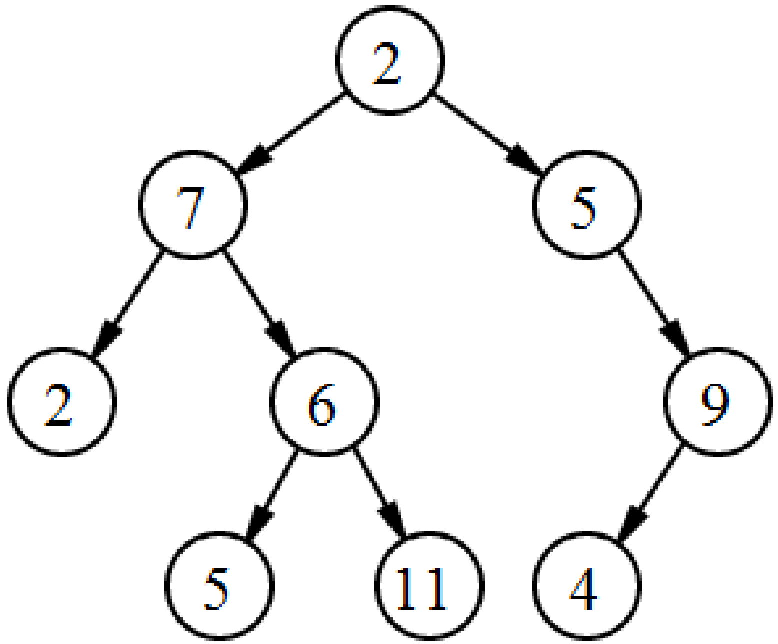
5. Tree

- A ***tree*** is a widely used data structure that simulates a hierarchical tree structure with a set of linked nodes.
- Every node contains a left pointer, a right pointer and a data element.
- Every tree has a root element pointed by a root pointer.
- If root = NULL, tree is empty.

Types of Data Structure

Tree

- A simple unordered tree; in this diagram, the node labelled 7 has two children, labelled 2 and 6, and one parent, labelled 2.
- The root node, at the top, has no parent.



Types of Data Structure

Tree

- Advantage: Provides quick search, insert, delete operations.

- Disadvantage: Complicated deletion algorithm.

- Example: Family Tree

Types of Data Structure

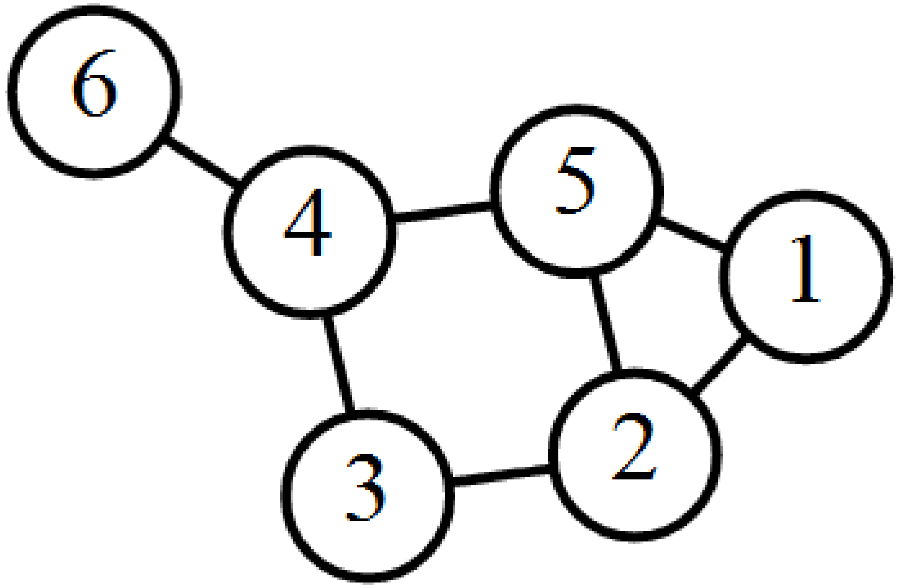
6. Graph

- A ***graph*** is a data structure that is meant to implement the graph concepts from mathematics.
- It is basically a collection of vertices(nodes) and edges that connect these vertices.
- A graph is often viewed as a generalization of the tree structure, where complex relationship can be represented.

Types of Data Structure

Graph

- Advantage: Best models real-world situations.
- Disadvantages: Some algorithms are slow and very complex.



Types of Data Structure

1. Linear Data Structure
   - Elements are stored sequentially
   - We can traverse either forward or backward
   - Examples: Arrays, Stacks, Queues, Linked list

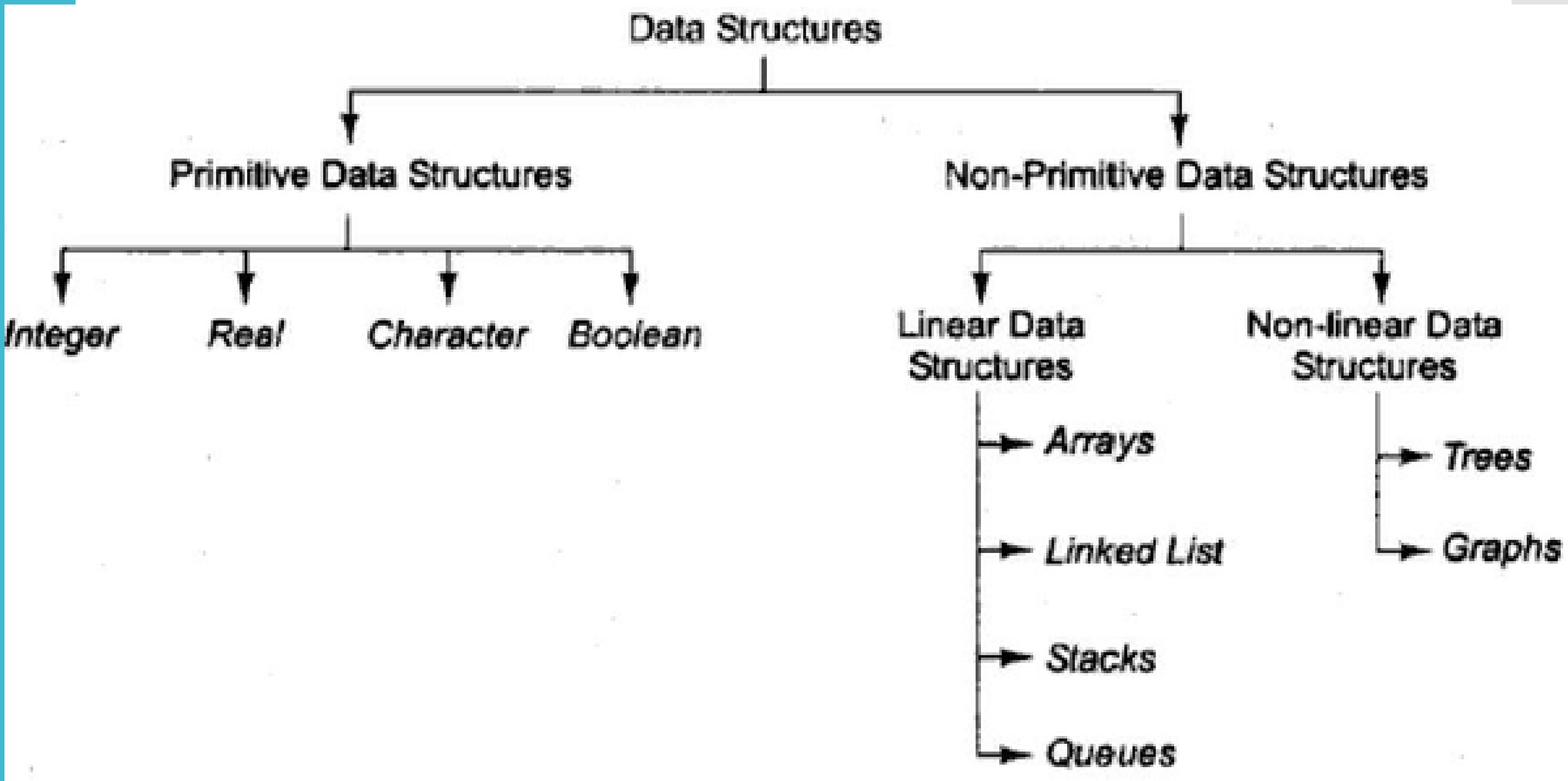
2. Nonlinear Data Structure
   - Not stored in sequential order
   - Branches to more than one node
   - Can’t be traversed in a single run
   - Examples: Tree, Graph

Abstract Data Type (ADT)

- ADT is the way we look at a Data Structure, focusing on what is does and ignoring how it does its job.
- Examples: Stack, Queue

- When ever end user uses Stack, he is concerned about only the type of data and operations that can be performed on it.
- Fundamentals of how the data is stored is invisible to users.
- They will have push() and pop() functions only.

Data Structure



Algorithm

- An algorithm provides a blueprint to write a program to solve a particular program.
- Algorithm is a set of instructions that solve a problem.
- It is possible to have multiple algorithms to tackle the same problem, but choice depends on time and space complexity.

Key features of an algorithm

- Any algorithm will be having finite steps.
- Algorithm exhibits three key features:

    1. Sequence
    2. Decision
    3. Repetition

Time and Space Complexity

- The **analysis of algorithms** is the determination of the number of resources (such as time and storage) necessary to execute them.
- Time Complexity of an algorithm is basically the running time of a program, as a function of the input size.
- Space Complexity of an algorithm is the amount of computer memory that is required during the program execution, as a function of input size.
- The space needed by a program depends on:
- *1)Fixed Part:* that varies from problem to problem. It includes instruction, constants, variables etc.
- *2)Variable Part*: that varies from problem to problem. It includes space needed for recursion, dynamic value allocation to variables.

Best, worst and average case

- **Best**, **Worst** and **Average cases** of a given algorithm express what there source usage is *at least*, *at most* and *on average*, respectively.

- In real-time computing, the worst-case execution time is often of particular concern since it is important to know how much time might be needed *in the worst case* to guarantee that the algorithm will always finish on time.

Best-case performance for algorithm

- The term *best-case performance* is used in computer science to describe the way of an algorithm behaves under *optimal conditions*.

- For example, the best case for a simple linear search on a list occurs when the desired element is the first element of the list.

Worst-case performance for algorithm

- This denotes the behavior of the algorithm with respect to the worst-possible case of the input instances.
- Worst-case running time of an algorithm is an *upper bound* on the running time for any input.
- This provides an assurance that this algorithm will never go beyond this time *limit.*

Average-case performance for algorithm

- Running time of an algorithm is an *estimate* of the running time for an ‘average’ input.
- It specifies the expected behavior of the algorithm when the input is randomly drawn from a given distribution.

Time-Space Trade-off

- There can be more than one algorithm to solve a particular problem.
- One may require less memory space and one may require less CPU time to execute.
- Hence, there exists a time-space trade-off among algorithms.
- So, *if space is a big constraint*, then one might choose a program that takes less space at the cost of more CPU time.
- On the contrary, *if time is a major constraint* then one might choose a program that minimum time to execute at the cost of more space.

Expressing Time & Space Complexity

- Time & Space Complexity can be expressed using function f(n), where n is the input size Required when:

1. We want to predict the rate of growth of complexity as size of the problem increases

2. Multiple algorithms available, but we need to find most efficient

- Most widely used notation to express this function f(n) is Big-Oh notation.

Expressing Time & Space Complexity

- Most widely used notation to express this function f(n) is Big-Oh notation.
- It provides upper bound for the complexity.

Find the time complexity of given example

```
1. for(i=0; i<10;i++)
      statement block;
```

Time Complexity
O(n)

```
2. for(i=0; i<100;i++)
      statement block;
```

Time Complexity
O(n)

```
3. for(i=0; i<100;i+=)
      statement block;
```

Time Complexity
O(n/2)

Find the time complexity of given example

Number of iterations in inner loop * Number of iterations in outer loop = Total number of iterations

4. Loop inside loop

```
for(i=0; i<10;i++)
        for(j=0; j<10;j++)
                statement block;
```

O(n*n)
OR
O(n^2)

Quiz

1. Linear Data Structure is ________

   (A) Tree

   (B) Graph

   (C) Stack

   (D) All of the above

2. None Linear Data Structure is ______

   (A) Stack

   (B) Graph

   (C) Queue

   (D) None of the above

Quiz

3. Which is/are Non Primitive Data Type ?
   (A) Stack
   (B) int
   (C) char
   (D) float
4. Stack is ___________
   (A) None Linear Data Structure
   (B) Last In Last Out
   (C) First In First Out
   (D) Last In First Out

Quiz

5. Queue is ______________

(A) None Linear Data Structure

(B) First In Last Out

(C) First In First Out

(D) Last In First Out

6. The following sequence of operations are performed on a stack : PUSH(1), PUSH(2), POP, PUSH(1), PUSH(2), POP, POP, POP, PUSH(2), POP. The sequence of values popped out is:

(A) 2, 1, 2, 1, 2

(B) 2, 2, 1, 1, 2

(C) 1, 2, 2, 1, 2

(D) 2, 2, 1, 2, 1

Quiz

7. Which is/are Primitive Data Type ?

(A) char

(B) Stack

(C) int

(D) options A and C

8. What is Array?

(A) Fixed sized number of elements

(B) Sequence collection

(C) Elements having same data type

(D) All of the above