Day 3 : Algorithms and Data Structures

```
Topic: Introduction to ADS
Date: 26/9/2024
Meeting ID:     832 1579 8576
Passcode:       806920
----------------------------------------
Topics:
        -Analysis of Algorithms
                -Time complexity
                -Space complexity
        -Arrays
        -Stack
        -Stack Applications
```

Time and Space complexity for Arrays:

```
1. Accessing an element 1-D Array: T=O(1) S=O(1)

2. Inserting at the end 1-D Array: T=O(1) S=O(1)

3. Inserting at the begining 1-D Array: T=O(n) S=O(n)

4. Searching in 1-D Array (Linear search): T=O(n) S=O(1)

5. Deleting an element in 1-D Array: T=O(n) S=O(1)

6. Transpose of a matrix in 2-D Array: T=O(m*n) S=O(m*n)

Ex:
//accessing
int []arr = {1,2,3,4,5};
int element = arr[2]; // O(1) : constant

Ex:
//insertion
int[] newArr = Arrays.copyOf(arr, arr.length+1);
newarr[arr.length] = 6; // O(1)

Ex:
int[][] matrix = {
        {1,2,3},
        {1,2,3},
        {1,2,3}
}
int element = matrix[1][1]; // O(1)
```

Stack:

```
-Linear data structure following LIFO (Last In First Out) principle.
-Insertion and deletion at the one end called as top of the stack.

LIFO (Last In First Out):
-The last element inserted is the first one to be deleted.
```

```
Representation of stack:
1. Fixed size stack: static : Arrays implementation
        -HAs a fixed size that cannot grow or shink.
        -Overflow occurs if stack is full.
        -Underflow occurs if stack is empty.

2. Dynamic size Stack: dynamic : Linked list Representation
        -Can grow and shrink into the stack.
```

Stack operations:

```
Insertion: Push()
Deletion: Pop()
Top position:peek()/tos()/top()
Underflow: isEmpty()
Overflow: isFull()
```

Stack Applications:

```
1. Balancing of symbols
2. String reversal
3. Redo/Undo
4. Recursion
5. Depth First search(DFS)
6. Backtracking
7. Expression evalaution
8. Memory management
```

String reverse

```
static void reverse(StringBuffer str)
        {
                int n = str.length();
                Stack s1 = new Stack();

                //push charaters in stack
                for(int i=0;i<n;i++)
                        s1.push(str.charAt(i));

                //pop one by one character and print
                for(int i=0;i<n;i++)
                {
                        char ch = (char)s1.pop();
                        str.setCharAt(i,ch);

                }
}
:
```

Balancing of symbols

Expression Evaluation:

```
Polish Notation:
1. Infix Notation : A + B :A,B=>OPRANDS, + =>OPERATOR
2. Prefix Notation : + A B
3. Postfix Notation : A B +

Operator Precedence:
---------------------
1. BODMAS Rule
2. Brackets, Exponential (^) (*/+-)

Example: Infix to prefix/Postfix
a+(b/c)-d

a+[(b+c)+(d+e)*f]/g

(a+b)*c/d+e^f/g

(a+b)/(c-d)
```

Conversion of Infix to Postfix: a+(b*c)

Postfix evaluation: 3 10 5 + *