Date.....
Page.....

DSA LAB-7

Theory

1) A. **Stack Underflow:**

Stack underflow happens when we try to pop (remove) an item from the stack, when nothing is actually there to remove. This will raise an alarm of sorts in the computer because we to it do something

Stack overflow:

Stack overflow happens when we try to push on more element into stack which is full (more than it can actually hold). we allocate where the stack is going be in memory and how big it can get. So, when we stick too much stuff there to try to remove nothing, we will generate a stack overflow or stack underflow message, respectively.

Date
Page

2) A.

Stacks	Array
1) Stack is based on LIFO principle i.e. the element inserted at the last, is the first element to come out of the list.	1) In the array of elements belong to indexes, i.e., if you want to get into the fourth element you have to write variable name with index.
2) Insertion and deletion in stacks takes place only from one end of the list called top.	2) Insertion and deletion in array can be done at any index in the array.
3) stack has a dynamic size	3) Array has fixed size
4) Stack can contain elements of different data type.	4) Array contains elements of same data type
5) We can do only linear Search	5) We can do both linear and binary search.

Date.....
Page.....

3) A. In implementation of stack as an array, array is fixed size. once we define size of array then in whole programme you can't increase or decrease a size even if we require. where in case of linked list these structures based on pointer arrangement. So, we can add as many nodes at any time.

So in case of linked list we create data structure is independent of size so it is in more favour.

4) A. The same way they are used in a program with recursive functions, stacks are used in non-recursive programs.

Non-recursion function are procedures or subroutines implemented in a programming language, whose implementation does not references itself.

Date
Page

5) A. a) Infinix Notation:

→ The traditional method of our writing of mathematical expressions is called as the infinix expressions.

→ Here the operator is fixed inside between the operands.

→ These expressions are easy to understand and evaluate for human beings. However computer finds it difficult to phrase - Information is needed about operator precendence and associativity rules, and brackets which override these rules.

b) Postfix Notation:

→ The postfix expression as the name suggests has the operator placed right after the two operands.

→ In the infinix expressions, it is difficult to keep track of the order precendence whereas here the postfix expression itself determines the precedence of operators

→ A postfix expression is parenthesis-free expression. For evaluation, we evaluate it from left-to-right.

Date.....
Page.....

c) Prefix Notation:

→ The prefix expression as the name suggests has the operator placed before the operand is specified.
→ It works entirely in same manner as the postfix expression.
→ while evaluating a prefix expression, the operators are applied to the operands immediately on the right of the operator.
→ For evaluation, we evaluate it from left-to-right.
Prefix expressions are called as polish notation.

Ex:

	Infinix	Prefix	Postfix
1)	(P+Q)*(M-N)	*+PQ-MN	PQ+MN-*
2)	(P+Q)/(M-N)-(A*B)	-/+PQ-MN*AB	PQ+MN-/AB*-

Date .....
Page .....

6)A. Infinix to Postfix

a) A + B * C / (E - F) : A B C * E F - / +

b) (A ^ B * (C + (D * E) - F)) / G : A B ^ C D E * + F - * G /

~~c) (A + B (+~~

c) (A + (B * C - (D / E ^ F) * G) * H) : A B C * D E F ^ / G * - H * +

Infinix to prefix

a) A + B * C / (E - F) : + A * B / C - E F

b) (A ^ B * (C + (D * E) - F)) / G : / * ^ A B + C - * D E F G

c) (A + (B * C - (D / E ^ F) * G) * H) : + A * - * B C * / D ^ E F G H