SEAT NO: A13A047

ROLL NO: 47

NAME: YASH SARANG

SEMESTER: III

SUBJECT: DS

PAGE NO: 1/19

Sarangpatil

Q1.

1) Option B: ABC*D/+E-F+

2) Option A: p->next = newNode; newNode->prev = p;
newNode->next = NULL; p = newNode.

3) Option C: ii and iii.

4) option B: 1 4 3 6 5 8 10 9 13
17 15 12 7.

5) Option C: i, ii, iv.

6) Option D: Array.

7) Option D: 50 -> 30 -> 10.

8) Option C: 2i+2.

9) Option C: (n*(n-1)/2).

10) Option D: -2.

Page No.:

Pg no. 3/19

Sarbanjeet

Q2. A

→

(i) Linear Data Structures

Data structures where the data elements are arranged sequentially or linearly where the elements are attached to its previous and next adjacent elements are called a linear data structure.

Linear data structures are easy to implement as the computer memory is arranged linearly.

Examples of Linear Data Structures would be Stacks, Arrays, Queues, Linked Lists, etc.

(ii) Non-Linear Data Structures.

Data structures in which the elements are not arranged sequentially or linearly are called non-linear data structures.

These data structures are complex to implement but they utilize computer memory efficiently.

Examples of non-linear data structures are Trees, Graphs, etc.

Page No.:

Pg no. 4/19

Sabyasachi

Q2. C.

→ Program to reverse a string using ~~st~~ stack.

```
#include <stdio.h>
#include <string.h>

#define max 100.
int top, stack [max];

void push (char x) {
    if (top == max-1)
        printf (" Stack Overflowed\n ");
    else
        stack [++top] = x
}

void pop () {
    printf (" %c ", stack [top--]);
}

int main () {
    char str[];
    printf (" Please enter the string to be
            reversed : ");
    scanf (" %s ", str);

    int len = strlen (str);
    int i;
```

Pg no. 5/19

Sarangesh

Q2.c.

```
for (i=0; i<len; i++)
    push (str[i]);

printf ("The reverse of your string
        would be: ");

for (i=0; i<len; i++)
    pop();

return 0;
}
```

Input :

Please enter the string to be reversed : India

Output :

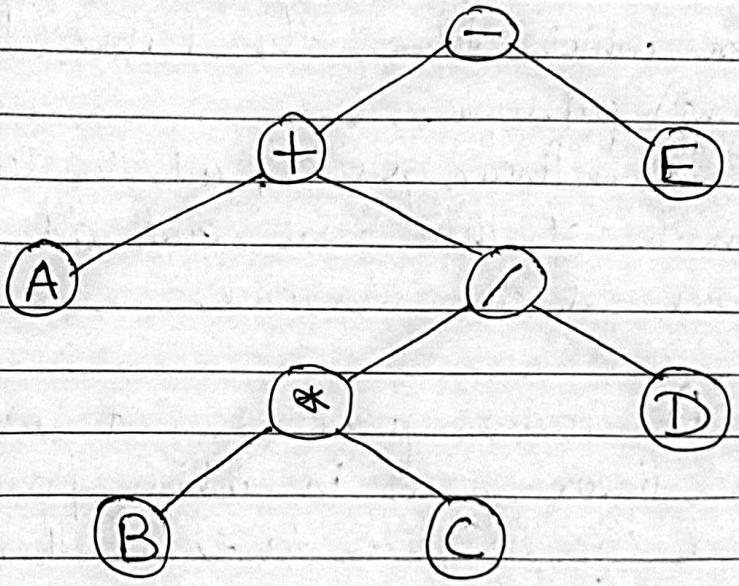
The reverse of your string would be : aidnI

Pg no. 6/19

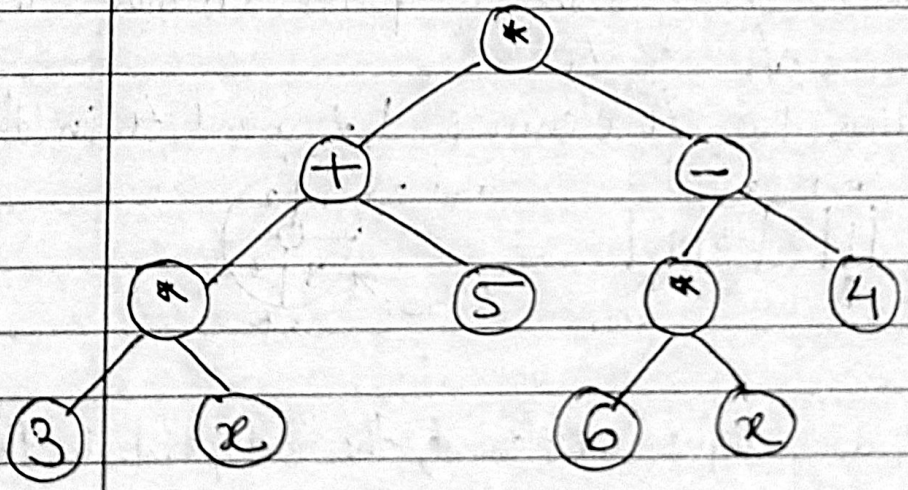
Sarangyash

Q2. F

i) A + B * C / D - E



ii) (3x + 5) (6x - 4)



Sarangyash

Q2. D.

→

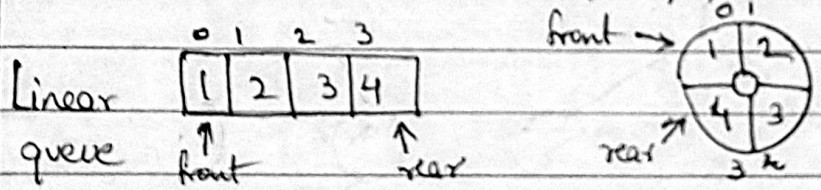
A Linear Queue is a linear data structure which follows the FIFO (first in first out) principle.

In a linear queue, all the deletions are made at the front and all insertions are made at the rear end.

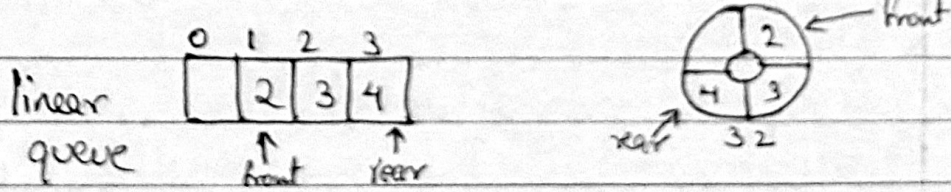
A circular queue is just a variation of the linear queue in which front and rear end are connected to each other to optimize the space wastage of the linear queue which makes it efficient.

Example: Let us take 2 queues (circular & linear) of size 4.

Let there be 4 elements 1, 2, 3 and 4 in them,

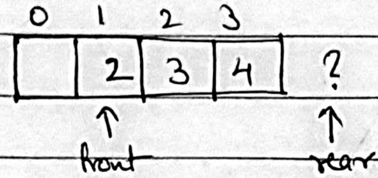


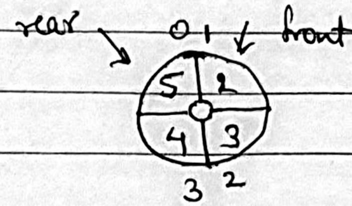
When we deque an element, the front points to the index 1.



Sarangyatha

Q2.D Now, if we ~~eo~~ enque 5 i.e insert an element then, the index of linear queue would go out of range whereas in a circular queue, it would ~~use~~ accomplish the empty 0th index.





circular queue.

linear queue

Altogether, circular queue has multiple advantages over a linear queue. such as

i) Ease of insertion and deletion.

ii) Efficient utilization of memory

iii) Ease of performing operations.

Saranyash

Q3. B.

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
# define Max 100.

float st [Max];
int top = -1;

void push (float st [], float val)
{
   if (top == Max - 1)
      printf ("\n Stack Overflowed");
   else {
      top ++;
      st [top] = val;
   }
}
 float pop (float st[])
{
  float val = -1;
  if (top == -1)
     printf (" Stack Underflow.\n");
  else {
     val = st [top];
     top --;
  } return val;
}
```

Pg. no. 10/19 Saranyasts

Q3.B
```
int main()
{
    float val;
    char exp[100];
    clrscr();
    printf("\n Enter any postfix expression:");
    gets(exp);

    val = evaluatePostfixExp(exp);
    printf("\n Value of the postfix expression
                = %.2f", val);
    getch();
    return 0;
}

float evaluatePostfixExp(char exp[])
{
    int i=0;
    float op1, op2, value;
    while(exp[i] != '\0')
    {
        if (isdigit(exp[i]))
            push(st, (float)(exp[i] - '0'));
        else {
            op2 = pop(st);
            op1 = pop(st);
            switch(exp[i])
            {
```

Pg no. 11/19

Sarang Yadav

Q3.B

```
        case '+':
            value = op1 + op2;
            break;

        case '-':
            value = op1 - op2;
            break;

        case '*':
            value = op1 * op2;
            break;

        case '/':
            value = op1 / op2;
            break;
        }
    push (st, value);
    }
    i++;
  }
  return (pop(st));
}
```

Page No.:

Pg no. 12/19

Saranyash

Q3. c.

→

	Binary Search Tree used when	AVL tree used when
i)	application involves less amount of data	searching on a higher priority.
ii)	insertion and deletion is easy due to no rotations	insertion and deletions are complex due to multiple rotations
iii)	searching is less efficient	searching is more efficient.
iv)	All binary trees are not AVL trees since they can be balanced or unbalanced	All AVL trees are binary search trees since they follow the conditions of binary search tree.

Continued...

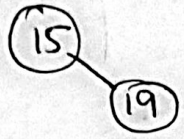
Yash Sarang Sarangpath

Q3. c.

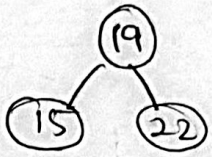
→ 15, 19, 22, 10, 3, 37, 25, 12, 13

1) 15

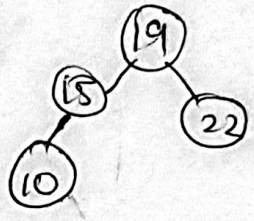
2)



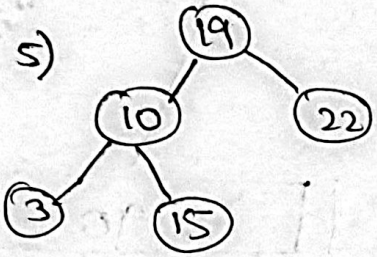
3)



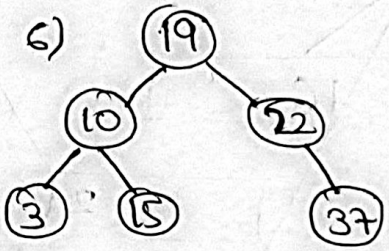
4)



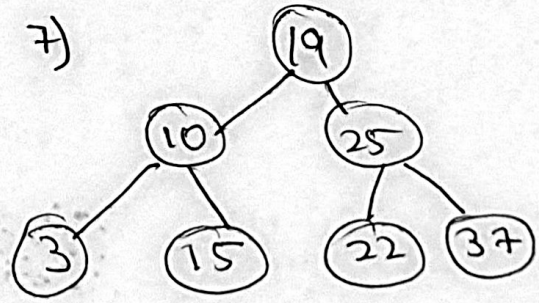
5)



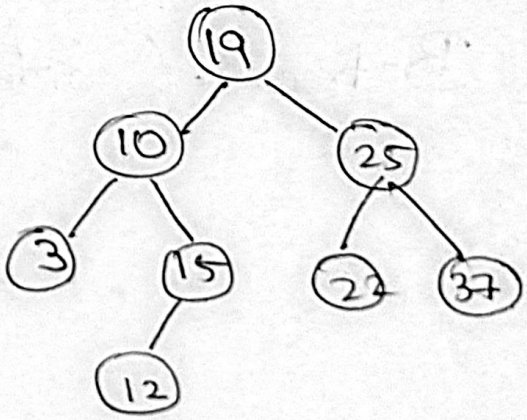
6)



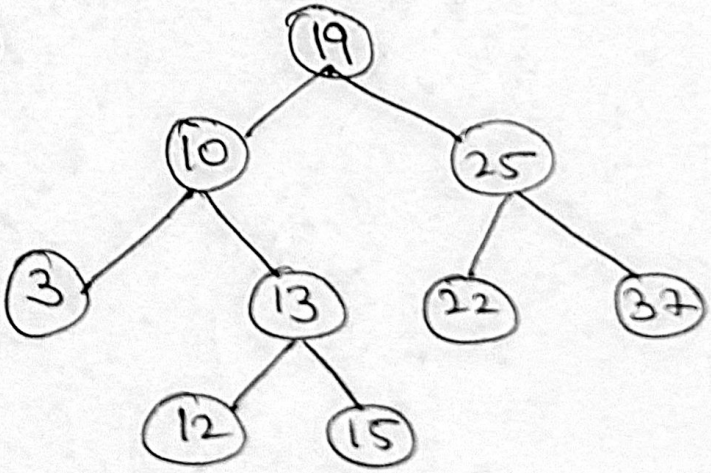
7)



8)



9)



Paragraph

Q4.A:) →

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *next;
}

void append (struct Node **head_ref, int data1)
{
    struct Node* new_node =
            (struct Node*) malloc(sizeof (struct Node));

    struct Node *last = *head_ref;

    new_node -> data = data1;
    new_node -> next = NULL;

    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }
    while (last -> next != NULL)
        last = last -> next;
    last -> next = new_node;
```

Sarangyadh

Q4.A.1)

```
    return;
}

int Mean (struct Node** head_ref)
{
    int mean = 0;
    int i = 0;
    struct Node* temp = *head_ref;

    while (temp -> next != NULL)
    {
        mean = mean + (temp -> data);
        i = i + 1;
    }

    mean = mean / i;
    return mean;
}
int main () {
    printf("Enter the values you want to
    insert in the list: ");
    int input[];
    i = 0;
    while (input[i] != '\0')
    {
        scanf("%i", &input[i]);
        i++;
    }
```

Page No.:

Pg no. 16/19 Sarangyash

Q4.A) i)

```
        for (i=0; i<n; i++)
        {
            append (input[i]);
        }
    int x=Mean();
        printf("The mean of the data is %.d", x);
}
```

Pg no. 14/19 Sarangpath

Qu A)

ii)

→ DFS (using Stack)

① Push 1 onto the stack. [1]

② Pop and print the top element of the stack (here 1) and push all the neighbors of 1 onto the stack. Repeat this step until there are no elements left to push on the stack.

print 1. [2]

③ [6, 3] print 2. ~~[illegible]~~

④ [5, 4, 3] print 6. ⑤ [4, 3] print 5 ⑥ [3] print 4.

⑦ [7] print 3 ⑧ [ ] print 7.

As there are no elements left to be pushed and the stack is empty we will end the traversal.

∴ DFS traversal will be
1, 2, 6, 5, 4, 3, 7

Pg no. 18/19

Sarangyadh

Q4. B. ii)

→ Sequence: 90, 27, 7, 9, 18, 21, 3, 16, 11.

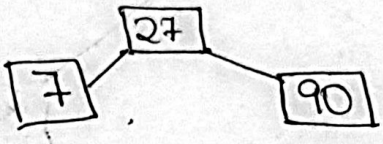
order = m = 3

max key = m-1 = 2.

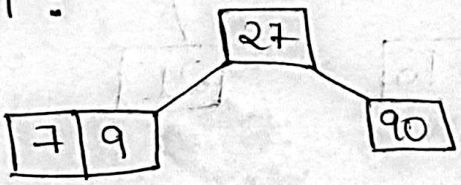
① Insert 90 : 90

② Insert 27 : 

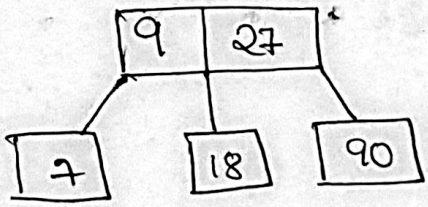
③ Insert 7 :



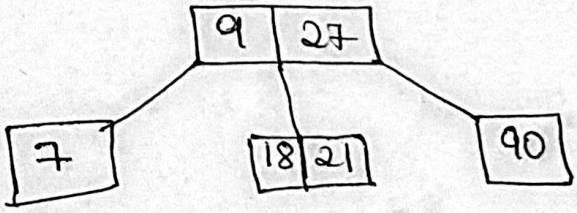
④ Insert 9 :



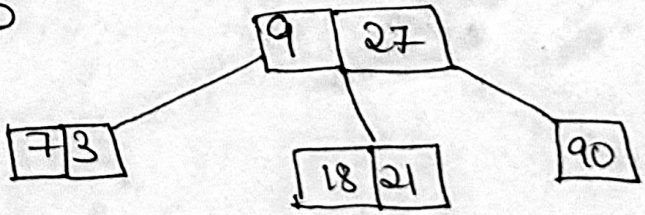
⑤ Insert 18 :



⑥ Insert 21 :

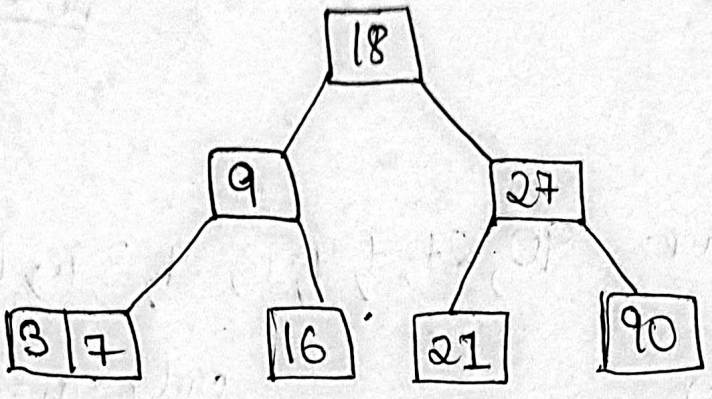


⑦ Insert 3



⑧ Insert 16

Saraswati



⑨ Insert 11