Assignment

Q-1.

(a) $A[\overset{201}{\overbrace{-100:100}}, \overset{56}{\overbrace{-5:50}}]$.

Base address = 10.
Size = 4.

$A[99, 49] = BA + Size[\text{no. of column}(i - lb_1) + (j - lb_2)]$

$= 10 + 4[56(99+100) + (49+5)]$.

$= 10 + 4(56 \times 199 + 54)$.

$= 10 + 4(11144 + 54)$

$= 10 + 4 \times 11198$

$=$ ~~10 +~~ 44802.

(b) The various ~~Ato~~ asymptotic ~~cm~~ notations are-

1 Big Oh notations (O)
2 Omega notation (Ω)
3 Theta notation (θ)

Big Oh notation- Big Oh notation is an asymptotic notation that measures the performance of an algorithm by simply providing the order of growth of the function

(c) The notations used in evaluation of arithmetic expression using prefix and postfix forms are-

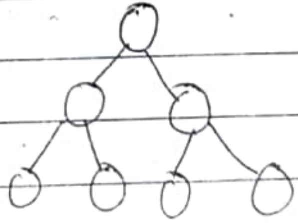
1 Prefix- operator <operand> <operand>
Eg - +AB, *AB.

2 Postfix - <operand> <operand> operator
Eg - AB+, AB*.

(d) The hashing ~~ft~~ functions based on various methods by which key value is found are-

1 K mod 10
2 K mod n.
3 Mid Square method
4 Folding method.

(e) Nodes = 7.



~~Heigt~~ Max. height = 3.

(f) $n = 10$

$$2^n + 1 = 2^{10} + 1$$
$$= 1025.$$

Total no. of moves = 1025.

(g) Connected graph- A connected graph is the one in which some path exists between every two vertices. There are no isolated nodes in connected graph

Strongly connected graph- A graph is said to be strongly connected if there is at least one directed ~~graph~~ path ~~s~~ from ~~the~~ every vertex to every other vertex

(h) Infix- A*(B+D)/E-F*(G+H/K).

Symbol	Stack	Postfix
A		A.
*	*	
(	*(	
B		AB
+	*(+	
)		AB+
/	/	AB+*
E		AB+*E
-	-	AB+*E/
F		AB+*E/F
*	-*	
(	-*(	
G		AB+*E/FG.
+	-*(+	
H		AB+*E/FGH

/   - * (+ /   AB+*E/FGH.

K)   AB+*E/FGHK/+*-

Postfix = AB+*E/FGHK/+*-

(f)(ii) For tree construction linked list is suitable because it is the most suitable and efficient data structure and it is easily accessible due to the concept of pointer used in it.

(g) The application of sparse matrix are-

1 The sparse matrices are useful for computing large scale operations that dense matrices can not handle.

2 It is used in ~~sol~~ solving partial differential equations.

Section-B

Q-2 AAA [5 : 50], BBB [-5 : 10], CCC [1 : 8]

(a) Length AAA = 50 - 5 + 1
= 46

~~(b)~~ Length BBB = 10 + 5 + 1
= 16

~~(c)~~ Length CCC = 8 - 1 + 1
= 8.

(b) Base = 300.

$w = 4$.

$AAA[15] = BA + w(i - lb)$

$= 300 + 4(15 - 5)$.

$= 300 + \cancel{40}40$

$= 340$.

$AAA[35] = 300 + 4(35 - 5)$.

$= 300 + 120$.

$= 420$.

AAA[55] will not have any location in array AAA because the ~~length~~ ~~size of the array is 46~~ greatest element in an array is 50.

3 In order to balance a tree, ~~there~~ there are four cases of rotations

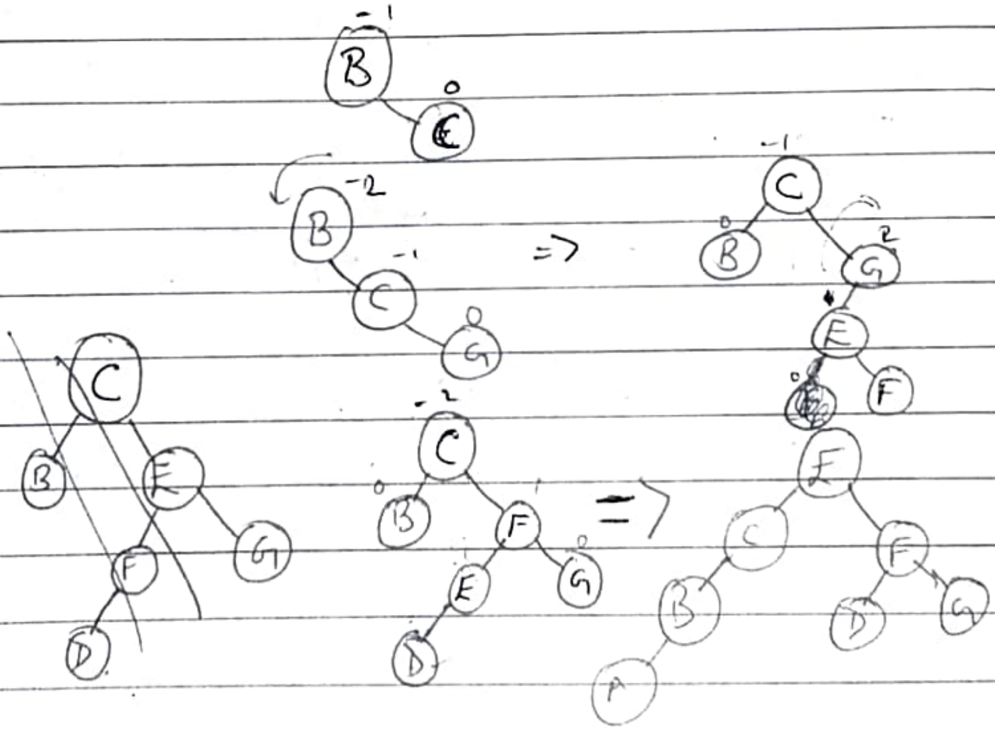
1 LL rotation - In LL rotation every node moves one position to left from the current position.

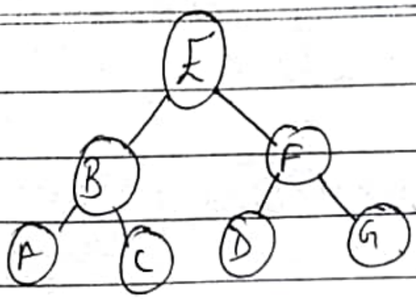
2 RR rotation: In RR rotation every node moves one position to right from the current position.

3 LR Rotation - The LR rotation is combination of single left rotation followed by single right rotation. In LR rotation, first every node moves one position to left then one position to right from the current position.

4 RL rotation - The RL rotation is the combination of single right rotation followed by single left rotation. In RL rotation, first every node moves one position to right then one position to left from the current position.

Construction of AVL tree
B, C, G, E, F, D, A.





4 A binary search tree is a binary tree. It can be represented by a linked data structure in which each node is an object. In addition to a key ~~fe~~ field, each node contains fields ~~right~~ left, right and P; ~~wt~~ which point to the nodes corr. to its left child, ~~th~~ its right child and its parent resp.

A non-empty binary search tree satisfies the following properties:-

(a) Every element has a key and no two elements have the same value.

(b) The keys, if any, in the left subtree of root are smaller than the key in the node.

(c) The keys, if any in the right subtree of the root are larger than the keys in the node.

(d) The left and right subtree of the root are also binary search tree.

Various operations of BST are-

1 **Searching**- Searching for a data in BST is much faster than in arrays or linked list. ~~Thee~~ The TREE-SEARCH algo. searches the tree root at x for a node whose key value equals to k. It returns a pointer to the node if it exists otherwise NIL.

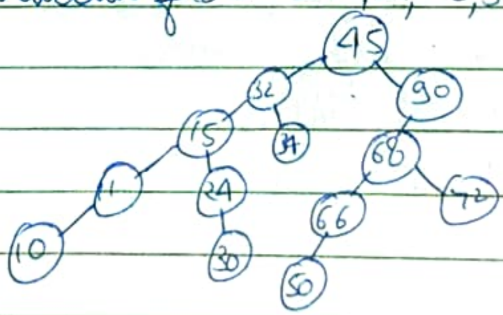
2 **Traversal** - All the traversal operations are applicable in ~~t~~ BST. The ~~two~~ inorder traversal on BST gives the sorted order of data in ascending order.

3. **Insertion**- To insert a new value w into a BST, we use the procedure TREE-INSERT.

4 **Deletion**- Deletion of a node ~~in B~~ from BST depends on the no. of its children.

~~Cons~~

Construction of BST = 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10



6 Implementation of stack using Linked List

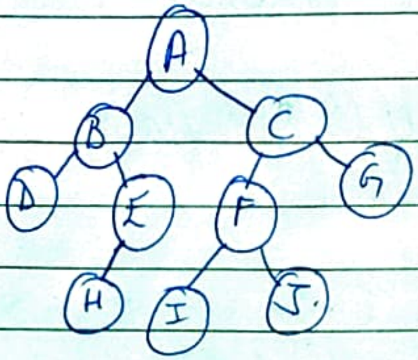
```
#include<stdio.h>
#include <alloc.h>.
struct node
 {
 int info;
struct node * link;
 };
  struct node* top;
  void main()
  {
 void create();
 void traverse();
 void push();
void pop();
create();
printf("stack is:");
 traverse();
  ~~pop()~~, push();
printf("After push the element in the stack is:"
traverse();
pop();
printf("After pop the element in the stack is:")
 traverse();
 getch();
}
void create()
{
```

```
struct node * ptr,* cpt;
char ch;
ptr = (structnode*) malloc (sizeof(struct node));
printf("Input first info");
scanf ("%d" &ptr -> info);
 ptr -> link = NULL;
 do
 {
 cpt =(struct node*) malloc(size of(struct node));
 printf (" Input next information");
scanf("% d", &cpt -> info);
 cpt -> link = ptr;
  ptr = cpt;
printf("Press <Y|N> for more information");
  ch = getch();
 }
   ~~while~~ while (ch=='y')
        { top = ptr;
        }
  void traverse ()
   {
    struct node* ptr;
  ~~pt~~ printf ("Traversing of stack : ");
   ptr = top;
  while ( ptr != NULL)
     {
      printf ("%d", ptr -> info);
      ptr = ptr -> link;
  ;}
```

```
void push()
{
 struct node* ptr;
 ptr=(struct node*)malloc(sizeof(struct node));
 if(ptr==NULL)
  {
   printf("Overflow");
   return;
  }
 printf("Enter new node");
 scanf("%d" &ptr->info);
 ptr->link=top;
 top=ptr;
}
void pop()
 {
  struct node* ptr;
  if(top==NULL)
   {
    printf("Underflow");
    return;
   }
  ptr=top;
  top=ptr->link;
  free(ptr);
 }
```

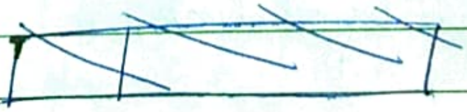
7 Binary Tree.

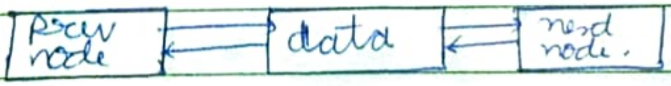
Inorder - DBHEAIFJCG
Preorder - ABDEHCFIJG.



Section - C

Q10 Doubly Linked List - The doubly or two-way linked list ~~uses~~ uses double set of pointers ~~o~~, One pointing to the next node and the other pointing to the ~~prec~~ preceding node. In doubly linked list, all nodes are ~~linke~~ linked together by multiple links which help in accessing both the successor and predecessor node for any arbitrary node within the list.





Program to create a doubly linked list.

```
#include<stdio.h>
#include<alloc.h>
 struct node
  {
   int info;
   struct node* lpt;
   struct node* rpt;
  };
struct node* first;
void main()
 {
  create();
  getch();
 }
void create()
 {
  struct node* ptr, *cpt;
  char ch;
 ptr =(struct node*)malloc(sizeof(struct node));
 printf("Input first node: ");
 scanf("%d", &ptr->info);
 ptr->lpt =NULL;
 first= ptr;
  do
   {
    cpt=(struct node*)malloc(sizeof(struct node));
 printf=("Input next node: ");
scanf ("%d", & cpt → info);
ptr → rpt = cpt;
cpt → lpt = ptr;
ptr = cpt;
printf("press Y/N for more node");
ch= getch();
}
while( ch == 'Y');
ptr → rpt = NULL;
}
```

12 A B-Tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertion and deletion in logarithmic time.

Applications of B-Tree

The main application of B-Tree is the organization of huge collection of records into a file structure. The organization should be in such a way that any record in it can be searched very efficiently ~~i.~~ i.e. insertion, deletion and modification operations can be carried out perfectly and efficiently

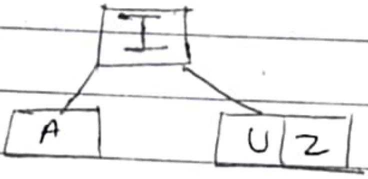
Insertion - Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V

Z → Z

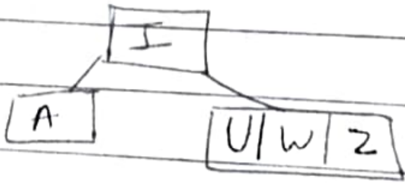
U → |U|Z|

A → |A|U|Z|

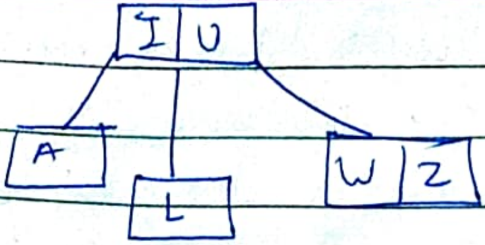
I → |A|I|U|Z|



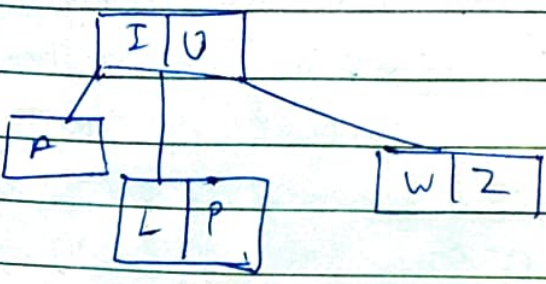
W →



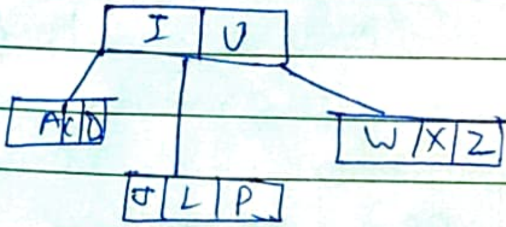
L →



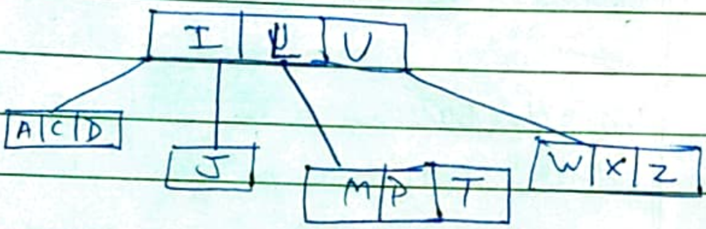
P →



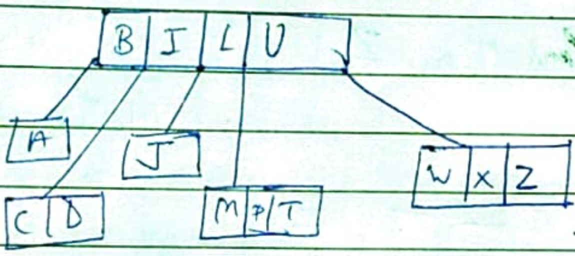
X, C, J, D

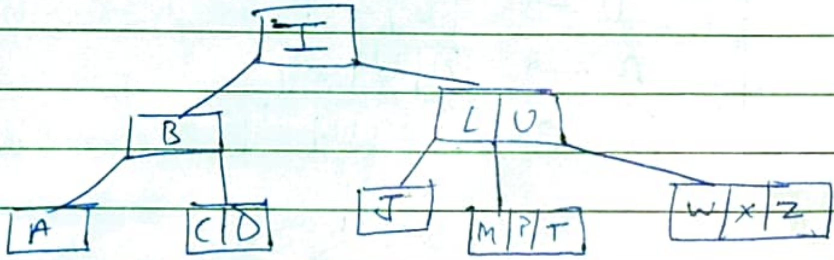


MT →

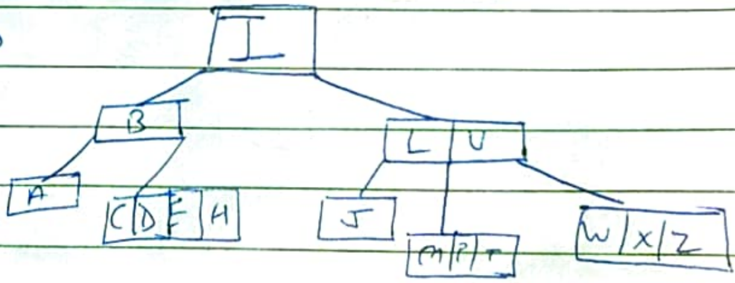


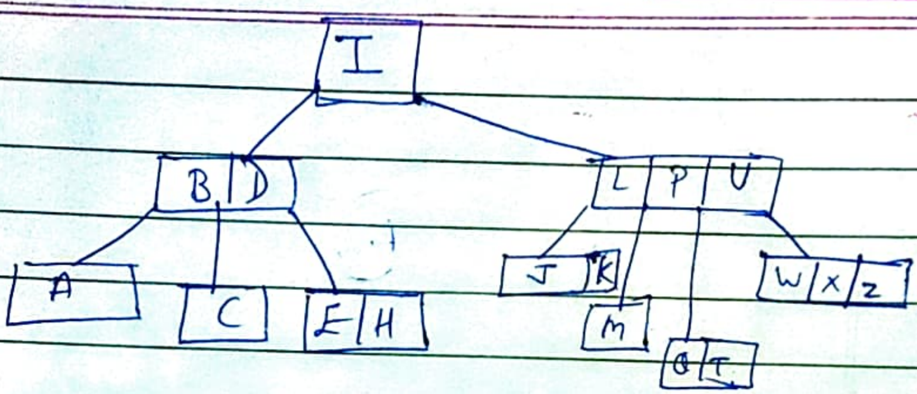
B →

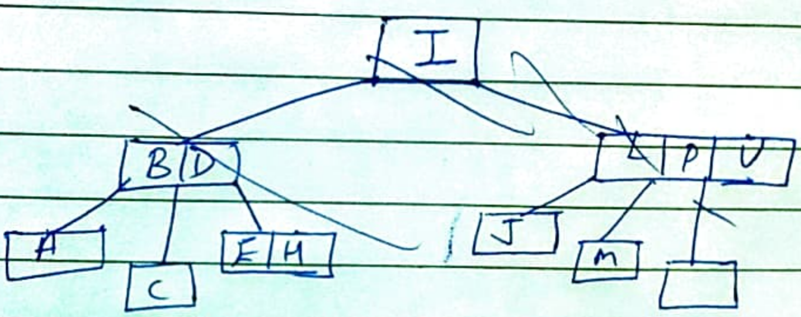


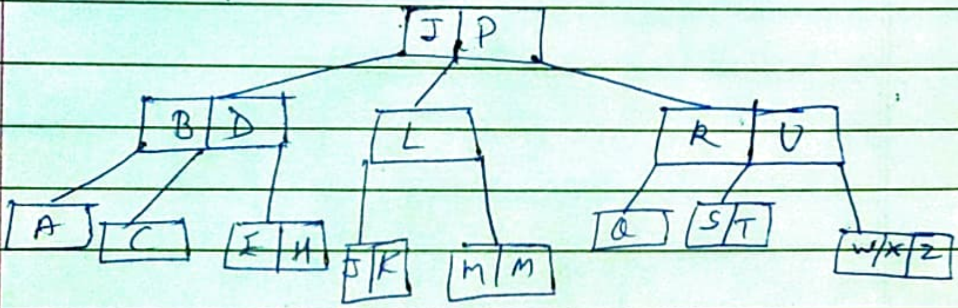


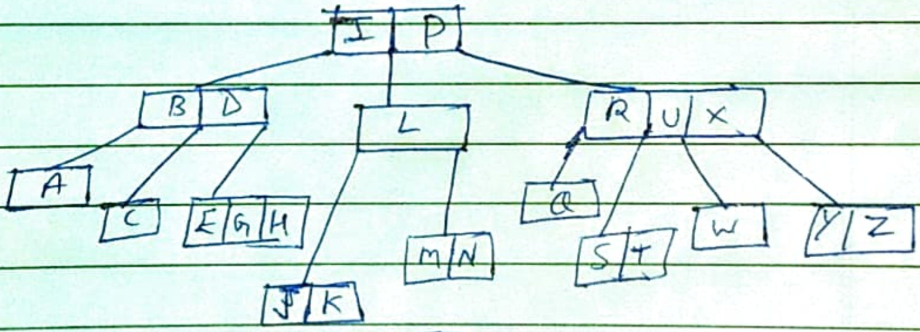
E, H →

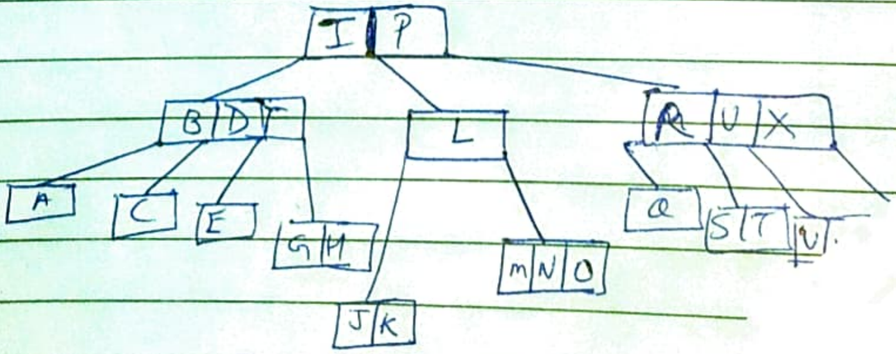



I
B D
L P U
A
C
E H
J K
M
Q T
W X Z


I
B D
L P U
A
C
E H
J
M


J P
B D
L
R U
A
C
E H
J K
M M
Q
S T
W X Z


I P
B D
L
R U X
A
C
E G H
Q
W
Y Z
M N
S T
J K


I P
B D F
L
R U X
A
C
E
G H
Q
S T
V
M N O
J K