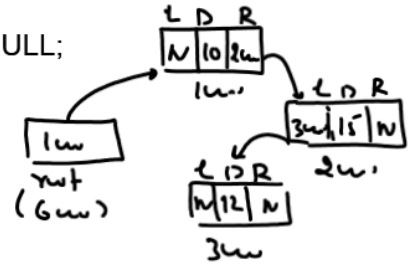
Creating a BST

```
void add(struct bst **ps,int x)
{
  struct bst *p,*temp,*prev;
  p=(struct bst *)malloc(sizeof(struct bst));
  if(p==NULL)
  {
    printf("Insufficient Memory");
    return;
 }
p->data=x;
p->left=p->right=NULL;
if(*ps==NULL)
{
  *ps=p;
  return;
}
```



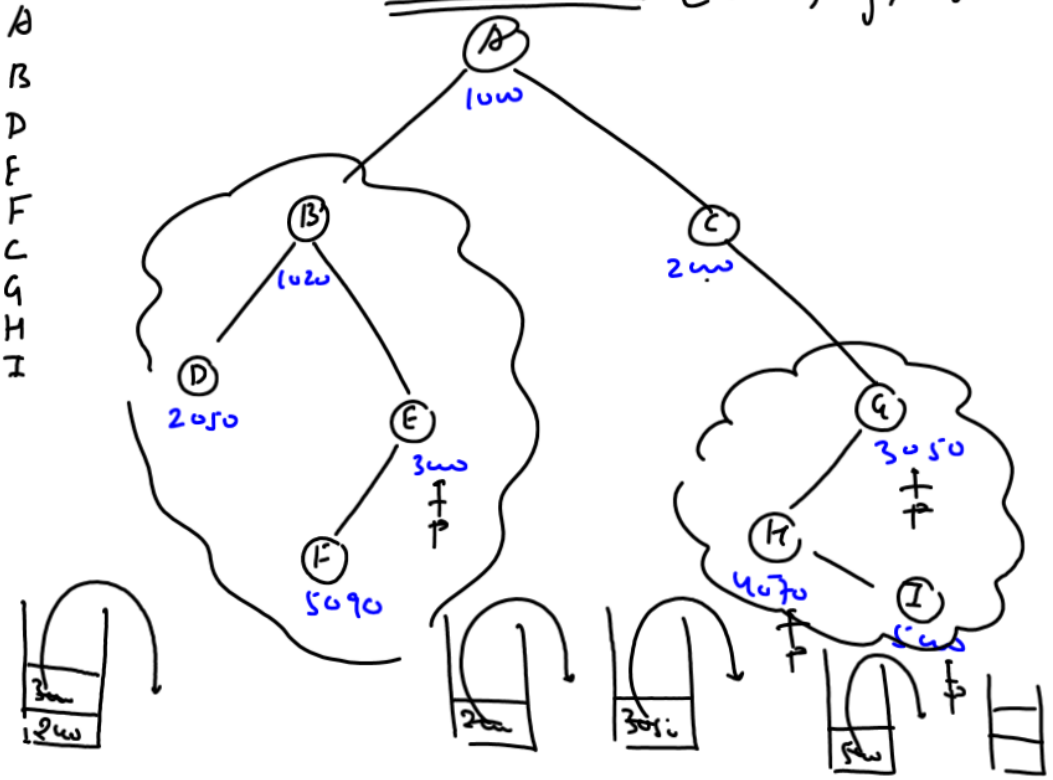

```
temp=*ps;
while(temp!=NULL)
{
  prev=temp;
  if(temp->data>x)
     temp=temp->left;
  else
     temp=temp->right;
}
if(prev->data>x)
    prev->left=p;
else
    prev->right=p;
}
```

Tree Traversal Algorithms

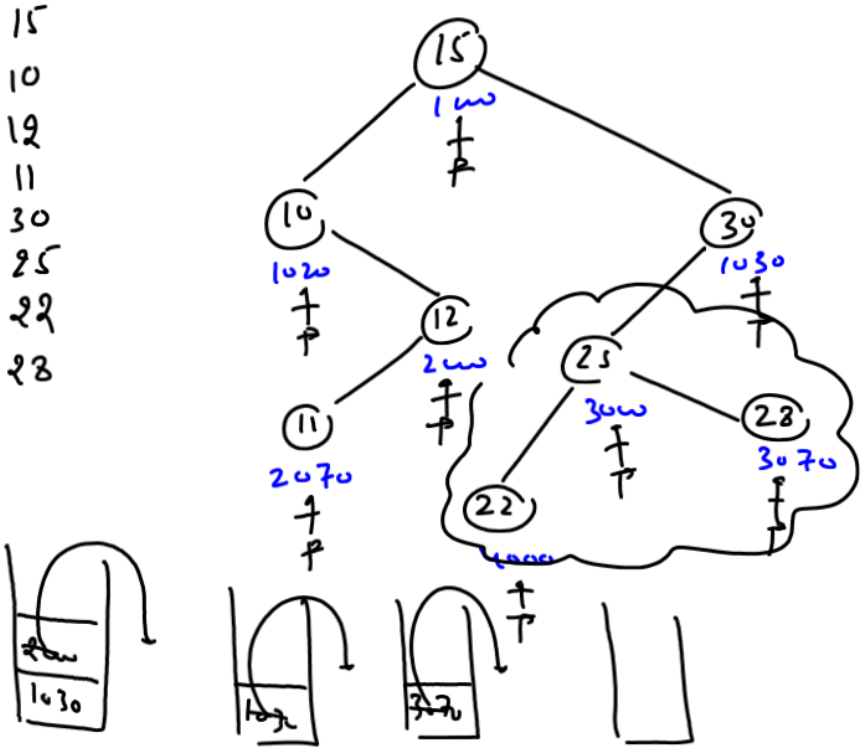
1. Preorder Traversal (Root, Left, Right)
2. Inorder Traversal (Left, Root, Right)
3. Postorder Traversal (Left, Right, Root)

Preorder Traversal (Root, Left, Right)

A
B
D
F
F
C
G
H
I



15
10
12
11
30
25
22
28



Algorithm For Preorder Traversal
==========

1. Check whether the TREE is empty or not.

2. If it is empty , then print EMPTY TREE and return.

3.Starting from the ROOT node do the following:

   a. Print the CURRENT node data
   b. If the node has right child , then PUSH it in the STACK
   c. Move towards LEFT

4. Repeat sept 3 until POINTER becomes NULL.

5.Check the STACK:

   a. If it is EMPTY then FINISH and return.
   b. POP the top node from the STACK and goto STEP 3

IMPLEMENTING PEORDER TRAVERSAL

```
struct bst
{
   struct bst *left;
   int data;
   struct bst *right;
};
struct Stack
{
    struct bst * arr[10];
    int tos;
};
void push(struct Stack *,struct bst *);
struct bst * pop(struct Stack *);
void add(struct bst **,int);
void preorder(struct bst *);
```

```
int main()
{
  struct bst *root=NULL;
   add(&root,15);
   add(&root,10);
   add(&root,12);
    ........
   preorder(root);
   return 0;
}
```