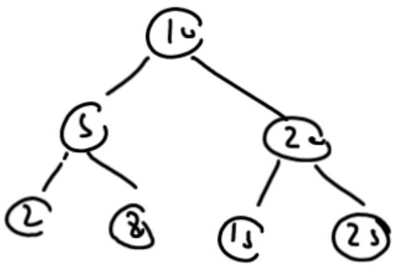
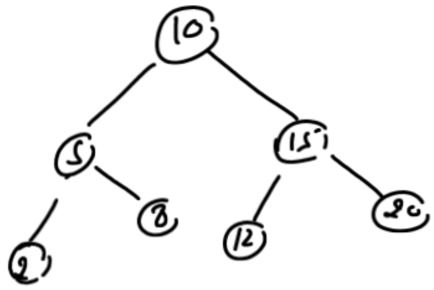
```
void preorder( struct bst *p)
{
    struct Stack S;
    if( p== NULL)
    {
        printf(" Empty Tree");
        return;
    }
    S.top=-1;

push_nodes:
    while( p!=NULL)
    {
        printf("\n %d", p->data);
        if( p->right != NULL)
            push(&S, p->right);
        p=p->left;
    }
    p= pop( &S);
    if( p == NULL)
        return;
    else
        goto push_nodes;
}
```

10
5
2
8
20
15
25



10, 5, 2, 8, 15, 12, 20

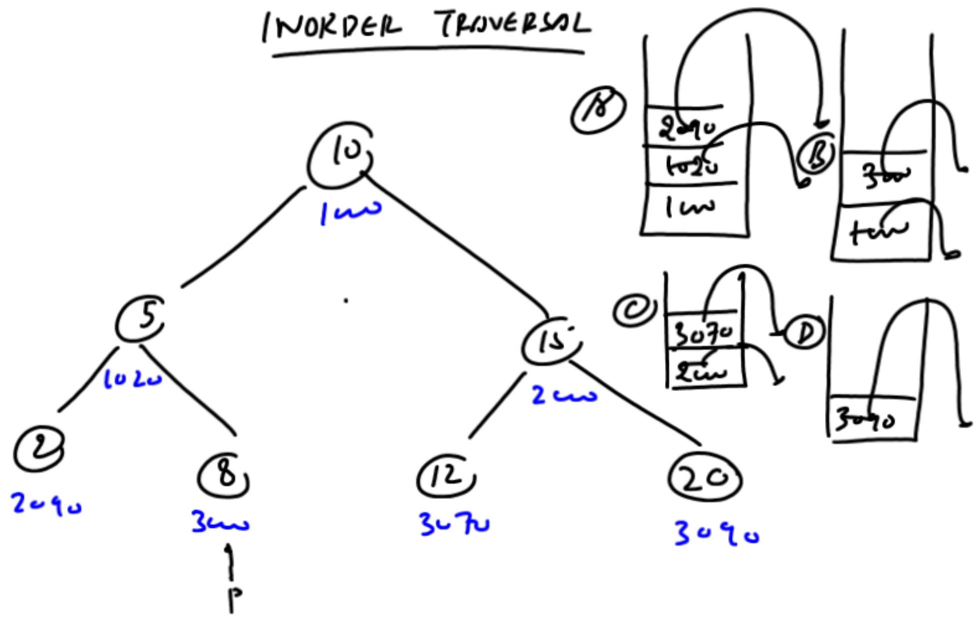


```
while ( p!=NULL)
{
    while ( p! = NULL)
    {
        printf("\n %d", p->data);
        if( p->right != NULL)
            push(&S, p->right);
        p=p->left;
    }
    p=pop(&S);
}
```

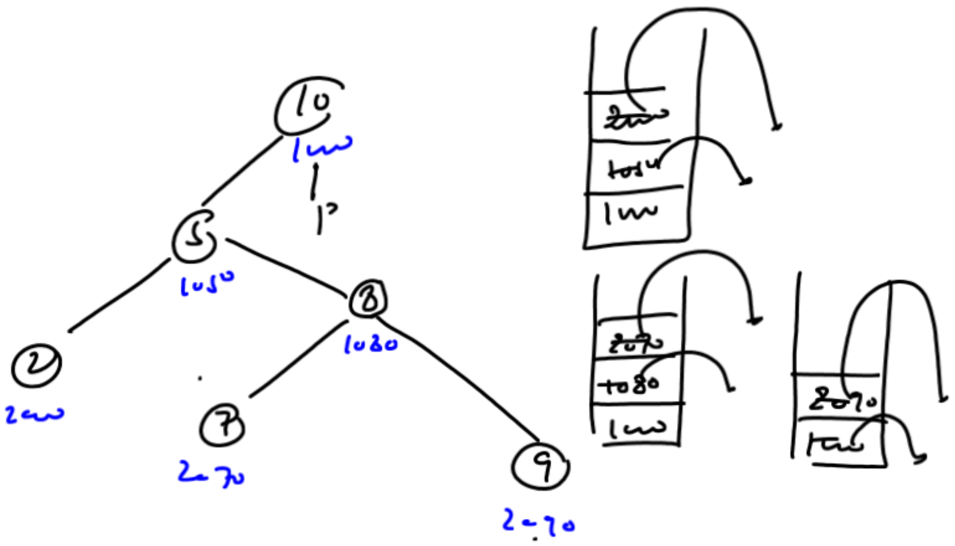
10
5
2
8
15
12
20

2
5
8
10
12
15
20

INORDER TRAVERSAL



2
5
7
8
9
10

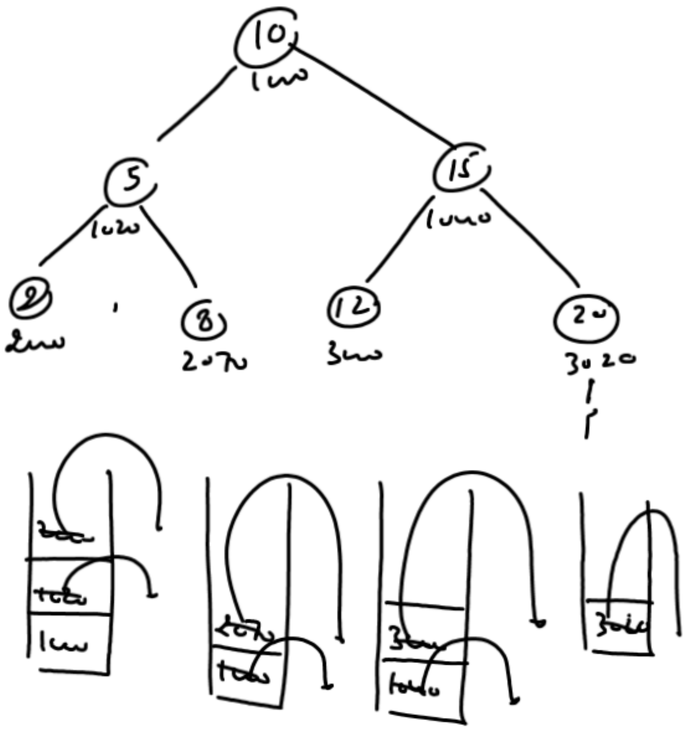


Algorithm For Inorder Traversal

1. Check whether the tree is empty or not .
2. If it is empty then print EMPTY TREE and return.
3. Proceed down to left most path pushing each node in the STACK.
4. STOP when pointer becomes NULL.
5. POP the top node from the STACK.
6. If we get NULL from STACK then finish and return.
7. If we get a NODE then:
    a. Print the data part.
    b. Check whether this node has a RIGHT child.
    c. If it has a RIGHT child then move the pointer to RIGHT and goto step 3.
    d. If no RIGHT child is present goto step 5

```
void inorder(struct bst *p)
{
   struct Stack S;
   if(p==NULL)
     {
         printf("Empty Tree");
         return;
     }
   S.tos=-1;
   push_nodes:
                       while(p!=NULL)
                       {
                          push(&S,p);
                          p=p->left;
                       }
                       p=pop(&S);
                       while(p!=NULL)
                       {
                           printf("\n%d",p->data);
                           if(p->right!=NULL)
                             {
                                  p=p->right;
                                  goto push_nodes;
                             }
                            p=pop(&S);
                        }
}
```

2
5
8
10
12
15
20


10
1000
5
1020
15
1040
2
2000
8
2020
12
3000
20
3020
1000
1020
2020
3000
1040
3020