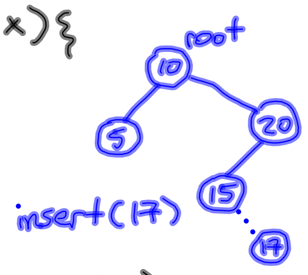
Creating a Binary Tree

```
/* Create a new node */
BTREE new_node (int data){
        BTREE p;
        p=(BTREE)malloc(sizeof(struct node));
        p->data=data;
        p->left=p->right=NULL;
        return p;
}
```

```
/* Insert data to the BT
 Rule: each right node will be greater
 than its parent and each left node
 will be less than its parent */
```

} BST



```
BTREE insert(BTREE root, int x){
      if (root==NULL)
          root= new_node(x);
      else {
          if(x<root->data)
              root->left=insert(root->left,x);
          else
              root->right=insert(root->right,x);
      }
      return root;
}
```

Ex: Get number from user till -1

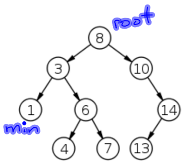
```
main( ){
        BTREE myroot=NULL;
        int i;
        scanf("%d", &i);
        while(i!=-1){
              myroot= insert(myroot,i);
              scanf("%d", &i);
        }
        inorder(myroot);
}
```

BST Traversal

```
void  inorder ( BTREE root){
        if( root != NULL) {
preorder    inorder ( root->left);
            printf ( "%d\t", root->data);
postorder   inorder ( root->right);
        }
}
```

Finding minimum element in a BST

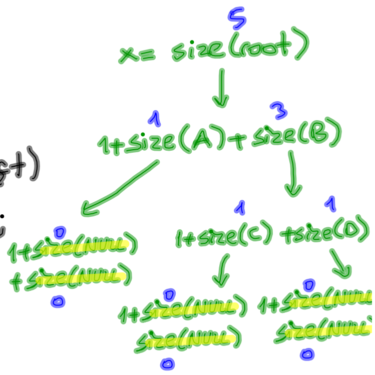
```
int  BSTmin ( BTREE root){
      if (root != NULL) {
         while ( root->left != NULL)
               root = root->left;
         return  root->data;
      }else
         return  -100;
}
```

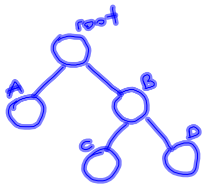


HW: Write the function above recursively

Finding the number of elements in a BT

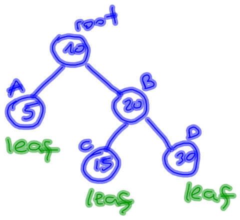
```
int  size ( BTREE root){
   if ( root == NULL)
        return 0;
   else
        return 1+size(root->left)
              +size (root->right);
}
```

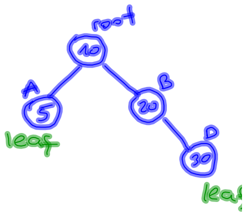




Finding the number of leaves in a BST.

```
int  leaves (BTREE root)
   if(root != NULL){
      if ( root->left == root->right)
            return 1;
   else
         return leaves(root->left)
               + leaves(root->right);
   }else
      return 0;
}
```


root
10
A
5
leaf
B
20
C
15
leaf
D
30
leaf


root
10
A
5
leaf
B
20
D
30
leaf