```
// C program to demonstrate the insert operation in binary search tree.
#include<stdio.h>
#include<stdlib.h>

struct node
{
 int key;
 struct node *left, *right;
};

// A utility function to create a neww BST node
struct node *newNode(int item)
{
 struct node *temp = (struct node *)malloc(sizeof(struct node));
 temp->key = item;
 temp->left = temp->right = NULL;
 return temp;
}

// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
 if (root != NULL)
 {
  inorder(root->left);
  printf("%d \n", root->key);
  inorder(root->right);
 }
}

/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
 /* If the tree is empty, return a new node */
 if (node == NULL) return newNode(key);

 /* Otherwise, recur down the tree */
 if (key < node->key)
  node->left = insert(node->left, key);
 else if (key > node->key)
  node->right = insert(node->right, key);

 /* return the (unchanged) node pointer */
 return node;
}

// Driver Program to test above functions
int main()
{
 /* Let us create following BST
   50
  /  \
```

```
 30  70
 / \ / \
20 40 60 80 */
 struct node *root = NULL;
 root = insert(root, 50);
 insert(root, 30);
 insert(root, 20);
 insert(root, 40);
 insert(root, 70);
 insert(root, 60);
 insert(root, 80);

 // print inoder traversal of the BST
 inorder(root);

 return 0;
}
```

Problem 2:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
 int data;
 struct node* leftlink;
 struct node* rightlink;
}*root=NULL;
void inorder(struct node *root)
{
 if(root==NULL)
  return;
 inorder(root->leftlink);
 printf("%d ",root->data);
 inorder(root->rightlink);
}
struct node* insert(struct node *root,int data)
{
 if(root==NULL)
 {
  root=(struct node*)malloc(sizeof(struct node));
  root->leftlink=root->rightlink=NULL;
  root->data=data;
 }
 else if(data<root->data)
 {
  root->leftlink=insert(root->leftlink,data);
 }
 else if(data>root->data)
 {
  root->rightlink=insert(root->rightlink,data);
 }
 return root;
}
int main()
```

```
{
 int n,i,data,s;
 printf("enter the no of elements of tree");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("enter the element of tree");
  scanf("%d",&data);
  root=insert(root,data);
 }
 printf("preorder :");
 inorder(root);
 printf("\n");
}
```

Problem 3:

```
#include<stdio.h>
main()
{
 int a[20],i,n,s,flag=0;
 printf("enter the no elements of array");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("enter %d element of array :",i+1);
  scanf("%d",&a[i]);
 }
 printf("enter the element to search:");
 scanf("%d",&s);
 for(i=0;i<n;i++)
 {
  if(a[i]==s)
  {
   printf("element found");
   flag=1;
  }
 }
 if(flag==0)
  printf("element not found");
}
```

Output

Problem 4:

```
#include<stdio.h>
main()
{
```

```
 int a[20],i,n,s,mid,j;
 printf("enter the no elements of array\n");
 scanf("%d",&n);
 printf("enter sorted array only:\n");
 for(i=0;i<n;i++)
 {
  printf("enter %d element of array :",i+1);
  scanf("%d",&a[i]);
 }
 printf("enter the element to search:");
 scanf("%d",&s);
 i=0;
 j=n-1;
 while(i<=j)
 {
  mid=(i+j)/2;
  if(a[mid]==s)
  {
   printf("element found");
   break;
  }
  else
  {
   if(s<a[mid])
   {
    j=mid-1;
   }
   else
   {
    i=mid+1;
   }
  }
 }
 if(i>j)
 {
  printf("element not found");
 }
}
```

Out put