LAB TASK – 2

1. Programs for summation of series 1+X+X^2+X^3+...with different time
   Complexities:

Method 1

```
#include <stdio.h>
#include <math.h>

int main() {
   int n, i;
   int x, sum = 0, a = 1;

   printf("Enter value of x: ");
   scanf("%d", &x);
   printf("Enter value of n: ");
   scanf("%d", &n);

   for (i = 0; i <= n; i++) {
      sum += a;
     a*= x;
 }

   printf("Sum of series  = %d\n", sum);
   return 0;
}
```

```
Enter value of x: 2
Enter value of n: 3
Sum of series  = 15


=== Code Execution Successful ===
```

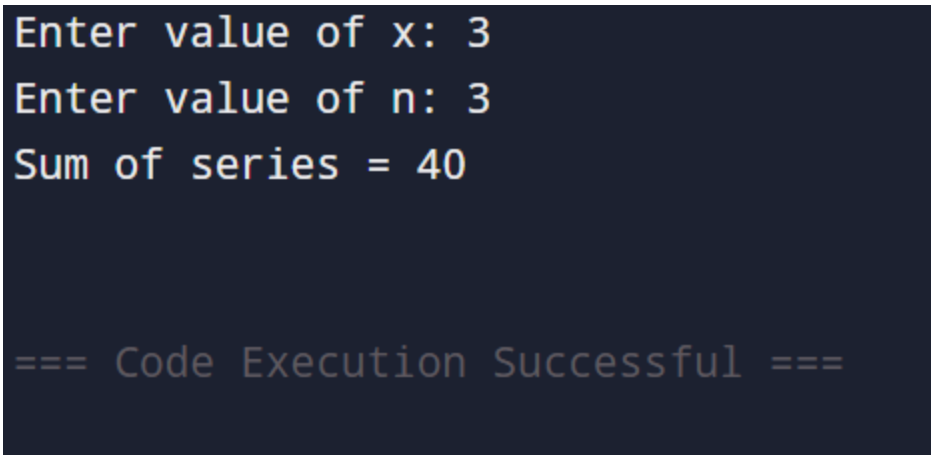
Method 2

```
#include <stdio.h>
#include <math.h>
int main() {
   int n, i;
   int x, sum = 0;

   printf("Enter value of x: ");
   scanf("%d", &x);
   printf("Enter value of n: ");
   scanf("%d", &n);

   for (i = 0; i <= n; i++) {
      sum += pow(x, i);
   }

   printf("Sum of series = %d\n", sum);
   return 0;
}
```



2. Create a Binary Search Tree and perform the insertion, deletion operations.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{ int data;
struct Node* left;
struct Node* right;
};
struct Node* newNode(int value)
{ struct Node* node = (struct Node*)malloc(sizeof(struct Node));
 node->data = value;
 node->left = node->right = NULL;
return node;
}
struct Node* insert(struct Node* root, int value)
{ if (root == NULL)
```

```
return newNode(value);

if                (value                <                root->data)
    root->left          =          insert(root->left,          value);
else            if            (value            >            root->data)
    root->right          =          insert(root->right,          value);

return                                                          root;


}

struct Node* minValueNode(struct Node* node)

{ struct Node* current = node;

 while (current && current->left != NULL) current = current->left;

 return current;

}

struct Node* deleteNode(struct Node* root, int key)

{

 if (root == NULL)

return root;

if                 (key                 <                 root->data)
    root->left          =          deleteNode(root->left,          key);
else            if            (key            >            root->data)
    root->right          =          deleteNode(root->right,          key);
else                                                                {
    //    Case    1:    node    with    one    child    or    no    child
    if           (root->left           ==           NULL)           {
        struct         Node*         temp         =         root->right;
        free(root);
        return                                                      temp;
    }
    else        if        (root->right        ==        NULL)        {
```

```
        struct Node* temp = root->left;
        free(root);
        return temp;
    }

    // Case 2: node with two children
    struct Node* temp = minValueNode(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data);
}
return root;


}

void inorder(struct Node* root)

{ if (root != NULL)

{ inorder(root->left);

printf("%d ", root->data);

inorder(root->right); }

}

int main()

{ struct Node* root = NULL;

root=insert(root,5);

root=insert(root,3);

root=insert(root,2);

root=insert(root,4);

root=insert(root,7);

root=insert(root,6);

root=insert(root,8);
```

```
printf("Inorder traversal after insertion: ");
inorder(root);
printf("\n");


root = deleteNode(root, 4);
printf("Inorder traversal after deleting 4: ");
inorder(root);
printf("\n");

root = deleteNode(root, 3);
printf("Inorder traversal after deleting 3: ");
inorder(root);
printf("\n");

root = deleteNode(root, 5);
printf("Inorder traversal after deleting 5: ");
inorder(root);
printf("\n");

return 0;


}
```

Output

```
Inorder traversal after insertion: 2 3 4 5 6 7 8
Inorder traversal after deleting 4: 2 3 5 6 7 8
Inorder traversal after deleting 3: 2 5 6 7 8
Inorder traversal after deleting 5: 2 6 7 8


=== Code Execution Successful ===
```