Write a c program to implement binary tree traversal

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data) {
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;

      while(1) {
         parent = current;


         if(data < parent->data) {
            current = current->leftChild;


            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }
         else {
            current = current->rightChild;


            if(current == NULL) {
               parent->rightChild = tempNode;
```

```
            parent->rightChild = tempNode;
            return;
         }
      }
   }
}

struct node* search(int data) {
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);

      if(current->data > data) {
         current = current->leftChild;
      }

      else {
         current = current->rightChild;
      }

      if(current == NULL) {
         return NULL;
      }
   }

   return current;
}

void pre_order_traversal(struct node* root) {
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}

void inorder_traversal(struct node* root) {
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}

void post_order_traversal(struct node* root) {
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}

int main() {
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

   for(i = 0; i < 7; i++)
      insert(array[i]);

   i = 31;
   struct node * temp = search(i);

   if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
   }else {
      printf("[ x ] Element not found (%d).\n", i);
   }

   i = 15;
   temp = search(i);

   if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
   }else {
      printf("[ x ] Element not found (%d).\n", i);
   }

   printf("\nPreorder traversal: ");
   pre_order_traversal(root);

   printf("\nInorder traversal: ");
   inorder_traversal(root);

   printf("\nPost order traversal: ");
   post_order_traversal(root);

   return 0;
}
```

```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
```

2) write a c program to implement AVL tree ?

```c
#include<stdio.h>
#include<stdlib.h>

struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};


int height(struct Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

int max(int a, int b)
{
    return (a > b)? a : b;
}


struct Node* newNode(int key)
{
    struct Node* node = (struct Node*)
                        malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return(node);
}

struct Node *rightRotate(struct Node *y)
{
    struct Node *x = y->left;
    struct Node *T2 = x->right;


    x->right = y;
    y->left = T2;
    y->height = max(height(y->left),
                    height(y->right)) + 1;
    x->height = max(height(x->left),
                    height(x->right)) + 1;

    return x;
}


struct Node *leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    struct Node *T2 = y->left;


    y->left = x;
    x->right = T2;

    x->height = max(height(x->left),
                    height(x->right)) + 1;
    y->height = max(height(y->left),
                    height(y->right)) + 1;

    return y;
}

int getBalance(struct Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}


struct Node* insert(struct Node* node, int key)
{

    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;


    node->height = 1 + max(height(node->left),
                           height(node->right));


    int balance = getBalance(node);
```

```c
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);


    if (balance < -1 && key > node->right->key)
        return leftRotate(node);


    if (balance > 1 && key > node->left->key)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }


    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }


    return node;
}

void preOrder(struct Node *root)
{
    if(root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}


int main()
{
struct Node *root = NULL;
root = insert(root, 10);
root = insert(root, 20);
root = insert(root, 30);
root = insert(root, 40);
root = insert(root, 50);
root = insert(root, 25);
printf("Preorder traversal of the constructed AVL"
       " tree is \n");
preOrder(root);
return 0;
}
```

```
Preorder traversal of the constructed AVL tree is
30 20 10 25 40 50
```

3) write a c program to implement hashing using linear probing technique?

```c
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10

int h[TABLE_SIZE]={NULL};

void insert()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
    {

     index=(hkey+i)%TABLE_SIZE;

     if(h[index] == NULL)
     {
        h[index]=key;
        break;
     }

    }

    if(i == TABLE_SIZE)

     printf("\nelement cannot be inserted\n");
}
void search()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
    index=(hkey+i)%TABLE_SIZE;
    if(h[index]==key)
    {
      printf("value is found at index %d",index);
      break;
    }
  }
  if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE_SIZE; i++)

  printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
    int opt,i;
    while(1)
    {
        printf("Press\n 1. Insert\n 2. Display \n 3. Search \n 4. Exit\n ");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

```
enter a value to insert into hash table
3
Press
 1. Insert
 2. Display
 3. Search
 4. Exit
 2

elements in the hash table are

at index 0       value =  0
at index 1       value =  1
at index 2       value =  2
at index 3       value =  2
at index 4       value =  3
```

4) write a c program to implement bubble sort?

```c
#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1])
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
    printf("%d\n", array[c]);

  return 0;
}
```

```
Enter number of elements
5
Enter 5 integers
23
54
12
3
69
Sorted list in ascending order:
3
12
23
54
69
```

5) write a c program to implement selection sort?

```c
#include <stdio.h>

void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;


    for (i = 0; i < n - 1; i++) {

        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;


        swap(&arr[min_idx], &arr[i]);
    }
}

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int arr[] = { 64, 25, 12, 22, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

```
Sorted array:
11 12 22 25 64

--------------------------------
```

6) write a c program to implement insertion sort?

```c
#include <stdio.h>
void insert(int a[], int n)
{
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;

        while(j>=0 && temp <= a[j])
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}

void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}
  int main()
{
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    insert(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
```

7) write a c program to implement quick sort?

```c
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;
      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
         i++;
         while(number[j]>number[pivot])
         j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }
      temp=number[pivot];
      number[pivot]=number[j];
      number[j]=temp;
      quicksort(number,first,j-1);
      quicksort(number,j+1,last);
   }
}
int main(){
   int i, count, number[25];
   printf("How many elements are u going to enter?: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
   scanf("%d",&number[i]);
   quicksort(number,0,count-1);
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
   printf(" %d",number[i]);
   return 0;
}
```

```
How many elements are u going to enter?: 5
Enter 5 elements: 1
56
78
3
40
Order of Sorted elements:  1 3 40 56 78
```

8) write a c program to implement merge sort?

```c
#include <stdio.h>
#define max 10
int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high) {
   int l1, l2, i;

   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }

   while(l1 <= mid)
      b[i++] = a[l1++];

   while(l2 <= high)
      b[i++] = a[l2++];

   for(i = low; i <= high; i++)
      a[i] = b[i];
}

void sort(int low, int high) {
   int mid;

   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
}

int main() {
   int i;

   printf("List before sorting\n");

   for(i = 0; i <= max; i++)
      printf("%d ", a[i]);

   sort(0, max);

   printf("\nList after sorting\n");

   for(i = 0; i <= max; i++)
      printf("%d ", a[i]);
}
```

```
List before sorting
10 14 19 26 27 31 33 35 42 44 0
List after sorting
0 10 14 19 26 27 31 33 35 42 44
--------------------------------
```