1)Linked Lists:

1.1)Singly Linked List(Insertion and Deletion):

Source Code:

#include <iostream>

using namespace std;

class node

{

public:

    int data;

    node \*next;

    node(int data)

    {

*this*->data = data;

*this*->next = NULL;

    }

};

void insertatfront(node \*&head, int d)

{

    node \*temp = head;

    node \*ptr = new node(d);

    if (head == NULL)

    {

        head = ptr;

    }

    ptr->next = head;

    head = ptr;

}

void insertatEnd(node \*&head, int d)

{

    node \*temp = head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    node \*ptr = new node(d);

    temp->next = ptr;

}

void insertatpos(node \*&head, int d, int pos)

{

    node \*temp = head;

    int i = 0;

    while (i < pos)

    {

        temp = temp->next;

        i++;

    }

    node \*ptr = new node(d);

    ptr->next = temp->next;

    temp->next = ptr;

}

void print(node \*&head)

{

    node \*temp = head;

    while (temp != NULL)

    {

        cout << temp->data << " ";

        temp = temp->next;

    }

}

void deletenode(int position, node \*&head)

{

    if (position == 1)

    {

        node \*temp = head;

        head = head->next;

        temp->next = NULL;

        delete temp;

    }

    else

    {

        node \*current = head;

        node \*previous = NULL;

        int count = 1;

        while (count < position)

        {

            previous = current;

            current = current->next;

            count++;

        }

        previous->next = current->next;

        current->next = NULL;

        delete current;

    }

}

int main()

{

    node \*head = new node(10);

    insertatfront(head, 69);

    print(head);

    cout << endl;

    insertatEnd(head, 58);

    print(head);

    cout << endl;

    insertatpos(head, 45, 1);

    print(head);

    cout<<endl;

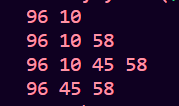
    deletenode(2,head);

    print(head);

    return 0;

}

OUTPUT:



1.2)Doubly Linked List(Insertion and Deletion):

Source Code:  
#include <iostream>

using namespace std;

class node {

public:

    int data;

    node\* next;

    node\* prev;

    node(int data) {

*this*->data = data;

*this*->next = NULL;

*this*->prev = NULL;

    }

    ~node() {

        cout << "Free " << data << endl;

    }

};

void print(node\* head) {

    node\* temp = head;

    while (temp != NULL) {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}

void insertatfront(node\* &head, int d) {

    node\* ptr = new node(d);

    ptr->next = head;

    if (head != NULL) {

        head->prev = ptr;

    }

    head = ptr;

}

void insertatEnd(node\* &head, int d) {

    node\* temp = head;

    node\* ptr = new node(d);

    if (temp == NULL) {

        head = ptr;

        return;

    }

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = ptr;

    ptr->prev = temp;

}

void insertatpos(node\* &head, int d, int pos) {

    if (pos <= 0) {

        cout << "Invalid position" << endl;

        return;

    }

    if (pos == 1 || head == NULL) {

        insertatfront(head, d);

        return;

    }

    node\* temp = head;

    node\* ptr = new node(d);

    int count = 1;

    while (count < pos - 1 && temp->next != NULL) {

        temp = temp->next;

        count++;

    }

    ptr->next = temp->next;

    if (temp->next != NULL) {

        temp->next->prev = ptr;

    }

    temp->next = ptr;

    ptr->prev = temp;

}

void deleteNode(node\* &head, int value) {

    if (head == NULL) {

        return;

    }

    if (head->data == value) {

        node\* temp = head;

        head = head->next;

        if (head != NULL) {

            head->prev = NULL;

        }

        delete temp;

        return;

    }

    node\* current = head;

    while (current != NULL && current->data != value) {

        current = current->next;

    }

    if (current == NULL) {

        cout << "Node with value " << value << " not found." << endl;

        return;

    }

    if (current->prev != NULL) {

        current->prev->next = current->next;

    }

    if (current->next != NULL) {

        current->next->prev = current->prev;

    }

    delete current;

}

int main() {

    node\* head = new node(10);

    insertatfront(head, 10);

    print(head);

    cout << endl;

    insertatEnd(head, 234);

    print(head);

    cout << endl;

    insertatpos(head,34,1);

    print(head);

    cout<<endl;

    insertatpos(head, 99, 2);

    print(head);

    cout << endl;

    deleteNode(head, 54);

    print(head);

    cout << endl;

    deleteNode(head, 10);

    print(head);

    cout << endl;

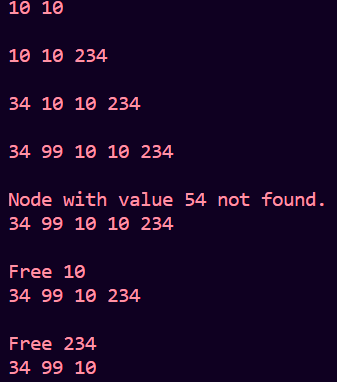
    deleteNode(head, 234);

    print(head);

    return 0;

}

OUTPUT:



2)Stack and Queue:  
  
2.1)Stack using array:

Source Code:  
#include<iostream>

using namespace std;

class Stack {

private:

    int\* arr;

    int capacity;

    int top;

public:

    Stack(int size) {

        capacity = size;

        arr = new int[capacity];

        top = -1;

    }

    ~Stack() {

        delete[] arr;

    }

    void push(int data) {

        if (isFull()) {

            cout << "OVERFLOW" << endl;

            return;

        }

        arr[++top] = data;     }

    int pop() {

        if (isEmpty()) {

            cout << "UNDERFLOW" << endl;

            return -1;

        }

        return arr[top--];

    }

    bool isEmpty() {

        return top == -1;

    }

    bool isFull() {

        return top == capacity - 1;

    }

};

int main() {

    Stack stack(5);

    stack.push(1);

    stack.push(2);

    stack.push(3);

    stack.push(4);

    cout << "Stack Elements: ";

    while (!stack.isEmpty()) {

        cout << stack.pop() << " ";

    }

    cout << endl;

    stack.push(5);

    stack.push(6);

    cout << "Stack Elements: ";

    while (!stack.isEmpty()) {

        cout << stack.pop() << " ";

    }

    cout << endl;

    return 0;

}

OUTPUT:



2.2)Stack Using Linked List:

Source Code:

#include<iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int data) {

*this*->data = data;

        next = NULL;

    }

};

class Stack {

private:

    Node\* top;

public:

    Stack() {

        top = NULL;

    }

    ~Stack() {

        while (!isEmpty()) {

            pop();

        }

    }

    void push(int data) {

        Node\* newNode = new Node(data);

        newNode->next = top;

        top = newNode;

    }

    int pop() {

        if (isEmpty()) {

            cout << "UNDERFLOW" << endl;

            return -1;

        }

        Node\* temp = top;

        int popVal = top->data;

        top = top->next;

        delete temp;

        return popVal;

    }

    bool isEmpty() {

        return top == NULL;

    }

};

int main() {

    Stack stack;

    stack.push(1);

    stack.push(2);

    stack.push(3);

    stack.push(4);

    cout << "Stack Elements: ";

    while (!stack.isEmpty()) {

        cout << stack.pop() << " ";

    }

    cout << endl;

    stack.push(5);

    stack.push(6);

    cout << "Stack Elements: ";

    while (!stack.isEmpty()) {

        cout << stack.pop() << " ";

    }

    cout << endl;

    return 0;

}

OUTPUT:



2.3)Queue using Array:  
Source Code:

#include<iostream>

using namespace std;

class Queue{

     int \*arr;

     int qfront,rear,size;

     public:

     Queue(){

        size=100001;

        arr=new int[size];

        qfront=0;

        rear=0;

     }

     void enqueue(int data){

        if(rear==size){

            cout<<"Queue is full."<<endl;

        }

        else{

            arr[rear]=data;

            rear++;

        }

     }

     bool isEmpty(){

        if(qfront==rear){

            return true;

        }

        else{

            return false;

        }

     }

     int deque(){

        if(qfront==rear){

            return -1;

        }

        else{

            int ans=arr[qfront];

            arr[qfront]=-1;

            qfront++;

            if(qfront==rear){

                qfront=0;

                rear=0;

            }

            return ans;

        }

     }

     int front(){

        if(qfront==rear){

            cout<<"Queue is empty."<<endl;

            return -1;

        }

        else{

            return arr[qfront];

        }

     }

};

int main(){

     Queue obj;

     obj.enqueue(12);

     obj.enqueue(13);

     obj.enqueue(14);

     obj.enqueue(15);

     int dequeuedValue = obj.deque();

     cout << "Dequeued value: " << dequeuedValue << endl;

     int frontValue = obj.front();

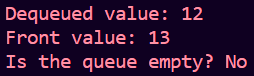
     cout << "Front value: " << frontValue << endl;

     bool emptyStatus = obj.isEmpty();

     cout << "Is the queue empty? " << (emptyStatus ? "Yes" : "No") << endl;

     return 0;

}

OUTPUT:  


2.4)Queue using Linked List:  
Source Code:

#include<iostream>

using namespace std;

struct Node{

     int data;

     Node\* next;

     Node(int d){

        data=d;

        next=NULL;

     }

};

struct Queue{

    Node \*rear,\*front;

    Queue(){front=rear=NULL;}

    void enqueue(int x){

        Node\*temp=new Node(x);

        if(rear==NULL){

            front=rear=temp;

            return;

        }

        rear->next=temp;

        rear=temp;

    }

    void deque(){

        if(front==NULL){

            return ;

        }

        Node\*temp=front;

        front=front->next;

        if(front==NULL){

            rear=NULL;

        }

        delete (temp);

    }

};

int main(){

    Queue q;

    q.enqueue(12);

    q.enqueue(13);

    q.deque();

    q.enqueue(14);

    q.deque();

    q.enqueue(13);

    q.deque();

    q.enqueue(16);

    q.enqueue(17);

    cout<<"Front of the Queue: "<<((q.front!=NULL)?(q.front)->data:-1)<<endl;

    cout<<"Rear of the Queue: "<<((q.rear!=NULL)?(q.rear)->data:-1);

    return 0;

}

OUTPUT:  


3)Binary Tree and Binary Search Tree

3.1)Binary Tree  
Source Code:

#include<iostream>

#include<queue>

using namespace std;

class node{

    public:

    int data;

    node\* left;

    node\* right;

    node(int d){

*this*->data=d;

*this*->left=NULL;

*this*->right=NULL;

    }

};

node\* BuildTree(node\* root){

   int data;

   cout<<"Enter the data: "<<endl;

   cin>>data;

   root=new node(data);

   if(data==-1){

    return NULL;

   }

   cout<<"Left of: "<<data<<endl;

   root->left=BuildTree(root->left);

   cout<<"Right of: "<<data<<endl;

   root->right=BuildTree(root->right);

   return root;

}

void LOT(node \*root){

    queue<node\*> q;

    q.push(root);

    q.push(NULL);

    while(!q.empty()){

        node\* temp=q.front();

        q.pop();

        if(temp==NULL){

            cout<<endl;

            if(!q.empty()){

                q.push(NULL);

            }

        }

        else{

            cout<<temp->data<<" ";

        if(temp->left){

            q.push(temp->left);

        }

        if(temp->right){

            q.push(temp->right);

        }

        }

    }

}

void IOT(node\* root){

    if(root==NULL){

        return ;

    }

    IOT(root->left);

    cout<<root->data<<" ";

    IOT(root->right);

}

void POT(node\* root){

if(root==NULL){

        return ;

    }

    cout<<root->data<<" ";

    POT(root->left);

    POT(root->right);

}

void POOT(node\* root){

    if(root==NULL){

        return ;

    }

    POOT(root->left);

    POOT(root->right);

    cout<<root->data<<" ";

}

int main(){

    node\* root=NULL;

*//1 3 7 -1 -1 11 -1 -1 5 17 -1 -1 -1*

    cout<<"Level order Traversal is: "<<endl;

    root=BuildTree(root);

    LOT(root);

    cout<<"In order Traversal is: "<<endl;

    IOT(root);

    cout<<endl;

    cout<<"Pre order Traversal is: "<<endl;

    POT(root);

    cout<<endl;

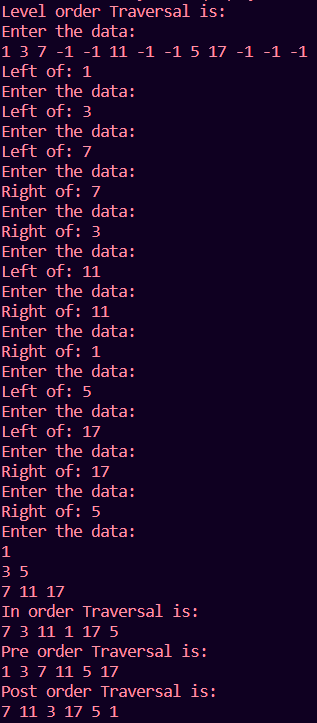
    cout<<"Post order Traversal is: "<<endl;

    POOT(root);

    return 0;

}

OUTPUT:



3.2)Binary Search Tree:  
Source Code:

#include<iostream>

#include<queue>

using namespace std;

class node{

    public:

    int data;

    node\* left,\*right;

    node(int d){

*this*->data=d;

*this*->left=NULL;

*this*->right=NULL;

    }

};

node\* insert(node\* root,int d){

    if(root==NULL){

        root=new node(d);

        return root;

    }

    if(d>root->data){

        root->right=insert(root->right,d);

    }

    else{

        root->left=insert(root->left,d);

    }

    return root;

}

void IOT(node\* root){

    if(root==NULL){

        return ;

    }

    IOT(root->left);

    cout<<root->data<<" ";

    IOT(root->right);

}

void POT(node\* root){

if(root==NULL){

        return ;

    }

    cout<<root->data<<" ";

    POT(root->left);

    POT(root->right);

}

void POOT(node\* root){

    if(root==NULL){

        return ;

    }

    POOT(root->left);

    POOT(root->right);

    cout<<root->data<<" ";

}

void takeInput(node\* &root){

    int data;

    cin>>data;

    while(data!=-1){

        root=insert(root,data);

        cin>>data;

    }

}

void LOT(node \*root){

    queue<node\*> q;

    q.push(root);

    q.push(NULL);

    while(!q.empty()){

        node\* temp=q.front();

        q.pop();

        if(temp==NULL){

            cout<<endl;

            if(!q.empty()){

                q.push(NULL);

            }

        }

        else{

            cout<<temp->data<<" ";

        if(temp->left){

            q.push(temp->left);

        }

        if(temp->right){

            q.push(temp->right);

        }

        }

    }

}

int main(){

   node\* root=NULL;

   cout<<"Enter data to create BST"<<endl;

   takeInput(root);

   cout<<"Printing the BST"<<endl;

   LOT(root);

   cout<<"Inorder Traversal "<<endl;

   IOT(root);

   cout<<endl;

   cout<<"Preorder Traversal "<<endl;

   POT(root);

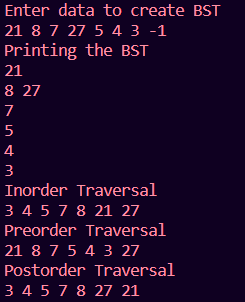
   cout<<endl;

   cout<<"Postorder Traversal "<<endl;

   POOT(root);

    return 0;

}

OUTPUT:  


4)Graphs:

4.1)BFS:  
Source Code:

#include <bits/stdc++.h>

using namespace std;

class Graph {

    int V;

    vector<list<int>> adj;

public:

    Graph(int V);

    void addEdge(int v, int w);

    void BFS(int s);

};

Graph::Graph(int V) {

*this*->V = V;

    adj.resize(V);

}

void Graph::addEdge(int v, int w) {

    adj[v].push\_back(w);

}

void Graph::BFS(int s) {

    vector<bool> visited;

    visited.resize(V, false);

    list<int> queue;

    visited[s] = true;

    queue.push\_back(s);

    while (!queue.empty()) {

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        for (auto adjacent : adj[s]) {

            if (!visited[adjacent]) {

                visited[adjacent] = true;

                queue.push\_back(adjacent);

            }

        }

    }

}

int main() {

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "

         << "(starting from vertex 2) \n";

    g.BFS(2);

    return 0;

}

OUTPUT:  


4.2)DFS:  
Source Code:

#include <bits/stdc++.h>

using namespace std;

class Graph {

public:

    map<int, bool> visited;

    map<int, list<int>> adj;

    void addEdge(int v, int w);

    void DFS(int v);

};

void Graph::addEdge(int v, int w) {

    adj[v].push\_back(w);

}

void Graph::DFS(int v) {

    visited[v] = true;

    cout << v << " ";

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            DFS(\*i);

}

int main() {

    Graph g;

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"

         << " (starting from vertex 2) \n";

    g.DFS(2);

    return 0;

}

OUTPUT:



5)Binary Search:

Source Code:

#include<bits/stdc++.h>

using namespace std;

int main(){

    int n,k;

    cout<<"Enter the size of the array\n";

    cin>>n;

    int arr[n];

    for(int i=0;i<n;i++){

        cin>>arr[i];

    }

    cout<<"Enter the element to be compared\n";

    cin>>k;

    int start=0,end=n-1,mid;

    mid=start+(end-start)/2;

    while(start<=end){

        if(arr[mid]==k){

            cout<<"Element is found at index: "<<mid<<endl;

            break;

        }

        else if(arr[mid]<k){

            start=mid+1;

        }

        else{

            end=mid-1;

        }

        mid=start+(end-start)/2;

    }

    if(start>end){

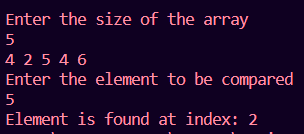
        cout<<"Element is not present in the array."<<endl;

    }

return 0;

}

OUTPUT:



6)Merge And Quick Sort:  
  
6.1)Merge Sort:

Source Code:

#include<iostream>

using namespace std;

void merge(int \*arr,int s,int e)

{

    int mid=(s+e)/2;

    int l1=mid-s+1;

    int l2=e-mid;

    int \*first=new int[l1];

    int \*second=new int[l2];

    int mainArrayIndex=s;

    for(int i=0;i<l1;i++){

        first[i]=arr[mainArrayIndex++];

    }

    int k=mid+1;

    for(int i=0;i<l2;i++){

        second[i]=arr[mainArrayIndex++];

    }

    int i1=0;

    int i2=0;

    mainArrayIndex=s;

    while(i1<l1&&i2<l2){

        if(first[i1]<second[i2]){

            arr[mainArrayIndex++]=first[i1++];

        }

        else{

            arr[mainArrayIndex++]=second[i2++];

        }

    }

    while(i1<l1){

        arr[mainArrayIndex++]=first[i1++];

    }

    while(i2<l2){

        arr[mainArrayIndex++]=second[i2++];

    }

}

void mergeSort(int \*arr,int s,int e){

    int mid=(s+e)/2;

  if(s>=e){

    return ;

  }

  mergeSort(arr,s,mid);

  mergeSort(arr,mid+1,e);

  merge(arr,s,e);

}

int main(){

 int arr[5]={2,5,1,6,9};

 int n=5;

 mergeSort(arr,0,n-1);

for(int i=0;i<n;i++){

    cout<<arr[i]<<" ";

}cout<<endl;

    return 0;

}

OUTPUT:



6.2)Quick Sort:  
Source Code:

#include<iostream>

using namespace std;

int partition(int arr[],int s,int e){

    int pivot=arr[s];

    int count=0;

    for(int i=s+1;i<=e;i++){

        if(arr[i]<=pivot){

            count++;

        }

    }

    int pivotIndex=s+count;

    swap(arr[pivotIndex],arr[s]);

    int i=s;

    int j=e;

    while(i<pivotIndex&&j>pivotIndex){

        while(arr[i]<pivot){

            i++;

        }

        while(arr[j]>pivot){

            j--;

        }

        if(i<pivotIndex&&j>pivotIndex){

            swap(arr[i++],arr[j--]);

        }

    }

    return pivotIndex;

}

void quickSort(int arr[],int s,int e){

 if(s>=e){

    return ;

 }

 int p=partition(arr,s,e);

 quickSort(arr,s,p-1);

 quickSort(arr,p+1,e);

}

int main(){

int arr[5]={2,4,1,6,9};

int n=5;

         quickSort(arr,0,n-1);

         for(int i=0;i<n;i++){

            cout<<arr[i]<<" ";

         }

         cout<<endl;

    return 0;

}

OUTPUT:

