Data Structures Practical File

Submitted by:

NITIN GOYAL – 2022UIT3045

VAISHANT SHARMA – 2022UIT3048

GAUTAM ARORA – 2022UIT3050

SAKSHAM MITTAL – 2022UIT3073

ISHAN MANGOTRA – 2022UIT3076

CLASS – INFORMATION TECHNOLOGY(IT-1)

SUBJECT – DATA STRUCTURES AND ALGORITHMS

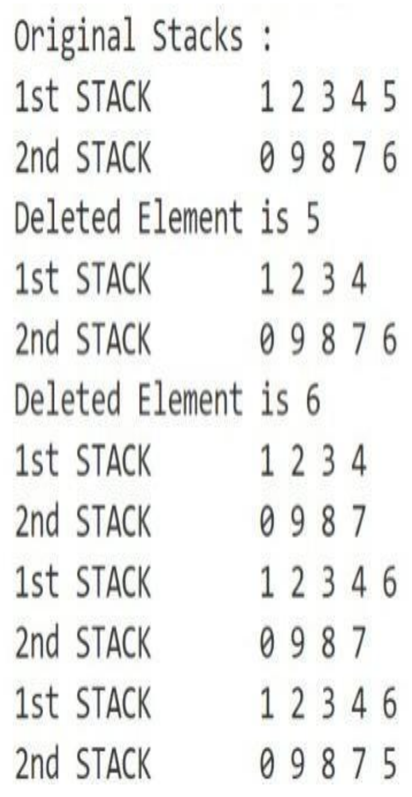
SUBJECT CODE – ITITC02

Submitted to:

Dr. Preeti Bansal

Index

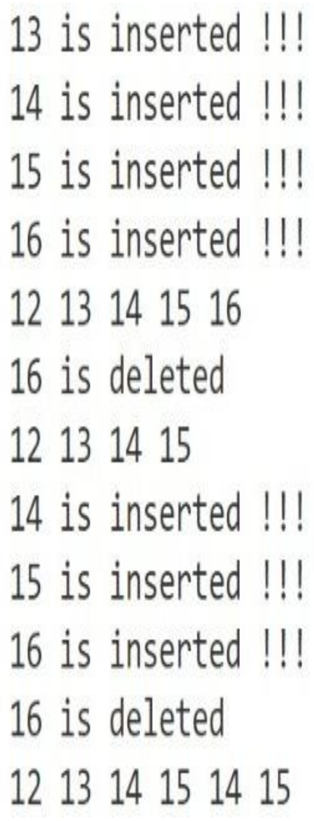
Q1.
```
#include <iostream>
using namespace std;
int top1,top2,MIN = 0,MAX = 99;

void push1(int arr[],int element) {
    if (top1 < top2 - 1) {
        top1++;
        arr[top1] = element;
    } else{
        cout << "Stack Overflow !!!" << endl;
    }
}

void push2(int arr[],int element) {
    if (top1 < top2 - 1) {
        top2--;
        arr[top2] = element;
    } else {
        cout << "Stack  Overflow !!!" << endl;
    }
}

void pop1(int arr[]) {
    if (top1 == MIN - 1) {
        cout << "Stack Underflow !!!" << endl;
    } else {
        int popped = arr[top1];
        top1--;
        cout << "Deleted Element is " << popped << endl;
    }
}

void pop2(int arr[]) {
    if (top2 == MAX + 1) {
        cout << "Stack Underflow" << endl;
    } else {
        int popped = arr[top2];
        top2++;
        cout << "Deleted Element is " << popped << endl;
    }
}

void traversal(int arr[]) {
    if (top1 == MIN-1) {
        cout << "FIRST STACK IS EMPTY !!!" << endl;
    } if(top1 != MIN-1) {
        cout << "1st STACK" << "\t";
        for (int i=0;i<=top1;i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    } if (top2 == MAX+1) {
        cout << "SECOND STACK IS EMPTY !!!" << endl;
    } if (top2 != MAX+1) {
        cout << "2nd STACK" << "\t";
        for (int i=9;i>=top2;i--) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }


}

int main(int argc, char const *argv[])
{
    int arr[] = {1,2,3,4,5,6,7,8,9,0};
    top1 = 4;
    top2 = 5;
    cout << "Original Stacks : " << endl;
    traversal(arr);
    pop1(arr);
    traversal(arr);
    pop2(arr);
    traversal(arr);
    push1(arr,6);
    traversal(arr);
    push2(arr,5);
    traversal(arr);


    return 0;
}
```



```
Original Stacks :
1st STACK      1 2 3 4 5
2nd STACK      0 9 8 7 6
Deleted Element is 5
1st STACK      1 2 3 4
2nd STACK      0 9 8 7 6
Deleted Element is 6
1st STACK      1 2 3 4
2nd STACK      0 9 8 7
1st STACK      1 2 3 4 6
2nd STACK      0 9 8 7
1st STACK      1 2 3 4 6
2nd STACK      0 9 8 7 5
```

Q2.

```
#include <iostream>
using namespace std;

class Node{
    public :
    int element;
    Node* next;



    Node(int element) {
        this -> element = element;
        this -> next = NULL;
    }
};

void PUSH(Node* &top,Node* &head,int element) {
    Node* temp = new Node(element);
    if (top == NULL) {
        head = temp;
        top = temp;
    } else {
        top -> next = temp;
        top = temp;
    }
    cout << element << " is inserted !!!" << endl;
}

void POP(Node* &head,Node* &top) {
    int popped = top->element;
    Node* temp = head;
    if (top == NULL) {
        cout << "STACK UNDERFLOW" << endl;
    } else if (temp -> next == top) {
        top = head;
        cout << popped << " is deleted" << endl;
    } else if (head == top) {
        top = NULL,head = NULL;
        cout << popped << " is deleted" << endl;
    } else {
        while (temp -> next != top) {
            temp = temp -> next;
        }
        top = temp;
        top -> next = NULL;
        cout << popped << " is deleted" << endl;
    }
}

void TRAVERSE(Node* &head) {
    Node* temp = head;
    if (head == NULL) {
        cout << "NO ELEMENT IN STACK" ;
    } else {
        while (temp != NULL) {
            cout << temp -> element << " ";
            temp = temp -> next;
        }
    }
    cout << endl;
}

int main(int argc, char const *argv[])
{
    Node* head = new Node(12);
    Node* top = head;
    PUSH(top,head,13);
    PUSH(top,head,14);
    PUSH(top,head,15);
    PUSH(top,head,16);
    TRAVERSE(head);
    POP(head,top);
    TRAVERSE(head);
    PUSH(top,head,14);
    PUSH(top,head,15);
    PUSH(top,head,16);
    POP(head,top);
    TRAVERSE(head);


    return 0;
}
```

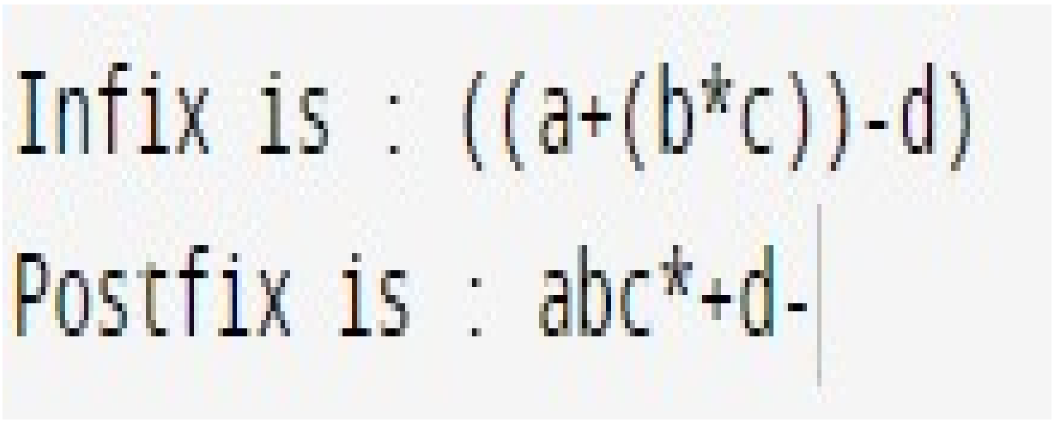


```
13 is inserted !!!
14 is inserted !!!
15 is inserted !!!
16 is inserted !!!
12 13 14 15 16
16 is deleted
12 13 14 15
14 is inserted !!!
15 is inserted !!!
16 is inserted !!!
16 is deleted
12 13 14 15 14 15
```

Q3.

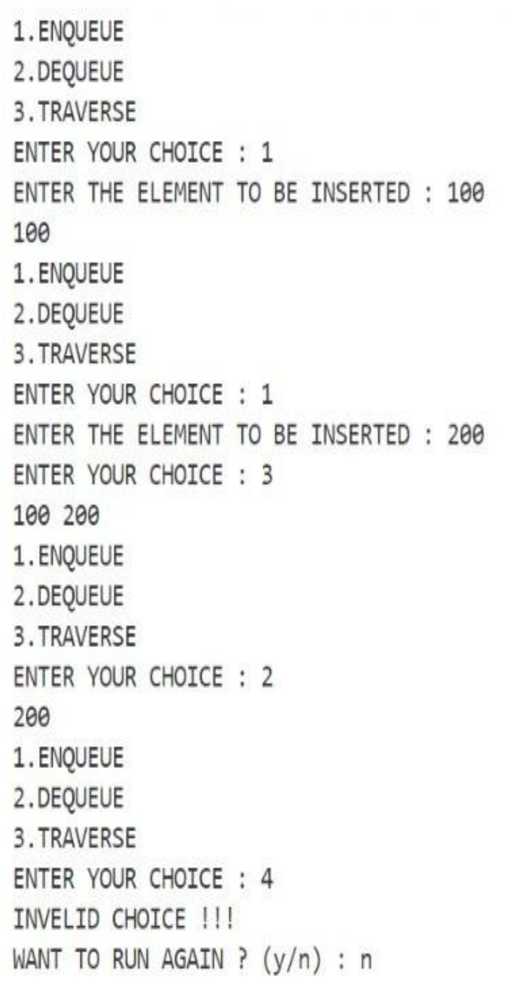
```
#include <iostream>
#include <stack>
using namespace std;

int priority(char alpha)
{
    if (alpha == '+' || alpha == '-')
        return 1;

    if (alpha == '*' || alpha == '/')
        return 2;

    if (alpha == '^')
        return 3;

    return 0;
}
string convert(string infix)
{
    int i = 0;
    string postfix = "";

    stack<int> s;

    while (infix[i] != '\0')
    {

        if (infix[i] >= 'a' && infix[i] <= 'z' || infix[i] >= 'A' && infix[i] <= 'Z')
        {
            postfix += infix[i];
            i++;
        }
    else if (infix[i] == '(')
    {
        s.push(infix[i]);
        i++;
    }

    else if (infix[i] == ')')
    {
        while (s.top() != '(')
        {
            postfix += s.top();
            s.pop();
        }
        s.pop();
        i++;
    }
    else
    {
        while (!s.empty() && priority(infix[i]) <= priority(s.top()))
        {
            postfix += s.top();
            s.pop();
        }
        s.push(infix[i]);
        i++;
    }
}
while (!s.empty())
{
    postfix += s.top();
    s.pop();
}

cout << "Postfix is : " << postfix;
```

```
    return postfix;
}

int main()
{
    string infix = "((a+(b*c))-d)";
    string postfix;
    postfix = convert(infix);

    return 0;
}
```


```
Infix is : ((a+(b*c))-d)
Postfix is : abc*+d-
```

Q4.

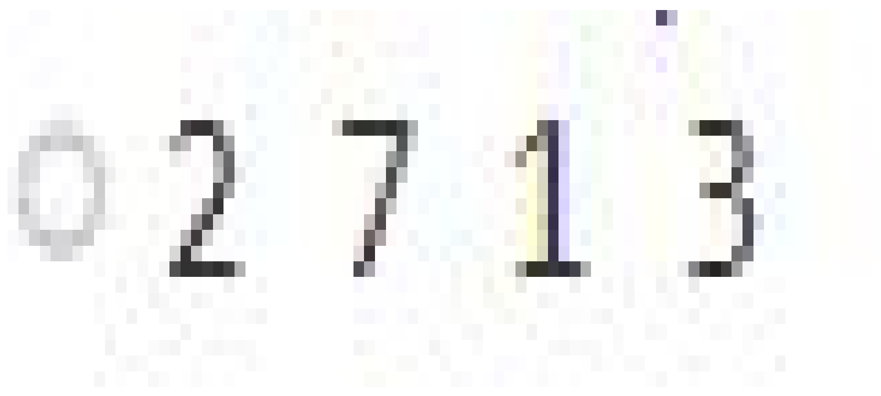
```
#include <iostream>
using namespace std;
int REAR=-1,FRONT=-1,SIZE=0;
int MENU(int arr[]);
int TRAVERSE(int arr[]) {
   if (REAR==-1) {
      cout << "NO ELEMENT IN QUEUE !!!\n";
   } else {
      for (int i = FRONT;i<=REAR;i++) {
         cout << arr[i] << " ";
      }
      cout << endl;
   }
   MENU(arr);
}
int ENQUEUE(int arr[]) {
   int element;
   cout << "ENTER THE ELEMENT TO BE INSERTED : ";
   cin >> element;
   if (FRONT==-1 && REAR==-1) {
      FRONT++,REAR++;
      arr[REAR] = element;
      SIZE++;TRAVERSE(arr);
   } else if (REAR==99) {
      cout << "STACK OVERFLOW !!!\n";
   } else {
      REAR++;
      arr[REAR] = element;
      SIZE++;TRAVERSE(arr);
   }
   MENU(arr);
}
int DEQUEUE(int arr[]) {
    int element;
    if (FRONT==-1 && REAR==-1) {
        cout << "QUEUE UNDERFLOW !!!\n";
    } else if (REAR==FRONT) {
        element=arr[REAR];
        FRONT=-1;REAR=-1;
        SIZE--;TRAVERSE(arr);
    } else {
        element=arr[REAR];
        FRONT++;
        SIZE--;TRAVERSE(arr);
    }
    MENU(arr);
}
int MENU(int arr[]) {
    while (1){
        cout << "1.ENQUEUE\n2.DEQUEUE\n3.TRAVERSE\n";
        int choice;
        cout << "ENTER YOUR CHOICE : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            ENQUEUE(arr);
            break;
        case 2:
            DEQUEUE(arr);
            break;
        case 3:
          TRAVERSE(arr);
            break;


        default:
            cout << "INVELID CHOICE !!!\n";
            break;
        }
        char ch;
        cout << "WANT TO RUN AGAIN ? (y/n) : ";
        cin >> ch;
        if (ch=='y') {
            continue;
        } else {
            return 0;
        }
    }
}
int main(int argc, char const *argv[])
{
    int arr[100];
    MENU(arr);
    return 0;
}
```


```
1.ENQUEUE
2.DEQUEUE
3.TRAVERSE
ENTER YOUR CHOICE : 1
ENTER THE ELEMENT TO BE INSERTED : 100
100
1.ENQUEUE
2.DEQUEUE
3.TRAVERSE
ENTER YOUR CHOICE : 1
ENTER THE ELEMENT TO BE INSERTED : 200
ENTER YOUR CHOICE : 3
100 200
1.ENQUEUE
2.DEQUEUE
3.TRAVERSE
ENTER YOUR CHOICE : 2
200
1.ENQUEUE
2.DEQUEUE
3.TRAVERSE
ENTER YOUR CHOICE : 4
INVELID CHOICE !!!
WANT TO RUN AGAIN ? (y/n) : n
```

Q5.

```
#include <stdio.h>
#include <stdlib.h>
// priority Node
typedef struct node {
  int data;
  int priority;
  struct node* next;
} Node;
Node* newNode(int d, int p) {
  Node* temp = (Node*)malloc(sizeof(Node));
  temp->data = d;
  temp->priority = p;
  temp->next = NULL;
  return temp;
}
int peek(Node** head) {
  return (*head)->data;
}
void pop(Node** head) {
  Node* temp = *head;
  (*head) = (*head)->next;
  free(temp);
}
void push(Node** head, int d, int p) {
  Node* start = (*head);
  Node* temp = newNode(d, p);
  if ((*head)->priority > p) {
    temp->next = *head;
    (*head) = temp;
  } else {
    while (start->next != NULL &&
    start->next->priority < p) {
      start = start->next;
```

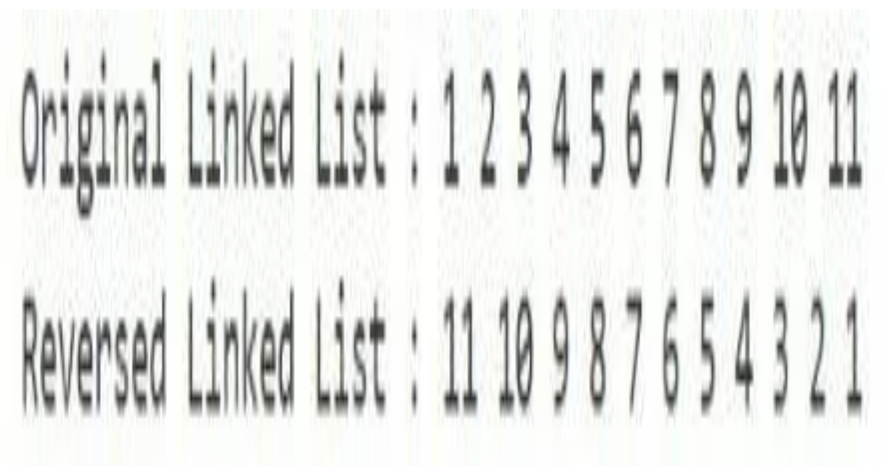
```
        }
        // Either at the ends of the list
        // or at required position
        temp->next = start->next;
        start->next = temp;
    }
}
// Function to check the queue is empty
int isEmpty(Node** head) {
    return (*head) == NULL;
}
// main function
int main() {
    Node* pq = newNode(7, 1);
    push(&pq, 1, 2);
    push(&pq, 3, 3);
    push(&pq, 2, 0);
    while (!isEmpty(&pq)) {
        printf("%d ", peek(&pq));
        pop(&pq);
    }
    return 0;
}
```


0 2 7 1 3

Q6.

```
#include <iostream>
using namespace std;
class Node{
  public :
  Node* next;
  int data;
  Node(int number){
    this -> next = NULL;
    this -> data = number;
  }
};
void TRAVERSE_FROM_START(Node* &head) {
  Node* temp = head;
  while(temp != NULL) {
    cout << temp-> data << " ";
    temp = temp -> next;
  }
  cout << endl;
}
void PUSH(Node* &tail,int number) {
  Node* element = new Node(number);
  tail -> next = element;
  tail = element;
}
void REVERSE(Node* &head,Node* &tail) {
  Node* temp = head;
  Node* temp2 = temp -> next;
  while(temp2 != NULL) {
    Node* temp3 = temp2 -> next;
    temp2 -> next = temp;
    temp = temp2;
    temp2 = temp3;
  }
```

```
    head -> next = NULL;
    tail = head;
    head = temp;
}
int main(int argc, char const *argv[])
{
    Node* head = new Node(1);
    Node* tail = head;
    PUSH(tail,2);
    PUSH(tail,3);
    PUSH(tail,4);
    PUSH(tail,5);
    PUSH(tail,6);
    PUSH(tail,7);
    PUSH(tail,8);
    PUSH(tail,9);
    PUSH(tail,10);
    PUSH(tail,11);
    cout << "Original Linked List : ";
    TRAVERSE_FROM_START(head);
    REVERSE(head,tail);
    cout << "Reversed Linked List : ";
    TRAVERSE_FROM_START(head);
    return 0;
}
```

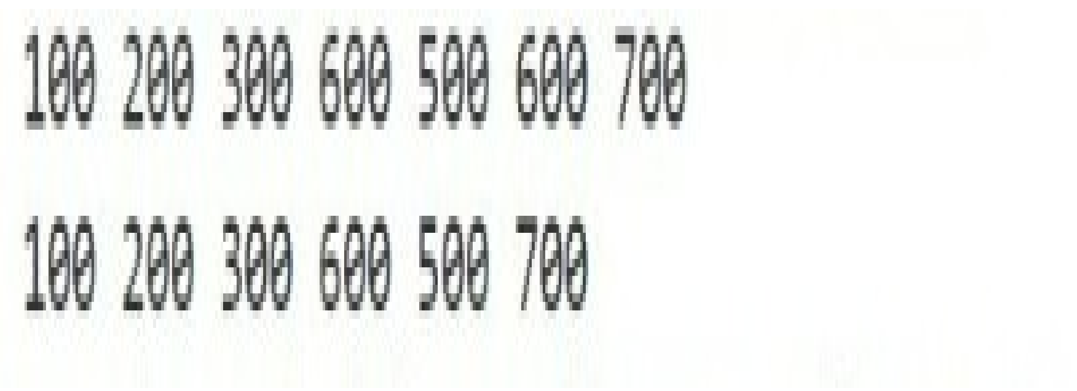


```
Original Linked List : 1 2 3 4 5 6 7 8 9 10 11
Reversed Linked List : 11 10 9 8 7 6 5 4 3 2 1
```

Q7.

```
#include <iostream>
using namespace std;

class Node{
    public :
    Node* next;
    int data;

    Node(int number){
        this -> next = NULL;
        this -> data = number;
    }
};

void removeRedundant(Node* &head) {
    Node *temp1 = head;
    while (temp1->next != NULL) {
        Node *temp2 = temp1->next;
        while (temp2 != NULL) {
            if (temp1->data == temp2->data) {
                Node *temp3 = temp1;
                while (temp3->next != temp2) {
                    temp3 = temp3->next;
                }
                temp3->next = temp2->next;
                temp2->next = NULL;
            }
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
    }
}
```

```
void PUSH(Node* &top,Node* &head,int element) {
    Node* temp = new Node(element);
    if (top == NULL) {
        head = temp;
        top = temp;
    } else {
        top -> next = temp;
        top = temp;
    }
}

void TRAVERSE(Node* &head) {
    Node* temp = head;
    if (head == NULL) {
        cout << "NO ELEMENT IN STACK" ;
    } else {
        while (temp != NULL) {
            cout << temp -> data << " ";
            temp = temp -> next;
        }
    }
    cout << endl;
}

int main()
{
    Node *head = new Node(100);
    Node *top = head;
    PUSH(top,head,200);
    PUSH(top,head,300);
    PUSH(top,head,600);
    PUSH(top,head,500);
    PUSH(top,head,600);
    PUSH(top,head,700);
    TRAVERSE(head);
    removeRedundant(head);
    TRAVERSE(head);



    return 0;
}
```

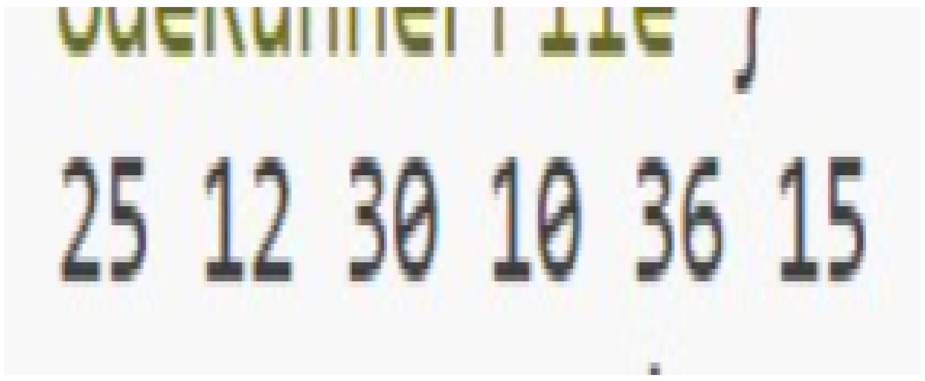


```
100 200 300 600 500 600 700
100 200 300 600 500 700
```

Q8.

```
#include <iostream>
using namespace std;
struct node {
   int data;
   node* left;
   node* right;
};
void BinaryTree2DoubleLinkedList(node* root, node** head)
{
   if (root == NULL)
       return;
   static node* prev = NULL;

   BinaryTree2DoubleLinkedList(root->left, head);

   if (prev == NULL)
       *head = root;
   else {
       root->left = prev;
       prev->right = root;
   }
   prev = root;

   BinaryTree2DoubleLinkedList(root->right, head);
}

node* newNode(int data)
{
   node* new_node = new node;
   new_node->data = data;
   new_node->left = new_node->right = NULL;
   return (new_node);
}
```

```
void printList(node* node)
{
    while (node != NULL) {
        cout << node->data << " ";
        node = node->right;
    }
}
int main()
{
    node* root = newNode(10);
    root->left = newNode(12);
    root->right = newNode(15);
    root->left->left = newNode(25);
    root->left->right = newNode(30);
    root->right->left = newNode(36);
    node* head = NULL;
    BinaryTree2DoubleLinkedList(root, &head);
    printList(head);
    return 0;
}
```



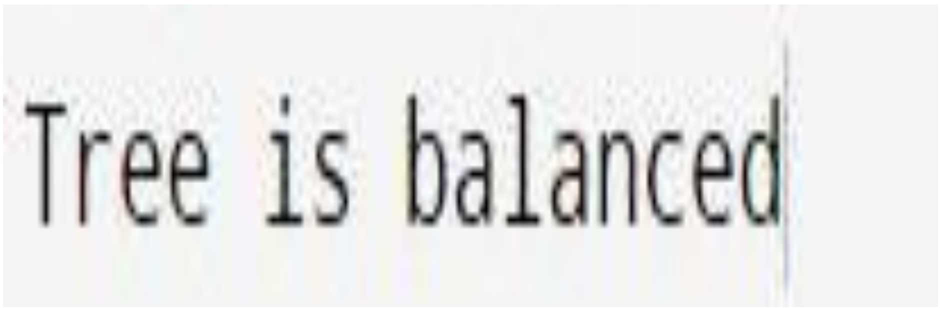
```
25 12 30 10 36 15
```

.

Q9

```
#include<iostream>
using namespace std;

struct Node {
   int data;
   Node *left, *right;
};

int height(Node* node) {
   if (node == NULL)
      return 0;
   return 1 + max(height(node->left), height(node->right));
}
bool isBalanced(Node* root) {
   int lh;
   int rh;

   if (root == NULL)
      return 1;

   lh = height(root->left);
   rh = height(root->right);

   if (abs(lh - rh) <= 1 && isBalanced(root->left) && isBalanced(root->right))
      return 1;

   return 0;
}

Node* newNode(int data) {
   Node* node = new Node;
   node->data = data;
   node->left = NULL;
```

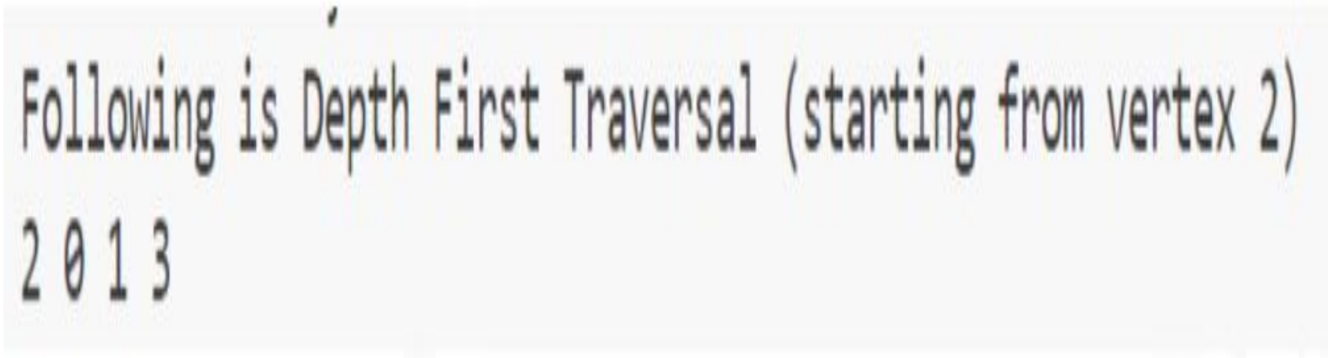
```
    node->right = NULL;

    return(node);
}

int main() {
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->left->left->left = newNode(7);

    if(isBalanced(root))
        cout << "Tree is balanced";
    else
        cout << "Tree is not balanced";

    return 0;
}
```



```
Tree is balanced
```

Q10.

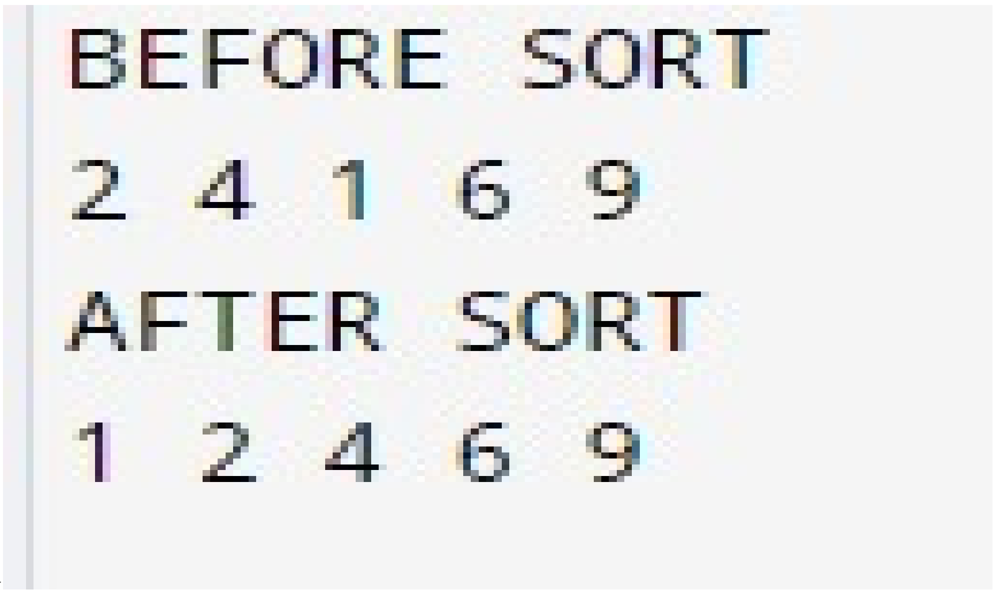
```
#include <iostream>
#include<vector>
#include<map>
#include<list>
using namespace std;


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;

}
```


```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

Q11.

```
#include <iostream>
using namespace std;

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

    int pivot = arr[s];
    int cnt = 0;
    for (int i = s + 1; i <= e; i++)
    {
        if (arr[i] <= pivot)
        {
            cnt++;
        }
    }
    // place pivot here
    int pivotIndex = s + cnt;
    swap(arr[pivotIndex], arr[s]);

    // left and right partition banao
    int i = s, 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)
{

    // base case
    if (s >= e)
    {
        return;
    }

    // partition
    int p = partition(arr, s, e);

    // Left side sort
    quickSort(arr, s, p - 1);

    // Right Side sort
    quickSort(arr, p + 1, e);
}

int main()
{
    int arr[5] = {2, 4, 1, 6, 9};
    int n = 5;

    cout<<"BEFORE SORT"<<endl;
```

```
 for (int i = 0; i < n; i++)
{
    cout << arr[i] << " ";
}
cout << endl;

quickSort(arr, 0, n - 1);

cout<<"AFTER SORT"<<endl;

for (int i = 0; i < n; i++)
{
    cout << arr[i] << " ";
}
cout << endl;
```



```
BEFORE SORT
2 4 1 6 9
AFTER SORT
1 2 4 6 9
```

}