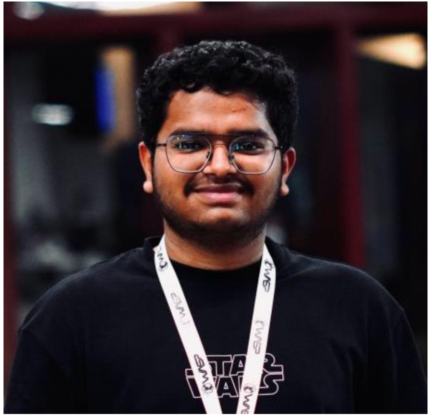
UCS301 Data Structures

Assignment 1- 9

Name: Dhruv Mittal

Roll no: 1024030364

Subgroup: 2C24



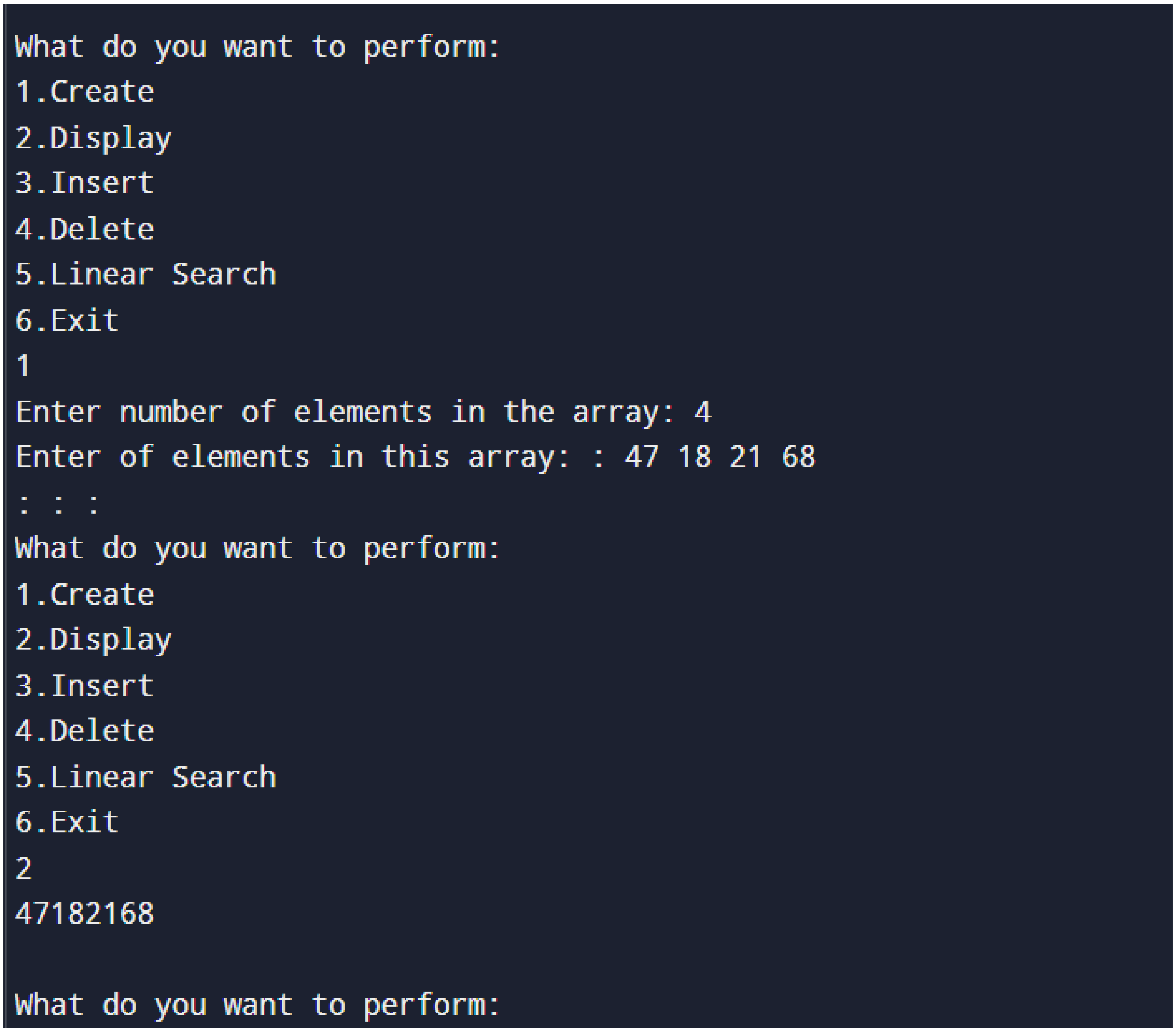
DSA ASSIGNMENT 1

QUESTION 1

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

int main(){

    int arr[100], numArray=0;

    while(true){
        cout<<("\nWhat do you want to perform: \n");
        cout<<("1.Create\n");
        cout<<("2.Display\n");
        cout<<("3.Insert\n");
        cout<<("4.Delete\n");
        cout<<("5.Linear Search\n");
        cout<<("6.Exit\n");
        int num;
        cin>>num;
        switch (num) {
            case 1:{
            cout<<("Enter number of elements in the array: ");
            cin>>numArray;
            cout<<("Enter of elements in this array: ");
            for (int i=0;i<numArray;i++){
                cout<<(": ");
                cin>>arr[i];
```

```
          }
          break;
}
case 2:{
   for (int i=0;i<numArray;i++){
          cout<<(arr[i]);
          }
          cout<<"\n";
          break;
}
case 3:{
   cout<<("At which position the number will be insterted: ");
   int pos=0,posNum=0;
   cin>>pos;
   cout<<("What number should be inserted: ");
   cin>>posNum;
   for (int i=numArray;i>=pos;--i){
          if(i>pos-1){
                    arr[i]=arr[i-1];
                    }
          }
          arr[pos-1]=posNum;
          numArray++;
          break;
}
case 4:{
   cout<<("At which position the number will be deleted: ");
   int pos=0,posNum=0;
```

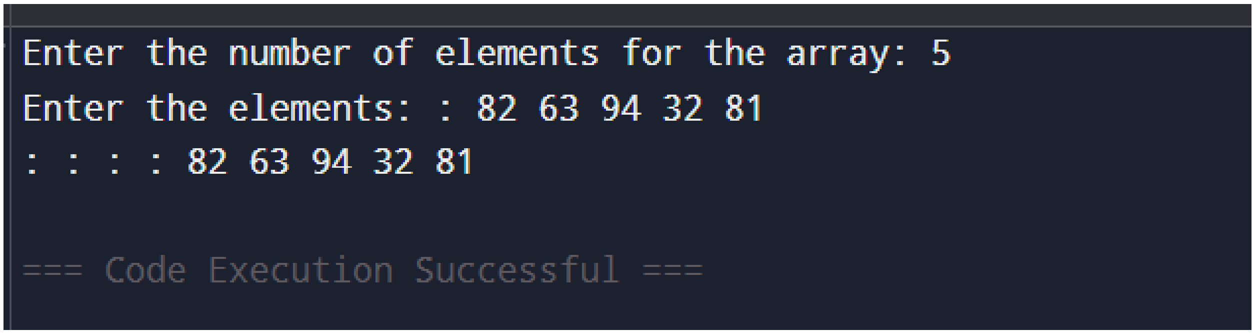
```
              cin>>pos;
              for (int i=0;i<numArray-1;i++){
                  if(i<pos-1){
                         arr[i]=arr[i];
                         }
                         else if(i>=pos-1){
                                arr[i]=arr[i+1];
                         }
                  }
                  numArray=numArray-1;
                  break;
            }
            case 5:{
              cout<<("Which number to find: ");
              int findNum=0;
              cin>>findNum;
bool found=false;
              for (int i=0;i<numArray;i++){
                  if(arr[i]==findNum){
                         cout<<("Element found on index")<<i;
    found=true;
                         }
                  }
if(found==false){
  cout<<"Not Found";
}
                  break;
            }
            case 6:{
               cout << "Exiting";
               return 0;
            }
            default:
                cout<<"Invalid code! Try again";
        }
    }
    return 0;
}
```



```
What do you want to perform:
1.Create
2.Display
3.Insert
4.Delete
5.Linear Search
6.Exit
1
Enter number of elements in the array: 4
Enter of elements in this array: : 47 18 21 68
: : :
What do you want to perform:
1.Create
2.Display
3.Insert
4.Delete
5.Linear Search
6.Exit
2
47182168

What do you want to perform:
```

QUESTION 2

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

int main(){

  int arr[100],filterArr[100],num;

  cout <<"Enter the number of elements for the array: ";
  cin >>num;

  cout<<"Enter the elements: ";
  for(int i=0;i<num;i++){
    cout<<": ";
    cin>> arr[i];
  }
  int count=0;
  for(int i=0;i<num;i++){
    for(int j=i+1;j<num;){
      if(arr[i]==arr[j]){
        for(int k=j;k<num;k++){
          arr[k]=arr[k+1];
        }
        num--;
      }else{
        j++;
      }
    }
    }
    for(int i=0;i<num;i++){
        cout<< arr[i]<<" ";
    }

    return 0;
}
```



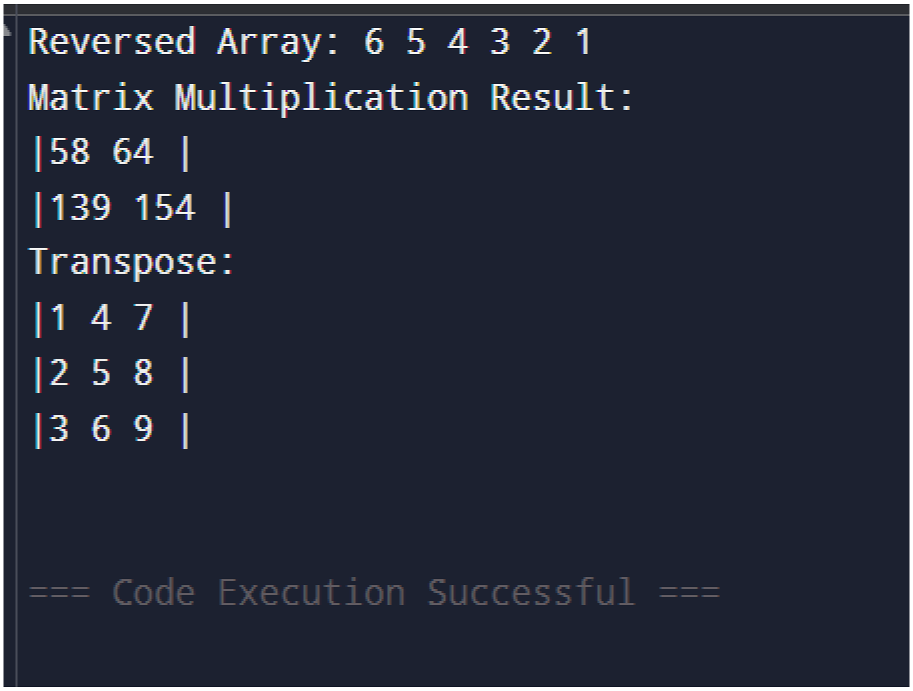
```
Enter the number of elements for the array: 5
Enter the elements: : 82 63 94 32 81
: : : : 82 63 94 32 81

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

QUESTION 3

int answer = 10000

QUESTION 4

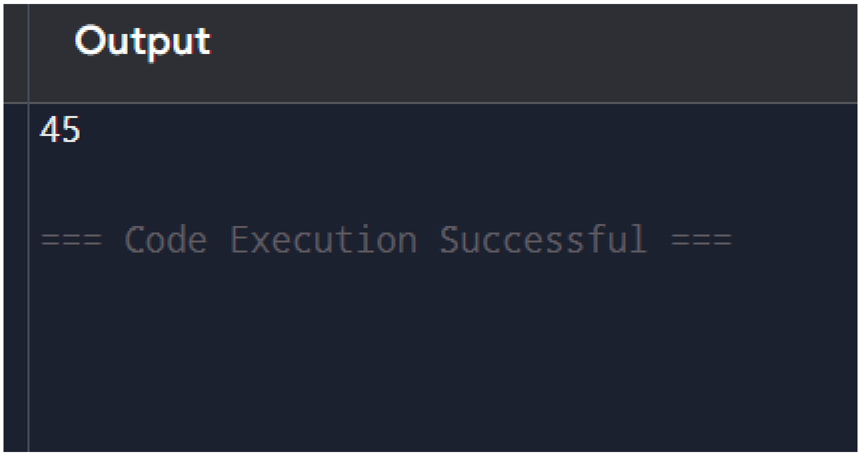
```
#include <iostream>
using namespace std;

int main(){

    int arr[6]={1,2,3,4,5,6};
    int temp=0;
    for(int i=0;i<3;i++){
        temp=arr[i];
        arr[i]=arr[5-i];
        arr[5-i]=temp;
    }
    cout << "Reversed Array: ";
    for(int i=0;i<6;i++){
        cout<< arr[i]<<" ";
    }
    cout<<"\n";

    int A[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    int B[3][2] = {
        {7, 8},
        {9, 10},
        {11, 12}
    };

    int C[2][2] = {0};

    cout << "Matrix Multiplication Result:\n";
    for(int i=0;i<2;i++){
        for (int j=0;j<2;j++){
            for (int k=0;k<3;k++){
                C[i][j]+=A[i][k]*B[k][j];
            }
        }
    }
    for (int i = 0; i < 2; i++) {
        cout<<"|";
        for (int j = 0; j < 2; j++) {
            cout << C[i][j] << " ";
        }
        cout<<"|\n";
    }

    int matrix[3][3]{
        {1,2,3},
        {4,5,6},
        {7,8,9}
    };
    temp=0;
    for(int i=0;i<3;i++){
      for(int j=i+1;j<3;j++){
        temp=matrix[i][j];
        matrix[i][j]=matrix[j][i];
        matrix[j][i]=temp;
      }
    }
    cout << "Transpose:\n";
    for(int i=0;i<3;i++){
      cout<<"|";
      for(int j=0;j<3;j++){
        cout<<matrix[i][j]<<" ";
      }cout << "|\n";
    }
    return 0;
}
```



```
Reversed Array: 6 5 4 3 2 1
Matrix Multiplication Result:
|58 64 |
|139 154 |
Transpose:
|1 4 7 |
|2 5 8 |
|3 6 9 |


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

QUESTION 5

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

int main(){

    int matrix[3][3]{
        {1,2,3},
        {4,5,6},
        {7,8,9}
    };
    int sum=0;

    for(int i=0;i<3;i++){
        for( int j=0;j<3;j++){
            sum=sum+matrix[i][j];
        }
    }
    cout<<sum;

    return 0;
}
```

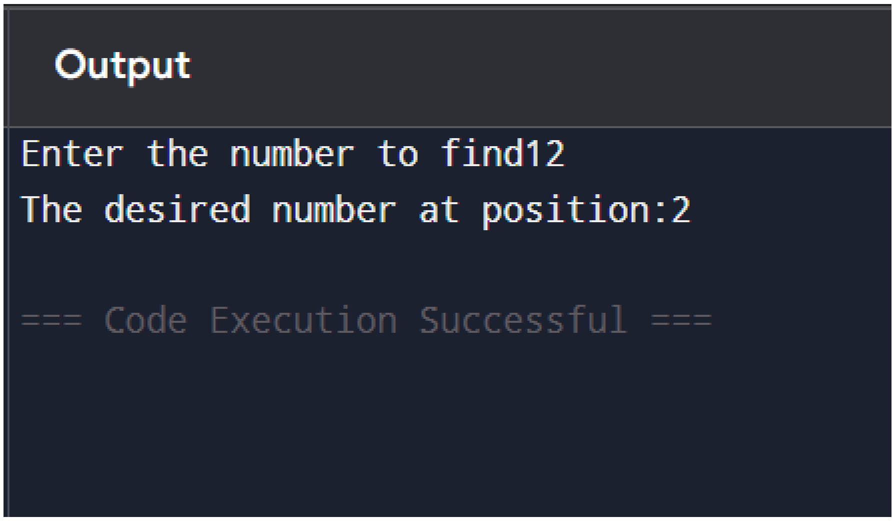


```
45


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

DSA ASSIGNMENT 2

QUESTION 1

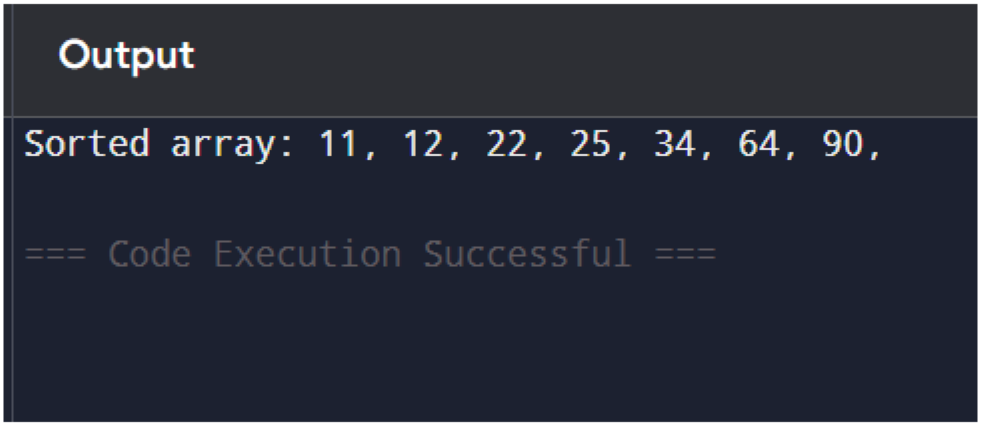
```
#include <iostream>
using namespace std;

int main(){

    int num_to_find;
    int arr[]={5,10,12,32,45,87};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout<<"Enter the number to find";
    cin>>num_to_find;

    int low=0,high=size-1,mid=0;
    bool found=false;

    while(low<high){
        mid=(high+low)/2;
        if(arr[mid]==num_to_find){
            cout<<"The desired number at position:"<<mid;
            found=true;
            break;
        }
        else if(arr[mid]>=num_to_find){

            high=mid-1;

        }

        else if(arr[mid]<=num_to_find){

            low=mid+1;

        }

    }

    if(found!=true){

        cout<<"Number not found";

    }


    return 0;

}
```



```
Enter the number to find12
The desired number at position:2

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

QUESTION 2

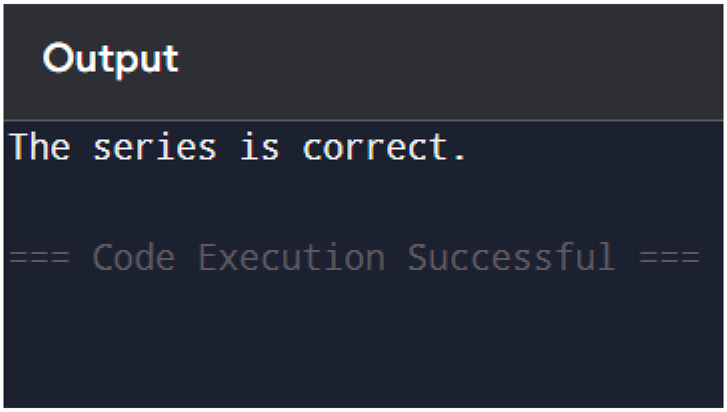
```
#include <iostream>
using namespace std;

int main(){

    int arr[]={64,34,25,12,22,11,90};
    int size=sizeof(arr)/sizeof(arr[0]);
    int temp=0;

    for(int i=0;i<size-1;i++){
        for(int j=0;j<size-1;j++){
            if(arr[j]>arr[j+1]){
                temp=arr[j+1];
                arr[j+1]=arr[j];
                arr[j]=temp;
            }
        }
    }
    cout<<"Sorted array: ";
    for(int i=0;i<size;i++){
        cout<<arr[i]<<", ";
    }

    return 0;
}
```



```
Sorted array: 11, 12, 22, 25, 34, 64, 90,


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

QUESTION 3

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

int main(){

    int arr[]={1,2,3,4,5,6,7,8,9,10,11,12,13};
    int size=sizeof(arr)/sizeof(arr[0]);

    for(int i=0;i<size;i++){
        if(arr[i]!=i+1){
            cout<<"The missing integer is: "<<i+1<<"(Linear search)";
            break;
        }
    }
    int low=0,high=size-1,mid=0;
    bool found=false;
    while(low<high){
        mid=(low+high)/2;
        if(arr[mid]==mid+1){
            low=mid+1;
        }
        else if(arr[mid]==mid+2 and arr[mid-1]==mid+1){
            high=mid-1;
        }
        else if(arr[mid]==mid+2 and arr[mid-1]==mid){
            cout<<"\nThe missing integer is: "<<mid+1<<"(Binary Search)";
            found=true;
            break;
        }
    }
    if(found!=true){
        cout<<"The series is correct.";
    }

    return 0;
}
```



```
The series is correct.

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

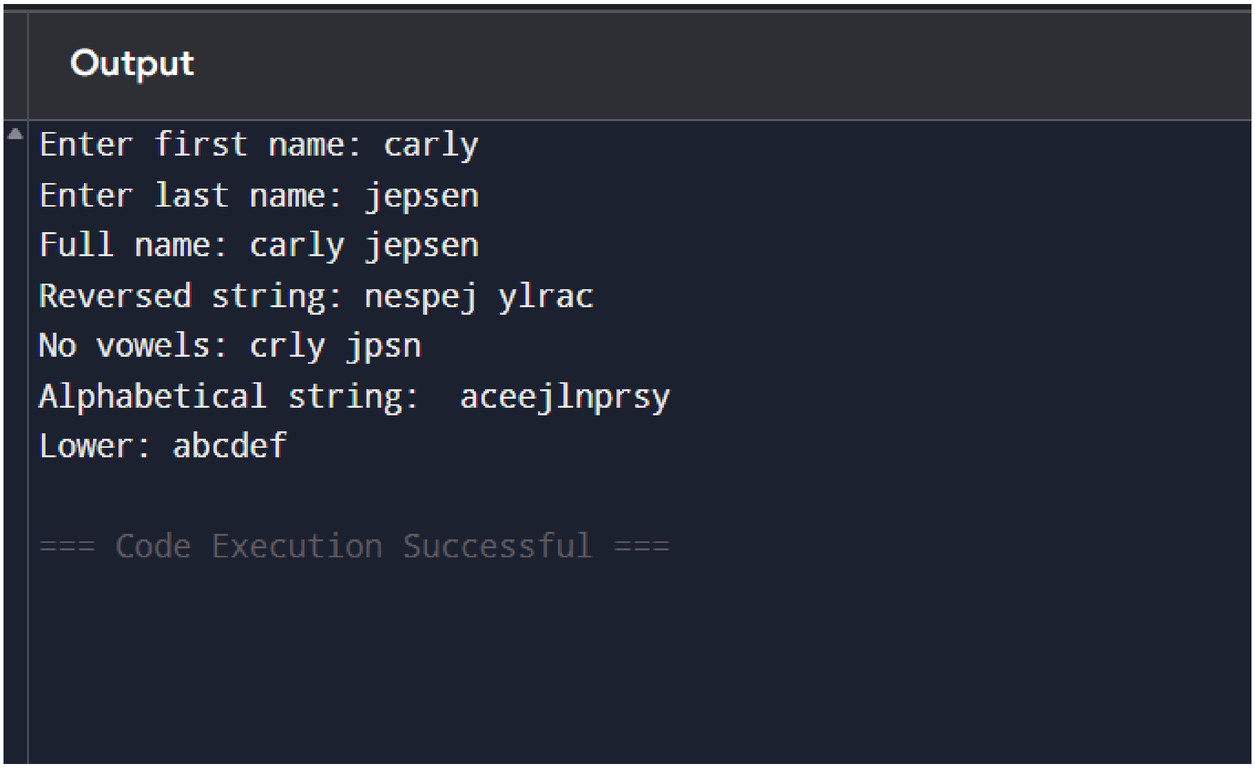
QUESTION 4

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

int main(){

    //(a) Write a program to concatenate one string to another string.

    string first_name,last_name,full_name;
    cout<<"Enter first name: ";
    cin>>first_name;
    cout<<"Enter last name: ";
    cin>>last_name;
    full_name=first_name+" "+last_name;
    cout<<"Full name: "<<full_name;

    int size=full_name.length();
    string reversed_string="";

    //(b) Write a program to reverse a string.

    for(int i=size-1;i>=0;i--){
        reversed_string=reversed_string+full_name[i];
    }
```

```
cout<<"\nReversed string: "<<reversed_string;

//(c) Write a program to delete all the vowels from the string.

char vowels[]={'a','e','i','o','u','A','E','I','O','U'};
string no_vowels="";

for(int i=0;i<size;i++){
    bool is_vowel=false;
    for(int j=0;j<10;j++){
        if(full_name[i]==vowels[j]){
            is_vowel=true;
            break;
        }
    }
    if(is_vowel!=true){
        no_vowels+=full_name[i];
    }
}
cout<<"\nNo vowels: "<<no_vowels;

//(d) Write a program to sort the strings in alphabetical order.

string is_alpha=full_name;
char temp;
```

```
for(int i=0;i<size-1;i++){
    for(int j=0;j<size-1;j++){

        int a=is_alpha[j];
        int b=is_alpha[j+1];

        if(65<=a && a<=90){
            a=a+32;
        }
        if(65<=b && b<=90){
            b=b+32;
        }

        if(a>b){
            temp=is_alpha[j];
            is_alpha[j]=is_alpha[j+1];
            is_alpha[j+1]=temp;
        }
    }
}
cout<<"\nAlphabetical string: "<<is_alpha;

//(e) Write a program to convert a character from uppercase to lowercase.

string upper="ABCDEF";
int size_upper=upper.length();
```

```
    for(int i=0;i<size_upper;i++){
        int c=upper[i];
        if(65<=c && c<=90){
            c=c+32;
            upper[i]=char(c);
        }
    }
    cout<<"\nLower: "<<upper;

    return 0;
}
```



```
Enter first name: carly
Enter last name: jepsen
Full name: carly jepsen
Reversed string: nespej ylrac
No vowels: crly jpsn
Alphabetical string:  aceejlnprsy
Lower: abcdef

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

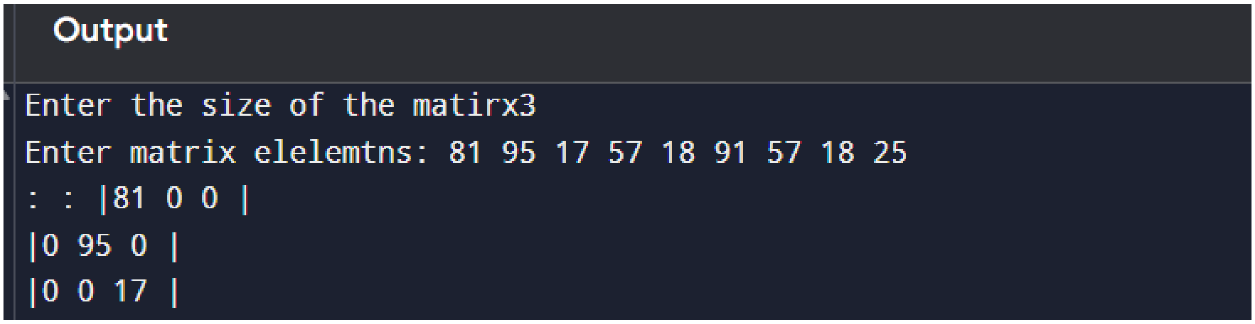
QUESTION 5

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

int main(){

    //(a) Diagonal Matrix.
    int size_diagonal;

    cout<<"Enter the size of the matirx";
    cin>>size_diagonal;

    int arr[size_diagonal],num;

    cout<<"Enter matrix elelemtns";
    for (int i=0;i<size_diagonal;i++){
        cout<<": ";
        cin>>num;
        arr[i]=num;
    }

    for (int i=0;i<size_diagonal;i++){
        cout<<"|";
        for (int j=0;j<size_diagonal;j++){
            if(i==j){
```

```
            cout<<arr[i]<<" ";
        }
        else{
        cout<<"0 ";
        }
    }
    cout<<"|\n";
}

//(b) Tri-diagonal Matrix.

int size_trigonal;

cout<<"Enter the size of the matirx: ";
cin>>size_trigonal;

int arr2[3*size_trigonal-2];

cout<<"Enter matrix elelemtns: ";
for(int i=0;i<3*size_trigonal-2;i++){
    cout<<": ";
    cin>>arr2[i];
}

int count=0;
for (int i=0;i<size_trigonal;i++){
    cout<<"|";
    for(int j=0;j<size_trigonal;j++){
        if(abs(i-j)<=1){
            cout<<arr2[count]<<" ";
            count+=1;
        }
        else{
            cout<<"0 ";
        }
    }
    cout<<"|\n";
}

//(c) Lower triangular Matrix.

cout<<"Enter the size of matrix: ";
int size_lower;
cin>>size_lower;

int arr3[size_lower*(size_lower+1)/2];

cout<<"Enter the non zero elements row wise: ";
for(int i=0;i<size_lower*(size_lower+1)/2;i++){
    cout<<": ";
    cin>>arr3[i];
}
```

```
count=0;
for(int i=0;i<size_lower;i++){
    cout<<"|";
    for(int j=0;j<size_lower;j++){
        if(i-j>=0){
            cout<<arr3[count]<<" ";
            count+=1;
        }
        else{
            cout<<"0 ";
        }
    }
    cout<<"|\n";
}

//(c) Upper triangular Matrix.

cout<<"Enter the size of matrix: ";
int size_upper;
cin>>size_upper;

int arr4[size_upper*(size_upper+1)/2];

cout<<"Enter the non zero elements row wise";
for(int i=0;i<size_upper*(size_upper+1)/2;i++){
    cout<<": ";
    cin>>arr4[i];
}

count=0;
for(int i=0;i<size_upper;i++){
    cout<<"|";
    for(int j=0;j<size_upper;j++){
        if(j-i>=0){
            cout<<arr4[count]<<" ";
            count+=1;
        }
        else{
            cout<<"0 ";
        }
    }
    cout<<"|\n";
}

//(e) Symmetric Matrix

cout<<"Enter the size of matrix: ";
int size_symmetric;
cin>>size_symmetric;

int arr5[size_symmetric*(size_symmetric+1)/2];
```

```
    cout<<"Enter the non zero elements row wise";

    for(int i=0;i<size_symmetric*(size_symmetric+1)/2;i++){

        cout<<": ";

        cin>>arr5[i];

    }


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

        cout<<"|";

        for(int j=0;j<size_symmetric;j++){

            int count1;

            if(j-i>=0){

                count1=i*size_symmetric-(i*(i-1))/2+(j-i);

            }

            else{

                count1=j*size_symmetric-(j*(j-1))/2+(i-j);

            }

            cout<<arr5[count1]<<" ";

        }

        cout<<"|\n";

    }


    return 0;

}
```



```
Enter the size of the matirx3
Enter matrix elelemtns: 81 95 17 57 18 91 57 18 25
: : |81 0 0 |
|0 95 0 |
|0 0 17 |
```

QUESTION 6

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

void transpose(int rows[],int cols[], int values[], int count){\

    int trows[100], tcols[100], tvalues[100];

    for (int i=0;i<count;i++){
        trows[i]=cols[i];
        tcols[i]=rows[i];
        tvalues[i]=values[i];
    }

    cout<<"\nTranspose of the matrix is: ";
    cout << "\nRows\tCol\tValue";
    for (int i=0;i<count;i++){
        cout<<"\n"<<trows[i]<<"\t"<<tcols[i]<<"\t"<<tvalues[i]<<"\t";
    }

}

void addition(int rows1[],int cols1[], int values1[], int count1,int rows2[],int cols2[], int values2[], int count2){
```

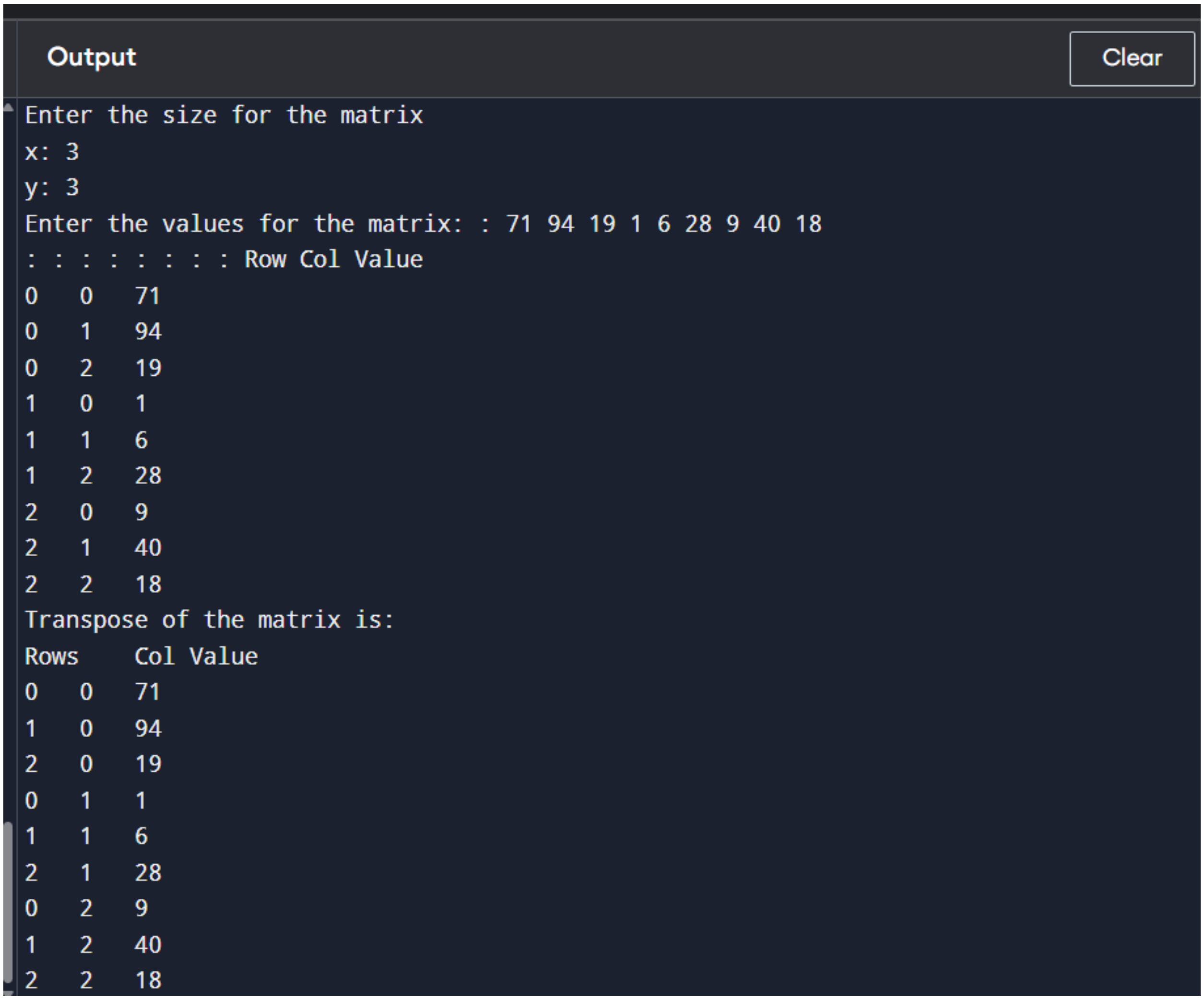
```
int rows3[count1+count2],cols3[count1+count2],values3[count1+count2],count3=0;


int i=0,j=0;

while(i<count1 && j<count2){

   if(rows1[i]==rows2[j] && cols1[i]==cols2[j]){

      rows3[count3]=rows1[i];

      cols3[count3]=cols1[i];

      values3[count3]=values1[i]+values2[j];

      count3++;

      i++;

      j++;

   }

   else if(rows1[i]<rows2[j]||rows1[i]==rows2[j]&&cols1[i]<cols2[j]){

      rows3[count3]=rows1[i];

      cols3[count3]=cols1[i];

      values3[count3]=values1[i];

      count3++;

      i++;

   }

   else if(rows1[i]>rows2[j]||rows1[i]==rows2[j]&&cols1[j]<cols1[i]){

      rows3[count3]=rows1[j];

      cols3[count3]=cols1[j];

      values3[count3]=values1[j];

      count3++;

      j++;

   }
```

```
    }

    while(i<count1){

        rows3[count3]=rows1[i];

        cols3[count3]=cols1[i];

        values3[count3]=values1[i];

        count3++;

        i++;

    }


    while(j<count2){

        rows3[count3]=rows2[j];

        cols3[count3]=cols2[j];

        values3[count3]=values2[j];

        count3++;

        j++;

    }


    cout<<"\nAdditon of both matrix is: \n";

    cout << "\nRows\tCol\tValue";

    for (int i=0;i<count3;i++){

        cout<<"\n"<<rows3[i]<<"\t"<<cols3[i]<<"\t"<<values3[i]<<"\t";

    }


}
```

```
void multiplication(int rows1[],int cols1[], int values1[], int count1,int rows2[],int cols2[], int
values2[], int count2,int sizeMatrix){

    int rows3[sizeMatrix],cols3[sizeMatrix],values3[sizeMatrix],count3=0;

    cout<<"\nMultiplication of both matrix is: \n";
    cout << "\nRows\tCol\tValue";
    for (int i=0;i<count3;i++){
        cout<<"\n"<<rows3[i]<<"\t"<<cols3[i]<<"\t"<<values3[i]<<"\t";
    }
}

int main(){

    int x,y;

    cout<<"Enter the size for the matrix";
    cout<<"\nx: ";
    cin>>x;

    cout<<"y: ";
    cin>>y;

    int arr[x][y];

    cout<<"Enter the values for the matrix: ";
```

```
for (int i=0;i<x;i++){
    for(int j=0;j<y;j++){
        cout<<": ";
        cin>>arr[i][j];
    }
}

int rows[x*y];
int cols[x*y];
int values[x*y];
int count=0;

for (int i=0;i<x;i++){
    for(int j=0;j<y;j++){
        if(arr[i][j]!=0){
            rows[count]=i;
            cols[count]=j;
            values[count]=arr[i][j];
            count++;
        }
    }
}
cout << "Row\tCol\tValue";
for (int i=0;i<count;i++){
    cout<<"\n"<<rows[i]<<"\t"<<cols[i]<<"\t"<<values[i]<<"\t";
}
```

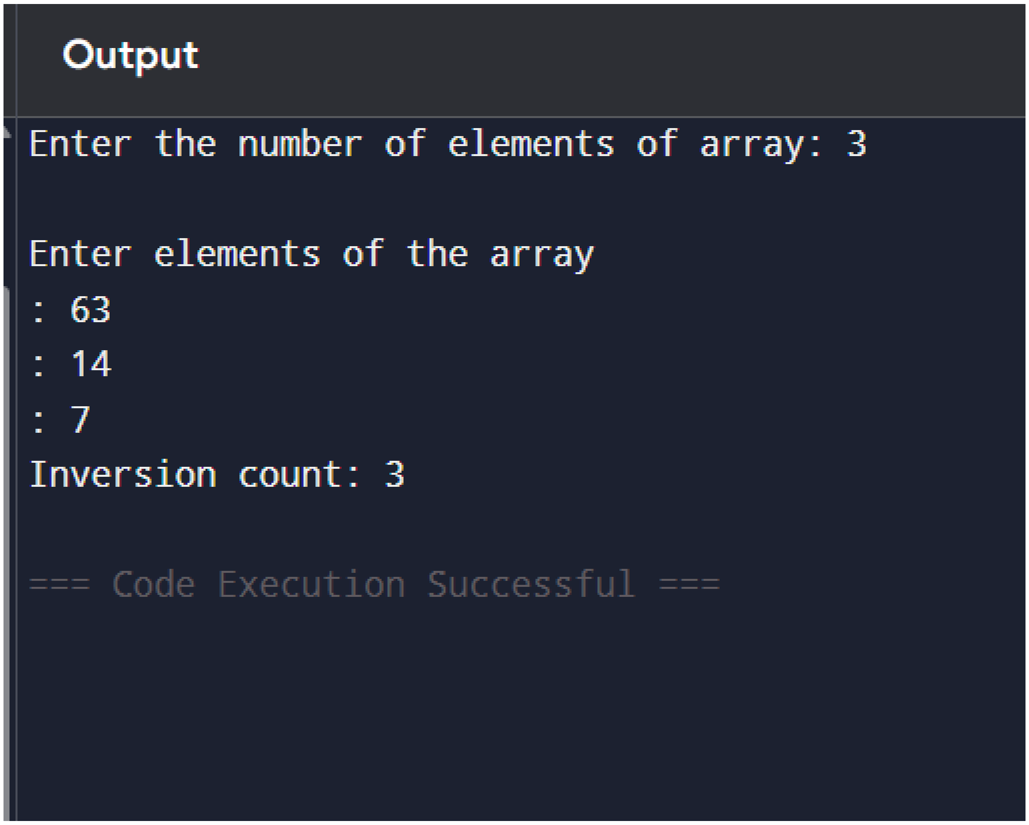
```
    transpose(rows,cols,values,count);

    addition(rows,cols,values,count,rows,cols,values,count);

    multiplication(rows,cols,values,count,rows,cols,values,count,x*y);


    return 0;

}
```



```
Enter the size for the matrix
x: 3
y: 3
Enter the values for the matrix: : 71 94 19 1 6 28 9 40 18
: : : : : : : : Row Col Value
0   0   71
0   1   94
0   2   19
1   0   1
1   1   6
1   2   28
2   0   9
2   1   40
2   2   18
Transpose of the matrix is:
Rows    Col Value
0   0   71
1   0   94
2   0   19
0   1   1
1   1   6
2   1   28
0   2   9
1   2   40
2   2   18
```

QUESTION 7

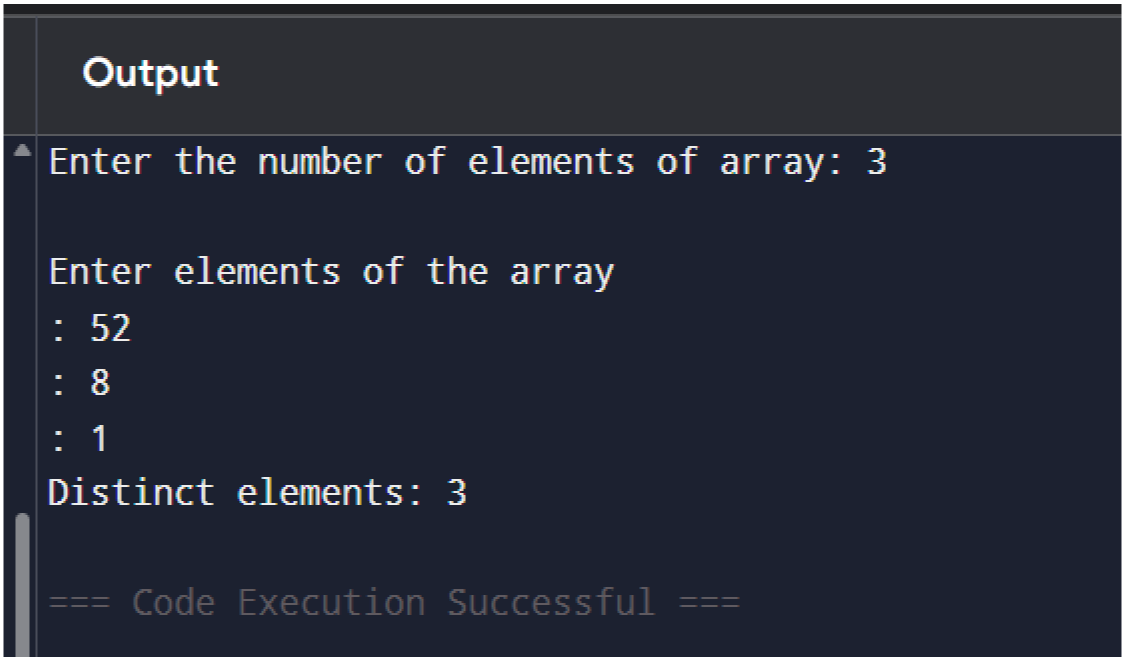
```
#include <iostream>
using namespace std;

int main(){

    int num;
    cout<<"Enter the number of elements of array: ";
    cin>>num;
    int arr[num];

    cout<<"\nEnter elements of the array\n";
    for (int i=0;i<num;i++){
        cout<<": ";
        cin>>arr[i];
    }

    int inversion_count=0;
    for (int i=0;i<num;i++){
        for (int j=i+1;j<num;j++){
            if(arr[i]>arr[j]){
                inversion_count+=1;
            }
        }
    }
    cout<<"Inversion count: "<<inversion_count;


    return 0;
}
```



```
Enter the number of elements of array: 3

Enter elements of the array
: 63
: 14
: 7
Inversion count: 3

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

QUESTION 8

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

int main(){

    int num;
    cout<<"Enter the number of elements of array: ";
    cin>>num;
    int arr[num];

    cout<<"\nEnter elements of the array\n";
    for (int i=0;i<num;i++){
        cout<<": ";
        cin>>arr[i];
    }

    int distinct_elements=0;
    for(int i=0;i<num;i++){
        bool found=false;
        for (int j=i+1;j<num;j++){
            if(arr[i]==arr[j]){
                found=true;
                break;
            }
        }
        if (found==false){
            distinct_elements+=1;
        }
    }

    cout<<"Distinct elements: "<<distinct_elements;

    return 0;
}
```



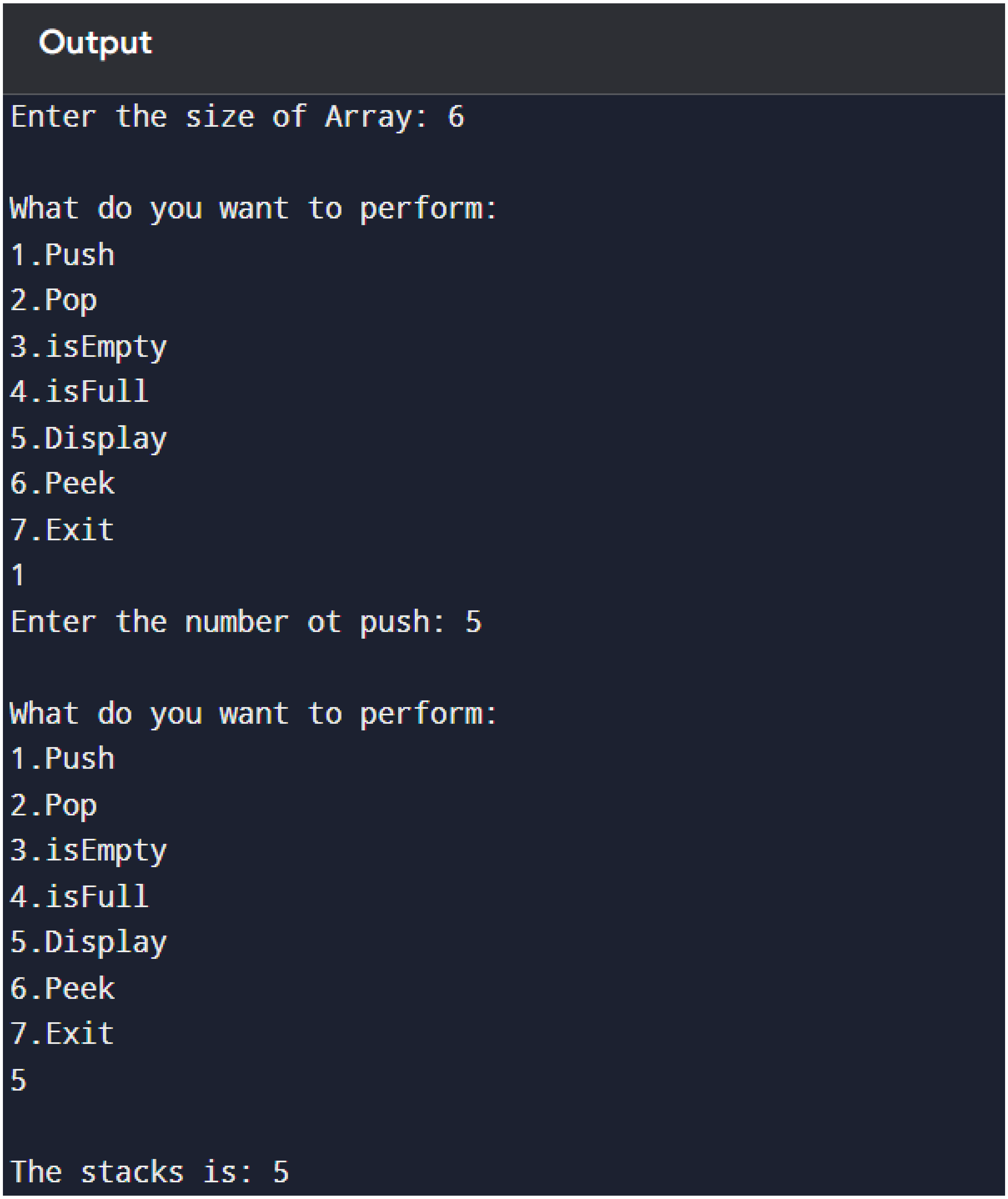
```
Enter the number of elements of array: 3

Enter elements of the array
: 52
: 8
: 1
Distinct elements: 3

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

DSA ASSIGNMENT 3

QUESTION 1

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

int main(){

   cout<<"Enter the size of Array: ";
   int size;
   cin>>size;

   int arr[size];
   int top=-1;

   while(true){
                    cout<<("\nWhat do you want to perform: \n");
                    cout<<("1.Push\n");
                    cout<<("2.Pop\n");
                    cout<<("3.isEmpty\n");
                    cout<<("4.isFull\n");
                    cout<<("5.Display\n");
      cout<<("6.Peek\n");
                    cout<<("7.Exit\n");
                    int num;
                    cin>>num;
                    switch (num) {
                        case 1:{
int num_push;
                        cout<<"Enter the number ot push: ";
cin>>num_push;
if(top==size-1){
    cout<<"Stack is full already";
}
else{
    top++;
    arr[top]=num_push;
}
break;
                        }
                        case 2:{
if(top==-1){
    cout<<"Stack is already empty";
}
else{
    top--;
    cout<<"Removed element: "<<arr[top+1];
}
break;
                        }
                        case 3:{
if(top==-1){
    cout<<"Stack is empty";
}
else{
        cout<<"Stack is not empty";
    }
    break;
                        }
                        case 4:{
    if(top==size-1){
        cout<<"Stack is full";
    }
    else{
        cout<<"Stack is not full";
    }
                            break;
                        }
                        case 5:{
    cout<<"\nThe stacks is: ";
    for(int i=0;i<=top;i++){
        cout<<arr[i]<<"\t";
    }
    break;

                        }
                        case 6:{
    cout<<arr[top];
    break;
                        }
case 7:{
    cout << "Exiting";
                            return 0;
```

```
          }
                              default:
                                   cout<<"Invalid code! Try again";
                    }
   }


   return 0;
}
```

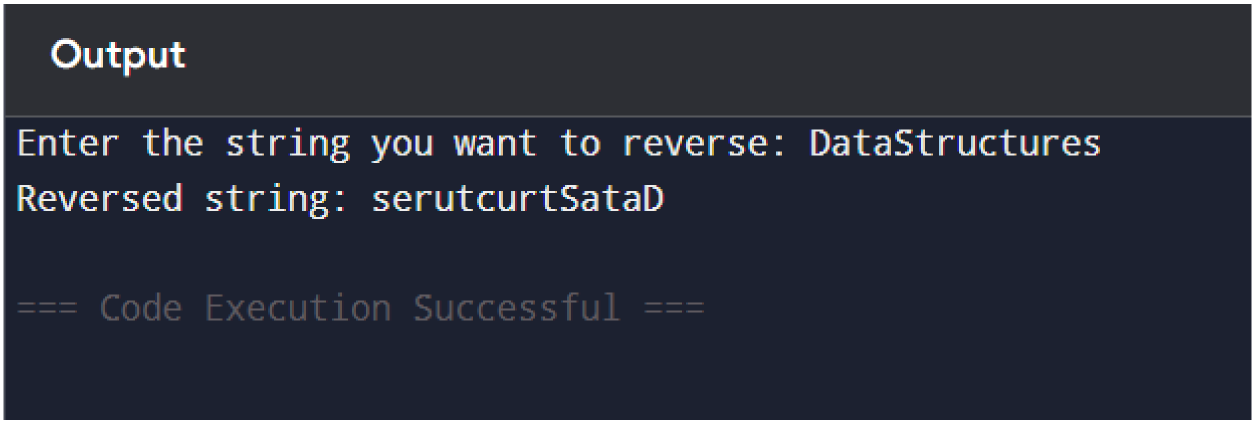


```
Enter the size of Array: 6

What do you want to perform:
1.Push
2.Pop
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
1
Enter the number ot push: 5

What do you want to perform:
1.Push
2.Pop
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
5

The stacks is: 5
```

```
}
```

QUESTION 2

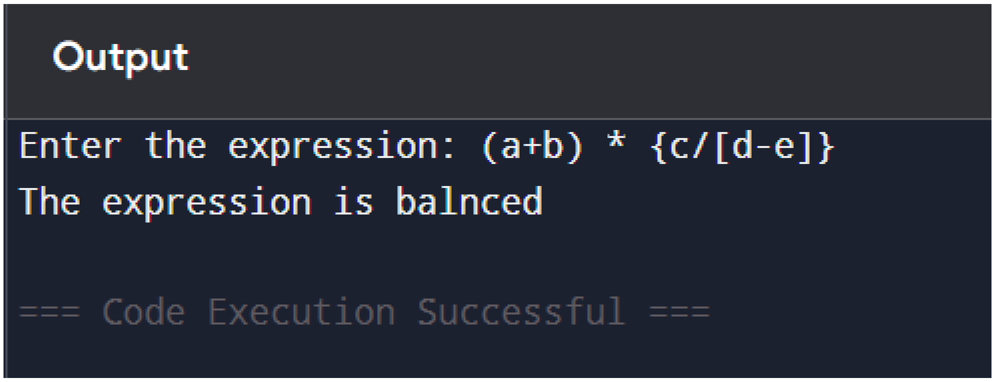
```
#include <iostream>
#include <stack>
using namespace std;

int main(){

    string word;
    stack<char> reverseWord;

    cout<<"Enter the string you want to reverse: ";
    cin>>word;

    for(char ch:word){
        reverseWord.push(ch);
    }

    cout<<"Reversed string: ";
    while(!reverseWord.empty()){
        cout<<reverseWord.top();
        reverseWord.pop();
    }

    return 0;
}
```



```
Enter the string you want to reverse: DataStructures
Reversed string: serutcurtSataD

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

QUESTION 3

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

bool isBalanced(string exp){
    stack<char> pt;

    for(char ch:exp){
        if(ch=='{' || ch=='(' || ch=='['){
            pt.push(ch);
        }
        else if(ch=='}' || ch==')' || ch==']'){
            if(pt.empty()){
                return false;
            }
            char top=pt.top();
            pt.pop();

            if(ch=='}' && top!='{' || ch==')' && top!='(' || ch==']' && top!='['){
                return false;
            }
        }
    }
    return pt.empty();
}

int main(){

    string word;
    stack<char> pt;

    cout<<"Enter the expression: ";
    cin>> word;

    bool balanced=isBalanced(word);

    if(balanced){
        cout<<"The expression is balanced";
    }
    else{
        cout<<"Expression is not balanced";
    }

    return 0;
}
```



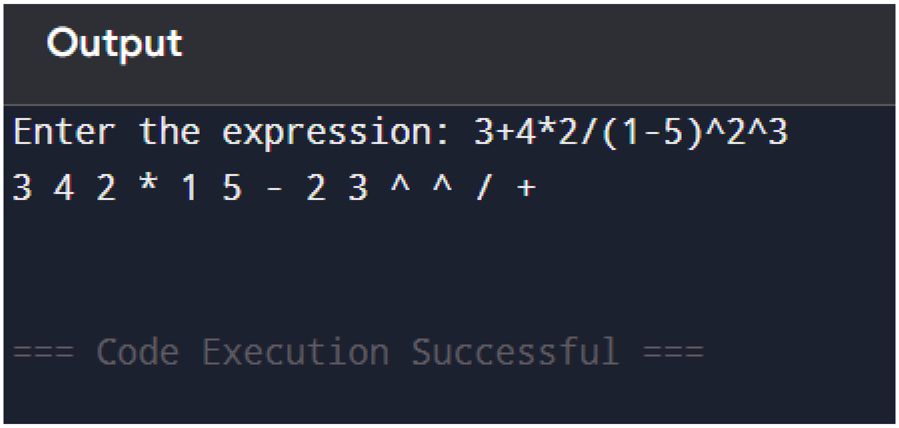
```
Enter the expression: (a+b) * {c/[d-e]}
The expression is balnced

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

QUESTION 4

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype>

using namespace std;

int prec(char op) {
  if (op == '+' || op == '-') return 1;
  if (op == '*' || op == '/') return 2;
  if (op == '^')        return 3;
  return -1; // non-operator
}

bool isOp(char c) {
  return c=='+' || c=='-' || c=='*' || c=='/' || c=='^';
}

// '^' is right-associative; others are left-associative.
bool isRightAssoc(char op) { return op == '^'; }

int main() {
  cout << "Enter the expression: ";
  string line;
  if (!getline(cin, line)) return 0;
```

```
    string out;
    stack<char> ops;

    auto flushOp = [&](char op){
        out += op;
        out += ' ';
    };

    for (size_t i = 0; i < line.size(); ++i) {
        char c = line[i];

        // skip spaces
        if (isspace(static_cast<unsigned char>(c))) continue;

        // numbers/identifiers (alnum and underscore) — emit as one token
        if (isdigit(static_cast<unsigned char>(c)) || isalpha(static_cast<unsigned char>(c)) || c ==
'_') {
            // collect the whole token
            string token;
            while (i < line.size()) {
                char d = line[i];
                if (isalnum(static_cast<unsigned char>(d)) || d == '_') {
                    token += d;
                    ++i;
                } else {
                    break;
                }
            }
            --i; // step back one because for-loop will ++i
    out += token;
    out += ' ';
}
else if (c == '(') {
    ops.push(c);
}
else if (c == ')') {
    bool foundOpen = false;
    while (!ops.empty()) {
        char t = ops.top(); ops.pop();
        if (t == '(') { foundOpen = true; break; }
        flushOp(t);
    }
    if (!foundOpen) {
        cerr << "Error: mismatched parentheses.\n";
        return 1;
    }
}
else if (isOp(c)) {
    // pop while top has higher precedence, or same precedence and current is left-assoc
    while (!ops.empty() && isOp(ops.top())) {
        char top = ops.top();
        int pt = prec(top), pc = prec(c);
        bool shouldPop = (pt > pc) || (pt == pc && !isRightAssoc(c));
        if (!shouldPop) break;
        ops.pop();
        flushOp(top);
    }
```

```
            ops.push(c);
        }
        else {
            cerr << "Error: invalid character '" << c << "'.\n";
            return 1;
        }
    }

    // pop remaining operators
    while (!ops.empty()) {
        if (ops.top() == '(' || ops.top() == ')') {
            cerr << "Error: mismatched parentheses.\n";
            return 1;
        }
        flushOp(ops.top());
        ops.pop();
    }

    // trim trailing space if you want (not necessary)
    cout << out << "\n";
    return 0;
}
```



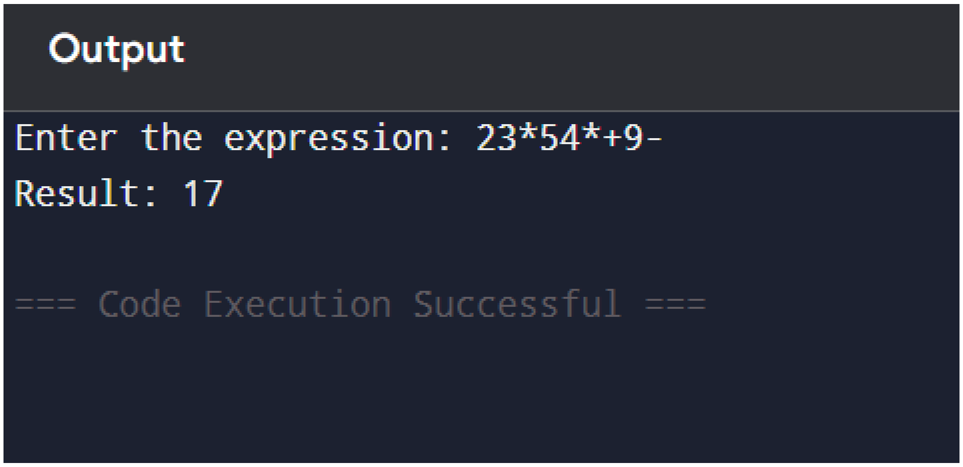
```
Enter the expression: 3+4*2/(1-5)^2^3
3 4 2 * 1 5 - 2 3 ^ ^ / +



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

QUESTION 5

```
#include <iostream>

#include <string>

#include <stack>

#include <math.h>

using namespace std;



bool isOperator(char c){

    if(c=='+' || c=='-' || c=='*' || c=='/' || c=='^'){

        return true;

    }

    else{

        return false;

    }

}


int main(){


    string expression;

    cout<<"Enter the expression: ";

    cin>>expression;

    stack<int> st;


    for(char c:expression){
```

```
if(isdigit(c)){
    st.push(c-'0');
}

else if(isOperator(c)){
    int num1= st.top();
    st.pop();
    int num2= st.top();
    st.pop();

    if(c=='+'){
        st.push(num2+num1);
    }
    else if(c=='-'){
        st.push(num2-num1);
    }
    else if(c=='*'){
        st.push(num2*num1);
    }
    else if(c=='/'){
        st.push(num2/num1);
    }
    else if(c=='^'){
        st.push(pow(num2,num1));
    }
}
```

```
    }
    int result=st.top();
    st.pop();
    if(!st.empty()){
        cout<<"Error!";
    }
    cout<<"Result: "<<result;

    return 0;
}
```



```
Enter the expression: 23*54*+9-
Result: 17

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

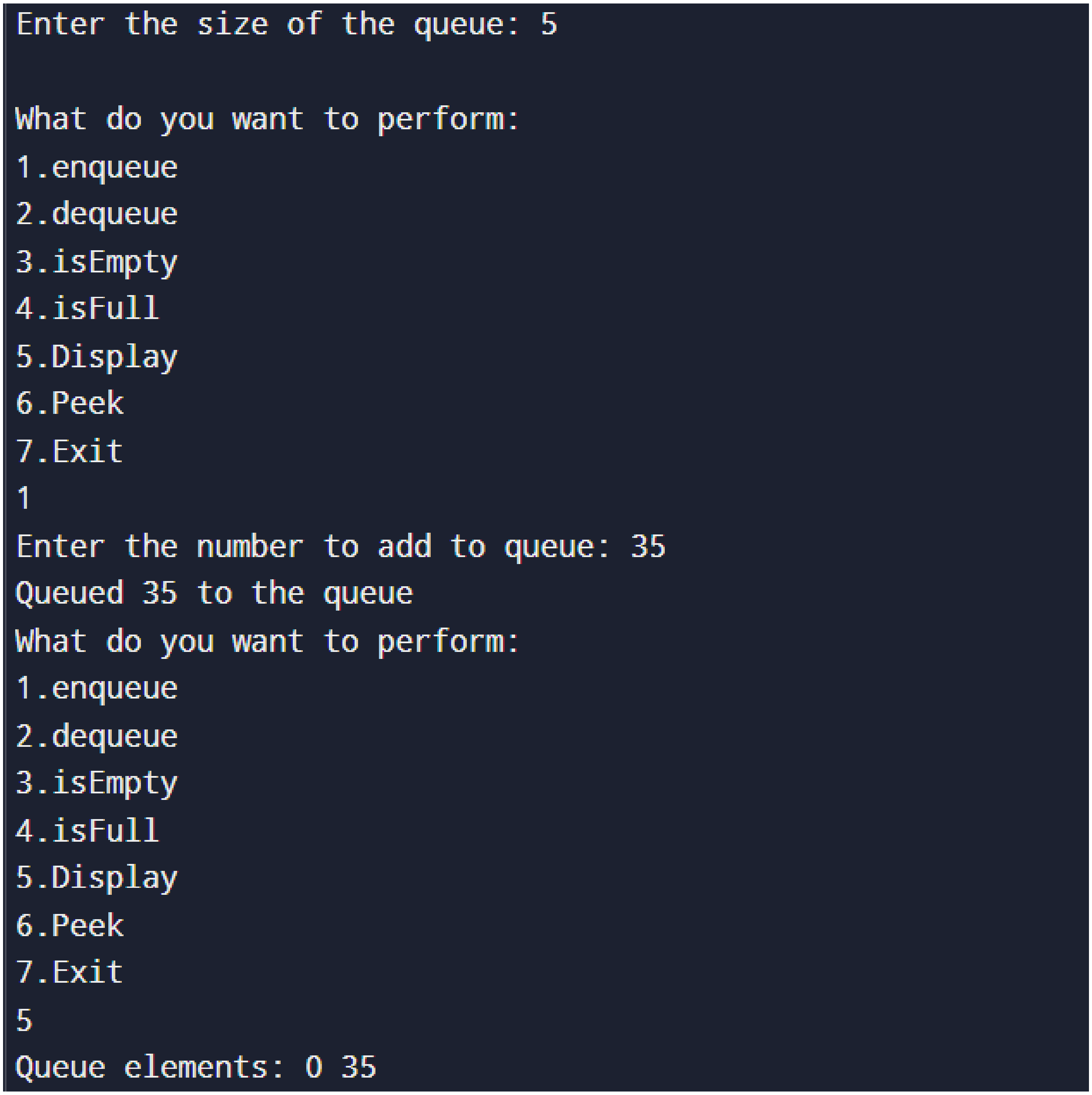
DSA Assignment 4

Q1

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

int main(){

    int size;
    cout<<"Enter the size of the queue: ";
    cin>>size;

    int qu[size];
    int sizern=0;
    int front=-1;
    int back=-1;

    while(true){
        cout<<("\nWhat do you want to perform: \n");
        cout<<("1.enqueue\n");
        cout<<("2.dequeue\n");
        cout<<("3.isEmpty\n");
        cout<<("4.isFull\n");
        cout<<("5.Display\n");
        cout<<("6.Peek\n");
        cout<<("7.Exit\n");
        int num;
        cin>>num;

        switch (num)
        {
        case 1:
            if(sizern<size){
                int numPush;
                back++;
                cout<<"Enter the number to add to queue: ";
                cin>>numPush;
                qu[back]=numPush;
                cout<<"Queued "<<numPush<<" to the queue";
                sizern++;
            }
```

```
   else{
      cout<<"queue is full!";
   }

   break;
case 2:
   if(sizern!=0){
      front++;
      int numPop=qu[front];
      sizern--;
      cout<<"Removed "<<numPop<<" from the queue";
      if(sizern==0){
         front=-1;
         back=-1;
      }
   }
   else{
      cout<<"The queue is already empty!";
   }
   break;
case 3:
   if(sizern==0){
      cout<<"The queue is empty";
   }
   else{
      "The queueue is not empty";
   }

   break;
case 4:
   if(sizern==size){
      cout<<"The queue is full";
   }
   else{
      cout<<"The queue is not full";
   }
   break;
case 5:
   if (sizern==0) {
      cout << "Queue is empty!";
   } else {
      cout << "Queue elements: ";
      for(int i=front;i<=back;i++){
         cout<<qu[i]<<" ";
```

```
                }
            }
            break;
        case 6:
            if (sizern==0) {
                cout << "Queue is empty!";
            } else {
                cout << "Front element: " << qu[front];
            }
            break;
            break;
        case 7:
            cout<<"Exiting! ";
            return 0;

        default:
            cout<<"Invalid code. Try again!";
            break;
        }
    }
}
```



```
Enter the size of the queue: 5

What do you want to perform:
1.enqueue
2.dequeue
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
1
Enter the number to add to queue: 35
Queued 35 to the queue
What do you want to perform:
1.enqueue
2.dequeue
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
5
Queue elements: 0 35
```

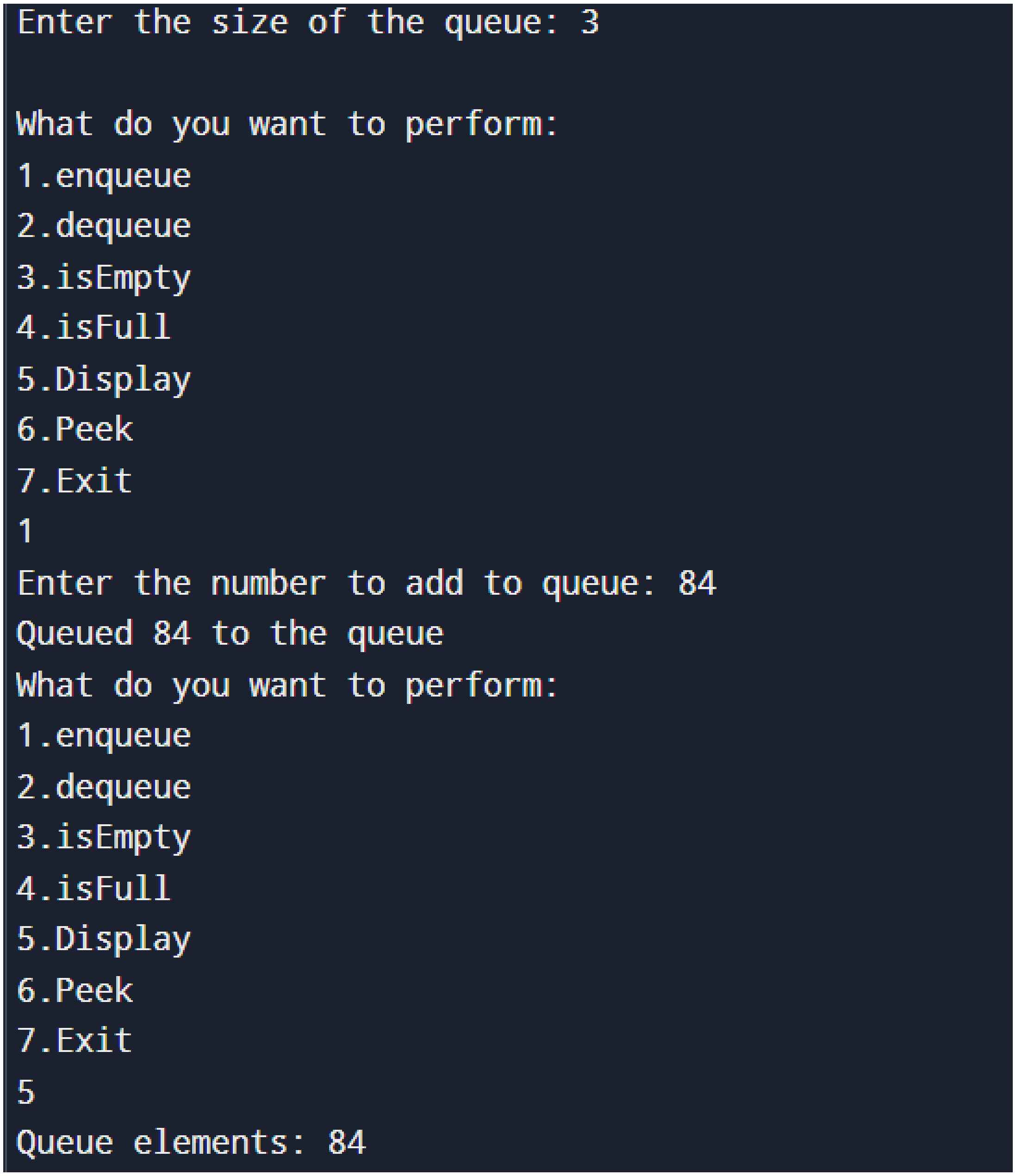
Q2

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

int main(){

   int size;
```

```
cout<<"Enter the size of the queue: ";
cin>>size;

int qu[size];
int sizern=0;
int front=-1;
int back=-1;

while(true){
    cout<<("\nWhat do you want to perform: \n");
    cout<<("1.enqueue\n");
    cout<<("2.dequeue\n");
    cout<<("3.isEmpty\n");
    cout<<("4.isFull\n");
    cout<<("5.Display\n");
    cout<<("6.Peek\n");
    cout<<("7.Exit\n");
    int num;
    cin>>num;

    switch (num)
    {
    case 1:
        if(sizern<size){
            if(front==-1){
                front=0;
            }
            int numPush;
            back=(back+1) % size;
            cout<<"Enter the number to add to queue: ";
            cin>>numPush;
            qu[back]=numPush;
            cout<<"Queued "<<numPush<<" to the queue";
            sizern++;
        }
        else{
            cout<<"queue is full!";
```

```
   }

   break;
case 2:
   if(sizern!=0){
      int numPop=qu[front];
      front=(front+1)%size;
      sizern--;
      cout<<"Removed "<<numPop<<" from the queue";
      if(sizern==0){
         front=-1;
         back=-1;
      }
   }
   else{
      cout<<"The queue is already empty!";
   }
   break;
case 3:
   if(sizern==0){
      cout<<"The queue is empty";
   }
   else{
      cout<<"The queueue is not empty";
   }

   break;
case 4:
   if(sizern==size){
      cout<<"The queue is full";
   }
   else{
      cout<<"The queue is not full";
   }
   break;
case 5:
   if (sizern==0) {
```

```
                cout << "Queue is empty!";
            } else {
                cout << "Queue elements: ";
                for(int i=0;i<sizern;i++){
                    cout<<qu[(front + i) % size] << " ";
                }
            }
            break;
        case 6:
            if (sizern==0) {
                cout << "Queue is empty!";
            } else {
                cout << "Front element: " << qu[front];
            }
            break;
            break;
        case 7:
            cout<<"Exiting! ";
            return 0;

        default:
            cout<<"Invalid code. Try again!";
            break;
        }
    }
}
```



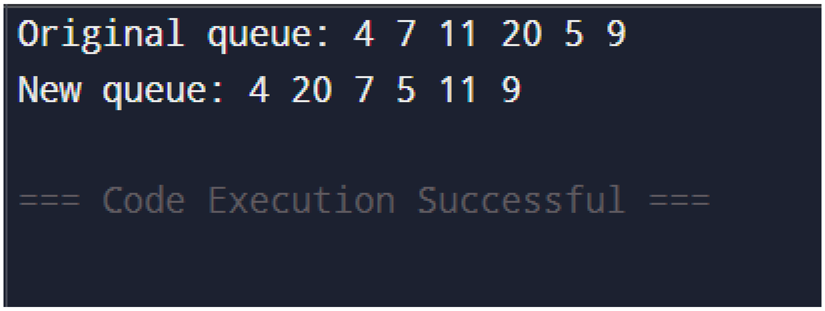
```
Enter the size of the queue: 3

What do you want to perform:
1.enqueue
2.dequeue
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
1
Enter the number to add to queue: 84
Queued 84 to the queue
What do you want to perform:
1.enqueue
2.dequeue
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
5
Queue elements: 84
```

Q3

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

int main(){

   queue<int> qu;
```

```
int arr[]={4,7,11,20,5,9};

int n=sizeof(arr)/sizeof(arr[0]);

for(int i=0;i<n;i++){
   qu.push(arr[i]);
}
queue<int> temp;
temp=qu;

cout<<"Original queue: ";
for(int i=0;i<n;i++){
   cout<<temp.front()<<" ";
   temp.pop();
}

temp=qu;
queue<int> newqu;

for(int i=0;i<n/2;i++){
   temp.pop();
}
queue<int> temp1=qu;

for(int i=0;i<n;i++){
   if(i%2==0){
      newqu.push(temp1.front());
      temp1.pop();
   }
   else{
      newqu.push(temp.front());
      temp.pop();
   }
}
queue<int> temp2=newqu;

cout<<"\nNew queue: ";
```

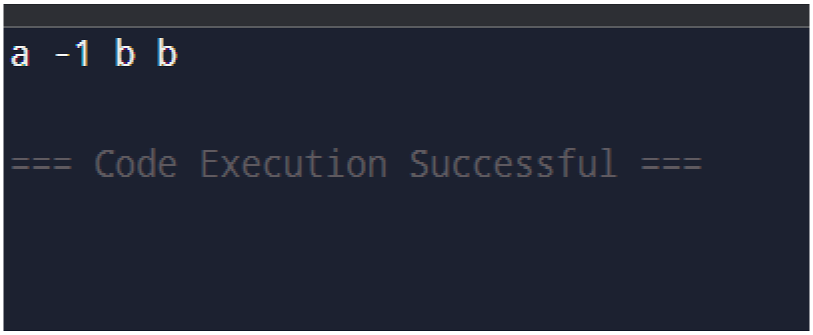
```
    for(int i=0;i<n;i++){
        cout<<temp2.front()<<" ";
        temp2.pop();
    }
    return 0;
}
```



```
Original queue: 4 7 11 20 5 9
New queue: 4 20 7 5 11 9

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

Q4

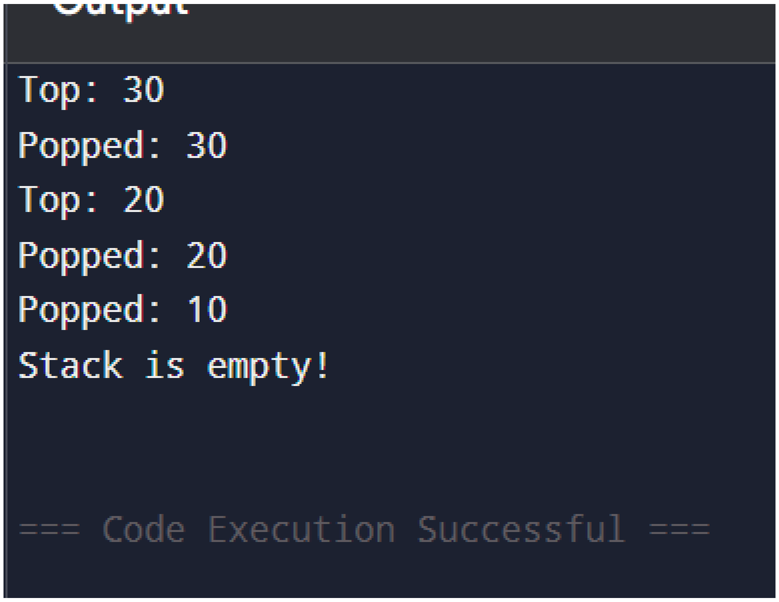
```
#include <iostream>
#include <queue>
using namespace std;

int main(){

    int freq[26] = {0};
    queue<char> qu;
    char arr[]={'a','a','b','c'};

    int size=sizeof(arr)/sizeof(arr[0]);

    for(int i=0;i<size;i++){
        char ch=arr[i];
        freq[ch-'a']++;
        qu.push(ch);
        while(freq[qu.front()-'a']>1){
            qu.pop();
        }
        if(qu.empty()){
            cout<<-1<<" ";
        }
        else{
        cout<<qu.front()<<" ";
        }
    }
    return 0;
}
```



```
a -1 b b

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

Q5 A

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

queue<int> q1, q2;

void push(int x) {
    q2.push(x);

    // Move all elements from q1 → q2
    while(!q1.empty()) {
        q2.push(q1.front());
        q1.pop();
    }
    swap(q1, q2);
}

void pop(){
    if (q1.empty()){
        cout<<"Stack is empty!\n";
        return;
    }
    cout<<"Popped: "<< q1.front() <<endl;
    q1.pop();
}

int top(){
    if (q1.empty()){
        cout<<"Stack is empty!\n";
        return -1;
    }
    return q1.front();
}

bool empty(){
    return q1.empty();
}

int main() {
    push(10);
    push(20);
    push(30);

    cout <<"Top: "<<top()<<endl;
    pop();
    cout <<"Top: "<<top()<<endl;
    pop();
    pop();
    pop();
}
```

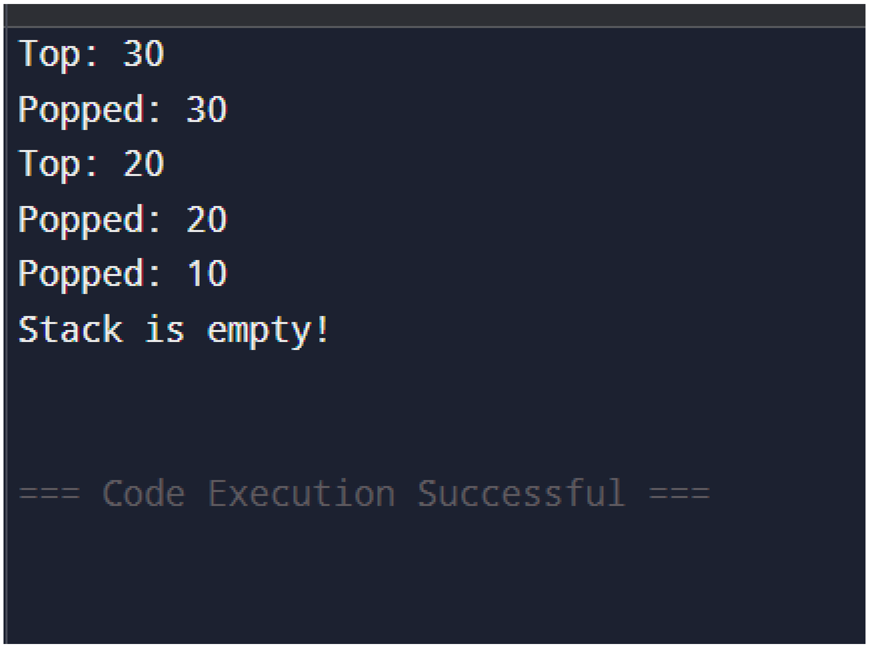


```
Top: 30
Popped: 30
Top: 20
Popped: 20
Popped: 10
Stack is empty!


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

Q5 B

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

queue<int> q;

void push(int x) {
   int size = q.size();
   q.push(x);

   // Rotate elements to make new element front
   for (int i=0; i<size; i++) {
      q.push(q.front());
      q.pop();
   }
}

void pop(){
   if (q.empty()){
      cout<<"Stack is empty!\n";
      return;
```

```
    }
    cout<<"Popped: "<<q.front()<<endl;
    q.pop();
}

int top(){
    if(q.empty()) {
        cout<<"Stack is empty!\n";
        return -1;
    }
    return q.front();
}

bool empty(){
    return q.empty();
}

int main(){
    push(10);
    push(20);
    push(30);

    cout<<"Top: "<<top()<<endl;
    pop();
    cout<<"Top: "<<top()<<endl;
    pop();
    pop();
    pop();
}
```



```
Top: 30
Popped: 30
Top: 20
Popped: 20
Popped: 10
Stack is empty!


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

DSA Assignment 5

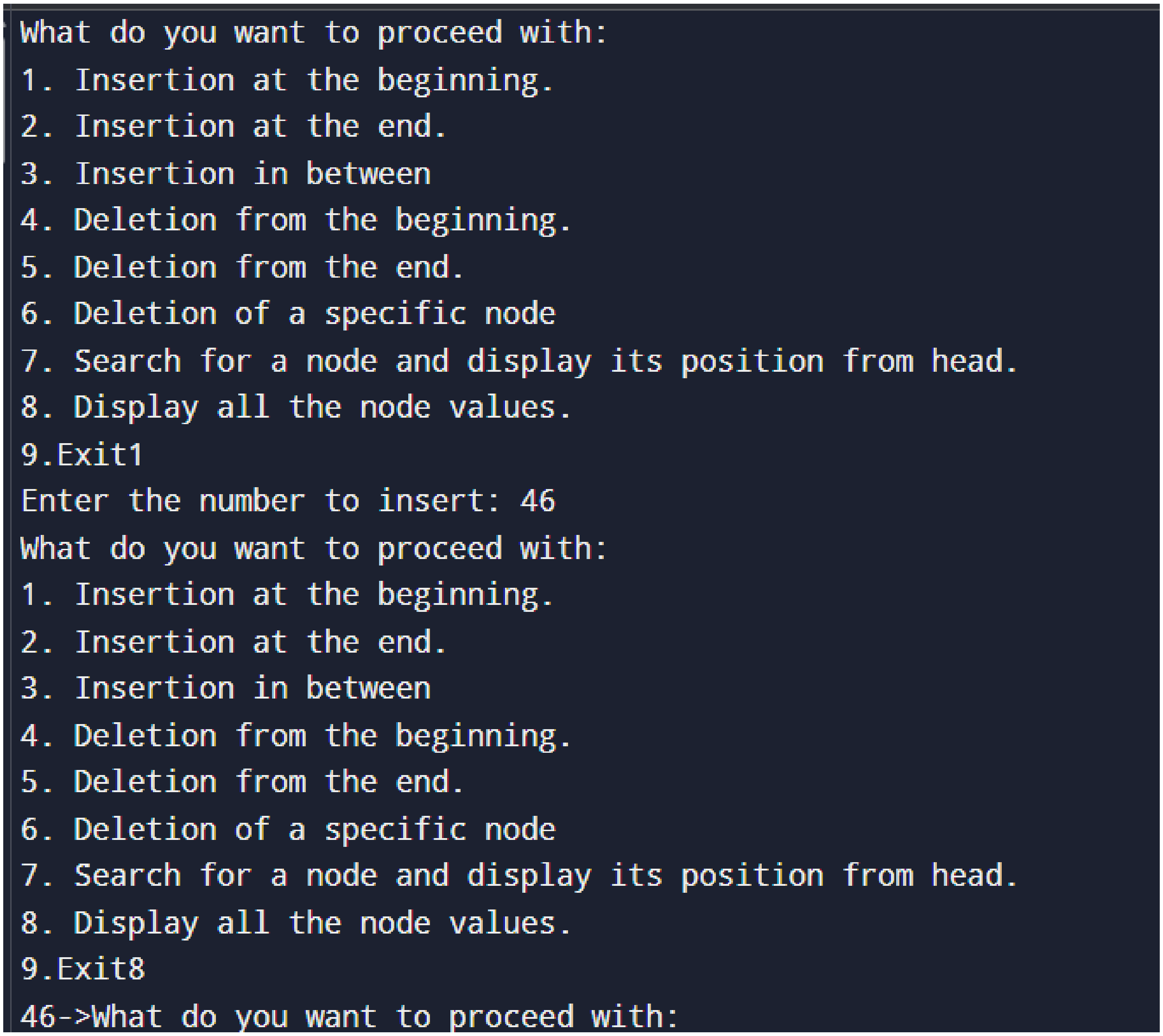
Q1

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

struct Node {
  int data;
  Node* next;
};

void InsertAtBeginning(Node*&head, int value){
  Node*newNode=new Node();
  newNode->data=value;
  newNode->next=head;
  head=newNode;
}

void InsertAtEnd(Node*&head, int value){
  Node*newNode= new Node();
  newNode->data=value;
  newNode->next=NULL;

  if(head==NULL){
    head=newNode;
    return;
  }
  Node*temp= head;
  while(temp->next!=NULL){
    temp=temp->next;
  }
  temp->next=newNode;

}
void InsertAtBetween(Node*&head, int value, int position){
  Node*newNode=new Node();
  newNode->data=value;
  newNode->next=NULL;

  if (position==0){
    newNode->next=head;
    head=newNode;
    return;
  }

  Node*temp=head;
  for (int i=0; i<position-1 && temp!=NULL; i++){
    temp=temp->next;
  }

  if (temp == NULL) {
    cout << "Position out of range!" << endl;
    delete newNode;
    return;
  }
  newNode->next=temp->next;
  temp->next=newNode;
}

void DeleteAtBeginning(Node*&head){
  if (head == NULL) {
    cout << "List is empty, nothing to delete." << endl;
    return;
  }
  head = head->next;
}
void DeleteAtEnd(Node*&head){
  if (head == NULL) {
    cout << "List is empty, nothing to delete." << endl;
    return;
  }
  if(head->next==NULL){
    delete head;
    head=NULL;
    return;

  }

  Node*temp=head;
  Node*temp2;
  while(temp->next!=NULL){
    temp2=temp;
    temp=temp->next;
  }
  delete temp;
  temp2->next=NULL;



}
void DeleteAtBetween(Node*&head, int position){
  if (head == NULL) {
    cout << "List is empty, nothing to delete." << endl;
    return;
  }
  if (position == 1) {
    head = head->next;
    return;
  }
  Node*temp=head;
  for(int i=0; i<position-2 && temp->next != NULL;i++){
    temp=temp->next;
  }
  Node*temp2=temp;
  temp=temp->next;
  temp2->next=temp->next;
  delete temp;

}
int searchNode(Node*&head, int value){
  Node*temp=head;
  int count=1;
  while (temp != NULL) {
    if (temp->data == value) {
        return count;
    }
    temp = temp->next;
    count++;
  }
  return -1;
}
void displayNode(Node*&head){
  Node*temp=head;
  while(temp!=NULL){
    cout<<temp->data<<"->";
    temp=temp->next;
  }

}



int main(){


  Node*head =NULL;
  int value,num=0;


  while(num!=9){
    cout<<"What do you want to proceed with: \n";
    cout<<"1. Insertion at the beginning.\n";
```

```
cout<<"2. Insertion at the end.\n";
cout<<"3. Insertion in between\n";
cout<<"4. Deletion from the beginning.\n";
cout<<"5. Deletion from the end.\n";
cout<<"6. Deletion of a specific node\n";
cout<<"7. Search for a node and display its position from head.\n";
cout<<"8. Display all the node values.\n";
cout<<"9.Exit";
cin>>num;
int tempNum;
int tempPos;
switch (num)
{
case 1:
   cout<<"Enter the number to insert: ";
   cin>>tempNum;
   InsertAtBeginning(head,tempNum);
   break;
case 2:
   cout<<"Enter the number to insert: ";
   cin>>tempNum;
   InsertAtEnd(head,tempNum);
   break;
case 3:
   cout<<"Enter the number to insert: ";
   cin>>tempNum;
   cout<<"Which position to insert";
   cin>>tempPos;
```

```
    InsertAtBetween(head,tempNum,tempPos);
    break;
case 4:
    DeleteAtBeginning(head);
    break;
case 5:
    DeleteAtEnd(head);
    break;
case 6:
    cout<<"Which position to delete";
    cin>>tempPos;
    DeleteAtBetween(head,tempPos);
    break;
case 7:
    cout<<"Which number to find";
    cin>>tempNum;
    tempPos=searchNode(head,tempNum);
    if(tempPos!=-1) {
        cout<<tempNum<<" found at position "<<tempPos<<endl;
    }
    else{
        cout<<tempNum<<"not found in the list."<<endl;
    }
    break;
case 8:
    displayNode(head);
    break;
case 9:
```

```
            cout<<"Exiting!";
            break;

        default:
        cout<<"Choose a valid option!";
            break;
        }
    }


    return 0;
}
```



```
What do you want to proceed with:
1. Insertion at the beginning.
2. Insertion at the end.
3. Insertion in between
4. Deletion from the beginning.
5. Deletion from the end.
6. Deletion of a specific node
7. Search for a node and display its position from head.
8. Display all the node values.
9.Exit1
Enter the number to insert: 46
What do you want to proceed with:
1. Insertion at the beginning.
2. Insertion at the end.
3. Insertion in between
4. Deletion from the beginning.
5. Deletion from the end.
6. Deletion of a specific node
7. Search for a node and display its position from head.
8. Display all the node values.
9.Exit8
46->What do you want to proceed with:
```

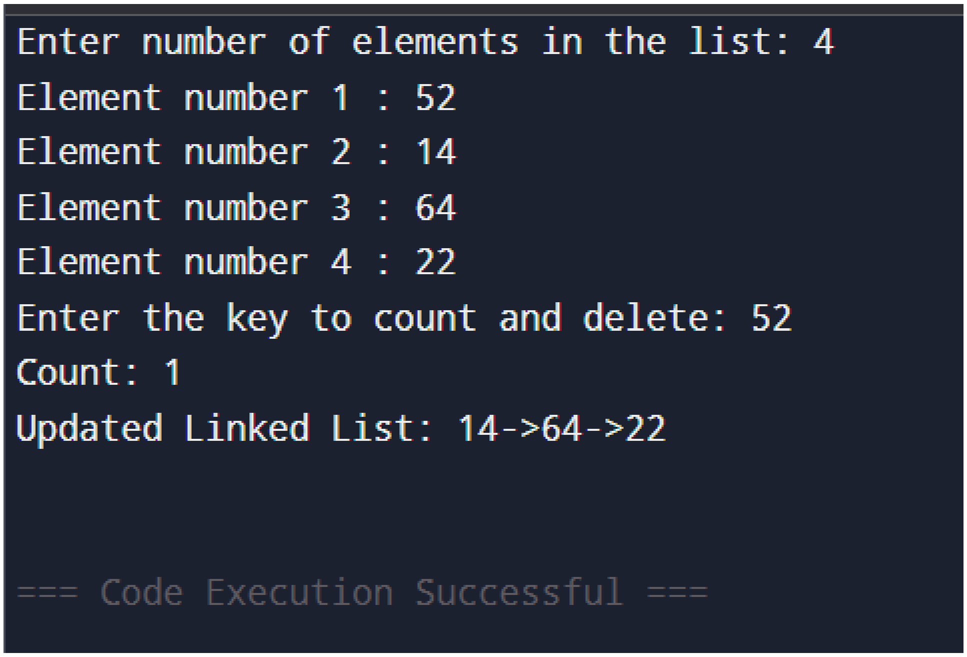
Q2

```
#include <iostream>

using namespace std;


struct Node {

   int data;

   Node* next;

};
```

```
int main(){
    int num=0;
    while(num<=0){
        cout<<"Enter number of elements in the list: ";
        cin>>num;
    }

    Node* head=NULL;
    int value;

    for(int i=0;i<num;i++){
        cout<<"Element number "<<i+1<<" : ";
        cin>>value;
        Node*temp=new Node;
        temp->data=value;
        temp->next=NULL;
        if (head == NULL) {
            head = temp;
        }
        else {
            Node*current=head;
            while(current->next != NULL){
                current=current->next;
            }
            current->next=temp;
        }
    }
```

```
int key;
cout<<"Enter the key to count and delete: ";
cin>>key;

Node*current =head;
Node*prev =NULL;
int count=0;

while(current!=NULL){
   if(current->data==key){
      count++;
      Node* toDelete =current;
      if(prev==NULL){
         head=current->next;
      }
      else{
         prev->next=current->next;
      }
      current=current->next;
      delete toDelete;
   }
   else{
      prev=current;
      current=current->next;
   }
}

cout<<"Count: "<<count<< endl;
```

```
  cout<<"Updated Linked List: ";
  current = head;

  while(current != NULL){
    cout<<current->data;
    if (current->next != NULL) cout << "->";
    current = current->next;
  }
  cout << endl;

  return 0;
}
```



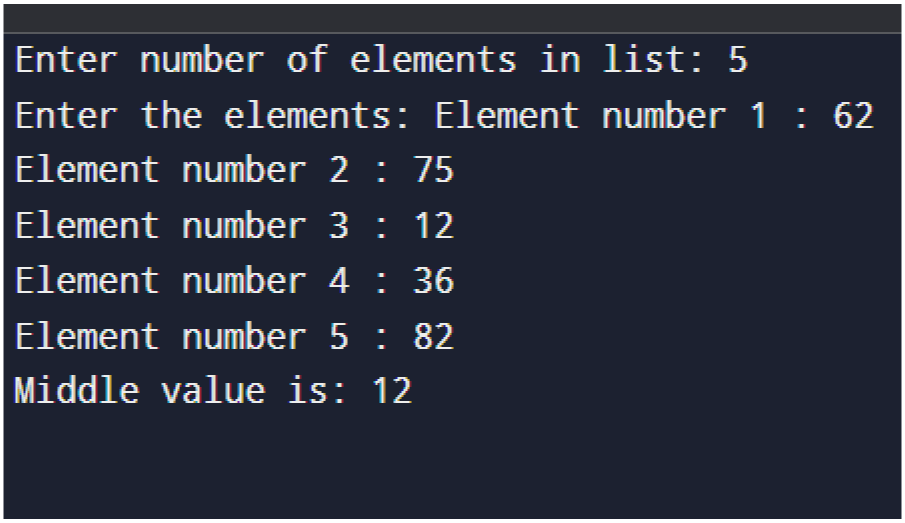
```
Enter number of elements in the list: 4
Element number 1 : 52
Element number 2 : 14
Element number 3 : 64
Element number 4 : 22
Enter the key to count and delete: 52
Count: 1
Updated Linked List: 14->64->22


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

Q3

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

struct Node{
   int data;
   Node* next;
};

void findMiddle(Node*&head){
   Node*temp=head;
   int count=0;
   while(temp!=NULL){
      temp=temp->next;
      count++;
   }
   temp=head;
   for(int i=0;i<count/2;i++){
      temp=temp->next;

   }
   cout<<"Middle value is: "<<temp->data<<endl;
}

int main(){

   int num=0;
   while(num%2==0){
      cout<<"Enter number of elements in list: ";
```

```
    cin>>num;
}
Node*head=NULL;

int arr[num];
int value=0;
cout<<"Enter the elements: ";
for(int i=0;i<num;i++){
    cout<<"Element number "<<i+1<<" : ";
    cin>>value;
    Node*temp=new Node;
    temp->data=value;
    temp->next=NULL;
    if (head == NULL) {
        head = temp;
    }
    else {
        Node*current=head;
        while(current->next != NULL){
            current=current->next;
        }
        current->next=temp;
    }
}
findMiddle(head);


return 0;
```

```
}
```



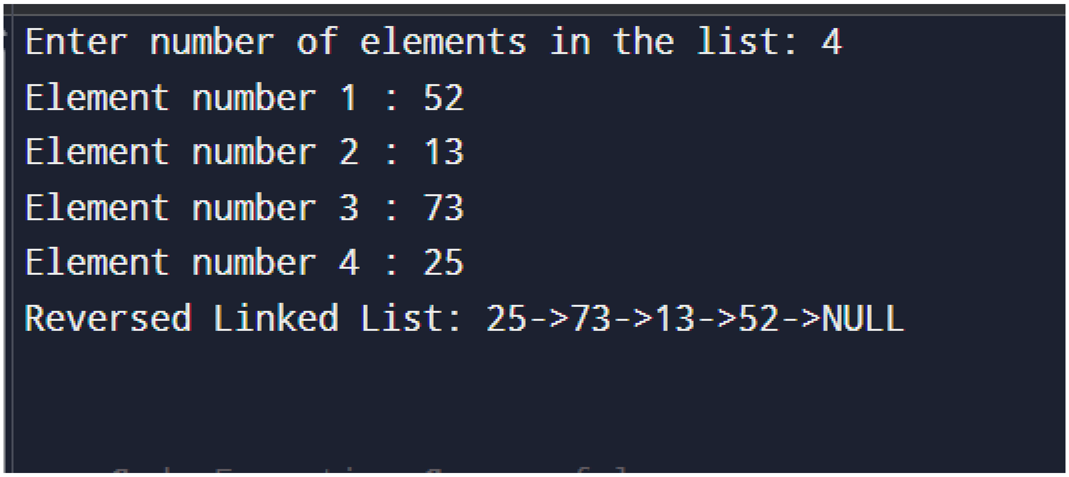
```
Enter number of elements in list: 5
Enter the elements: Element number 1 : 62
Element number 2 : 75
Element number 3 : 12
Element number 4 : 36
Element number 5 : 82
Middle value is: 12
```

Q4

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

struct Node {
   int data;
   Node* next;
};

int main(){
   int num=0;
   while(num <= 0){
      cout<<"Enter number of elements in the list: ";
      cin>>num;
   }
```

```
Node* head =NULL;
int value;

for(int i=0;i<num;i++){
    cout<<"Element number "<<i+1<<" : ";
    cin>>value;
    Node*temp=new Node;
    temp->data=value;
    temp->next=NULL;
    if (head==NULL) {
        head= temp;
    }
    else {
        Node*current=head;
        while(current->next != NULL){
            current=current->next;
        }
        current->next=temp;
    }
}

Node*prev= NULL;
Node*current= head;
Node*next= NULL;

while(current!=NULL){
    next=current->next;
    current->next= prev;
```

```
        prev=current;
        current=next;
    }
    head = prev;

    cout<< "Reversed Linked List: ";
    current=head;
    while(current != NULL){
        cout<<current->data;
        if (current->next != NULL) cout<<"->";
        current = current->next;
    }
    cout << "->NULL" << endl;

    return 0;
}
```



```
Enter number of elements in the list: 4
Element number 1 : 52
Element number 2 : 13
Element number 3 : 73
Element number 4 : 25
Reversed Linked List: 25->73->13->52->NULL
```

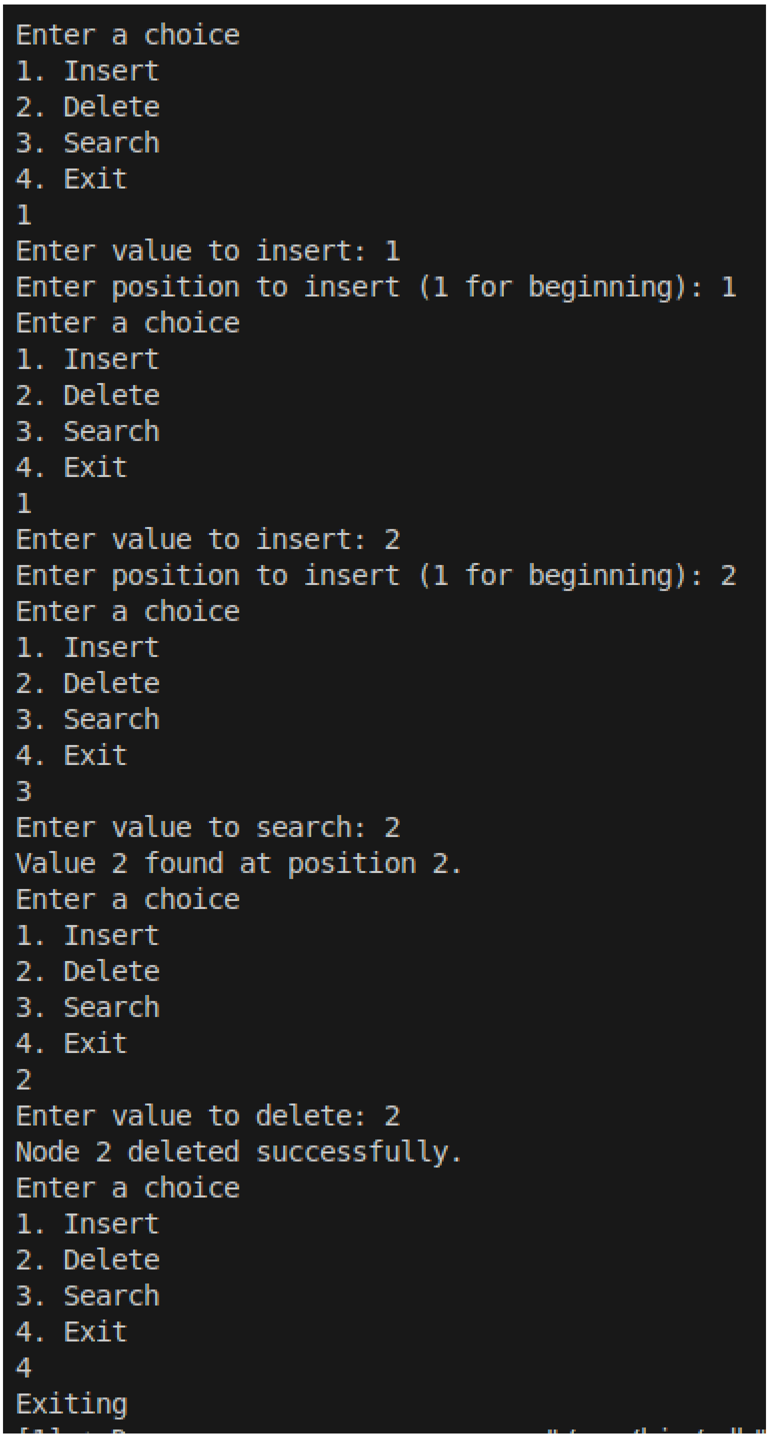
DSA ASSIGNMENT 6

QUESTION 1

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

struct Node{
   int data;
   Node* next;
   Node* prev;
};
void Insert(Node*&head,int value,int pos){
   Node*newNode=new Node();
   newNode->data=value;
   if (head==NULL){
      newNode->next=newNode;
      newNode->prev=newNode;
      head=newNode;
      return;
   }
   else if(pos==1){
      Node*last=head->prev;
      newNode->next=head;
      newNode->prev=last;
      head->prev=newNode;
      last->next=newNode;
      head=newNode;
   }
   else{
      Node*temp=head;
      int count=0;
      while (count<pos-1&&temp->next!=head) {
         temp=temp->next;
         count++;
      }
      Node*nextNode=temp->next;
      temp->next=newNode;
      newNode->prev=temp;
      newNode->next=nextNode;
      nextNode->prev=newNode;
      return;
   }
}
void Delete(Node*&head,int value){
   if(head==NULL){
      cout<<"List is empty, nothing to delete";
```

```
    }
    Node*temp=head;
    Node*toDelete=nullptr;

    do{
        if(temp->data==value) {
            toDelete=temp;
            break;
        }
        temp=temp->next;
    } while(temp!=head);

    if (!toDelete){
        cout<<"Value "<<value<<" not found.\n";
        return;
    }

    if(toDelete->next==toDelete){
        delete toDelete;
        head=nullptr;
    }
    else{
        Node*prevNode=toDelete->prev;
        Node*nextNode=toDelete->next;
        prevNode->next=nextNode;
        nextNode->prev=prevNode;

        if(toDelete == head)
            head=nextNode;

        delete toDelete;
    }

    cout<<"Node "<<value<<" deleted successfully.\n";
}
void Search(Node*&head, int value){
    Node*temp=head;
    int pos=1;
    do{
        if(temp->data == value) {
            cout<<"Value "<<value<<" found at position "<<pos<< ".\n";
            return;
        }
        temp=temp->next;
        pos++;
    } while(temp!=head);

    cout<< "Value "<<value<<" not found.\n";
}
```

```
int main(){
    Node* head = nullptr;
    int choice, value, pos;
    while(true){
        cout<<"Enter a choice\n";
        cout<<"1. Insert\n";
        cout<<"2. Delete\n";
        cout<<"3. Search\n";
        cout<<"4. Exit\n";
        cin >> choice;
        switch(choice){
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                cout << "Enter position to insert (1 for beginning): ";
                cin >> pos;
                Insert(head, value, pos);
                break;

            case 2:
                cout << "Enter value to delete: ";
                cin >> value;
                Delete(head, value);
                break;

            case 3:
                cout << "Enter value to search: ";
                cin >> value;
                Search(head, value);
                break;

            case 4:
                cout << "Exiting\n";
                return 0;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    }
}
```

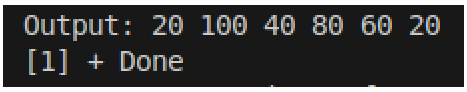


```
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
1
Enter value to insert: 1
Enter position to insert (1 for beginning): 1
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
1
Enter value to insert: 2
Enter position to insert (1 for beginning): 2
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
3
Enter value to search: 2
Value 2 found at position 2.
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
2
Enter value to delete: 2
Node 2 deleted successfully.
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
4
Exiting
```

QUESTION 2

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

struct Node {
   int data;
   Node* next;
};

void Insert(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   if(head==nullptr){
      head=newNode;
      head->next=head;
      return;
   }
   Node*temp=head;
   while (temp->next!=head)
      temp=temp->next;
   temp->next=newNode;
   newNode->next=head;
}

void displayCircular(Node* head) {
   if(head==nullptr){
      cout<<"List is empty.\n";
      return;
   }
   Node*temp=head;
   do{
      cout<<temp->data<< " ";
      temp=temp->next;
   } while(temp!=head);

   cout<<head->data<<endl;
}

int main() {
   Node*head=nullptr;

   Insert(head,20);
   Insert(head,100);
   Insert(head,40);
   Insert(head,80);
   Insert(head,60);
   cout<<"Output: ";
```

```
    displayCircular(head);

    return 0;
}
```

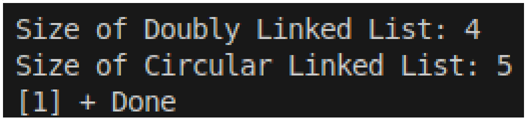


```
Output: 20 100 40 80 60 20
[1] + Done
```

QUESTION 3

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

struct Node{
   int data;
   Node* next;
   Node* prev;
};

void InsertDoubly(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   newNode->next=nullptr;
   newNode->prev=nullptr;

   if(head==nullptr){
      head=newNode;
      return;
   }

   Node*temp=head;
   while(temp->next!=nullptr)
      temp=temp->next;

   temp->next=newNode;
   newNode->prev=temp;
}

int sizeDoubly(Node* head){
   int count=0;
   Node* temp=head;
   while(temp){
      count++;
      temp=temp->next;
   }
   return count;
}

struct NodeC{
   int data;
   NodeC* next;
};

void InsertCircular(NodeC*& head, int value) {
   NodeC*newNode=new NodeC();
   newNode->data=value;
```

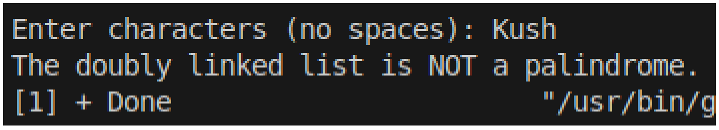
```
    if(head==nullptr){
        head=newNode;
        head->next=head;
        return;
    }
    NodeC*temp=head;
    while (temp->next!=head)
        temp=temp->next;
    temp->next=newNode;
    newNode->next=head;
}
int sizeCircular(NodeC*head2) {
    if(!head2)
        return 0;

    int count=0;
    NodeC*temp=head2;
    do{
        count++;
        temp=temp->next;
    } while(temp!=head2);

    return count;
}

int main() {
    Node* head=nullptr;
    InsertDoubly(head,10);
    InsertDoubly(head,20);
    InsertDoubly(head,30);
    InsertDoubly(head,40);

    cout<<"Size of Doubly Linked List: "<<sizeDoubly(head)<< endl;

    NodeC* head2=nullptr;
    InsertCircular(head2, 20);
    InsertCircular(head2, 100);
    InsertCircular(head2, 40);
    InsertCircular(head2, 80);
    InsertCircular(head2, 60);

    cout<<"Size of Circular Linked List: "<<sizeCircular(head2)<<endl;
    return 0;
}
```



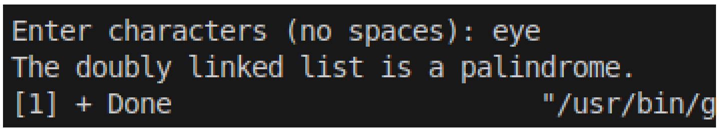
```
Size of Doubly Linked List: 4
Size of Circular Linked List: 5
[1] + Done
```

QUESTION 4

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

struct Node {
   char data;
   Node* next;
   Node* prev;
};

void Insert(Node*& head, char value) {
   Node*newNode=new Node();
   newNode->data=value;
   newNode->next=nullptr;
   newNode->prev=nullptr;

   if(head==nullptr){
      head=newNode;
      return;
   }

   Node*temp=head;
   while(temp->next!=nullptr)
      temp=temp->next;

   temp->next=newNode;
   newNode->prev=temp;
}

bool isPalindrome(Node*head) {
   if(head==nullptr)
      return true;

   Node*left=head;
   Node*right=head;
   while(right->next!=nullptr)
      right=right->next;

   while(left!=right && right->next!=left) {
      if (left->data!=right->data)
         return false;
      left=left->next;
      right=right->prev;
   }
   return true;
}

int main() {
   Node*head=nullptr;
   string input;

   cout<<"Enter characters (no spaces): ";
   cin>>input;

   for(char ch : input)
      Insert(head,ch);

   if(isPalindrome(head))
      cout<<"The doubly linked list is a palindrome."<<endl;
   else
      cout<<"The doubly linked list is NOT a palindrome."<<endl;

   return 0;
}
```



```
Enter characters (no spaces): Kush
The doubly linked list is NOT a palindrome.
[1] + Done                       "/usr/bin/g
```

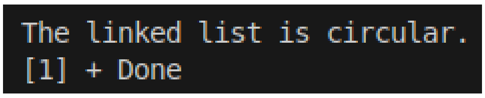


```
Enter characters (no spaces): eye
The doubly linked list is a palindrome.
[1] + Done                       "/usr/bin/g
```

QUESTION 5

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

struct Node {
   int data;
   Node* next;
};

void Insert(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   if(head==nullptr){
      head=newNode;
      head->next=head;
      return;
   }
   Node*temp=head;
   while (temp->next!=head)
      temp=temp->next;
   temp->next=newNode;
   newNode->next=head;
}

bool isCircular(Node* head){
   if(head==nullptr)
      return false;

   Node*temp=head->next;
   while(temp!=nullptr && temp!=head)
      temp=temp->next;

   return(temp==head);
}

int main() {
   Node* head=nullptr;

   Insert(head,10);
   Insert(head,20);
   Insert(head,30);
   Insert(head,40);
   Insert(head,50);

   if (isCircular(head))
      cout<<"The linked list is circular."<<endl;
   else
```

```
        cout<<"The linked list is NOT circular."<<endl;

    return 0;
}
```



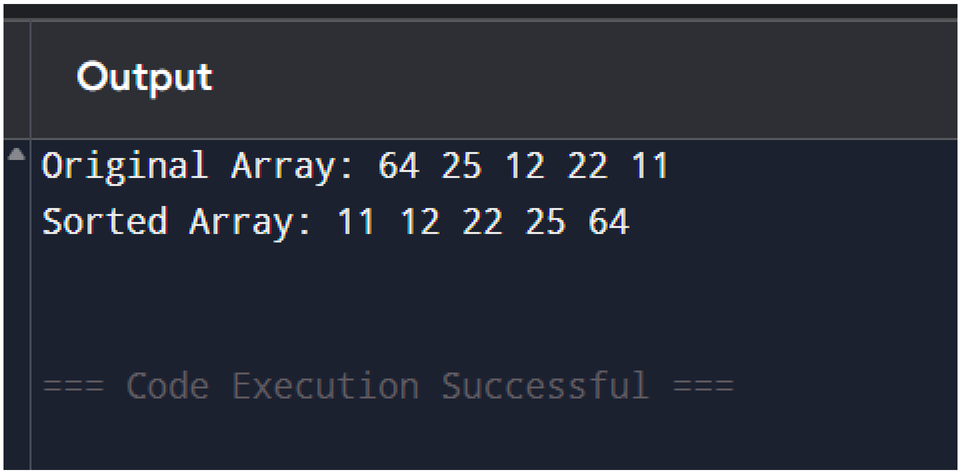
```
The linked list is circular.
[1] + Done
```

DSA ASSIGNMENT 7

QUESTION 1

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

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;

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

        swap(arr[i], arr[minIndex]);
    }
}

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        bool swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        if (!swapped) break;
    }
}
void merge(int arr[], int l, int mid, int r) {
    int n1 = mid - l + 1;
    int n2 = r - mid;

    int a[n1], b[n2];

    for (int i = 0; i < n1; i++) a[i] = arr[l + i];
    for (int i = 0; i < n2; i++) b[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (a[i] <= b[j]) arr[k++] = a[i++];
        else arr[k++] = b[j++];
    }

    while (i < n1) arr[k++] = a[i++];
    while (j < n2) arr[k++] = b[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l >= r) return;

    int mid = l + (r - l) / 2;

    mergeSort(arr, l, mid);
    mergeSort(arr, mid + 1, r);
    merge(arr, l, mid, r);
}
int partition(int arr[], int l, int r) {
    int pivot = arr[r];
    int i = l - 1;

    for (int j = l; j < r; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[r]);
    return i + 1;
}
```

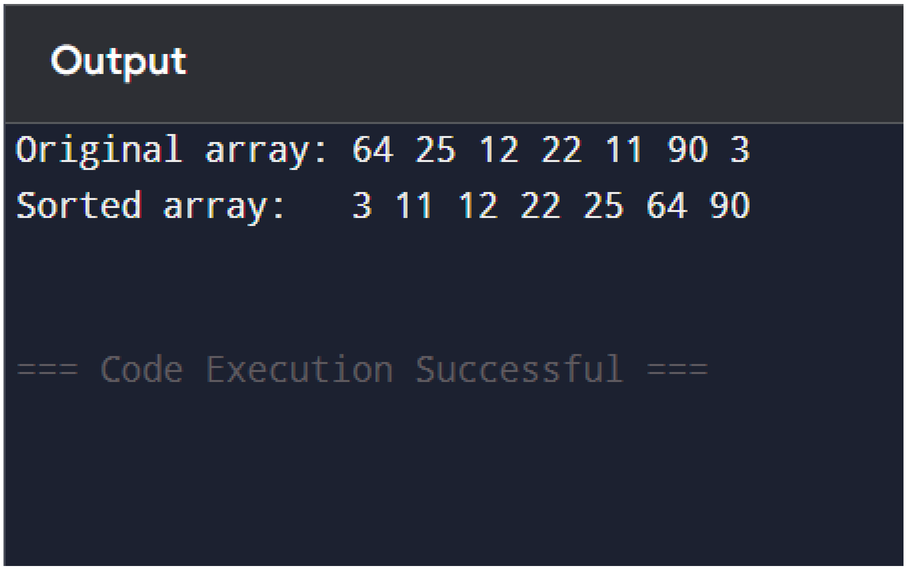
```
void quickSort(int arr[], int l, int r) {
    if (l < r) {
        int pi = partition(arr, l, r);
        quickSort(arr, l, pi - 1);
        quickSort(arr, pi + 1, r);
    }
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = 5;

    cout << "Original Array: ";
    printArray(arr, n);
    selectionSort(arr, n);
    // insertionSort(arr, n);
    // bubbleSort(arr, n);

    // mergeSort(arr, 0, n-1);

    quickSort(arr, 0, n-1);

    cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```



```
Original Array: 64 25 12 22 11
Sorted Array: 11 12 22 25 64


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

QUESTION 2

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

void improvedSelectionSort(int arr[], int n) {
    int left = 0, right = n - 1;

    while (left < right) {

        int minIndex = left;
        int maxIndex = right;

        if (arr[minIndex] > arr[maxIndex])
            swap(arr[minIndex], arr[maxIndex]);

        for (int i = left + 1; i < right; i++) {

            if (arr[i] < arr[minIndex])
                minIndex = i;

            else if (arr[i] > arr[maxIndex])
                maxIndex = i;
        }

        swap(arr[left], arr[minIndex]);

        if (maxIndex == left)
            maxIndex = minIndex;

        swap(arr[right], arr[maxIndex]);
        left++;
        right--;
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {64, 25, 12, 22, 11, 90, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    improvedSelectionSort(arr, n);

    cout << "Sorted array:   ";
    printArray(arr, n);

    return 0;
}
```

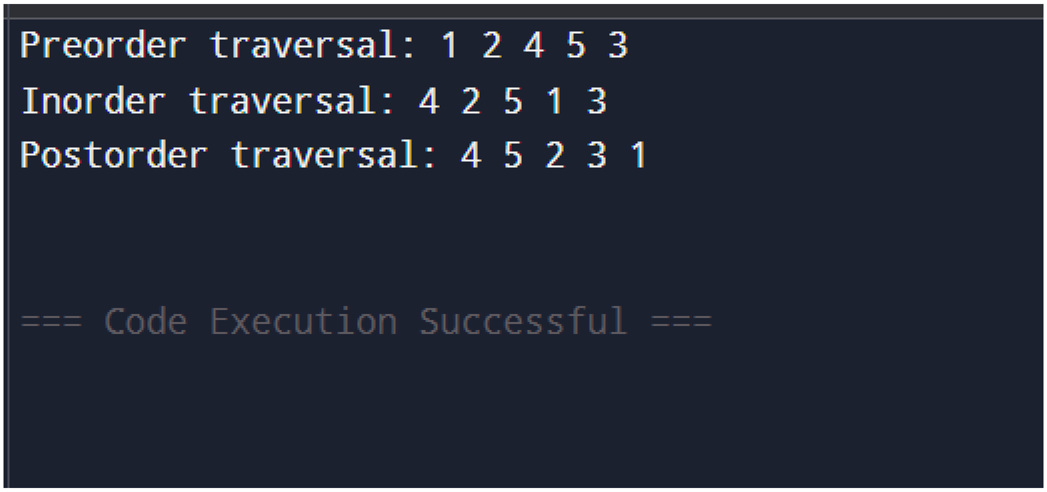


```
Original array: 64 25 12 22 11 90 3
Sorted array:   3 11 12 22 25 64 90


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

DSA ASSIGNMENT 8

QUESTION 1

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

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

Node* createNode(int value) {
    Node* newNode=new Node();
    newNode->data=value;
    newNode->left=nullptr;
    newNode->right=nullptr;
    return newNode;
}

void preorder(Node* root) {
    if(root==nullptr)
        return;
    cout<<root->data<<" ";
    preorder(root->left);
    preorder(root->right);
}

void inorder(Node* root) {
    if(root==nullptr)
        return;
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);
}

void postorder(Node* root) {
    if(root==nullptr)
        return;
    postorder(root->left);
    postorder(root->right);
    cout<<root->data<<" ";
}

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

    cout<<"Preorder traversal: ";
    preorder(root);
    cout<<endl;

    cout<<"Inorder traversal: ";
    inorder(root);
    cout<<endl;
    cout<<"Postorder traversal: ";
    postorder(root);
    cout<<endl;

    return 0;
}
```



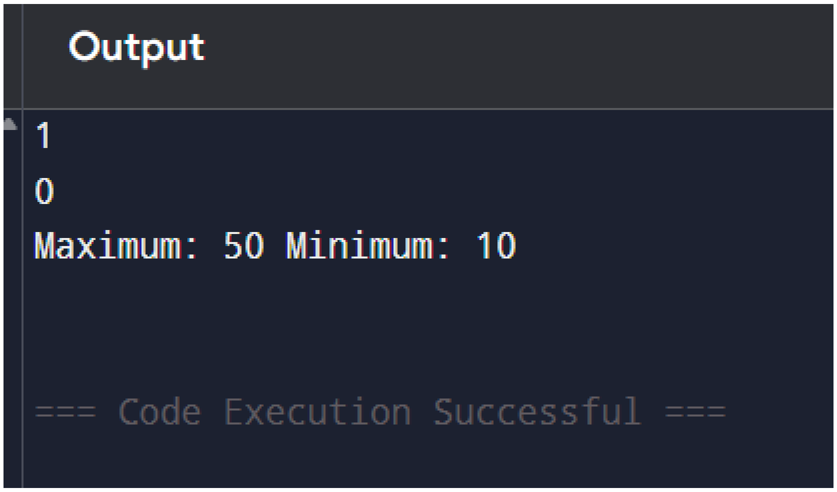
```
Preorder traversal: 1 2 4 5 3
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1


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

QUESTION 2

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

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

    Node(int val) {
        data = val;
        left = right = nullptr;
    }
};

Node* insertNode(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insertNode(root->left, value);
    }
    else if (value > root->data) {
        root->right = insertNode(root->right, value);
    }
    return root;
}
```

```
bool searchRecursive(Node* root, int value) {
    if (root == nullptr) {
        return false;
    }
    if (root->data == value) {
        return true;
    }
    if (value < root->data) {
        return searchRecursive(root->left, value);
    } else {
        return searchRecursive(root->right, value);
    }

    return false; // FIX for warning
}

bool searchNonRecursive(Node* root, int value) {
    Node* curr = root;

    while (curr != nullptr) {
        if (value == curr->data) {
            return true;
        }
        else if (value < curr->data) {
            curr = curr->left;
        }
        else {
            curr = curr->right;
        }
    }
    return false;
}
```

```
int maximumElement(Node* root) {
    if (root == nullptr)
        return -1;

    Node* curr = root;
    while (curr->right != nullptr) {
        curr = curr->right;
    }
    return curr->data;
}

int minimumElement(Node* root) {
    if (root == nullptr)
        return -1;

    Node* curr = root;
    while (curr->left != nullptr) {
        curr = curr->left;
    }
    return curr->data;
}

Node* inorderSuccessor(Node* root, Node* target) {
    Node* succ = nullptr;

    while (root != nullptr) {
        if (target->data < root->data) {
            succ = root;
            root = root->left;
        }
        else if (target->data > root->data) {
            root = root->right;
        }
        else {
            if (root->right != nullptr) {
                Node* temp = root->right;
                while (temp->left != nullptr)
                    temp = temp->left;
                succ = temp;
            }
            break;
        }
    }
    return succ;
}

Node* inorderPredecessor(Node* root, Node* target) {
    Node* pred = nullptr;

    while (root != nullptr) {
        if (target->data > root->data) {
            pred = root;
            root = root->right;
        }
        else if (target->data < root->data) {
            root = root->left;
        }
        else {
            if (root->left != nullptr) {
                Node* temp = root->left;
                while (temp->right != nullptr)
                    temp = temp->right;
                pred = temp;
            }
            break;
        }
    }
    return pred;
}

int main() {
    Node* root = nullptr;
    root = insertNode(root, 10);
    root = insertNode(root, 40);
    root = insertNode(root, 30);
    root = insertNode(root, 50);
    root = insertNode(root, 20);

    bool found;
    found = searchNonRecursive(root, 20);
    cout << found << endl;

    found = searchRecursive(root, 60);
    cout << found << endl;

    int max, min;
    max = maximumElement(root);
    min = minimumElement(root);
    cout << "Maximum: " << max << " Minimum: " << min << endl;

    return 0;
}
```



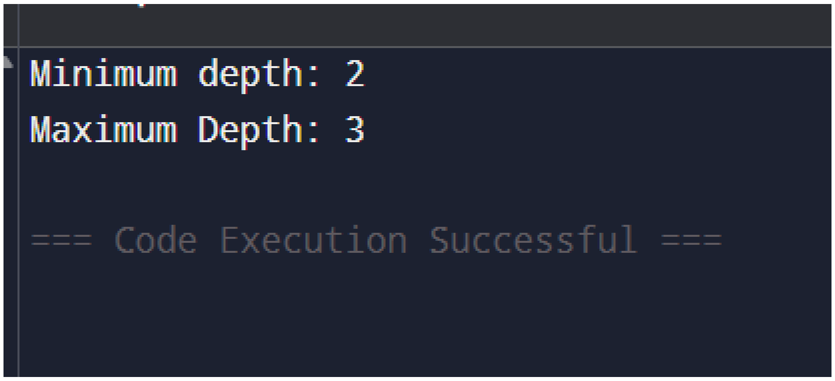
```
1
0
Maximum: 50 Minimum: 10



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

QUESTION 3

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

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

    Node(int val){
        data=val;
        left=right=nullptr;
    }
};

Node* insertElement(Node*root, int value){
    if (root==nullptr){
        return new Node(value);
    }
    if(value==root->data){
        return root;}
    if(value<root->data){
        root->left=insertElement(root->left, value);
    }
    else if(value>root->data){
        root->right=insertElement(root->right,value);
    }
    return root;
}
```

```
Node* findMin(Node* root) {
    while(root->left!=nullptr)
        root=root->left;
    return root;
}

Node* deleteElement(Node*root, int value){
    if(root==nullptr){
        return root;
    }
    if (value<root->data){
        root->left=deleteElement(root->left, value);
    }
    else if(value>root->data){
        root->right=deleteElement(root->right,value);
    }
    else{
        if(root->left==nullptr && root->right==nullptr){
            delete root;
            root=nullptr;
        }
        else if(root->left==nullptr){
            Node*temp=root;
            root=root->right;
            delete temp;
        }
        else if(root->right==nullptr){
            Node*temp=root;
            root=root->left;
            delete temp;
        }
        else{
            Node*temp=findMin(root->right);
            root->data=temp->data;
            root->right=deleteElement(root->right,temp->data);
        }

    }
    return root;
}

int maxDepth(Node*root){
    if (root==nullptr){
        return 0;
    }
    int leftDepth=maxDepth(root->left);
    int rightDepth=maxDepth(root->right);

    return 1+max(leftDepth,rightDepth);
}

int minDepth(Node*root){
    if (root==nullptr){
        return 0;
    }
    if (root->left==nullptr){
        return 1+minDepth(root->left);
    }
    if(root->right==nullptr){
        return 1+minDepth(root->right);
    }

    return 1+min(minDepth(root->left), minDepth(root->right));
```

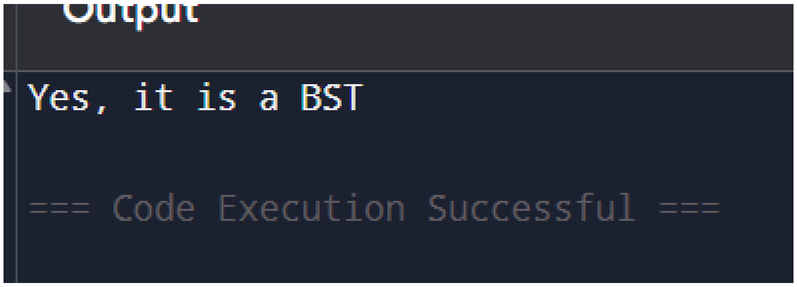
```
}

int main(){
    Node*root=nullptr;
    root=insertElement(root,10);
    root=insertElement(root,40);
    root=insertElement(root,30);
    root=insertElement(root,50);
    root=insertElement(root,20);

    root=deleteElement(root,10);
    int maxmDepth=maxDepth(root);
    int minmDepth=minDepth(root);

    cout<<"Minimum depth: "<<minmDepth<<"\nMaximum Depth: "<<maxmDepth;

    return 0;
}
```



```
Minimum depth: 2
Maximum Depth: 3


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

QUESTION 4

```
#include <iostream>
#include <limits.h>
using namespace std;

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

  Node(int value){
    data=value;
    left=right=nullptr;
  }
};

bool isBSTUtil(Node* root, int minVal, int maxVal){
  if (root == nullptr){
    return true;
  }

  if (root->data <= minVal || root->data >= maxVal){
    return false;
  }

  return isBSTUtil(root->left, minVal, root->data) && isBSTUtil(root->right, root->data, maxVal);
}

bool isBST(Node* root){
  return isBSTUtil(root, INT_MIN, INT_MAX);
}

int main(){
  Node* root=new Node(10);
  root->left=new Node(5);
  root->right=new Node(20);
  root->left->left=new Node(3);
  root->left->right=new Node(7);

  if(isBST(root))
    cout << "Yes, it is a BST";
  else
    cout << "No, it is NOT a BST";

  return 0;
}
```

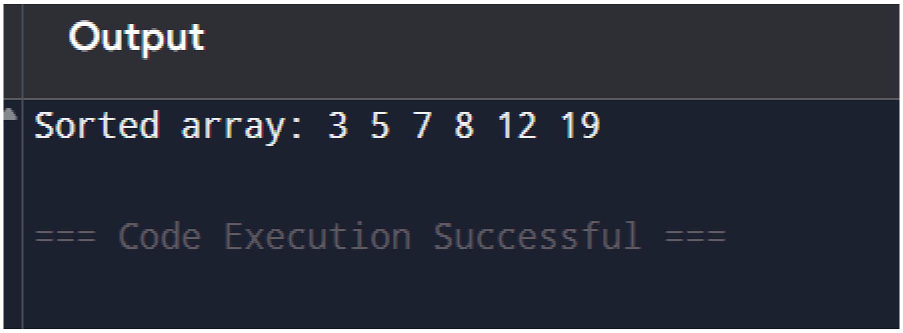


```
Yes, it is a BST

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

QUESTION 5

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

void heapifyMax(int arr[], int n, int i) {
  int largest=i;
  int left=2*i+1;
  int right=2*i+2;

  if(left < n && arr[left] > arr[largest])
    largest = left;

  if(right < n && arr[right] > arr[largest])
    largest = right;

  if(largest != i){
    swap(arr[i], arr[largest]);
    heapifyMax(arr, n, largest);
  }
}

void heapSortIncreasing(int arr[], int n){
  for (int i=n/2-1;i>=0;i--)
    heapifyMax(arr, n, i);

  for (int i=n-1;i>0;i--) {
    swap(arr[0], arr[i]);
    heapifyMax(arr, i, 0);
  }
}
```

```
void heapifyMin(int arr[], int n, int i) {
    int smallest=i;
    int left=2*i+1;
    int right=2*i+2;
    if (left<n && arr[left]<arr[smallest])
        smallest = left;

    if (right< n && arr[right]<arr[smallest])
        smallest=right;

    if (smallest!=i){
        swap(arr[i], arr[smallest]);
        heapifyMin(arr, n, smallest);
    }
}

void heapSortDecreasing(int arr[], int n){
    for (int i=n/2-1;i>=0;i--)
        heapifyMin(arr, n, i);

    for (int i=n-1;i>0;i--){
        swap(arr[0], arr[i]);
        heapifyMin(arr, i, 0);
    }
}
int main(){
    int arr[]={12, 3, 19, 8, 5, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSortIncreasing(arr, n);

    cout << "Sorted array: ";
    for (int x : arr) cout << x << " ";
}
```

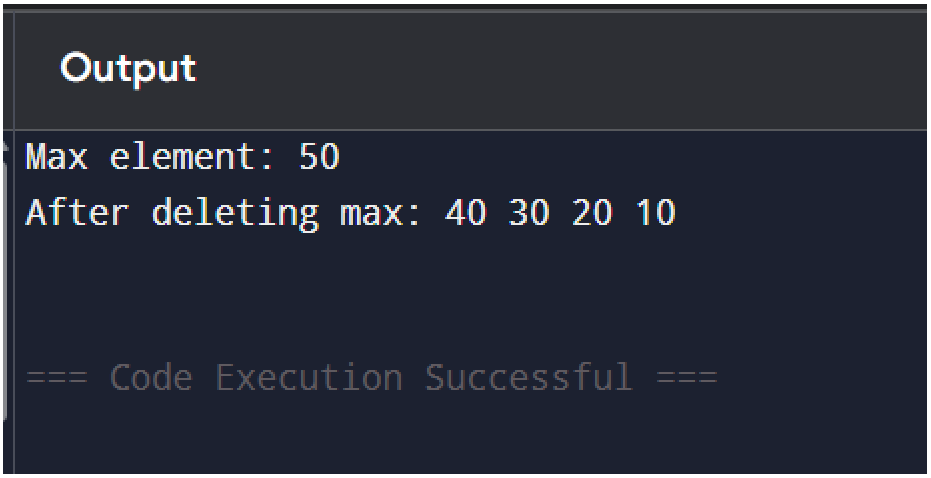


```
Sorted array: 3 5 7 8 12 19


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

QUESTION 6

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

#define MAX 100

int heapArr[MAX];
int heapSize = 0;

void heapifyUp(int index){
    while (index > 0){
        int parent = (index - 1) / 2;
        if (heapArr[parent] < heapArr[index]) {
            swap(heapArr[parent], heapArr[index]);
            index = parent;
        } else break;
    }
}
void heapifyDown(int index){
    while (true){
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int largest = index;

        if (left < heapSize && heapArr[left] > heapArr[largest])
            largest = left;

        if (right < heapSize && heapArr[right] > heapArr[largest])
            largest = right;

        if (largest != index) {
            swap(heapArr[index], heapArr[largest]);
            index = largest;
        } else break;
    }
}

void insertElement(int value){
    if (heapSize >= MAX) {
        cout << "Heap overflow!\n";
        return;
    }
    heapArr[heapSize] = value;
    heapifyUp(heapSize);
    heapSize++;
}
int getMax(){
    if (heapSize == 0){
        cout << "Heap empty!\n";
        return -1;
    }
    return heapArr[0];
}

void deleteMax(){
    if (heapSize == 0){
        cout << "Heap empty!\n";
        return;
    }

    heapArr[0] = heapArr[heapSize - 1];
    heapSize--;

    heapifyDown(0);
```

```
}


void printHeap(){
    for (int i = 0; i < heapSize; i++)
        cout << heapArr[i]<<" ";
    cout << endl;
}
int main(){
    insertElement(40);
    insertElement(10);
    insertElement(50);
    insertElement(30);
    insertElement(20);

    cout<<"Max element: "<<getMax()<<endl;

    deleteMax();
    cout << "After deleting max: ";
    printHeap();

    return 0;
}
```



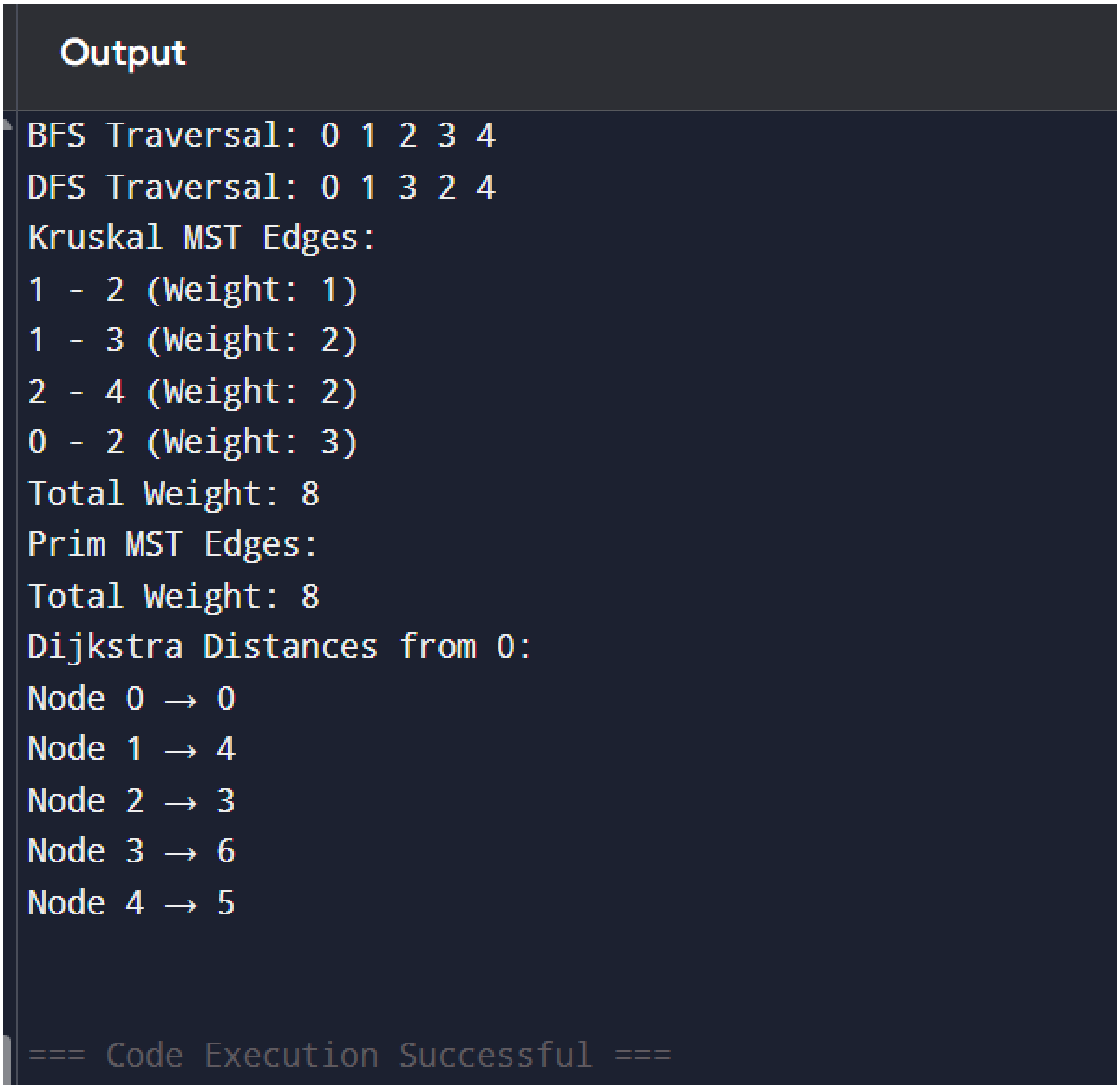
```
Max element: 50
After deleting max: 40 30 20 10


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

DSA ASSIGNMENT 9

QUESTION 1

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
#include <climits>
using namespace std;

void BFS(int start, vector<vector<int>>& adj, int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    cout << "BFS Traversal: ";

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int next : adj[node]) {
            if (!visited[next]) {
                visited[next] = true;
                q.push(next);
            }
        }
    }
    cout << endl;
}

void DFSUtil(int node, vector<vector<int>>& adj, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int next : adj[node]) {
        if (!visited[next])
            DFSUtil(next, adj, visited);
    }
}

void DFS(int start, vector<vector<int>>& adj, int n) {
    vector<bool> visited(n, false);
    cout << "DFS Traversal: ";
    DFSUtil(start, adj, visited);
    cout << endl;
}

int findParent(int node, vector<int>& parent) {
    if (node == parent[node]) return node;
    return parent[node] = findParent(parent[node], parent);
}

void unionSet(int u, int v, vector<int>& parent, vector<int>& rank) {
    u = findParent(u, parent);
    v = findParent(v, parent);

    if (u != v) {
        if (rank[u] < rank[v]) parent[u] = v;
        else if (rank[v] < rank[u]) parent[v] = u;
        else {
            parent[v] = u;
            rank[u]++;
        }
    }
}

void Kruskal(int n, vector<vector<int>>& edges) {
    sort(edges.begin(), edges.end(), [](auto& a, auto& b){
        return a[2] < b[2];
    });

    vector<int> parent(n), rank(n, 0);
    for (int i = 0; i < n; i++) parent[i] = i;

    cout << "Kruskal MST Edges:\n";
    int mstCost = 0;

    for (auto edge : edges) {
        int u = edge[0], v = edge[1], w = edge[2];

        if (findParent(u, parent) != findParent(v, parent)) {
            cout << u << " - " << v << " (Weight: " << w << ")\n";
            mstCost += w;
            unionSet(u, v, parent, rank);
        }
    }
    cout << "Total Weight: " << mstCost << "\n";
}

void Prim(int n, vector<vector<pair<int,int>>>& adj) {
    vector<int> key(n, INT_MAX);
    vector<bool> inMST(n, false);
    key[0] = 0;

    cout << "Prim MST Edges:\n";
    int mstCost = 0;

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

        for (int j = 0; j < n; j++)
            if (!inMST[j] && (u == -1 || key[j] < key[u]))
                u = j;

        inMST[u] = true;
        mstCost += key[u];

        for (auto x : adj[u]) {
            int v = x.first;
            int w = x.second;

            if (!inMST[v] && w < key[v])
                key[v] = w;
        }
    }

    cout << "Total Weight: " << mstCost << "\n";
}

void Dijkstra(int start, int n, vector<vector<pair<int,int>>>& adj) {
    vector<int> dist(n, INT_MAX);
    dist[start] = 0;
```

```
        },



    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int d = pq.top().first;
        int node = pq.top().second;
        pq.pop();

        if (d > dist[node]) continue;

        for (auto edge : adj[node]) {
            int next = edge.first;
            int w = edge.second;

            if (dist[node] + w < dist[next]) {
                dist[next] = dist[node] + w;
                pq.push({dist[next], next});
            }
        }
    }

    cout << "Dijkstra Distances from " << start << ":\n";
    for (int i = 0; i < n; i++)
        cout << "Node " << i << " → " << dist[i] << endl;
}
int main() {
    int n = 5;

    vector<vector<int>> adjList = {
        {1, 2},
        {0, 3},
        {0, 3, 4},
        {1, 2},
        {2}
    };

    BFS(0, adjList, n);
    DFS(0, adjList, n);

    vector<vector<int>> edges = {
        {0,1,4}, {0,2,3}, {1,2,1},
        {1,3,2}, {2,3,4}, {2,4,2}, {3,4,3}
    };
    Kruskal(5, edges);

    vector<vector<pair<int,int>>> adjWeighted = {
        {{1,4},{2,3}},
        {{0,4},{2,1},{3,2}},
        {{0,3},{1,1},{3,4},{4,2}},
        {{1,2},{2,4},{4,3}},
        {{2,2},{3,3}}
    };
    Prim(5, adjWeighted);

    Dijkstra(0, 5, adjWeighted);

    return 0;
}
```



```
BFS Traversal: 0 1 2 3 4
DFS Traversal: 0 1 3 2 4
Kruskal MST Edges:
1 - 2 (Weight: 1)
1 - 3 (Weight: 2)
2 - 4 (Weight: 2)
0 - 2 (Weight: 3)
Total Weight: 8
Prim MST Edges:
Total Weight: 8
Dijkstra Distances from 0:
Node 0 → 0
Node 1 → 4
Node 2 → 3
Node 3 → 6
Node 4 → 5



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