Design and Analysis of Algorithms Assignment - 1

Name: Dhanraj Kore

Div: TY B

Roll No: 60

Batch : B-3

Section 1: Numbers

.

Q.1 Print all prime numbers up to n.
Ans.

Brute Force Approach:

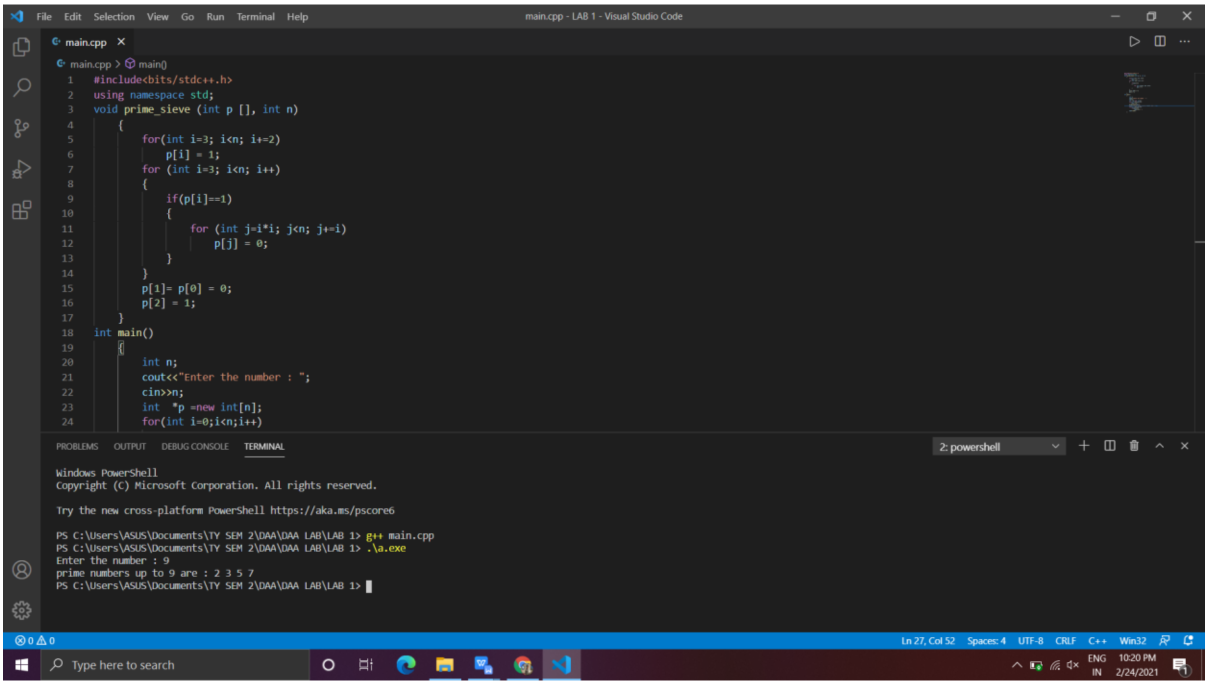
```
#include<bits/stdc++.h>
using namespace std;
int isPrime (int n)
{
	If(n==2)
		return true;
	for (int i=2;i<sqrt(n);i++)
	{
		if(n%i == 0)
			return false;
	}
	return true;
}
int main ()
{
	for( int i=2; i<=n; i++)
	{
		if(isPrime(i))
			cout<<i<<" ";
	}
}
```

Time Complexity: O(n^3/2)
Space Complexity: O(1)

Optimal Approach: Sieve of Eratosthenes

```
#include <bits/stdc++.h>
using namespace std;
void prime_sieve(int p[], int n)
{
    for (int i = 3; i < n; i += 2)
        p[i] = 1;
    for (int i = 3; i < n; i++)
    {
        if (p[i] == 1)
        {
            for (int j = i * i; j < n; j += i)
                p[j] = 0;
        }
    }
    p[1] = p[0] = 0;
    p[2] = 1;
}
int main()
{
    int n;
    cout << "Enter the number : ";
    cin >> n;
    int *p = new int[n];
    for (int i = 0; i < n; i++)
        p[i] = 0;
    prime_sieve(p, n);
    cout << "prime numbers up to " << n << " are : ";
    for (int i = 0; i < n; i++)
        if (p[i] == 1)
            cout << i << " ";
    cout << endl;
}
```

Time Complexity:: O(nlog(logn))
Space Complexity: O(n)



Q.2 Number of zeros at the end n!

Ans.

Brute Force Approach:

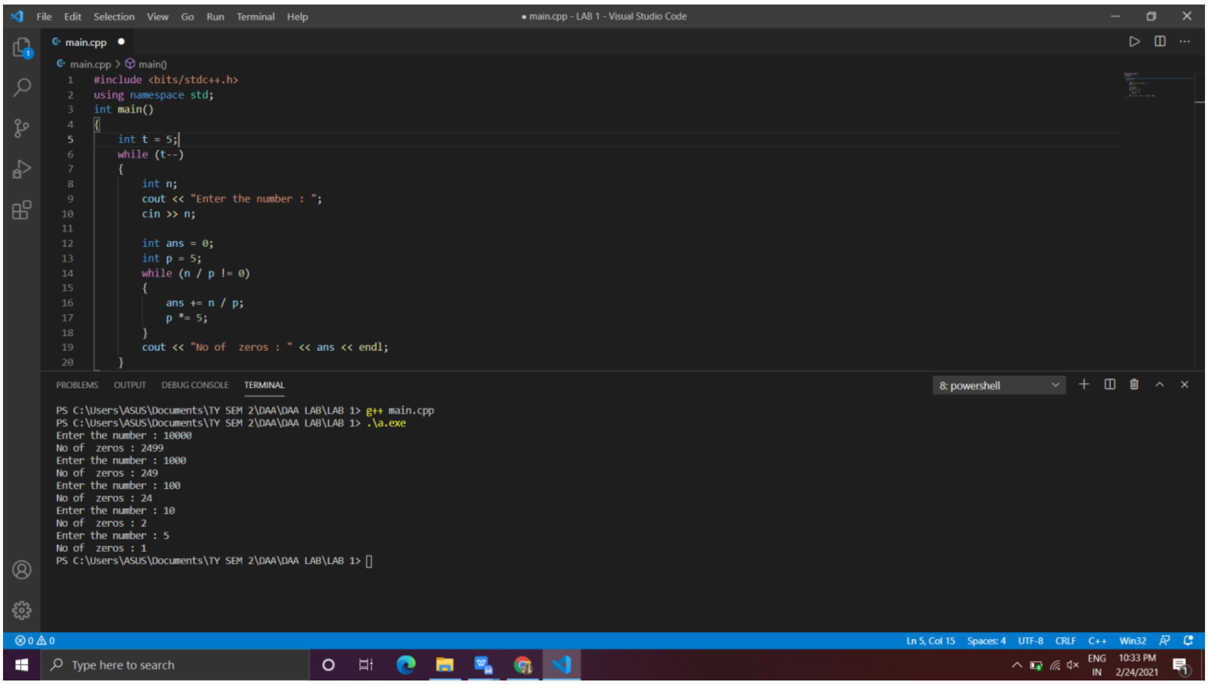
```
#include<bits/stdc++.h>
using namespace std;
int main()
{
        int n;
        cin>>n;
        fact = 1;
        for(int i=n ; i>=1;i--)
                fact *=i;
        int cnt=0;
        int t=0;
        while(t==0)
        {
                If(fact % 10 == 0)
                        cnt++;
                fact /=10;
        }
        cout<<cnt;
}
```

This method is not feasible because when we take a large number say 100000 as an input it will be very difficult to count number of trailing zeroes in its factorial.

Optimal Approach:

This solution works perfectly because number of zeroes trailing in factorial of any number is equal to number of occurrences of 5 in factorial of that number.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int t = 5;
    while (t--)
    {
        int n;
        cout << "Enter the number : ";
        cin >> n;

        int ans = 0;
        int p = 5;
        while (n / p != 0)
        {
            ans += n / p;
            p *= 5;
        }
        cout << "No of  zeros : " << ans << endl;
    }
}
```



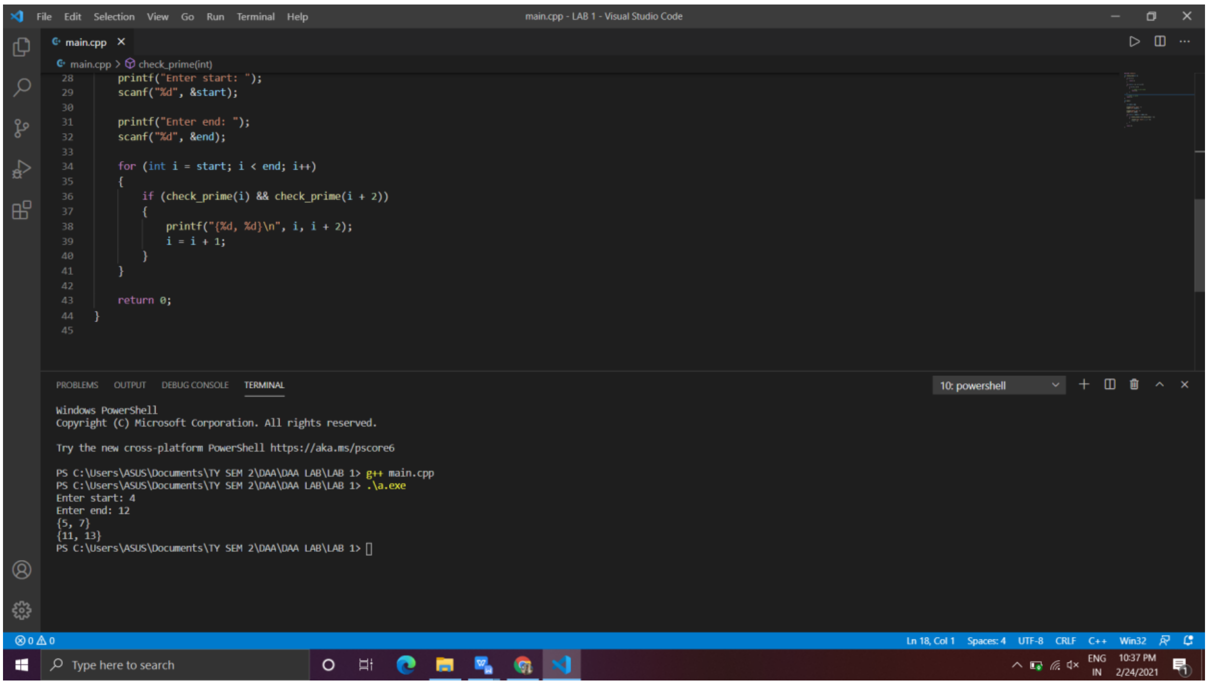
Q.3 Printing twin prime numbers in the given range
Ans.

A Twin prime are those numbers which are prime and having a difference of two (2) between the two prime numbers.

Approach : Using **Sieve of Eratosthenes**

```c
#include <stdio.h>

int check_prime(int n)
{
    if (n == 1)
    {
        return 0;
    }
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
        {
            // number is not prime
            return 0;
        }
    }
    // number is prime
    return 1;
}
int main()
{
    int start, end;
    printf("Enter start: ");
    scanf("%d", &start);
    printf("Enter end: ");
    scanf("%d", &end);
    for (int i = start; i < end; i++)
    {
        if (check_prime(i) && check_prime(i + 2))
        {
            printf("{%d, %d}\n", i, i + 2);
            i = i + 1;
        }
    }
    return 0;
}
```

Time Complexity : O(n*log(log(n)))
Space Complexity : O(1)



Section 2: Graphs

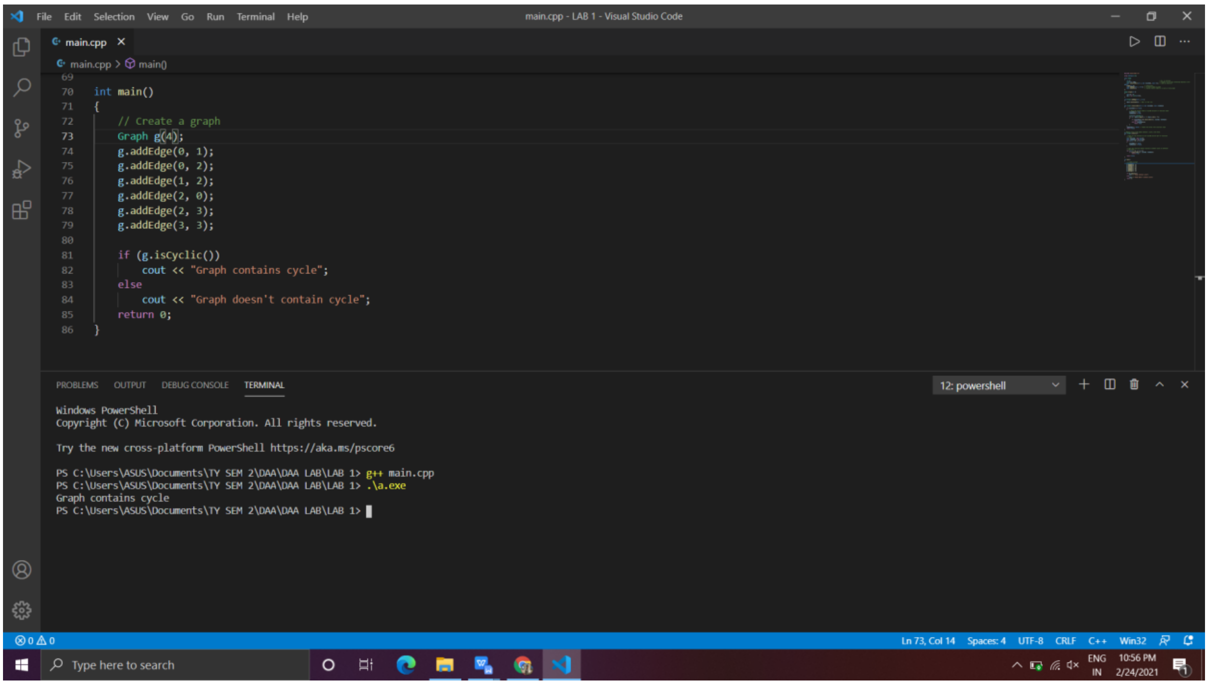
Q.1 Test if graph has a cycle?

Ans.

For every visited vertex ‘v’, if there is an adjacent ‘u’ such that u is already visited and u is not parent of v, then there is a cycle in graph.

```
#include<bits/stdc++.h>

using namespace std;

class Graph
{
    int V;          // No. of vertices
    list<int> *adj;     // Pointer to an array containing adjacency lists
    bool isCyclicUtil(int v, bool visited[], bool *rs);  // used by isCyclic()
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // to add an edge to graph
    bool isCyclic();    // returns true if there is a cycle in this graph
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}
```

```cpp
bool Graph::isCyclicUtil(int v, bool visited[], bool *recStack)
{
    if(visited[v] == false)
    {
        // Mark the current node as visited and part of recursion stack
        visited[v] = true;
        recStack[v] = true;

        list<int>::iterator i;
        for(i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            if ( !visited[*i] && isCyclicUtil(*i, visited, recStack) )
                return true;
            else if (recStack[*i])
                return true;
        }

    }
    recStack[v] = false;  // remove the vertex from recursion stack
    return false;
}

// Returns true if the graph contains a cycle, else false.
bool Graph::isCyclic()
{
    // Mark all the vertices as not visited and not part of recursion
    // stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for(int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle in different
    // DFS trees
    for(int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}

int main()
{
    // Create a graph
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    if(g.isCyclic())
        cout << "Graph contains cycle";
    else
        cout << "Graph doesn't contain cycle";
    return 0;
}
```



Time Complexity: O(V+E).
Space Complexity: O(V).

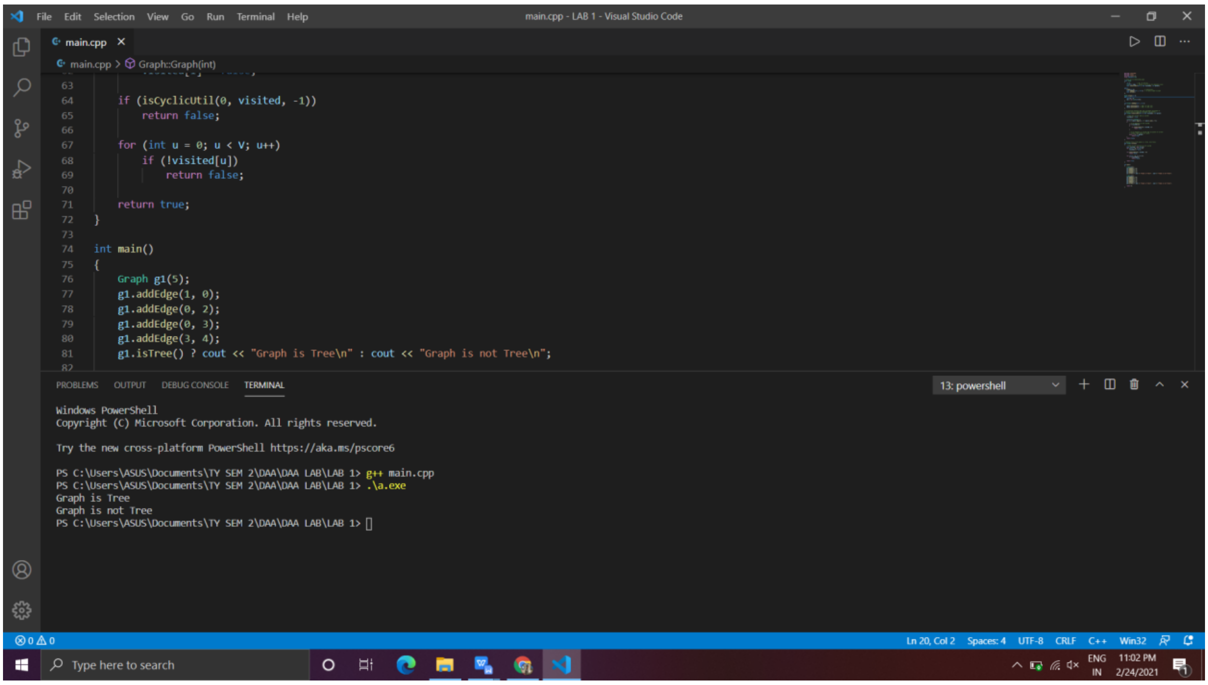
Q.2 Test if graph is a tree?

Ans.

An undirected graph is a tree if it has no cycle and the graph is connected.

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

// Class for an undirected graph
class Graph
{
    int V;          // No. of vertices
    list<int> *adj; // Pointer to an array for adjacency lists
    bool isCyclicUtil(int v, bool visited[], int parent);
public:
    Graph(int V);               // Constructor
    void addEdge(int v, int w); // to add an edge to graph
    bool isTree();
};
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
```

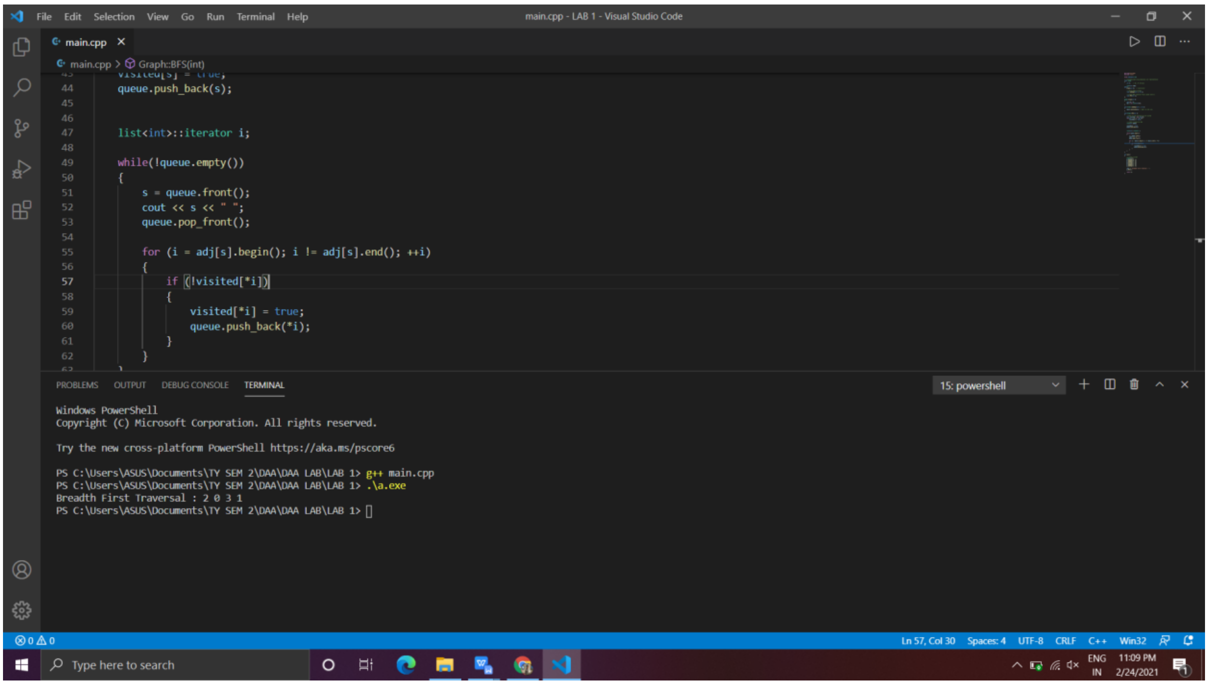
```
    adj[v].push_back(w); // Add w to v’s list.
    adj[w].push_back(v); // Add v to w’s list.
}


bool Graph::isCyclicUtil(int v, bool visited[], int parent)
{
    // Mark the current node as visited
    visited[v] = true;
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
    {
        // If an adjacent is not visited,
        if (!visited[*i])
        {
            if (isCyclicUtil(*i, visited, v))
                return true;
        }
        // If an adjacent is visited and not parent of current
        // vertex, then there is a cycle.
        else if (*i != parent)
            return true;
    }
    return false;
}
// Returns true if the graph is a tree, else false.
bool Graph::isTree()
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    if (isCyclicUtil(0, visited, -1))
        return false;
    for (int u = 0; u < V; u++)
        if (!visited[u])
            return false;
    return true;
}

int main()
{
    Graph g1(5);
    g1.addEdge(1, 0);
    g1.addEdge(0, 2);
    g1.addEdge(0, 3);
    g1.addEdge(3, 4);
    g1.isTree() ? cout << "Graph is Tree\n" : cout << "Graph is not Tree\n";

    Graph g2(5);
    g2.addEdge(1, 0);
    g2.addEdge(0, 2);
    g2.addEdge(2, 1);
    g2.addEdge(0, 3);
    g2.addEdge(3, 4);
    g2.isTree() ? cout << "Graph is Tree\n" : cout << "Graph is not Tree\n";

    return 0;
}
```



Q.3 BFS.
Ans.

```
#include<iostream>
#include <list>

using namespace std;

//  directed graph using adjacency list representation
class Graph
{
    int V;    // No. of vertices

    list<int> *adj;
public:
    Graph(int V);  // Constructor

    // add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}
```

```cpp
void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Create a queue for BFS
    list<int> queue;

    visited[s] = true;
    queue.push_back(s);

    list<int>::iterator i;

    while(!queue.empty())
    {
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}

int main()
{
    // Create a graph
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Breadth First Traversal : ";
    g.BFS(2);

    return 0;
}
```



Section 3: Array

Q.1 Rotating array.

Ans.

Brute Force Approach:

```
void rotate (int A [])
{
        for (int i=0; i<d; i++)
        {
                Int temp = A [0];
                for (int j=0; j<n; j++)
                {
                        A[j] = A[j+1];
                }
                A[n-1] = temp;
        }
}
```

Time Complexity: O (n * d)

Space Complexity: O (1)

Optimal Approach:

In this approach, first, array from index 0 to d-1 is reversed. Then array from d to n-1 is reversed and at last array from 0 to n-1 is reversed which will give rotated array by d positions.

```
void Reverse (int A [], int a, int b)
{
        int i=a, j=b;
        while(i<j)
```

```
        {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
                i++; j--;
        }
}
void rotate (int A [])
{
        Reverse (A, 0, d-1);
        Reverse (A, d, n-1);
        Reverse (A,0, n-1);
}
```
Time Complexity: O (n)
Space Complexity: O (1)

Q.2 find min, max.
Ans.

Brute Force Approach:
```
void find_minmax (int A [], int n)
{
        int max = A [0];
        int min = A [0];
        for (int i=1; i<n; i++)
        {
                if (max < A[i])
                        max = A[i];
                else if (min > A[i])
                        min = A[i];
        }
        cout<<min<<" "<<max;
}
```
Time Complexity: O(n)

Optimal approach:
```
struct Pair
{
        int min;
        int max;
};

struct Pair getMinMax (int arr [], int low, int high)
{
        struct Pair minmax, mml, mmr;
        int mid;
        if (low == high)
        {
```

```
            minmax.max = arr[low];
            minmax.min = arr[low];
           return minmax;
        }
        if (high == low + 1)
        {
            if (arr[low] > arr[high])
          {
             minmax.max = arr[low];
             minmax.min = arr[high];
          }
          else
          {
             minmax.max = arr[high];
             minmax.min = arr[low];
          }
          return minmax;
        }
        mid = (low + high) / 2;
        mml = getMinMax (arr, low, mid);
        mmr = getMinMax (arr, mid + 1, high);

        if (mml.min < mmr.min)
          minmax.min = mml.min;
        else
          minmax.min = mmr.min;

        if (mml.max > mmr.max)
          minmax.max = mml.max;
        else
          minmax.max = mmr.max;

        return minmax;
}
```

Time Complexity: O(n)

Total Number of comparisons: 2T(n/2) + 2

Q.3 Find missing element.

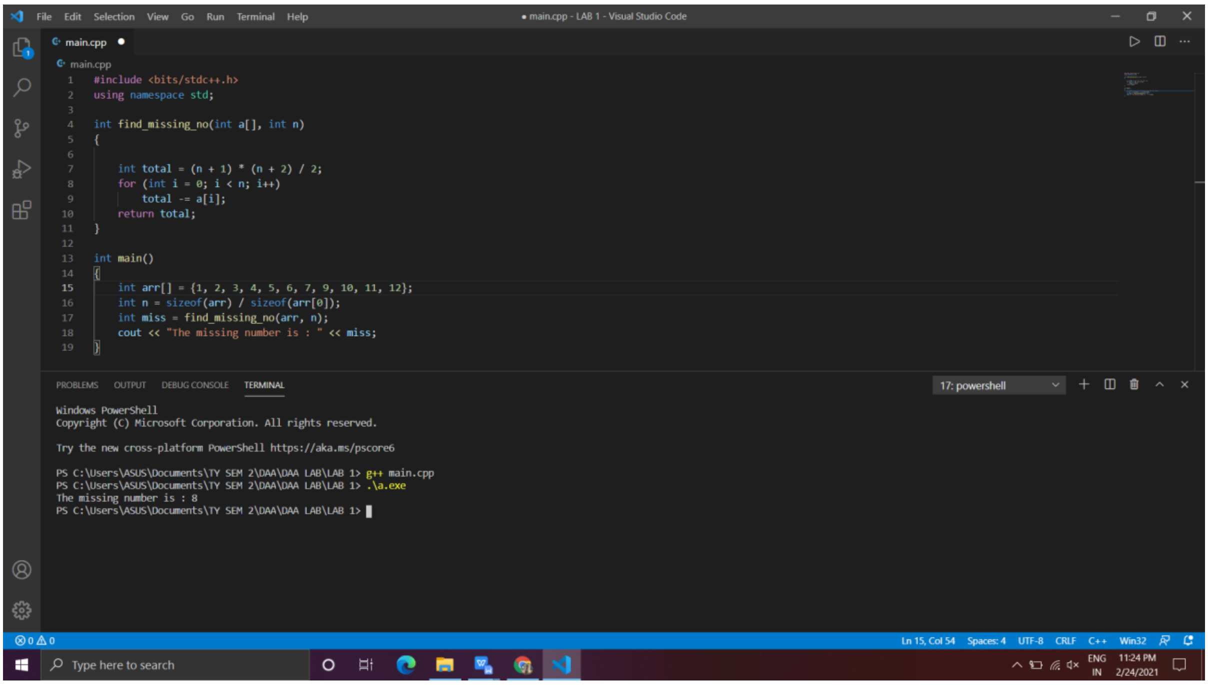
Ans.

Using summation formula - n*(n+1)/2. - total_sum

```
int find_missing_no(int a[], int n)
{

   int total = (n + 1) * (n + 2) / 2;
   for (int i = 0; i < n; i++)
      total -= a[i];
```

```
        return total;
    }
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int find_missing_no(int a[], int n)
{
    int total = (n + 1) * (n + 2) / 2;
    for (int i = 0; i < n; i++)
        total -= a[i];
    return total;
}
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int miss = find_missing_no(arr, n);
    cout << "The missing number is : " << miss;
}
```



Time Complexity: O(n)
Space Complexity: O(1)

Section 4: Matrix

Q.1 Print entries in spiral fashion.
Ans.

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
vector <int> v;
If (matrix. size () == 0)
```

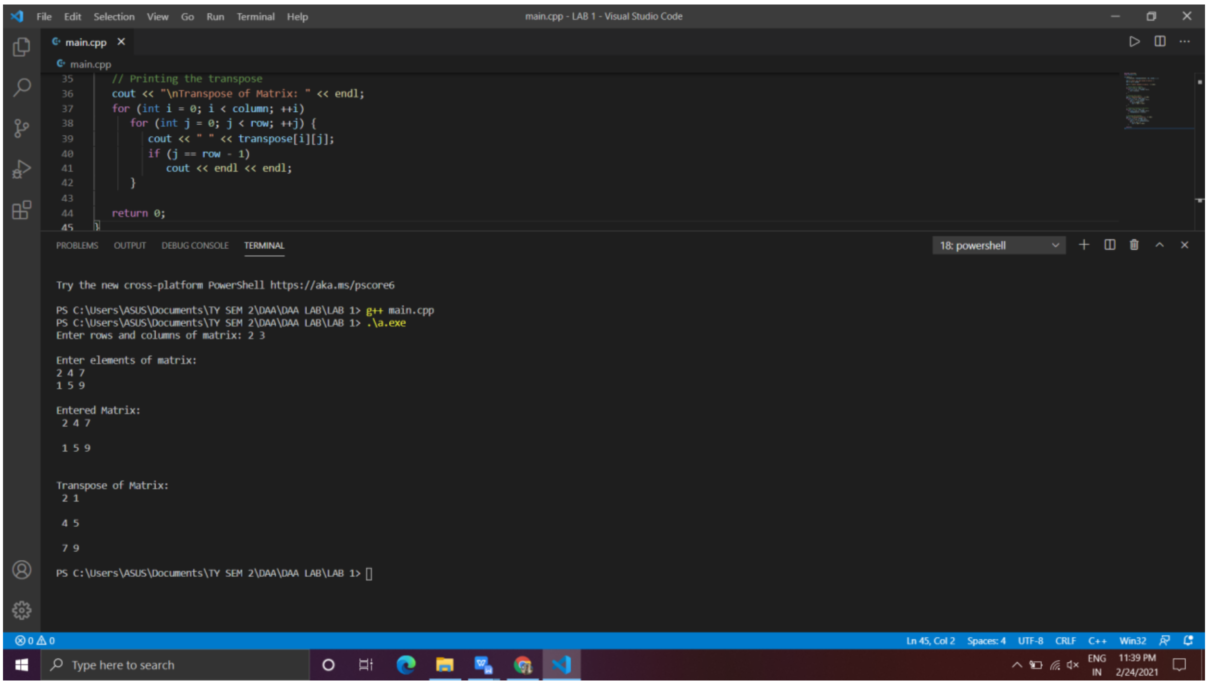
```
        return v;
    int top = 0;
    int bottom = matrix. size ()-1;
    int left = 0;
    int right = matrix [0]. size ()-1;
    int size = matrix. size () * matrix [0]. size ();
    while (v. size () < size)
    {
        for (int i=left; i<=right && v. size () < size; i++)
            v.push_back(matrix[top][i]);
        top++;
        for (int i=top; i<=bottom && v. size () < size; i++)
            v.push_back(matrix[i][right]);
        right--;
        for (int i=right; i>=left && v. size () < size; i--)
            v.push_back(matrix[bottom][i]);
        bottom--;
        for (int i=bottom; i>=top && v. size () < size; i--)
            v.push_back(matrix[i][left]);
        left++;
    }
    return v;
}
```

Time Complexity: O(n^2)
Space Complexity: O(n)

Q.2 Transpose.
Ans.

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

int main() {
   int a[10][10], transpose[10][10], row, column, i, j;
   cout << "Enter rows and columns of matrix: ";
   cin >> row >> column;
   cout << "\nEnter elements of matrix: " << endl;

   for (int i = 0; i < row; ++i) {
      for (int j = 0; j < column; ++j) {
         cin >> a[i][j];
      }
   }
   // Printing the a matrix
   cout << "\nEntered Matrix: " << endl;
   for (int i = 0; i < row; ++i) {
      for (int j = 0; j < column; ++j) {
         cout << " " << a[i][j];
         if (j == column - 1)
            cout << endl << endl;
      }
```

```
    }



    for (int i = 0; i < row; ++i)
       for (int j = 0; j < column; ++j) {
          transpose[j][i] = a[i][j];
       }
    // Printing the transpose
    cout << "\nTranspose of Matrix: " << endl;
    for (int i = 0; i < column; ++i)
       for (int j = 0; j < row; ++j) {
          cout << " " << transpose[i][j];
          if (j == row - 1)
             cout << endl << endl;
       }
    return 0;
}
```

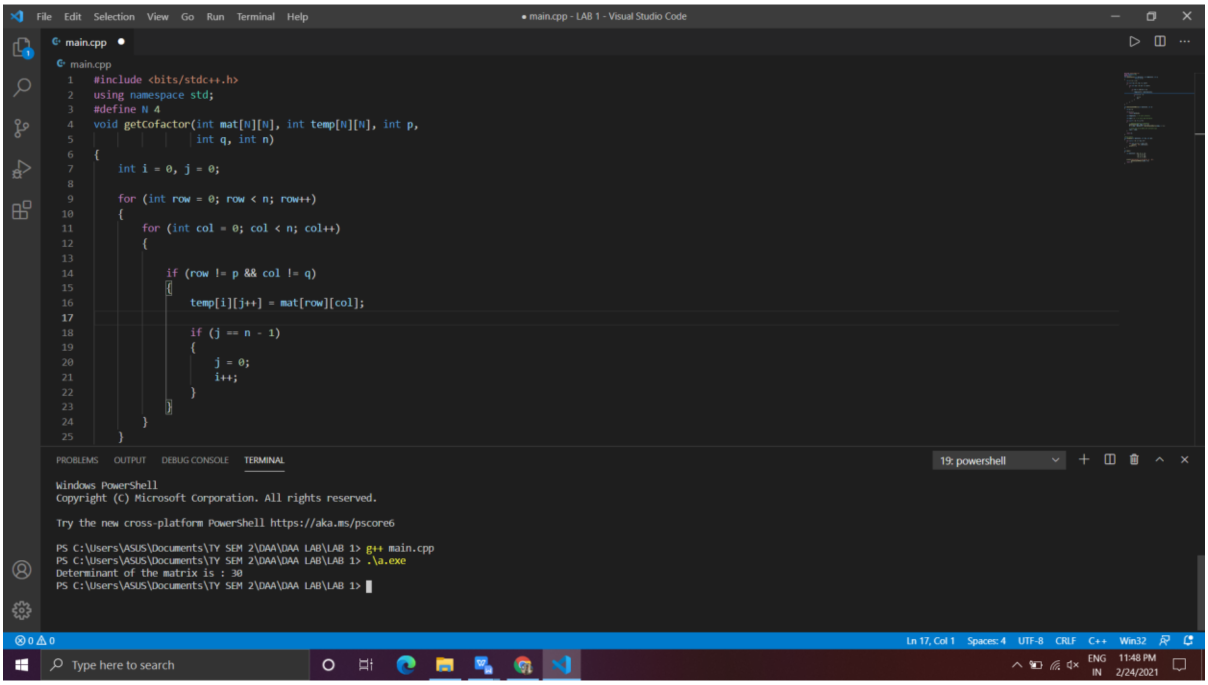


Time Complexity : O(m*n)
Space Complexity : O(m*n)

Q.3 Determinant.
Ans

```
        #include <bits/stdc++.h>
using namespace std;
#define N 4
void getCofactor(int mat[N][N], int temp[N][N], int p,
                 int q, int n)
{
    int i = 0, j = 0;
```

```
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            if (row != p && col != q)
            {
                temp[i][j++] = mat[row][col];
                if (j == n - 1)
                {
                    j = 0;
                    i++;
                }
            }
        }
    }
}
int determinantOfMatrix(int mat[N][N], int n)
{
    int D = 0;
    if (n == 1)
        return mat[0][0];
    int temp[N][N]; // To store cofactors
    int sign = 1; // To store sign multiplier
    for (int f = 0; f < n; f++)
    {
        // Getting Cofactor of mat[0][f]
        getCofactor(mat, temp, 0, f, n);
        D += sign * mat[0][f] * determinantOfMatrix(temp, n - 1);
        // terms are to be added with alternate sign
        sign = -sign;
    }
    return D;
}

//Print matrix
void display(int mat[N][N], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            printf("  %d", mat[i][j]);
        printf("n");
    }
}
int main()
{
    int mat[N][N] = {{1, 0, 2, -1},
                     {3, 0, 0, 5},
                     {2, 1, 4, -3},
                     {1, 0, 5, 0}};
    printf("Determinant of the matrix is : %d",
           determinantOfMatrix(mat, N));
    return 0;
}
```



Section 5: Vectors

Q.1 Find the angle between vectors.
Ans.

```
float angle_between_vectors (int A [], int B [])
{
        Float dot_product=0;
        For (int i=0; i<3; i++)
                dot_product += A[i]*B[i];
        mag_A = A [0] * A [1] *A [2];
        mag_B = B [0] * B [1] *B [2];
        return acos (dot_pproduct/ (mag_A * mag_B);
}
```

Q.2 Projection of vector A on vector B.
Ans.

```
void angle_between_vectors (int A [], int B [])
{
        Float dot_product= 0;
        For (int i=0; i<3; i++)
                dot_product += A[i]*B[i];
        mag_B = B [0] *B [1] *B [2];
        cout<< dot_product/(mag_B*mag_B) *B [0] <<" i+" <<
        dot_product/(mag_B*mag_B) *B [1] <<" j +" <<
        dot_product/(mag_B*mag_B) *B [2] <<" k +";
}
```

Q.3 Dot product of two vectors.

Ans.

```
float dot-product (int A [], int B [])
{
    Float dot_product=0;
    For (int i=0; i<3; i++)
        dot_product += A[i]*B[i];
    return dot_product;
}
```