

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:: $O(n \log(\log n))$

Space Complexity: $O(n)$

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 void prime_sieve (int p [], int n)
4 {
5     for(int i=3; i<n; i+=2)
6         p[i] = 1;
7     for (int i=3; i<n; i++)
8     {
9         if(p[i]==1)
10         {
11             for (int j=i*i; j<n; j+=i)
12                 p[j] = 0;
13         }
14     }
15     p[1]= p[0] = 0;
16     p[2] = 1;
17 }
18 int main()
19 {
20     int n;
21     cout<<"Enter the number : ";
22     cin>>n;
23     int *p =new int[n];
24     for(int i=0;i<n;i++)
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\VASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp
PS C:\Users\VASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> .\a.exe
Enter the number : 9
prime numbers up to 9 are : 2 3 5 7
PS C:\Users\VASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1>
```

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;
}
```

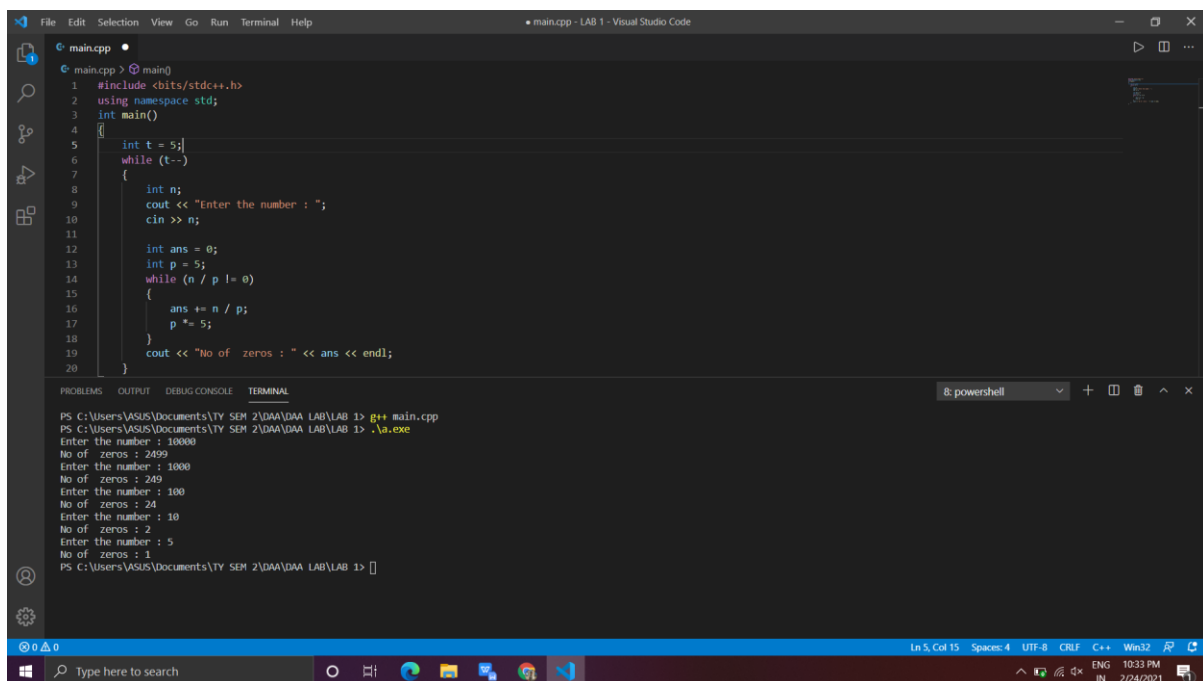
This method is not feasible because when we take a large number say 100000 as an input it will be very difficult to count number of trailing zeroes in its factorial.

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.

```
#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;
    }
}
```



The screenshot shows the Visual Studio Code editor with the C++ code from the previous block. The terminal window at the bottom shows the program's execution. It prompts the user to enter a number, and for each input, it calculates and displays the number of trailing zeros in its factorial. The inputs and outputs shown are:

Input	Output (No of zeros)
10000	2499
1000	249
100	24
10	2
5	1

The terminal also shows the command prompt path: `PS C:\Users\VASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp` and `PS C:\Users\VASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> .\a.exe`.

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**

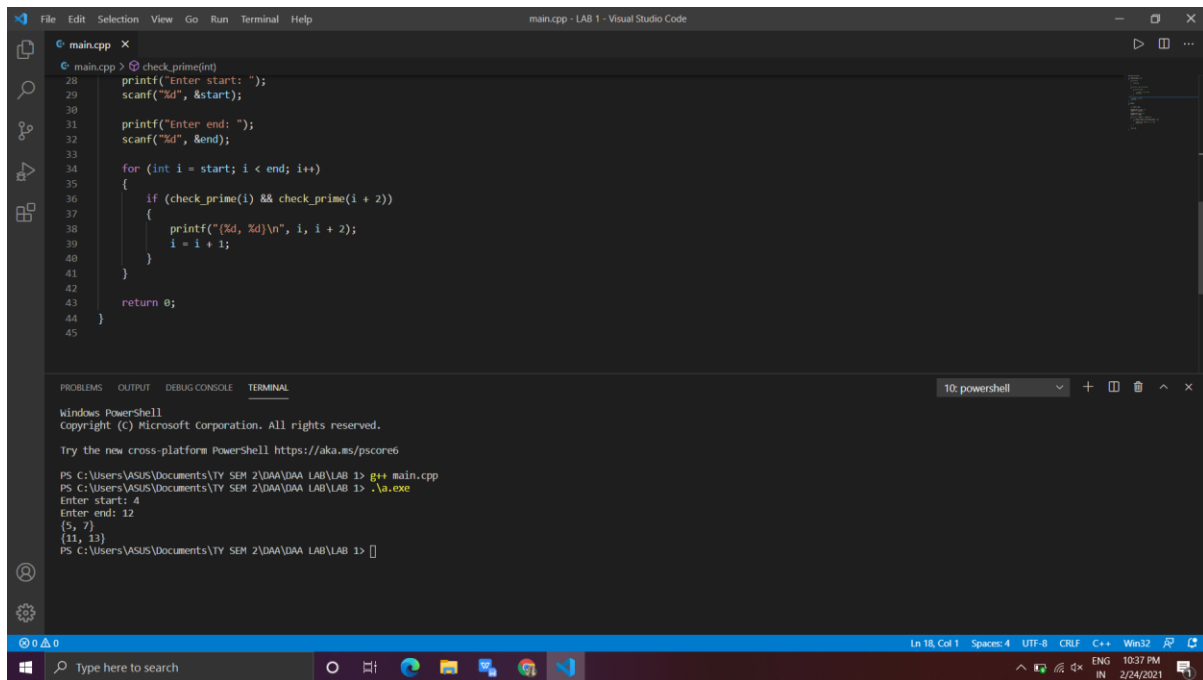
```
#include <stdio.h>

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 : $O(n \cdot \log(\log(n)))$

Space Complexity : $O(1)$



```
main.cpp x
28 check_prime(int)
29 {
30     printf("Enter start: ");
31     scanf("%d", &start);
32
33     printf("Enter end: ");
34     scanf("%d", &end);
35
36     for (int i = start; i < end; i++)
37     {
38         if (check_prime(i) && check_prime(i + 2))
39         {
40             printf("%d, %d\n", i, i + 2);
41             i = i + 1;
42         }
43     }
44     return 0;
45 }
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> ./a.exe
Enter start: 4
Enter end: 12
{5, 7}
{11, 13}
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 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.
}
```

```

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;
    }

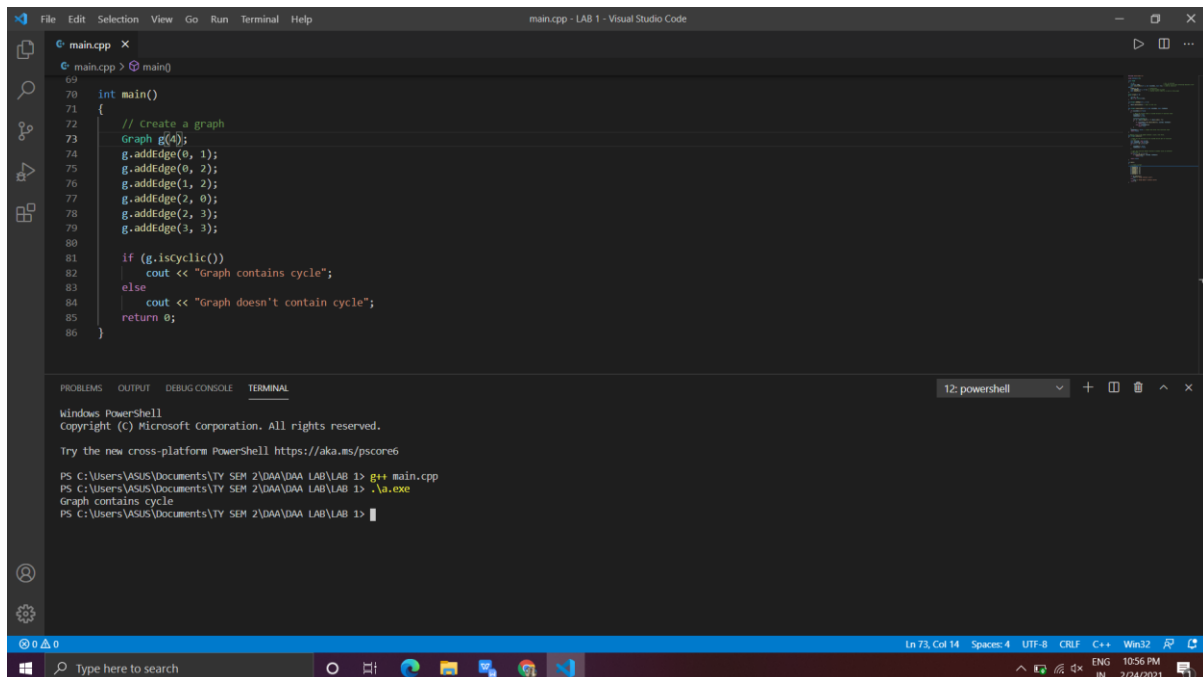
    // Call the recursive helper function to detect cycle in different
    // DFS trees
    for(int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    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;
}

```



```
main.cpp x
main.cpp > main()
99
70 int main()
71 {
72     // Create a graph
73     Graph g(4);
74     g.addEdge(0, 1);
75     g.addEdge(0, 2);
76     g.addEdge(1, 2);
77     g.addEdge(2, 0);
78     g.addEdge(2, 3);
79     g.addEdge(3, 3);
80
81     if (g.isCyclic())
82         cout << "Graph contains cycle";
83     else
84         cout << "Graph doesn't contain cycle";
85     return 0;
86 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

12: powershell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> .\a.exe
Graph contains cycle
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> |

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;
}

```

The screenshot shows the Visual Studio Code editor with a C++ file named `main.cpp`. The code implements a function `isCyclicUtil` to check for cycles in a graph and a `main` function to test it. The graph has 5 vertices and 4 edges: (1,0), (0,2), (0,3), and (3,4). The output of the program is "Graph is Tree". Below the editor, a terminal window shows the command prompt where the program was compiled and executed.

```
main.cpp - LAB 1 - Visual Studio Code
main.cpp x
main.cpp > Graph:Graph(int)
63
64 if (isCyclicUtil(0, visited, -1))
65     return false;
66
67 for (int u = 0; u < V; u++)
68     if (!visited[u])
69         return false;
70
71 return true;
72 }
73
74 int main()
75 {
76     Graph g1(5);
77     g1.addEdge(1, 0);
78     g1.addEdge(0, 2);
79     g1.addEdge(0, 3);
80     g1.addEdge(3, 4);
81     g1.isTree() ? cout << "Graph is Tree\n" : cout << "Graph is not Tree\n";
82 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
13: powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ASUS\Documents\TY SEM 2\DAAD\LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DAAD\LAB\LAB 1> .\a.exe
Graph is Tree
Graph is not Tree
PS C:\Users\ASUS\Documents\TY SEM 2\DAAD\LAB\LAB 1> []
```

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.
}
```

```

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;
}

```

The screenshot shows the Visual Studio Code editor with a C++ file named `main.cpp`. The code implements a Breadth First Traversal (BFS) algorithm. It starts by creating a queue and pushing the source node `s`. Then, it enters a loop that continues until the queue is empty. Inside the loop, it pops the front element `s`, prints it, and then iterates through its adjacent nodes. For each adjacent node `i`, if it has not been visited, it marks it as visited and pushes it into the queue. The terminal window at the bottom shows the command prompt where the program is compiled and executed, resulting in the output: `Breadth First Traversal : 2 0 3 1`.

```
main.cpp
44  queue.push_back(s);
45
46
47  list<int>::iterator i;
48
49  while(!queue.empty())
50  {
51      s = queue.front();
52      cout << s << " ";
53      queue.pop_front();
54
55      for (i = adj[s].begin(); i != adj[s].end(); ++i)
56      {
57          if (!visited[*i])
58          {
59              visited[*i] = true;
60              queue.push_back(*i);
61          }
62      }
63  }
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> ./a.exe
Breadth First Traversal : 2 0 3 1
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> |
```

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;
}

```

```

#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;
}

```

The screenshot shows the Visual Studio Code editor with the same C++ code as above. The terminal window at the bottom shows the command prompt output:

```

PS C:\Users\ASUS\Documents\TY SEM 2\DA\DA LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DA\DA LAB\LAB 1> .\a.exe
The missing number is : 8
PS C:\Users\ASUS\Documents\TY SEM 2\DA\DA LAB\LAB 1>

```

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 << "\nEnter 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;
}

```

The screenshot shows the Visual Studio Code editor with a C++ file named `main.cpp`. The code implements a function to calculate the transpose of a matrix. The terminal window shows the execution of the program, where the user enters the dimensions of the matrix (2 3) and the elements of the matrix (2 4 7 and 1 5 9). The output displays the original matrix and its transpose.

```

// 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;
}

```

Terminal Output:

```

PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> g++ main.cpp
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1> .\a.exe
Enter rows and columns of matrix: 2 3
Enter elements of matrix:
2 4 7
1 5 9
Entered Matrix:
2 4 7
1 5 9
Transpose of Matrix:
2 1
4 5
7 9
PS C:\Users\ASUS\Documents\TY SEM 2\DAADAA LAB\LAB 1>

```

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;
}

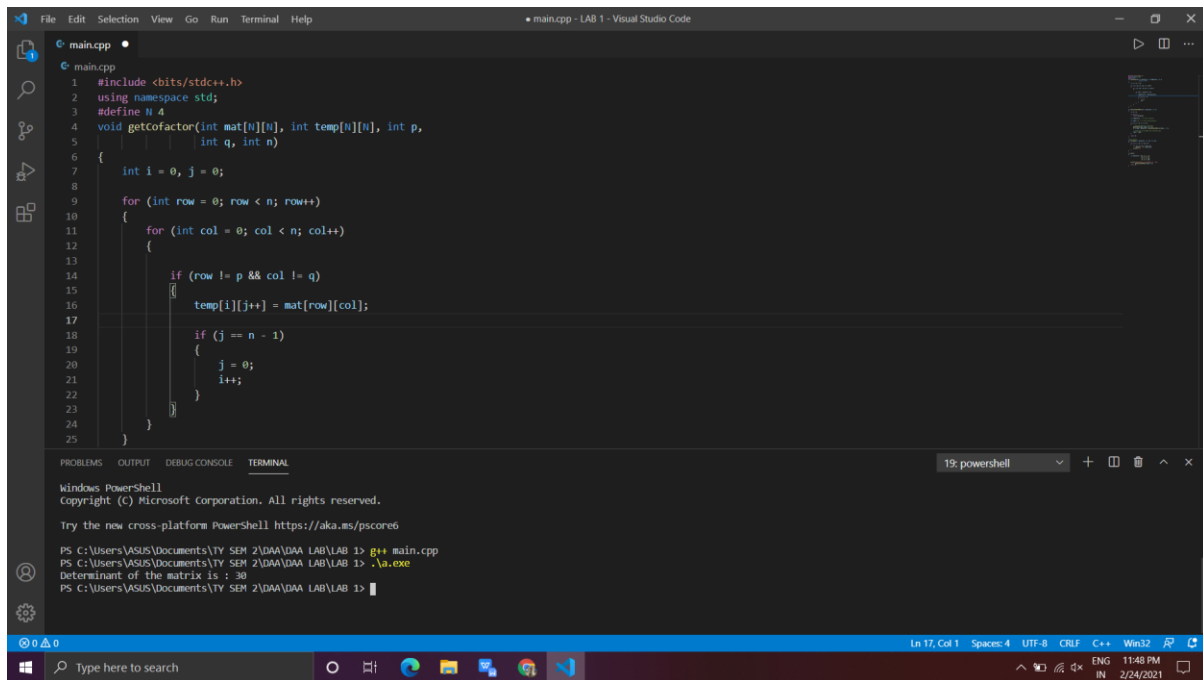
//Print matrix
void display(int mat[N][N], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            printf(" %d", mat[i][j]);
        printf("\n");
    }
}

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;  
}
```