WEEK – 8 .

PRIYANSHI

2016927

E

Q1.

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


#define ll long long
#define INF INT_MAX


using namespace std;


int prims(int **arr, int n)
{
    vector<bool> visited(n, false);
    vector<int> weight(n, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
min_heap;
    int src = 0;
    weight[src] = 0;
    min_heap.push(make_pair(weight[src], src));
    while (!min_heap.empty())
    {
```

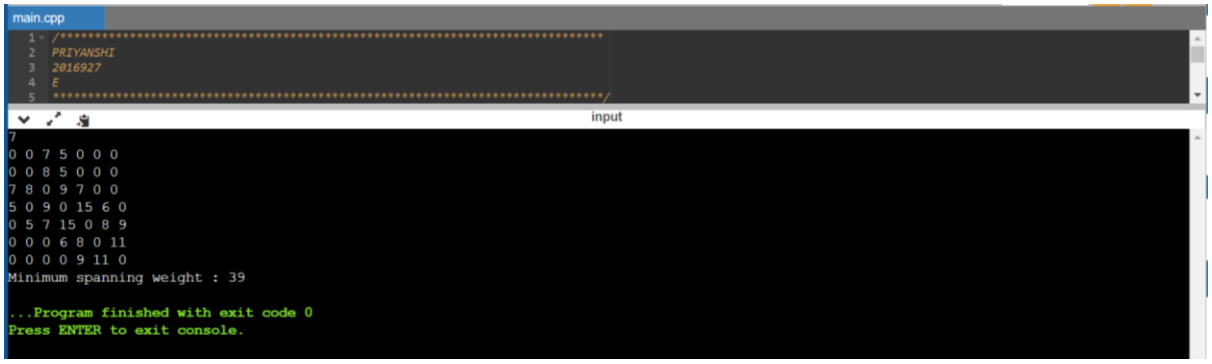
```
        int u = min_heap.top().second;
        min_heap.pop();
        if (!visited[u])
        {
            visited[u] = true;
            for (int v = 0; v < n; ++v)
            {
                if (!visited[v] && arr[u][v] != 0 && arr[u][v] < weight[v])
                {
                    weight[v] = arr[u][v];
                    min_heap.push(make_pair(weight[v], v));
                }
            }
        }
    }

    int sum = 0;
    for (auto i : weight)
        sum += i;
    return sum;
}

int main()
```

```
{
    int n;
    cin >> n;
    int **arr;
    arr = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        arr[i] = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> arr[i][j];
    cout << "Minimum spanning weight : " << prims(arr, n);
    return 0;
}
```

OUTPUT



Q2.

```
#include <bits/stdc++.h>
#define NIL -1
```

```
using namespace std;

int findParent(vector<int> parent, int u)
{
    if (parent[u] < 0)
        return u;
    return findParent(parent, parent[u]);
}

bool UnionByWeight(vector<int> &parent, int u, int v)
{
    int pu = findParent(parent, u);
    int pv = findParent(parent, v);
    if (pu != pv)
    {
        if (parent[pu] <= parent[pv])
        {
            parent[pu] += parent[pv];
            parent[pv] = pu;
        }
        else
        {
            parent[pv] += parent[pu];
```

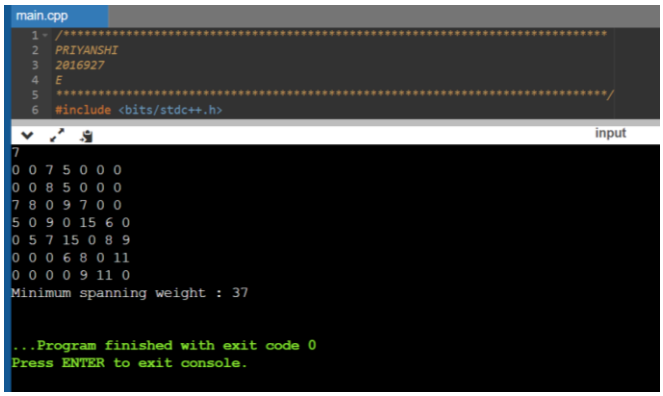
```
            parent[pu] = pv;
        }
        return true;
    }
    return false;
}

int kruskals(int **graph, int n)
{
    vector<pair<int, pair<int, int>>> G;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (graph[i][j] != 0)
                G.push_back(make_pair(graph[i][j], make_pair(i, j)));
    sort(G.begin(), G.end());
    vector<int> parent(n, NIL);
    int s = 0;
    for (auto i : G)
    {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (UnionByWeight(parent, u, v))
```

```
            s += w;
    }
    return s;
}

int main()
{
    int n;
    cin >> n;
    int **graph;
    graph = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        graph[i] = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> graph[i][j];
    cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;
    return 0;
}
```

OUTPUT



Q3.

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

#define NIL -1


using namespace std;


int findParent(vector<int> parent, int u)
{
    if (parent[u] < 0)
        return u;
    return findParent(parent, parent[u]);
}


bool UnionByWeight(vector<int> &parent, int u, int v)
{
    int pu = findParent(parent, u);
    int pv = findParent(parent, v);
```

```
    if (pu != pv)
    {
        if (parent[pu] <= parent[pv])
        {
            parent[pu] += parent[pv];
            parent[pv] = pu;
        }
        else
        {
            parent[pv] += parent[pu];
            parent[pu] = pv;
        }
        return true;
    }
    return false;
}

int kruskals(int **graph, int n)
{
    vector<pair<int, pair<int, int>>> G;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (graph[i][j] != 0)
```

```
            G.push_back(make_pair(graph[i][j], make_pair(i, j)));
    sort(G.begin(), G.end(), greater<pair<int, pair<int, int>>>());
    vector<int> parent(n, NIL);
    int s = 0;
    for (auto i : G)
    {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (UnionByWeight(parent, u, v))
            s += w;
    }
    return s;
}

int main()
{
    int n;
    cin >> n;
    int **graph;
    graph = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        graph[i] = (int *)malloc(n * sizeof(int));
```

```
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> graph[i][j];
    cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;
    return 0;
}
```

OUTPUT