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

struct Graph {
    int V;
    vector<vector<int>> adj;

    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void BFS(int start) {
        vector<bool> visited(V, false);
        queue<int> q;
        visited[start] = true;
        q.push(start);

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

            #pragma omp parallel for
            for (int i = 0; i < adj[u].size(); i++) {
                int v = adj[u][i];
                if (!visited[v]) {
                    #pragma omp critical
                    {
                        visited[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        cout << endl;
    }

    void DFS(int start) {
        vector<bool> visited(V, false);
```

```
        #pragma omp parallel
        {
            #pragma omp single nowait
            {
                DFSUtil(start, visited);
            }
        }
        cout << endl;
    }

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

        # pragma omp parallel for
        for (int i = 0; i < adj[u].size(); i++) {
            int v = adj[u][i];
            if (!visited[v]) {
                DFSUtil(v, visited);
            }
        }
    }
};

int main() {
    int V;
    cout << "Enter the number of vertices: ";
    cin >> V;

    Graph g(V);

    int edgeCount;
    cout << "Enter the number of edges: ";
    cin >> edgeCount;

    cout << "Enter the edges (in format 'source destination'): \n";
    for (int i = 0; i < edgeCount; i++) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }

    cout << "BFS traversal starting from node 0: ";
    g.BFS(0);
```

```
    cout << "DFS traversal starting from node 0: ";
    g.DFS(0);

    return 0;
}
```

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

void swap(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void bubble(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int first = i % 2;
#pragma omp parallel for shared(a, first)
        for (int j = first; j < n - 1; j += 2)
        {
            if (a[j] > a[j + 1])
            {
                swap(a[j], a[j + 1]);
            }
        }
    }
}
```

```
void merge(int a[], int i1, int j1, int i2, int j2)
{
    int temp[1000];
    int i, j, k;
    i = i1;
    j = i2;
    k = 0;


    while (i <= j1 && j <= j2)
    {
        if (a[i] < a[j])
        {
            temp[k++] = a[i++];
        }
        else
        {
            temp[k++] = a[j++];
        }
    }
    while (i <= j1)
    {
        temp[k++] = a[i++];
    }
    while (j <= j2)
    {
        temp[k++] = a[j++];
    }
    for (i = i1, j = 0; i <= j2; i++, j++)
    {
        a[i] = temp[j];
    }


}

void mergesort(int a[], int i, int j)
{
    int mid;
    if (i < j)
    {
        mid = (i + j) / 2;

#pragma omp parallel sections
```

```
        {
#pragma omp section
            {
                mergesort(a, i, mid);
            }

#pragma omp section
            {
                mergesort(a, mid + 1, j);
            }
        }
        merge(a, i, mid, mid + 1, j);
    }
}

int main()
{
    int n, i;
    cout << "\nEnter size of Array : ";
    cin >> n;
    int a[n];
    cout << "\nEnter elements : \n";
    for (i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    mergesort(a, 0, n - 1);
    cout << "\nSorted array is : ";
    for (i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }

     bubble(a, n);

    cout << "\nSorted array is : \n";
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << endl;
    }

    return 0;
}
```

```cpp
#include <iostream>
#include <omp.h>
#include <climits>
using namespace std;
void min_reduction(int arr[], int n)
{
    int min_value = INT_MAX;
#pragma omp parallel for reduction(min : min_value)
    for (int i = 0; i < n; i++)
    {
        if (arr[i] < min_value)
        {
            min_value = arr[i];
        }
    }
    cout << "Minimum value: " << min_value << endl;
}
void max_reduction(int arr[], int n)
{
    int max_value = INT_MIN;
#pragma omp parallel for reduction(max : max_value)
    for (int i = 0; i < n; i++)
    {
        if (arr[i] > max_value)
        {
            max_value = arr[i];
        }
    }
    cout << "Maximum value: " << max_value << endl;
}
void sum_reduction(int arr[], int n)
{
    int sum = 0;
#pragma omp parallel for reduction(+ : sum)
    for (int i = 0; i < n; i++)
    {
        sum += arr[i];
    }
    cout << "Sum: " << sum << endl;
}
void average_reduction(int arr[], int n)
{
    int sum = 0;
```

```
#pragma omp parallel for reduction(+ : sum)
    for (int i = 0; i < n; i++)
    {
        sum += arr[i];
    }
    cout << "Average: " << (double)sum / (n) << endl;
}
int main()
{
    int  n;
    cout << "\n enter total no of elements=>";
    cin >> n;
    int arr[n];
    cout << "\n enter elements=>";

    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    min_reduction(arr, n);
    max_reduction(arr, n);
    sum_reduction(arr, n);
    average_reduction(arr, n);
}
```