```
#include <iostream>
#include <queue>
#include <omp.h>
#include <iomanip> // For spacing
using namespace std;
class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = right = nullptr;
    }
};
class BreadthFS {
public:
    Node* insert(Node* root, int data);
    void bfsTreeView(Node* root);
};

// Level-order insertion
Node* BreadthFS::insert(Node* root, int data) {
    if (!root)
        return new Node(data);

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        Node* temp = q.front();
        q.pop();

        if (!temp->left) {
            temp->left = new Node(data);
            return root;
        }
        else
            q.push(temp->left);

        if (!temp->right) {
            temp->right = new Node(data);
            return root;
        }
        else
            q.push(temp->right);
```

```
    }

    return root;
}

// Tree-view BFS traversal using OpenMP
void BreadthFS::bfsTreeView(Node* root) {
    if (!root) return;

    queue<Node*> q;
    q.push(root);

    int level = 0;
    cout << "\nBFS Traversal using OpenMP (Tree View):\n";

    while (!q.empty()) {
        int size = q.size();

        // Print indentation for each level
        cout << setw(8 * (4 - level)) << ""; // Adjust spacing based on level

        #pragma omp parallel for
        for (int i = 0; i < size; i++) {
            Node* node;

            #pragma omp critical
            {
                node = q.front();
                q.pop();
                cout << node->data << "\t";

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        cout << "\n";
        level++;
    }
}
int main() {
    Node* root = nullptr;
    BreadthFS tree;
    int data;
    char ans;

    do {
        cout << "\nEnter data: ";
```

```
        cin >> data;

        root = tree.insert(root, data);

        cout << "Do you want to insert one more node? (y/n): ";
        cin >> ans;
    } while (ans == 'y' || ans == 'Y');

    tree.bfsTreeView(root);

    return 0;
}
```

Output:

```
C:\Users\user\Desktop\openmp>g++ -fopenmp -o bfs_openmp bfs_openmp.cpp

C:\Users\user\Desktop\openmp>bfs_openmp.exe

Enter data: 5
Do you want to insert one more node? (y/n): y

Enter data: 10
Do you want to insert one more node? (y/n): y

Enter data: 4
Do you want to insert one more node? (y/n): y

Enter data: 3
Do you want to insert one more node? (y/n): n

BFS Traversal using OpenMP (Tree View):
                            5
                    10      4
            3

C:\Users\user\Desktop\openmp>
```

```
#include <iostream>
#include <omp.h>
using namespace std;
// Swap two elements
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
// Odd-Even Transposition Sort using OpenMP
void bubble(int *a, int n) {
    for (int i = 0; i < n; i++) {
        int phase = i % 2;
        #pragma omp parallel for shared(a, phase)
        for (int j = phase; j < n - 1; j += 2) {
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
        }
    }
}
int main() {
    int *a, n;
    cout << "\nEnter total number of elements: ";
    cin >> n;
    a = new int[n];
    cout << "\nEnter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    bubble(a, n);
    cout << "\nSorted array:\n";
    for (int i = 0; i < n; i++) {
        cout << a[i] << endl;
    }

    delete[] a;
    return 0;
}
```

Output:

```
C:\Users\user\Desktop\openmp>g++ -fopenmp -o bubble_sort bubble_sort.cpp

C:\Users\user\Desktop\openmp>bubble_sort.exe

Enter total number of elements: 6

Enter elements:
8 4 45 2 1 7

Sorted array:
1
2
4
7
8
45
```

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

void merge(int* arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int* L = new int[n1];
    int* R = new int[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    while (i < n1)
        arr[k++] = L[i++];

    while (j < n2)
        arr[k++] = R[j++];

    delete[] L;
    delete[] R;
}
```

```
void mergeSort(int* arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        #pragma omp parallel sections
        {
            #pragma omp section
            mergeSort(arr, l, m);

            #pragma omp section
            mergeSort(arr, m + 1, r);
        }

        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    double start, stop;

    cout << "Given array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    start = omp_get_wtime();

    #pragma omp parallel
    {
        #pragma omp single
        mergeSort(arr, 0, n - 1);
    }

    stop = omp_get_wtime();

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    cout << "Time taken: " << (stop - start) << " seconds" << endl;

    return 0;
}
```

Output:

```
C:\Users\user\Desktop\openmp>g++ -fopenmp -o merge_sort merge_sort.cpp

C:\Users\user\Desktop\openmp>merge_sort.exe
Given array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13
Time taken: 0.00199986 seconds

C:\Users\user\Desktop\openmp>
```

```
#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 *arr, n;
    cout << "\nEnter total number of elements: ";
    cin >> n;
    arr = new int[n];

    cout << "\nEnter 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);

    delete[] arr;
    return 0;
}
```

OUTPUT:

```
C:\Users\user\Desktop\openmp>g++ -fopenmp -o min_max min_max.cpp

C:\Users\user\Desktop\openmp>min_max.exe

Enter total number of elements: 6

Enter elements: 4
12
26
36
54
65
Minimum value: 4
Maximum value: 65
Sum: 197
Average: 32.8333

C:\Users\user\Desktop\openmp>
```