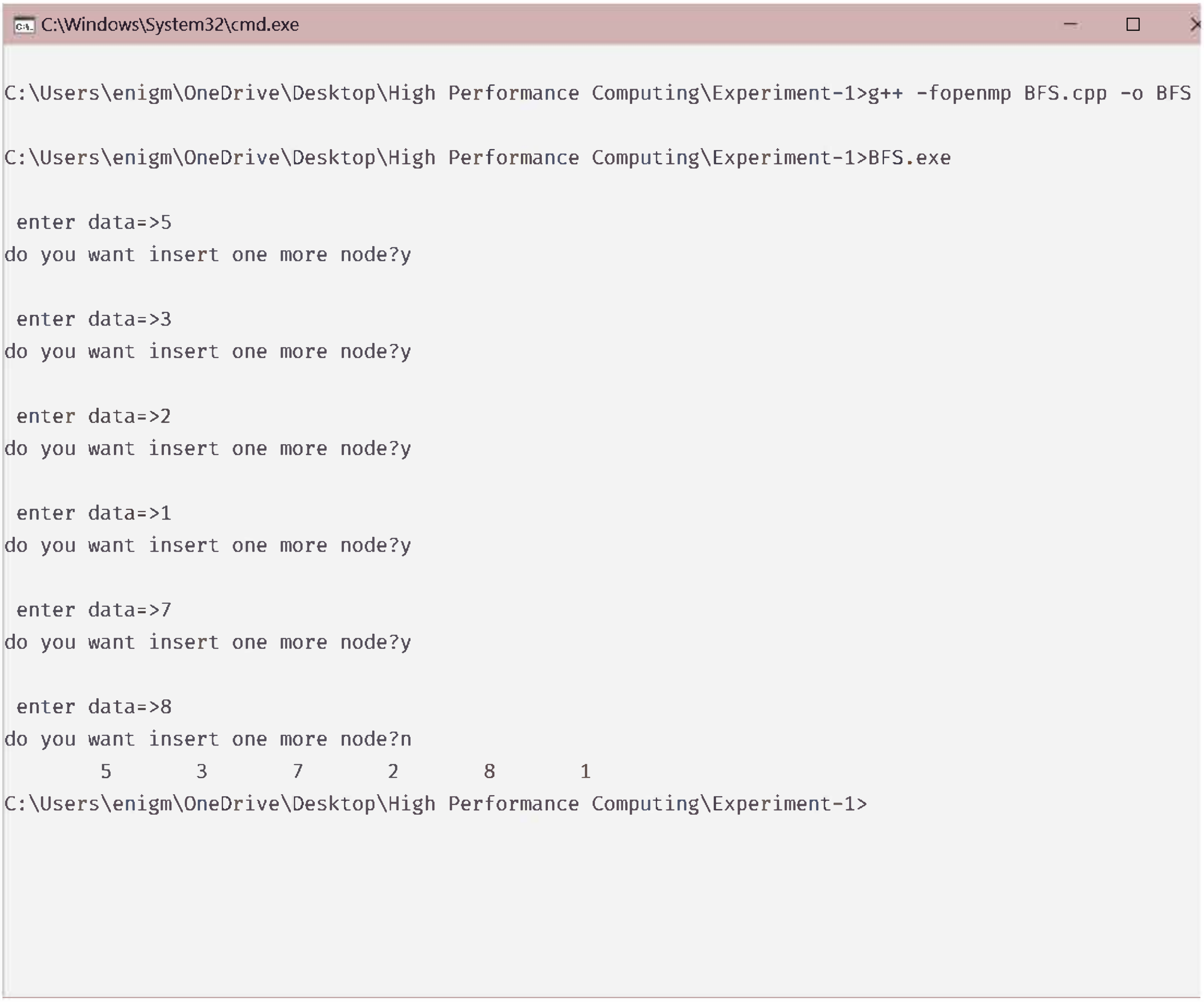
```
#include<iostream>

#include<stdlib.h>
#include<queue>
#include <omp.h>

using namespace std;
class node
{
   publi
          node *left,
        int

};

class Breadthfs
{

 publi

     node *insert(node *,
   void bfs(node

};

node *insert(node* root, int value)
{
    if (!root)
             root=new
                  root-
                   root-
              root->data=
             return


        // Insert
    if (value > root->data)
              root->right = insert(root->right,

    else if (value < root->data)
              root->left = insert(root->left,



        return
}

void bfs(node *head)
```

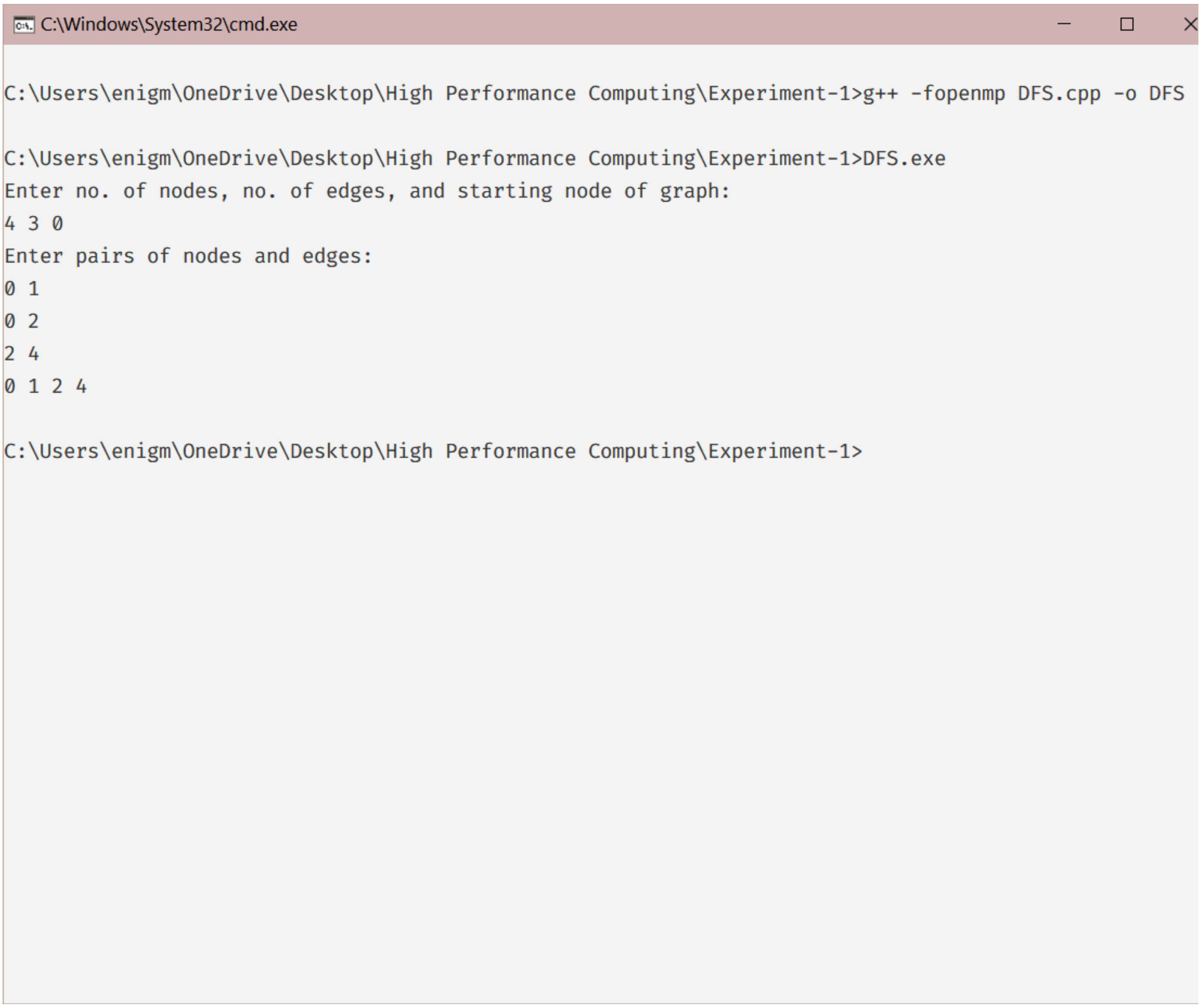
```
{

     queue<node*> q;
     q.push(head);

     int qSize;

     while (!q.empty())
     {
         qSize = q.size();
         #pragma omp parallel for
                //creates parallel threads
         for (int i = 0; i < qSize; i++)
         {
             node* currNode;
             #pragma omp critical
             {
               currNode = q.front();
               q.pop();
               cout<<"\t"<<currNode->data;

             }// prints parent node
             #pragma omp critical
             {
             if(currNode->left)// push parent's left node in queue
                 q.push(currNode->left);
             if(currNode->right)
                 q.push(currNode->right);
             }// push parent's right node in queue

         }
     }

}

int main(){

    node *root=NULL;
    int data;
    char ans;

    do
    {
     cout<<"\n enter data=>";
     cin>>data;
     root=insert(root,data);
     cout<<"do you want insert one more node?";
     cin>>ans;

    }while(ans=='y'||ans=='Y');

    bfs(root);
    return 0;
}
```



```
C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>g++ -fopenmp BFS.cpp -o BFS

C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>BFS.exe

 enter data=>5
do you want insert one more node?y

 enter data=>3
do you want insert one more node?y

 enter data=>2
do you want insert one more node?y

 enter data=>1
do you want insert one more node?y

 enter data=>7
do you want insert one more node?y

 enter data=>8
do you want insert one more node?n
        5       3       7       2       8       1
C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>
```

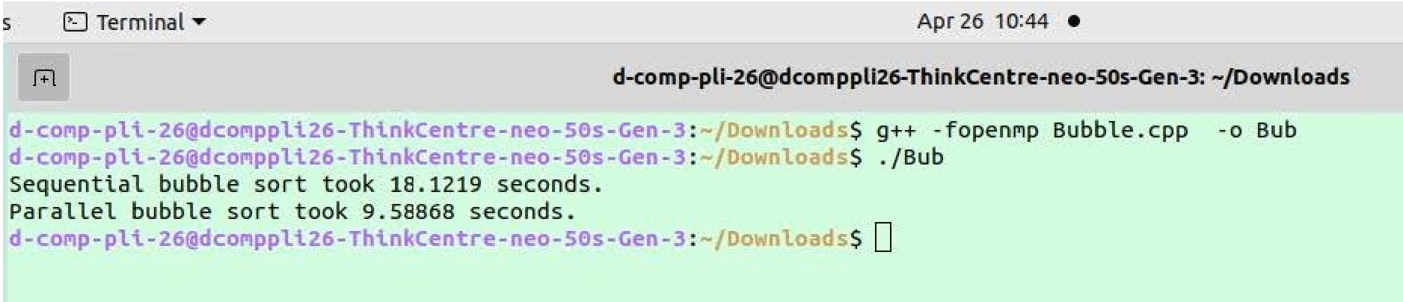
```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;

const int MAX = 100000;
vector<int> graph[MAX];
bool visited[MAX];

void dfs(int node) {
    stack<int> s;
    s.push(node);

    while (!s.empty()) {
        int curr_node = s.top();

        if (!visited[curr_node]) {
            visited[curr_node] = true;

        s.pop();
    cout<<curr_node<<" ";

            #pragma omp parallel for
            for (int i = 0; i < graph[curr_node].size(); i++) {
                int adj_node = graph[curr_node][i];
                if (!visited[adj_node]) {
                    s.push(adj_node);
                }
            }
        }
    }
}

int main() {
    int n, m, start_node;
    cout<<"Enter no. of Node,no. of Edges and Starting Node of graph:\n";
    cin >> n >> m >> start_node;
         //n: node,m:edges
        cout<<"Enter pair of node and edges:\n";

    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;

//u and v: Pair of edges
```

```
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    dfs(start_node);



    return 0;
}
```



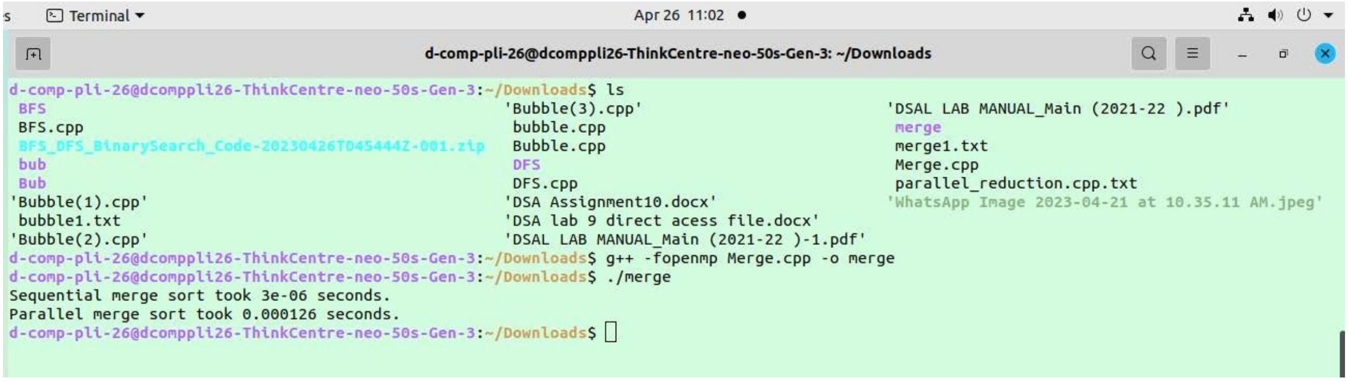
```
C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>g++ -fopenmp DFS.cpp -o DFS

C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>DFS.exe
Enter no. of nodes, no. of edges, and starting node of graph:
4 3 0
Enter pairs of nodes and edges:
0 1
0 2
2 4
0 1 2 4

C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-1>
```

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

void bubble(int *, int);
void swap(int &, int &);


void bubble(int *a, int n)
{
    int swapped;
    for(  int i = 0;  i < n;  i++ )
     {
        int first = i % 2;
        swapped=0;
        #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 ]  );
                    swapped=1;
             }
              }
              if(swapped==0)
              break;
     }
}


void swap(int &a, int &b)
{

    int test;
    test=a;
    a=b;
    b=test;

}

int main()
{

    int *a,n;
    cout<<"\n enter total no of elements=>";
    cin>>n;
    a=new int[n];
    cout<<"\n enter elements=>";
for(int i=0;i<n;i++)
{
    cin>>a[i];
}

double start_time = omp_get_wtime(); // start timer for sequential algorithm
bubble(a,n);
double end_time = omp_get_wtime(); // end timer for sequential algorithm

cout<<"\n sorted array is=>";
for(int i=0;i<n;i++)
{
    cout<<a[i]<< "  ";
}
cout << endl;
cout << "Time taken by sequential algorithm: " << end_time - start_time << " seconds"
<< endl;

start_time = omp_get_wtime(); // start timer for parallel algorithm
bubble(a,n);
end_time = omp_get_wtime(); // end timer for parallel algorithm

cout<<"\n sorted array is=>";
for(int i=0;i<n;i++)
{
    cout<<a[i]<<"  ";
}
cout << endl;
cout << "Time taken by parallel algorithm: " << end_time - start_time << " seconds" <<
 endl;

return 0;
}
```


s
Terminal ▾
Apr 26 10:44 ●
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3: ~/Downloads

```
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ g++ -fopenmp Bubble.cpp  -o Bub
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ ./Bub
Sequential bubble sort took 18.1219 seconds.
Parallel bubble sort took 9.58868 seconds.
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ 
```

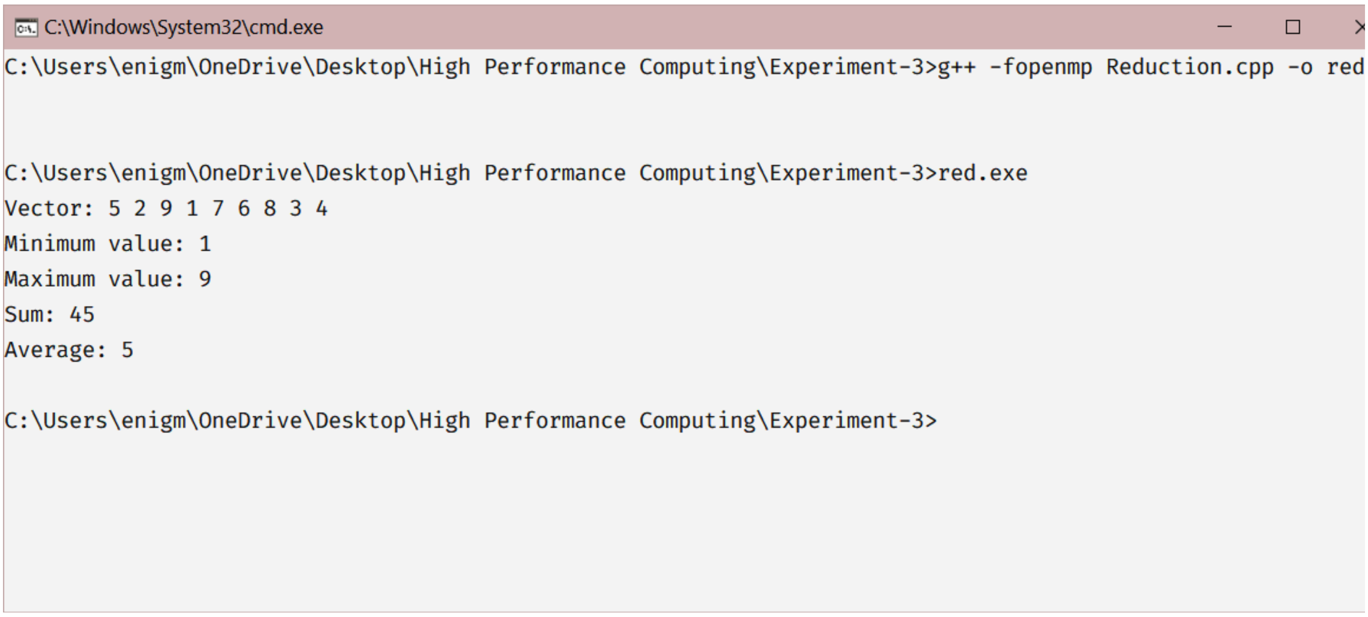
```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

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);
    }

}

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];
    }
}


int main()
{
    int *a,n,i;
    double start_time, end_time, seq_time, par_time;

    cout<<"\n enter total no of elements=>";
    cin>>n;
    a= new int[n];

    cout<<"\n enter elements=>";
    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }

    // Sequential algorithm
    start_time = omp_get_wtime();
    mergesort(a, 0, n-1);
    end_time = omp_get_wtime();
    seq_time = end_time - start_time;
    cout << "\nSequential Time: " << seq_time << endl;

    // Parallel algorithm
    start_time = omp_get_wtime();
    #pragma omp parallel
    {
        #pragma omp single
        {
            mergesort(a, 0, n-1);
        }
    }
    end_time = omp_get_wtime();
    par_time = end_time - start_time;
    cout << "\nParallel Time: " << par_time << endl;

    cout<<"\n sorted array is=>";
    for(i=0;i<n;i++)
    {
        cout<<a[i] << "  ";
    }

    return 0;
}

// 29 14 87 42 61 99 5 76 33 18
```



s
Terminal
Apr 26 11:02
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3: ~/Downloads

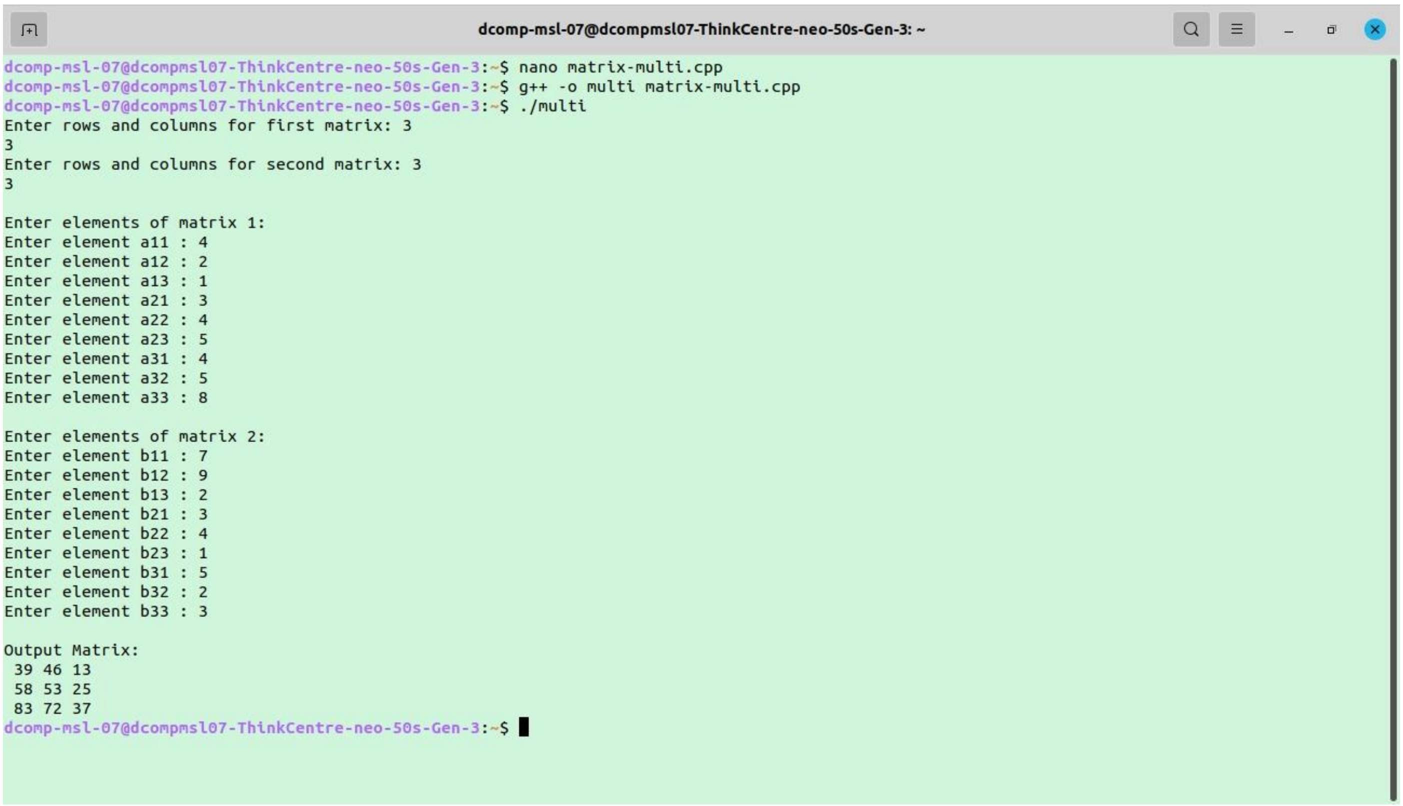
```
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ ls
 BFS                                                 'Bubble(3).cpp'                           'DSAL LAB MANUAL_Main (2021-22 ).pdf'
 BFS.cpp                                              bubble.cpp                                merge
 BFS_DFS_BinarySearch_Code-20230426T045444Z-001.zip   Bubble.cpp                                merge1.txt
 bub                                                  DFS                                       Merge.cpp
 Bub                                                  DFS.cpp                                   parallel_reduction.cpp.txt
'Bubble(1).cpp'                                      'DSA Assignment10.docx'                   'WhatsApp Image 2023-04-21 at 10.35.11 AM.jpeg'
 bubble1.txt                                         'DSA lab 9 direct acess file.docx'
'Bubble(2).cpp'                                      'DSAL LAB MANUAL_Main (2021-22 )-1.pdf'
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ g++ -fopenmp Merge.cpp -o merge
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ ./merge
Sequential merge sort took 3e-06 seconds.
Parallel merge sort took 0.000126 seconds.
d-comp-pli-26@dcomppli26-ThinkCentre-neo-50s-Gen-3:~/Downloads$ 
```

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <climits>

using namespace std;

int min_reduction(vector<int>& arr) {
  int min_value = INT_MAX;
  #pragma omp parallel for reduction(min: min_value)
  for (int i = 0; i < arr.size(); i++) {
    if (arr[i] < min_value) {
      min_value = arr[i];
    }
  }
  return min_value;
}

int max_reduction(vector<int>& arr) {
  int max_value = INT_MIN;
  #pragma omp parallel for reduction(max: max_value)
  for (int i = 0; i < arr.size(); i++) {
    if (arr[i] > max_value) {
      max_value = arr[i];
    }
  }
  return max_value;
}

int sum_reduction(vector<int>& arr) {
  int sum = 0;
   #pragma omp parallel for reduction(+: sum)
   for (int i = 0; i < arr.size(); i++) {
    sum += arr[i];
  }
  return sum;
}

double average_reduction(vector<int>& arr) {
  int sum = 0;
  #pragma omp parallel for reduction(+: sum)
  for (int i = 0; i < arr.size(); i++) {
    sum += arr[i];
  }
  return (double)sum / arr.size();
}

int main() {
  vector<int> arr;
  arr.push_back(5);
  arr.push_back(2);
  arr.push_back(9);
  arr.push_back(1);
  arr.push_back(7);
  arr.push_back(6);
  arr.push_back(8);
  arr.push_back(3);
  arr.push_back(4);

  int min_value = min_reduction(arr);
  int max_value = max_reduction(arr);
  int sum = sum_reduction(arr);
  double average = average_reduction(arr);

  cout << "Vector: ";
    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << " ";
    }

  cout << "\nMinimum value: " << min_value << endl;
  cout << "Maximum value: " << max_value << endl;
  cout << "Sum: " << sum << endl;
  cout << "Average: " << average << endl;
}
```



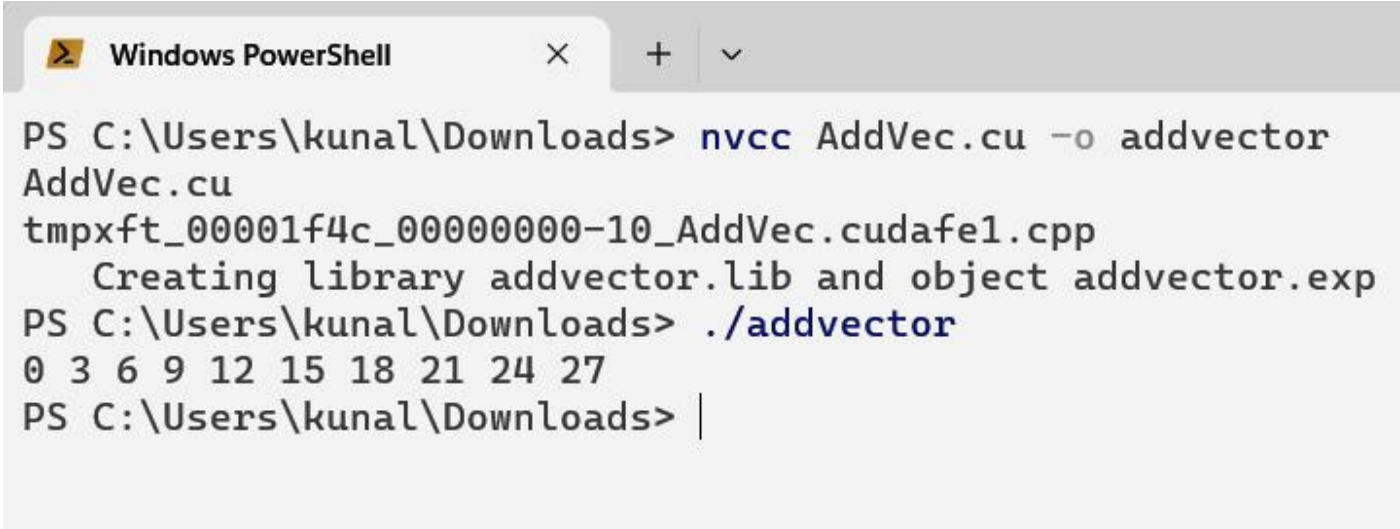
```
C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-3>g++ -fopenmp Reduction.cpp -o red


C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-3>red.exe
Vector: 5 2 9 1 7 6 8 3 4
Minimum value: 1
Maximum value: 9
Sum: 45
Average: 5

C:\Users\enigm\OneDrive\Desktop\High Performance Computing\Experiment-3>
```

```
#include <cuda_runtime.h>
#include <iostream>

__global__ void matmul(int* A, int* B, int* C, int N) {
    int Row = blockIdx.y * blockDim.y + threadIdx.y;
    int Col = blockIdx.x * blockDim.x + threadIdx.x;

    if (Row < N && Col < N) {
        int Pvalue = 0;
        for (int k = 0; k < N; k++) {
            Pvalue += A[Row * N + k] * B[k * N + Col];
        }
        C[Row * N + Col] = Pvalue;
    }
}

int main() {
    int N = 512;
    int size = N * N * sizeof(int);
    int* A, * B, * C;
    int* dev_A, * dev_B, * dev_C;

    // Allocate memory on the host
    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);

    // Allocate memory on the device
    cudaMalloc(&dev_A, size);
    cudaMalloc(&dev_B, size);
    cudaMalloc(&dev_C, size);

    // Initialize matrices A and B
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i * N + j] = i * N + j;
            B[i * N + j] = j * N + i;
        }
    }

    // Copy data from host to device
    cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);

    // Set the block and grid dimensions
    dim3 dimBlock(16, 16);
    dim3 dimGrid(N / dimBlock.x, N / dimBlock.y);

    // Launch the kernel
    matmul<<<dimGrid, dimBlock>>>(dev_A, dev_B, dev_C, N);

    // Copy the result from device to host
    cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost);

    // Print the result
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            std::cout << C[i * N + j] << " ";
        }
        std::cout << std::endl;
    }

    // Free memory
    cudaFree(dev_A);
    cudaFree(dev_B);
    cudaFree(dev_C);
    cudaFreeHost(A);
    cudaFreeHost(B);
    cudaFreeHost(C);

    return 0;
}
```



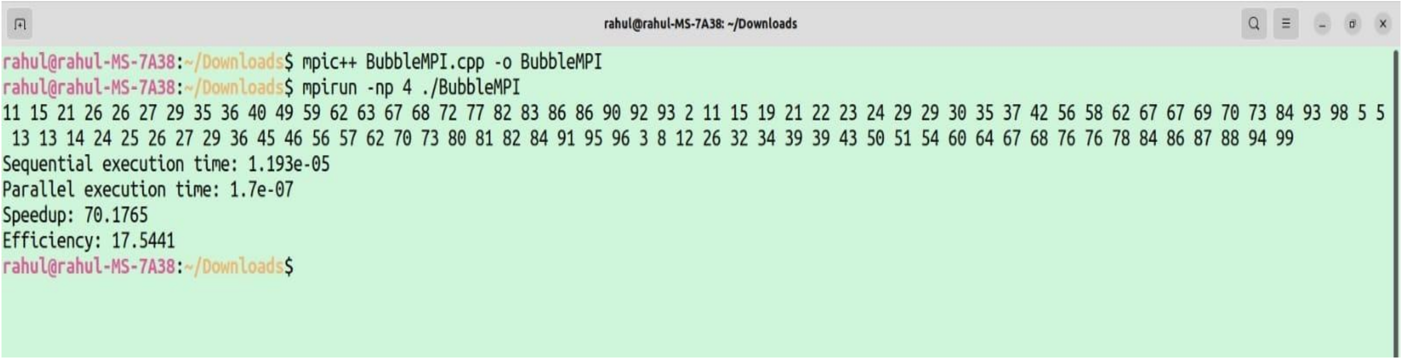
```
dcomp-msl-07@dcompmsl07-ThinkCentre-neo-50s-Gen-3:~$ nano matrix-multi.cpp
dcomp-msl-07@dcompmsl07-ThinkCentre-neo-50s-Gen-3:~$ g++ -o multi matrix-multi.cpp
dcomp-msl-07@dcompmsl07-ThinkCentre-neo-50s-Gen-3:~$ ./multi
Enter rows and columns for first matrix: 3
3
Enter rows and columns for second matrix: 3
3

Enter elements of matrix 1:
Enter element a11 : 4
Enter element a12 : 2
Enter element a13 : 1
Enter element a21 : 3
Enter element a22 : 4
Enter element a23 : 5
Enter element a31 : 4
Enter element a32 : 5
Enter element a33 : 8

Enter elements of matrix 2:
Enter element b11 : 7
Enter element b12 : 9
Enter element b13 : 2
Enter element b21 : 3
Enter element b22 : 4
Enter element b23 : 1
Enter element b31 : 5
Enter element b32 : 2
Enter element b33 : 3

Output Matrix:
 39 46 13
 58 53 25
 83 72 37
dcomp-msl-07@dcompmsl07-ThinkCentre-neo-50s-Gen-3:~$ 
```

```
#include <iostream>
#include <cuda_runtime.h>

__global__ void addVectors(int* A, int* B, int* C, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        C[i] = A[i] + B[i];
    }
}

int main() {
    int n = 1000000;
    int* A, * B, * C;
    int size = n * sizeof(int);

    // Allocate memory on the host
    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);

    // Initialize the vectors
    for (int i = 0; i < n; i++) {
        A[i] = i;
        B[i] = i * 2;
    }

    // Allocate memory on the device
    int* dev_A, * dev_B, * dev_C;
    cudaMalloc(&dev_A, size);
    cudaMalloc(&dev_B, size);
    cudaMalloc(&dev_C, size);

    // Copy data from host to device
    cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);

    // Launch the kernel
    int blockSize = 256;
    int numBlocks = (n + blockSize - 1) / blockSize;
    addVectors<<<numBlocks, blockSize>>>(dev_A, dev_B, dev_C, n);

    // Copy data from device to host
    cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost);

    // Print the results
    for (int i = 0; i < 10; i++) {
        std::cout << C[i] << " ";
    }
    std::cout << std::endl;
    // Free memory
    cudaFree(dev_A);
    cudaFree(dev_B);
    cudaFree(dev_C);
    cudaFreeHost(A);
    cudaFreeHost(B);
    cudaFreeHost(C);

    return 0;
}
```



```
PS C:\Users\kunal\Downloads> nvcc AddVec.cu -o addvector
AddVec.cu
tmpxft_00001f4c_00000000-10_AddVec.cudafe1.cpp
   Creating library addvector.lib and object addvector.exp
PS C:\Users\kunal\Downloads> ./addvector
0 3 6 9 12 15 18 21 24 27
PS C:\Users\kunal\Downloads>
```

```
#include <iostream>
#include <algorithm>
#include <mpi.h>

using namespace std;

// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high- 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Function to perform quicksort on the partition
void quicksort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quicksort(arr, low, pivot - 1);
        quicksort(arr, pivot + 1, high);
    }
}

int main(int argc, char *argv[]) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int n = 100;
    int* arr = new int[n];
    int* recvbuf = new int[n];
    int* sendbuf = new int[n];
    // Fill the array with random values
    if (rank == 0) {
        for (int i = 0; i < n; i++) {
            arr[i] = rand() % 100;
        }
    }
    // Divide the array into equal-sized partitions for each process
    int sub_arr_size = n / size;
    int* sub_arr = new int[sub_arr_size];
    MPI_Scatter(arr, sub_arr_size, MPI_INT, sub_arr, sub_arr_size, MPI_INT, 0,
MPI_COMM_WORLD);
    // Sort the partition using quicksort
    quicksort(sub_arr, 0, sub_arr_size - 1);
    // Gather the sorted partitions from each process
    MPI_Gather(sub_arr, sub_arr_size, MPI_INT, recvbuf, sub_arr_size, MPI_INT, 0,
MPI_COMM_WORLD);
    if (rank == 0) {
        // Print the sorted array
        for (int i = 0; i < n; i++) {
            cout << recvbuf[i] << " ";
        }
        cout << endl;
        // Measure the execution time of the parallel quicksort algorithm
        double start_time = MPI_Wtime();
        // Perform the above steps again
        double end_time = MPI_Wtime();
        double parallel_execution_time = end_time - start_time;
        // Measure the execution time of the sequential quicksort algorithm
        start_time = MPI_Wtime();
        quicksort(arr, 0, n - 1);
        end_time = MPI_Wtime();
        double sequential_execution_time = end_time - start_time;
        // Calculate speedup and efficiency
        double speedup = sequential_execution_time / parallel_execution_time;
        double efficiency = speedup / size;
        cout << "Sequential execution time: " << sequential_execution_time << endl;
        cout << "Parallel execution time: " << parallel_execution_time << endl;
        cout << "Speedup: " << speedup << endl;
        cout << "Efficiency: " << efficiency << endl;
    }
    MPI_Finalize();
    return 0;
}
```



```
rahul@rahul-MS-7A38:~/Downloads$ mpic++ BubbleMPI.cpp -o BubbleMPI
rahul@rahul-MS-7A38:~/Downloads$ mpirun -np 4 ./BubbleMPI
11 15 21 26 26 27 29 35 36 40 49 59 62 63 67 68 72 77 82 83 86 86 90 92 93 2 11 15 19 21 22 23 24 29 29 30 35 37 42 56 58 62 67 67 69 70 73 84 93 98 5 5
 13 13 14 24 25 26 27 29 36 45 46 56 57 62 70 73 80 81 82 84 91 95 96 3 8 12 26 32 34 39 39 43 50 51 54 60 64 67 68 76 76 78 84 86 87 88 94 99
Sequential execution time: 1.193e-05
Parallel execution time: 1.7e-07
Speedup: 70.1765
Efficiency: 17.5441
rahul@rahul-MS-7A38:~/Downloads$
```