# MIN-MAX

```
#include <iostream>
#include<vector>
#include<limits.h>
#include<omp.h>
using namespace std;
void minP(const vector<int>&arr)
{
  int min=INT_MAX;
  #pragma omp parallel reduction(min:min_val)
  for(int i=0;i<arr.size();i++)
  {
    if(arr[i]<min)
    {
      min=arr[i];
    }
  }
  cout<<min<<endl;
}
void maxP(const vector<int>&arr)
{
  int max=INT_MIN;
  #pragma omp parallel reduction(max:max_val)
  for(int i=0;i<arr.size();i++)
  {
    if(arr[i]>max)
    {
       max=arr[i];
    }
  }
  cout<<max<<endl;
}
int sumP(const vector<int>&arr)
{
  int sum=0;
```

```
    for(int i=0;i<arr.size();i++)
    {
        sum += arr[i];
    }
    return sum;
}
void avgP(const vector<int>&arr)
{
    int avg = sumP(arr);
    cout<<(double)avg/arr.size()<<endl;
}
int main() {
    // Write C++ code here
    vector<int>arr={10,2,4,31,5};
    minP(arr);
    maxP(arr);
    int s = sumP(arr);
    cout<<s<<endl;
    avgP(arr);
    return 0;
}
```

# PBFS PDFS

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

struct Node
{
   int id;
   vector<Node*> neighbor;
};
void PBFS(Node* startN)
{
   queue<Node*>q;
   vector<bool>visited(startN->id+1,false);
   q.push(startN);
   visited[startN->id]=true;
   #pragma omp parallel
   while(!q.empty())
   {
      #pragma omp for
      {
         for(int i=0;i<q.size();i++)
         {
            Node* currentN;
            #pragma omp critical
            {
               currentN=q.front();
               q.pop();
            }
            cout<<"Visited Node: "<<currentN->id<<endl;;
              for(Node* neighborN:currentN->neighbor)
              {
                 if(!visited[neighborN->id])
```

```
                    {
                        visited[neighborN->id]=true;
                        q.push(neighborN);
                    }
                }
            }
        }
    }
}
void PDFS(Node* currentN,vector<bool>&visited)
{
    cout<<"Visited Node"<<currentN->id<<endl;
    visited[currentN->id]=true;
    #pragma omp for
    {
        for(int i=0;i<currentN->neighbor.size();i++)
        {
         Node* nN=currentN->neighbor[i];
         if(!visited[nN->id])
         {
             #pragma omp task
             {
                 PDFS(nN,visited);
             }
         }
        }
    }

}
int main()
{
    Node* node1 = new Node{ 1 };
    Node* node2 = new Node{ 2 };
    Node* node3 = new Node{ 3 };
    Node* node4 = new Node{ 4 };
    Node* node5 = new Node{ 5 };
    Node* node6 = new Node{ 6 };
    Node* node7 = new Node{ 7 };

    Node* node8 = new Node{ 8 };


    node1->neighbor= { node2,node5 };

    node2->neighbor = { node1,node3 };

    node3->neighbor = { node2,node4 };

    node4->neighbor = { node3 };

    node5->neighbor = { node1,node6 };

    node6->neighbor = { node5,node7,node8 };

    node7->neighbor = { node6 };

    node8->neighbor = { node6 };


    cout<<"Parallel BFS"<<endl;

    PBFS(node1);

    vector<bool> visited(5,false);

    PDFS(node1,visited);


    delete node1;

    delete node2;

    delete node3;

    delete node4;

    delete node5;

    delete node6;

    delete node7;

    delete node8;

    return 0;

}
```

# Parallel BFS

```
#include<iostream>

#include<omp.h>

#include<vector>

using namespace std;


void PB(vector<int>&arr)

{

    int n =arr.size();

    bool sorted=false;

    #pragma omp parallel

    {

        while(!sorted)

        {

            sorted=true;

            #pragma omp for

            {

                for(int i=0;i<n-1;i++)

                {

                    if(arr[i]>arr[i+1])

                    {

                        swap(arr[i],arr[i+1]);

                        sorted=false;

                    }  }

            }  }

    } }

int main()

{

    vector<int> arr={4,6,2,3};

    PB(arr);

    for(int num: arr)

        cout<<num<<endl;


    return 0;

}
```