Assignment 1(A)

```
#include<iostream>
#include<stdlib.h>
#include<queue>
using namespace std;
class node
{
 public:

 node *left, *right;
 int data;
};
class Breadthfs
{
public:
node *insert(node *, int);
void bfs(node *);
};
node *insert(node *root, int data)
// inserts a node in tree
{
 if(!root)
 {

 root=new node;

 root->left=NULL;
 root->right=NULL;
 root->data=data;
 return root;
 }
 queue<node *> q;
 q.push(root);

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

```
 if(temp->left==NULL)
 {

 temp->left=new node;
 temp->left->left=NULL;
 temp->left->right=NULL;
 temp->left->data=data;
 return root;
 }
 else
 {
 q.push(temp->left);
 }
 if(temp->right==NULL)
 {

 temp->right=new node;
 temp->right->left=NULL;
 temp->right->right=NULL;

 temp->right->data=data;
 return root;
 }
 else
 {
 q.push(temp->right);
 }
 }

}
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;
}
```

Output:

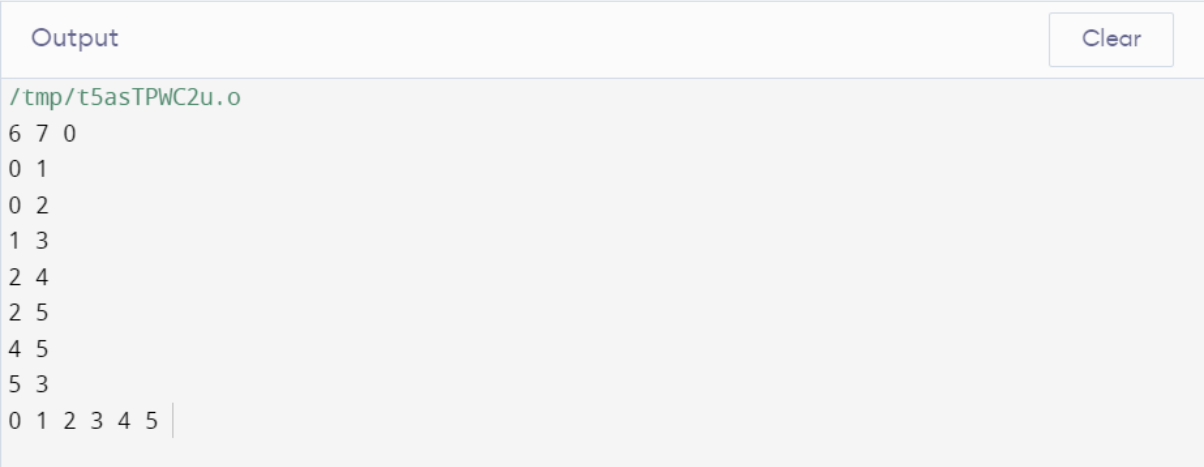
```
Output                                                    Clear

/tmp/cLvpQYNGOW.o
enter data=>5
do you want insert one more node?y
enter data=>9
do you want insert one more node?y
enter data=>8
do you want insert one more node?y
enter data=>3
do you want insert one more node?y
enter data=>7
do you want insert one more node?n
5   9   8   3   7
```

Assignment 1(B)

```
#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();
                s.pop();
                if (!visited[curr_node]) {
                        visited[curr_node] = true;
                #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;
cin >> n >> m >> start_node;
 //n: node,m:edges
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);
for (int i = 0; i < n; i++) {
 if (visited[i]) {
 cout << i << " ";
 }
}
return 0;
}
```

Output:

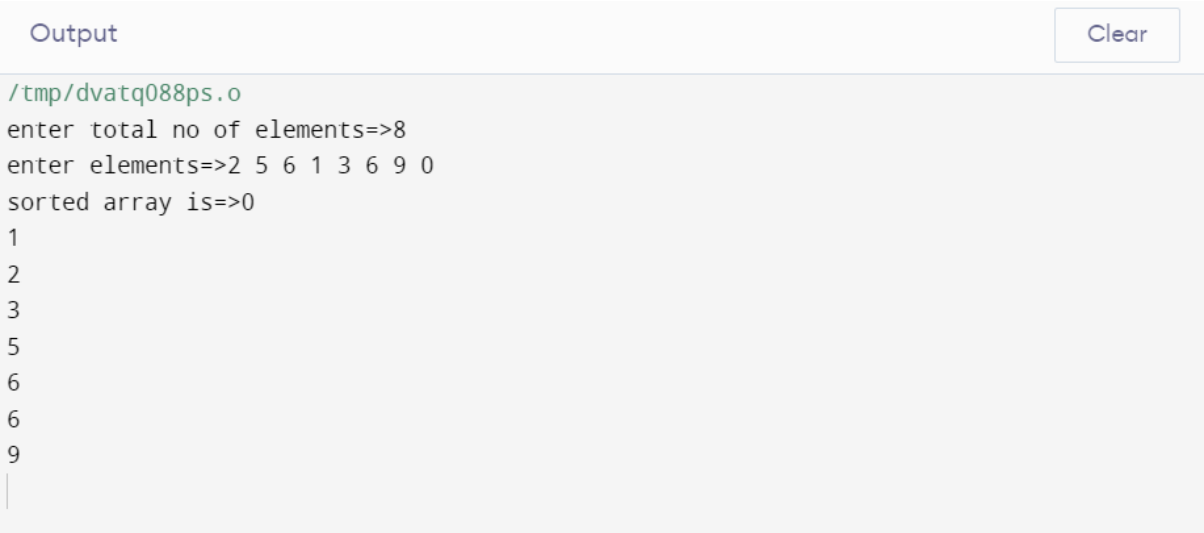


Assignment 2(A)

```
#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)
{
        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 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];
        }

        bubble(a,n);

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

        return 0;
}
```

Output:



Assignment 2(B)

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

using namespace std;

void merge(vector<int>& arr, int left, int middle, int right) {
   int i, j, k;
   int n1 = middle - left + 1;
   int n2 = right - middle;

   vector<int> L(n1), R(n2);

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

   i = 0;
   j = 0;
   k = left;
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++;
      }
      else {
         arr[k] = R[j];
         j++;
      }
      k++;
   }

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

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

void mergeSort(vector<int>& arr, int left, int right) {
    if (left >= right) {
        return;
    }

    int middle = left + (right - left) / 2;

    #pragma omp parallel sections num_threads(2)
    {
        #pragma omp section
        {
            mergeSort(arr, left, middle);
        }

        #pragma omp section
        {
            mergeSort(arr, middle + 1, right);
        }
    }

    merge(arr, left, middle, right);
}

int main() {
    vector<int> arr = { 12, 11, 13, 5, 6, 7 };
    int n = arr.size();

    mergeSort(arr, 0, n - 1);

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

```
    cout << endl;

    return 0;
}
```

Output: