Laboratory Program 1

Write program to do the following:
a. Print all the nodes reachable from a given starting node in a digraph using the BFS method.
b. Check whether a given graph is connected or not using the DFS method.

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

// Function to perform Breadth First Search (BFS)
void BFS(vector<vector<int>>& graph, int startNode) {
    int numNodes = graph.size();
    vector<bool> visited(numNodes, false);  // Keep track of visited nodes
    queue<int> q;  // Queue for BFS traversal

    visited[startNode] = true;  // Mark the start node as visited
    q.push(startNode);  // Enqueue the start node

    cout << "Nodes reachable from " << startNode << " using BFS: ";
    while (!q.empty()) {
        int currentNode = q.front();
        q.pop();
        cout << currentNode << " ";

        // Enqueue all the unvisited neighbors of the current node
        for (int neighbor = 0; neighbor < numNodes; ++neighbor) {
            if (graph[currentNode][neighbor] && !visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    cout << endl;
}

// Depth First Search (DFS) helper function
void DFSUtil(vector<vector<int>>& graph, int node, vector<bool>& visited) {
    visited[node] = true;

    // Recursive call for all unvisited neighbors
    for (int neighbor = 0; neighbor < graph.size(); ++neighbor) {
        if (graph[node][neighbor] && !visited[neighbor]) {
```

```
            DFSUtil(graph, neighbor, visited);
        }
    }
}

// Function to perform Depth First Search (DFS)
bool DFS(vector<vector<int>>& graph) {
    int numNodes = graph.size();
    vector<bool> visited(numNodes, false);  // Keep track of visited nodes

    DFSUtil(graph, 0, visited);  // Start DFS traversal from node 0

    // Check if all nodes were visited
    for (bool v : visited) {
        if (!v) {
            return false;  // Graph is not connected
        }
    }
    return true;  // Graph is connected
}

int main() {
    int numNodes;
    cout << "Enter the number of nodes in the graph: ";
    cin >> numNodes;

    vector<vector<int>> graph(numNodes, vector<int>(numNodes));

    cout << "Enter the adjacency matrix (0 for no edge, 1 for edge exists):" << endl;
    for (int i = 0; i < numNodes; ++i) {
        for (int j = 0; j < numNodes; ++j) {
            cin >> graph[i][j];
        }
    }

    int startNode;
    cout << "Enter the starting node for BFS: ";
    cin >> startNode;

    BFS(graph, startNode);

    bool isConnected = DFS(graph);
    if (isConnected) {
        cout << "The graph is connected." << endl;
```

```
    } else {
        cout << "The graph is not connected." << endl;
    }

    return 0;
}
```

Output

```
Enter the number of nodes in the graph: 3
Enter the adjacency matrix (0 for no edge, 1 for edge exists):
0 0 0
1 0 0
0 1 0
Enter the starting node for BFS: 1
Nodes reachable from 1 using BFS: 1 0
The graph is not connected.
```

Laboratory Program 2

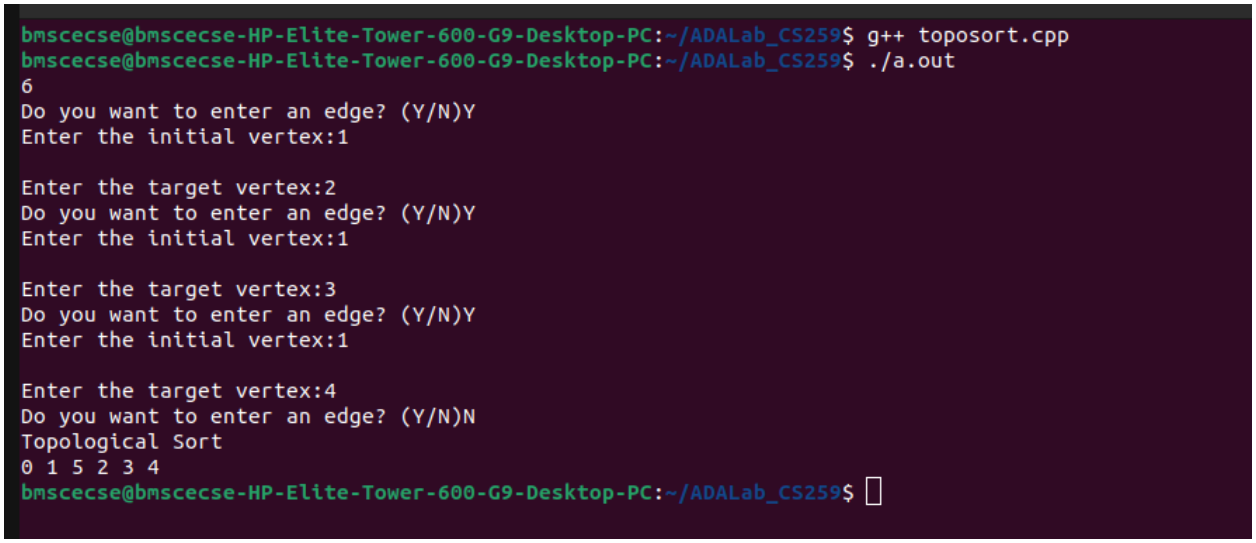
Write a program to obtain the Topological ordering of vertices in a given digraph.

```
#include <bits/stdc++.h>
using namespace std;
class Graph {
    int V;
    list<int>* adj;

public:

    Graph(int V);
    void addEdge(int u, int v);
    void topologicalSort();
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int u, int v)
{
    adj[u].push_back(v);
}
void Graph::topologicalSort()
{
    vector<int> in_degree(V, 0);
    for (int u = 0; u < V; u++) {
            list<int>::iterator itr;
            for (itr = adj[u].begin();
                    itr != adj[u].end(); itr++)
                    in_degree[*itr]++;
    }
    queue<int> q;
    for (int i = 0; i < V; i++)
            if (in_degree[i] == 0)
                    q.push(i);
    int cnt = 0;
    vector<int> top_order;
    while (!q.empty()) {
```

```
            int u = q.front();
            q.pop();
            top_order.push_back(u);
            list<int>::iterator itr;
            for (itr = adj[u].begin();
                        itr != adj[u].end(); itr++)
                        if (--in_degree[*itr] == 0)
                                    q.push(*itr);

            cnt++;
      }
      if (cnt != V) {
            cout << "There exists a cycle in the graph\n";
            return;
      }
      for (int i = 0; i < top_order.size(); i++)
            cout << top_order[i] << " ";
      cout << endl;
}
int main()
{
            int n;
            cin>>n;
      Graph g(n);
      for(int i=0;i<n;i++)
      {
            cout<<"Do you want to enter an edge? (Y/N)";
            char ch;
            cin>>ch;
            if(ch=='N')
            break;
            else if(ch=='Y')
            {
            int from,to;
            cout<<"Enter the initial vertex:";
            cin>>from;
            cout<<endl<<"Enter the target vertex:";
            cin>>to;
            g.addEdge(from,to);
            }
      }

      cout << "Topological Sort\n";
      g.topologicalSort();
```

```
    return 0;
}
```

Output



Laboratory Program 3

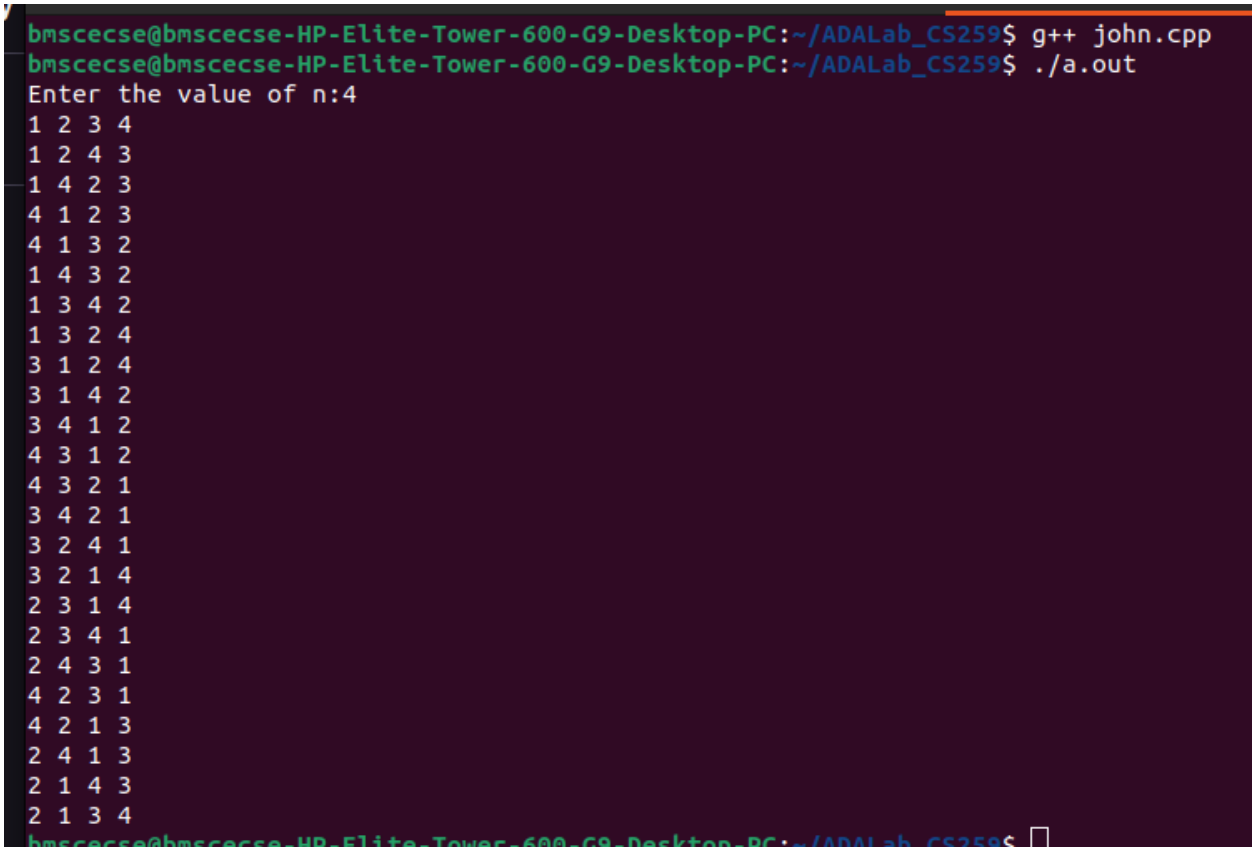
Implement Johnson Trotter algorithm to generate permutations.

```
#include <bits/stdc++.h>
#define LEFT_TO_RIGHT  1
#define RIGHT_TO_LEFT 0
using namespace std;

int searchArr(int a[], int n, int mobile)
{
	for (int i = 0; i < n; i++)
	if (a[i] == mobile)
	return i + 1;

	return -1;
}


int getMobile(int a[], bool dir[], int n)
{
	int mobile_prev = 0, mobile = 0;
	for (int i = 0; i < n; i++) {
	// direction 0 represents RIGHT TO LEFT.
	if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
	if (a[i] > a[i - 1] && a[i] > mobile_prev) {
		mobile = a[i];
		mobile_prev = mobile;
	}
	}

		if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
	if (a[i] > a[i + 1] && a[i] > mobile_prev) {
		mobile = a[i];
		mobile_prev = mobile;
	}
	}
	}

	if (mobile == 0 && mobile_prev == 0)
	return 0;
	else
	return mobile;
}
```

```
void printOnePerm(int a[], bool dir[], int n)
{
	int mobile = getMobile(a, dir, n);
	int pos = searchArr(a, n, mobile);

	if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
	swap(a[pos - 1], a[pos - 2]);

	else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
	swap(a[pos], a[pos - 1]);

	for (int i = 0; i < n; i++) {
	if (a[i] > mobile) {
	if (dir[a[i] - 1] == LEFT_TO_RIGHT)
		dir[a[i] - 1] = RIGHT_TO_LEFT;
	else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
		dir[a[i] - 1] = LEFT_TO_RIGHT;
	}
	}

	for (int i = 0; i < n; i++)
	cout << a[i]<<" ";
	cout <<endl;
}

int fact(int n)
{
	int res = 1;
	for (int i = 1; i <= n; i++)
	res = res * i;
	return res;
}

void printPermutation(int n)
{
	int a[n];
	bool dir[n];

	for (int i = 0; i < n; i++) {
	a[i] = i + 1;
	cout << a[i]<<" ";
	}
	cout << endl;
```

```
        for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

        for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}
int main()
{
   int n;
   cout<<"Enter the value of n:";
   cin>>n;
   printPermutation(n);
   return 0;
}
```

Output



Laboratory Program 4

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

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


void merge(int array[], int const left, int const mid,int const right)
{
    int const subArrayOne = mid - left + 1;
    int const subArrayTwo = right - mid;


    auto *leftArray = new int[subArrayOne],
            *rightArray = new int[subArrayTwo];


    for (auto i = 0; i < subArrayOne; i++)
            leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
            rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;


    while (indexOfSubArrayOne < subArrayOne
            && indexOfSubArrayTwo < subArrayTwo) {
            if (leftArray[indexOfSubArrayOne]
                    <= rightArray[indexOfSubArrayTwo]) {
                    array[indexOfMergedArray]
                            = leftArray[indexOfSubArrayOne];
                    indexOfSubArrayOne++;
            }
            else {
                    array[indexOfMergedArray]
                            = rightArray[indexOfSubArrayTwo];
                    indexOfSubArrayTwo++;
            }
            indexOfMergedArray++;
    }
```

```
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray]
            = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }


    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
    delete[] leftArray;
    delete[] rightArray;
}


void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return;

    int mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}


void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

int main()
{
    int n;
    cout<<"Enter the number of elements:";
```

```
        cin>>n;
        int arr[n];
    clock_t start,end;
        //cout<<"Enter the array:";
        for(int i=0;i<n;i++){
        //cin>>arr[i];
        arr[i]=rand()%100;}
        int arr_size = sizeof(arr) / sizeof(arr[0]);
    cout << "Given array is \n";
    printArray(arr, arr_size);
        start=clock();
    mergeSort(arr, 0, arr_size - 1);
        end=clock();
    cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    cout<<endl<<"Time taken is:"<<(double)(end-start)/CLOCKS_PER_SEC<<endl;
    return 0;
}
```

Output

```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/ADALab_CS259$ g++ mergesort.cpp
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/ADALab_CS259$ ./a.out
Enter the number of elements:500
Given array is
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70 13 26 91 80 56 73 62 70 96 81 5 25 84 27 36 5 46 29 13
 57 24 95 82 45 14 67 34 64 43 50 87 8 76 78 88 84 3 51 54 99 32 60 76 68 39 12 26 86 94 39 95 70 34 78 67 1 97 2 17 92 52 56 1 80 86 41 65 89 44 19 40 29 31 17 97 71 81 75 9 27 67 56 97 53 86 65 6 83 19
 24 28 71 32 29 3 19 70 68 8 15 40 49 96 23 18 45 46 51 21 55 79 88 64 28 41 50 93 0 34 64 24 14 87 56 43 91 27 65 59 36 32 51 37 28 75 7 74 21 58 95 29 37 35 93 18 28 43 11 28 29 76 4 43 63 13 38 6 40 4
 18 28 88 69 17 17 96 24 43 70 83 90 99 72 25 44 90 5 39 54 86 69 82 42 64 97 7 55 4 48 11 22 28 99 43 46 68 40 22 11 10 5 1 61 30 78 5 20 36 44 26 22 65 8 16 82 58 24 37 62 24 0 36 52 99 79 50 68 71 73
31 81 30 33 94 60 63 99 81 99 96 59 73 13 68 90 95 26 66 84 40 90 84 76 42 36 7 45 56 79 18 87 12 48 72 59 9 36 10 42 87 6 1 13 72 21 55 19 99 21 4 39 11 40 67 5 28 27 50 84 58 20 24 22 69 96 81 30 84 92
 72 72 50 25 85 22 99 40 42 98 13 98 90 24 90 9 81 19 36 32 55 94 4 79 69 73 76 50 55 60 42 79 84 93 5 21 67 4 13 61 54 26 59 44 2 2 6 84 21 42 68 28 89 72 8 58 98 36 8 53 48 3 33 33 48 90 54 67 46 68 29
 0 46 88 97 49 90 3 33 63 97 53 92 86 25 52 96 75 88 57 29 36 60 14 21 60 4 28 27 50 48 56 2 94 97 99 43 39 2 28 3 0 81 47 38 59 51 35 34 39 92 15 27 4 29 49 64 85 29 43 35 77 0 38 71 49 89 67 88 92 95 4
3 44 29 90 82 40 41 69 26 32

Sorted array is
0 0 0 0 0 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 6 6 6 6 7 7 7 8 8 8 8 8 9 9 9 10 10 11 11 11 11 11 11 12 12 13 13 13 13 13 13 13 14 14 14 15 15 15 15 16 17 17 17 17 18 18 18 18 19 1
9 19 19 19 19 20 20 21 21 21 21 21 21 21 21 21 22 22 22 22 22 22 23 23 24 24 24 24 24 24 24 24 25 25 25 25 26 26 26 26 26 26 26 26 27 27 27 27 27 27 27 28 28 28 28 28 28 28 28 28 28 28 29 29 29 29 29
29 29 29 29 29 29 29 29 29 30 30 30 30 31 31 32 32 32 32 32 33 33 33 33 34 34 34 34 35 35 35 35 35 36 36 36 36 36 36 36 36 36 37 37 37 37 38 38 38 39 39 39 39 39 39 40 40 40 40 40 40 40 40 41 41 41 42
 42 42 42 42 42 43 43 43 43 43 43 43 43 43 43 44 44 44 44 44 45 45 45 46 46 46 46 47 48 48 48 48 48 49 49 49 49 49 50 50 50 50 50 50 51 51 51 51 52 52 52 53 53 53 54 54 54 54 55 55 55 55 55 55 56 56 5
6 56 56 56 56 57 57 58 58 58 58 58 59 59 59 59 59 59 60 60 60 60 60 60 61 61 62 62 62 62 63 63 63 63 64 64 64 64 64 65 65 65 65 66 67 67 67 67 67 67 67 67 67 67 68 68 68 68 68 68 68 68 69 69 69 69 69 69 70
70 70 70 70 71 71 71 71 72 72 72 72 72 72 72 73 73 73 73 73 74 75 75 75 76 76 76 76 77 77 78 78 78 79 79 79 79 79 80 80 81 81 81 81 81 81 81 82 82 82 82 82 83 83 83 84 84 84 84 84 84 84 84 84 85 85 86
 86 86 86 86 86 87 87 87 87 88 88 88 88 88 88 88 89 89 89 90 90 90 90 90 90 90 90 90 90 91 91 92 92 92 92 92 92 93 93 93 93 93 94 94 94 94 95 95 95 95 95 96 96 96 96 96 96 97 97 97 97 97 97 97 98 98 98 9
8 99 99 99 99 99 99 99 99 99 99

Time taken is:0.000237
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/ADALab_CS259$
```

```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/ADALab_CS259$ ./a.out
Enter the number of elements:6
Given array is
83 86 77 15 93 35

Sorted array is
15 35 77 83 86 93

Time taken is:1.6e-05
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/ADALab_CS259$
```

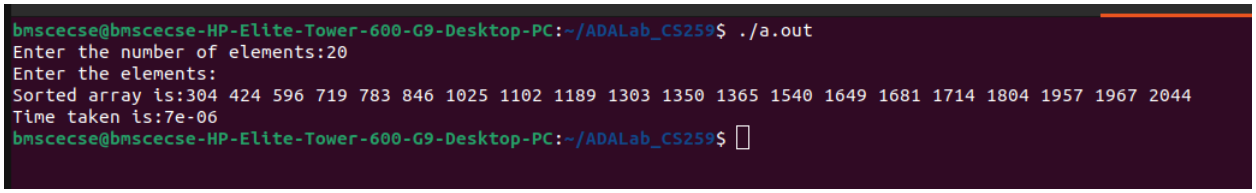
Laboratory Program 5

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```
#include<bits/stdc++.h>
#include<time.h>
using namespace std;
int partition(int arr[],int low,int high)
{
  int p=arr[high];
  int i=low-1;
  for(int j=low;j<=high-1;j++)
  {
        if(arr[j]<p)
        {
        i++;
        swap(arr[i],arr[j]);
        }
  }
  swap(arr[i+1],arr[high]);
  return i+1;
}
void quicksort(int arr[],int low,int high)
{
  if(low<high)
  {
        int pi=partition(arr,low,high);
        quicksort(arr,low,pi-1);
        quicksort(arr,pi+1,high);
  }
}
int main()
{
  clock_t start,end;
  int n;
  cout<<"Enter the number of elements:";
  cin>>n;
  int arr[n];
  cout<<"Enter the elements:";
  for(int i=0;i<n;i++)
  arr[i]=rand()/1000000;
  start=clock();
  quicksort(arr,0,n-1);
```

```
    end=clock();
    cout<<endl<<"Sorted array is:";
    for(int i=0;i<n;i++)
    cout<<arr[i]<<" ";
    cout<<"\nTime taken is:"<<(double)(end-start)/CLOCKS_PER_SEC<<endl;
    return 0;
}
```

Output



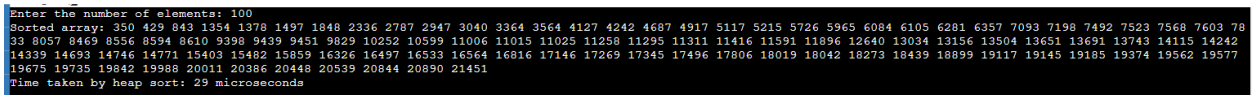
Laboratory Program 6

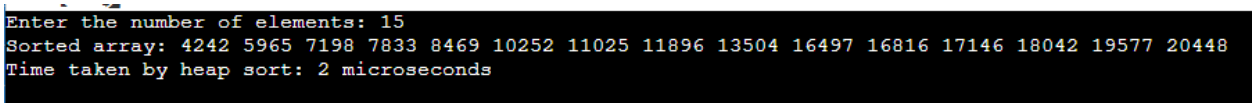
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;
using namespace std::chrono;

// Function to heapify a subtree rooted at 'root'
// n is the size of the heap
void heapify(vector<int>& arr, int n, int root) {
    int largest = root;  // Initialize largest as root
    int left = 2 * root + 1;  // Left child
    int right = 2 * root + 2;  // Right child

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != root) {
        swap(arr[root], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Function to perform heap sort
void heapSort(vector<int>& arr) {
    int n = arr.size();

    // Build a max heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Extract elements from the heap one by one
    for (int i = n - 1; i >= 0; i--) {
        // Move the current root to the end
```

```
        swap(arr[0], arr[i]);

        // Call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

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

    vector<int> arr(n);

    for (int i = 0; i < n; i++)
        arr[i]=rand()/100000;

    // Measure the time taken
    auto start = high_resolution_clock::now();

    heapSort(arr);

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

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

    cout << "Time taken by heap sort: " << duration.count() << " microseconds" << endl;

    return 0;
}
```

Output


Enter the number of elements: 100
Sorted array: 350 429 843 1354 1378 1497 1848 2336 2787 2947 3040 3364 3564 4127 4242 4687 4917 5117 5215 5726 5965 6084 6105 6281 6357 7093 7198 7492 7523 7568 7603 78
33 8057 8469 8556 8594 8610 9398 9439 9451 9829 10252 10599 11006 11015 11025 11258 11295 11311 11416 11591 11896 12640 13034 13156 13504 13651 13691 13743 14115 14242
14339 14693 14746 14771 15403 15482 15859 16326 16497 16533 16564 16816 17146 17269 17345 17496 17806 18019 18042 18273 18439 18899 19117 19145 19185 19374 19562 19577
19675 19735 19842 19988 20011 20386 20448 20539 20844 20890 21451
Time taken by heap sort: 29 microseconds


Enter the number of elements: 15
Sorted array: 4242 5965 7198 7833 8469 10252 11025 11896 13504 16497 16816 17146 18042 19577 20448
Time taken by heap sort: 2 microseconds

Laboratory Program 7

Implement 0/1 Knapsack problem using dynamic programming.

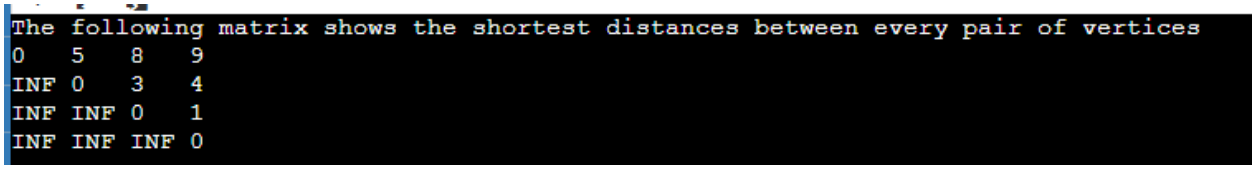
Laboratory Program 8

Implement All Pair Shortest paths problem using Floyd's algorithm

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

// Number of vertices in the graph
#define V 4
#define INF 99999
void printSolution(int dist[][V]);

void floydWarshall(int dist[][V])
{

    int i, j, k;
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][j] > (dist[i][k] + dist[k][j])
                    && (dist[k][j] != INF
                        && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}

void printSolution(int dist[][V])
{
    cout << "The following matrix shows the shortest "
            "distances"
            " between every pair of vertices \n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dist[i][j] << "   ";
```

```
        }
        cout << endl;
    }
}
int main()
{
    int graph[V][V] = { { 0, 5, INF, 10 },
                        { INF, 0, 3, INF },
                        { INF, INF, 0, 1 },
                        { INF, INF, INF, 0 } };

    floydWarshall(graph);
    return 0;
}
```

Output



Laboratory Program 9

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

Laboratory Program 10

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Laboratory Program 11

Implement "N-Queens Problem" using Backtracking.