**Date:** 24/1/2024
**Roll No. and Name:** 22BCE510 (Aarshit Jolapara)
**Course Code and Name:** 2CSDE56 – Graph Theory

**Practical No.:** 3
**AIM:** Write a Program to Use Havel-Hakimi theorem and check whether the given degree sequence is graphical or not.

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

vector<vector<int>> graph;
void printGraph(vector<vector<int>> graph){
    for(int i=0;i<graph.size();i++){
        cout << i << " ::  ";
        for(int j=0;j<graph[i].size();j++){
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }
}

bool isSimple(vector<pair<int, int>> deg){
    int n = deg.size();

    graph.clear();
    graph.resize(n);

    // Check if sum of all degrees is even
    int sum = 0;
    for(int i=0;i<n;i++){
        sum += deg[i].first;
    }
    if(sum%2 != 0) return false;

    for(int i=0;i<n;i++){
        // Check if degree is greater than or equal to n-i
        if(deg[i].first >= n-i) return false;
```

```
        // if degree is negative
        if(deg[i].first < 0) return false;
        // if degree is 0, then all the remaining degrees should be 0
        if(deg[i].first == 0) break;

        for(int j=i+1;j<=i+deg[i].first;j++){
            deg[j].first--;
            // if degree becomes negative after decrementing
            if(deg[j].first < 0) return false;

            // add edge between i and j
            graph[deg[i].second].push_back(deg[j].second);
            graph[deg[j].second].push_back(deg[i].second);

        }

        // rearrange the sequence
        sort(deg.begin()+i+1,deg.end(), greater<pair<int,int>>());
    }

    return true;
}

int main() {

    int n;
    cin >> n;

    while(n--){
        int size; // size of degree sequence
        cin >> size;

        vector<pair<int,int>> nodesWithDegree(size);
        for(int i=0;i<size;i++){
            int tmp;
            cin >> tmp;
            nodesWithDegree[i] = {tmp, i};
        }
```

```
        sort(nodesWithDegree.begin(), nodesWithDegree.end(),
greater<pair<int,int>>());

        if(isSimple(nodesWithDegree)) {
            cout << "Graph is simple" << endl;
            printGraph(graph);
        } else cout << "Graph is not simple" << endl;

        cout << endl;
    }
    return 0;
}
```

**Input:**

```
5
4
3 3 2 1
8
5 4 3 2 2 2 1 1
5
3 3 2 2 1
6
2 3 5 3 3 2
10
9 8 7 6 5 4 3 2 1 1
```

**Output:**

```
5
4
3 3 2 1
Graph is not simple

8
5 4 3 2 2 2 1 1
Graph is simple
0 ::  1 2 5 4 3
1 ::  0 2 7 6
2 ::  0 1 3
3 ::  0 2
4 ::  0 5
5 ::  0 4
6 ::  1
7 ::  1

5
3 3 2 2 1
Graph is not simple

6
2 3 5 3 3 2
Graph is simple
0 ::  2 1
1 ::  2 4 0
2 ::  4 3 1 5 0
3 ::  2 4 5
4 ::  2 3 1
5 ::  2 3

10
9 8 7 6 5 4 3 2 1 1
Graph is not simple
```