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

int maxMeetings(int start[], int end[], int n)
   {

       vector<pair<int,int>>vp;
       for ( int  i = 0; i < n ; i++ ){
           vp.push_back({ end[i], start[i] });
       }

       sort( vp.begin(), vp.end());
       int ans = 1;
       int j = 0;
       for ( int i = 1; i < n ; i ++ ){
           if ( vp[j].first < vp[i].second ){
               ans ++;
               j = i;
           }

       }
       return ans ;
   }
   int main(){
       int n;
       cin >> n;
       int start[n];
       int end[n];
       for ( int  i= 0; i< n; i++ ){
           cin >> start[i];
       }
       for ( int  i= 0; i< n; i++ ){
           cin >> end[i];
       }
       clock_t starter, ender;


       starter = clock();
       int ans =  maxMeetings(start,end,6);
       ender = clock();


       double time_taken = double(ender - starter) / double(CLOCKS_PER_SEC);
       cout << "Time taken by program is : " << fixed
        << time_taken << setprecision(5);
        cout << " sec " << endl;

       // O(n*logn)
```

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

class Node{
    public:
    int data;
    char ch;
    Node* left;
    Node* right;
    // we cannot give argument char ch to constructor otherwise it will be
using char ch of defined in Node not arguments'
    Node ( char c, int d ){
        data = d;
        left = nullptr;
        ch = c;
        right = nullptr;
    }
};
class cmp {
    public:
    bool operator()(Node*a, Node*b ){
        return a->data > b->data;
    }
};

class huffman {
    public:


    void traverse( Node* root, vector<pair<char,string>> &ans, string temp ){

        if ( root -> left == nullptr && root -> right == nullptr ){
            ans.push_back ( { root->ch,temp } );
            return;
        }

        traverse ( root -> left, ans, temp+ '0' );
        traverse ( root-> right , ans, temp + '1' );
    }


    vector< pair < char, string > > huffman_encoding ( string s, vector<int>
freq, int N ){

        priority_queue< Node*, vector<Node*> , cmp > pq;

        for ( int  i = 0 ;i < N ; i++ ){
            Node* newNode = new Node(s[i],freq[i]);
```

```
            pq.push ( newNode );
        }

        while ( pq.size() > 1 ){
            Node* l = pq.top();
            pq.pop();
            Node* r = pq.top();
            pq.pop();
            Node* newNode = new Node('$', l->data + r->data );
            newNode->left = l;
            newNode->right = r;
            pq.push(newNode);
        }

        Node* root = pq.top();
        vector<pair<char,string>> ans;
        string temp;
        traverse( root, ans, temp );
        return ans;

    }
};



int main(){
    int N = 5;
    string s = "qptad";
    vector<int> freq = { 3,23,30,12,18 };
    huffman obj;
     clock_t starter, ender;

    starter = clock();
    vector< pair < char, string > > ans = obj.huffman_encoding ( s,freq , N );
    ender = clock();

    double time_taken = double(ender - starter) / double(CLOCKS_PER_SEC);
    cout << "Time taken by program is : " << fixed
         << time_taken << setprecision(5);
    cout << " sec " << endl;


    for ( int i= 0; i< N; i++ ){
        cout << ans[i].first << " " << ans[i].second << endl;
    }
    // O(n*logn)

}
```

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

class DSU
{
    vector<int> size, parent;

public:
    // constructor
    DSU(int n)
    {
        size.resize(n + 1, 1);
        parent.resize(n + 1);
        for (int i = 0; i <= n; i++)
        {
            parent[i] = i;
        }
    }

    int find_ulp(int node)
    { // O(log(n))
        if (parent[node] == node)
            return node;
        else
            return parent[node] = find_ulp(parent[node]);
        // path compression as well as finding the parent
    }

    void unionBySize(int u, int v)
    {
        int ulp_u = find_ulp(u);
        int ulp_v = find_ulp(v);

        if (ulp_u == ulp_v)
            return;

        if (size[ulp_u] >= size[ulp_v])
        {
            size[ulp_u] += size[ulp_v];
            parent[ulp_v] = ulp_u;
        }

        else
        {
            size[ulp_v] += size[ulp_u];
            parent[ulp_u] = ulp_v;
        }
    }
```

```
};

int kruskalMST(int n, vector<vector<int>> &edges, vector<pair<int, int>> &mst)
{

    vector<pair<int, pair<int, int>>> edge;
    for (int i = 0; i < edges.size(); i++)
    {
        edge.push_back({edges[i][2], {edges[i][0], edges[i][1]}});
    }

    sort(edge.begin(), edge.end());
    int mstWt = 0;
    DSU ds(n);

    for (auto &it : edge)
    {
        int wt = it.first;
        int u = it.second.first;
        int v = it.second.second;

        if (ds.find_ulp(u) != ds.find_ulp(v))
        {
            mstWt += wt;
            mst.push_back({u, v});
            ds.unionBySize(u, v);
        }
    }
    return mstWt;
}

void pairprinter(vector<pair<int, int>> mst)
{
    for (int i = 0; i < mst.size(); i++)
    {
        cout << mst[i].first << " " << mst[i].second << endl;
    }
    cout << endl;
}
int main()
{

    int m;
    cout << "enter the num of edges : ";
    cin >> m;
    int n;
    cout << "enter the num of nodes : ";
    cin >> n;
```

```
    vector<vector<int>> edges;
    vector<pair<int, int>> mst;

    for (int i = 0; i < m; i++)
    {
        // vector<int>v;
        int x, y, z;
        cin >> x >> y >> z;
        edges.push_back({x, y, z});
    }
    clock_t starter, ender;

    starter = clock();
    cout << "minimum reparation cost: " << kruskalMST(n, edges, mst) << endl;
    ender = clock();

    double time_taken = double(ender - starter) / double(CLOCKS_PER_SEC);
    cout << "Time taken by program is : " << fixed
         << time_taken << setprecision(5);
    cout << " sec " << endl;

    cout << "we need to repair following roads" << endl;
    pairprinter(mst);

    // O(E*logE + E*logV)
}
```