**1)**

1)

//1.1(O(n^2))(выбором)

#include <iostream>

using namespace std;

void print\_array(int arr[], int n)

{

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

{

cout << arr[i] << " ";

}

cout << endl;

}

void selection\_sort(int arr[], int n, bool ascending)

{

int comp\_count = 0;

int swap\_count = 0;

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

{

int min\_idx = i;

for (int j = i + 1; j < n; j++)

{

comp\_count++;

if ((ascending && arr[j] < arr[min\_idx]) || (!ascending && arr[j] > arr[min\_idx]))

{

min\_idx = j;

}

}

if (min\_idx != i)

{

swap(arr[i], arr[min\_idx]);

swap\_count++;

}

}

cout << "\nKol-vo sravneniy: " << comp\_count << endl;

cout << "Kol-vo obmenov: " << swap\_count << endl;

}

int main()

{

int n;

cout << "Kol-vo elementov: ";

cin >> n;

int arr[n];

cout << "Massiv: ";

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

{

cin >> arr[i];

}

cout << "Pervishniy massiv: ";

print\_array(arr, n);

cout << "Po vozrastaniy: ";

selection\_sort(arr, n, true);

print\_array(arr, n);

cout << "Po ubuvaniy: ";

selection\_sort(arr, n, false);

print\_array(arr, n);

return 0;

}

2)

//1.2(O(n + k))(подсчётом)

#include <iostream>

using namespace std;

void count\_sort\_asc(int arr[], int n)

{

int max\_val = arr[0];

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

{

if (arr[i] > max\_val)

{

max\_val = arr[i];

}

}

int count[max\_val+1] = {0};

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

{

count[arr[i]]++;

}

for (int i = 1; i <= max\_val; i++)

{

count[i] += count[i-1];

}

int sorted\_arr[n];

for (int i = n-1; i >= 0; i--)

{

sorted\_arr[--count[arr[i]]] = arr[i];

}

cout << "Po vozrastaniy: ";

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

{

cout << sorted\_arr[i] << " ";

}

cout << endl;

}

void count\_sort\_desc(int arr[], int n)

{

int max\_val = arr[0];

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

{

if (arr[i] > max\_val)

{

max\_val = arr[i];

}

}

int count[max\_val+1] = {0};

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

{

count[arr[i]]++;

}

for (int i = max\_val-1; i >= 0; i--)

{

count[i] += count[i+1];

}

int sorted\_arr[n];

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

{

sorted\_arr[--count[arr[i]]] = arr[i];

}

cout << "Po ubuvaniy: ";

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

{

cout << sorted\_arr[i] << " ";

}

cout << endl;

}

int main()

{

int n;

cout << "Kol-vo elementov v massive: ";

cin >> n;

int arr[n];

cout << "Massiv: ";

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

{

cin >> arr[i];

}

cout << "Pervichniy massiv: ";

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

{

cout << arr[i] << " ";

}

cout << endl;

count\_sort\_asc(arr, n);

count\_sort\_desc(arr, n);

return 0;

}

3)

a)Нерекурсивный:

#include <iostream>  
#include <vector>  
using namespace std;  
  
void bucketSort(vector<int>& nums, bool ascending) {  
    int n = nums.size();  
    vector<vector<int>> buckets(n);  
    for (int i = 0; i < n; i++) {  
        int index = n \* nums[i];  
        buckets[index].push\_back(nums[i]);  
    }  
    for (int i = 0; i < n; i++) {  
        sort(buckets[i].begin(), buckets[i].end());  
    }  
    int index = 0;  
    if (ascending) {  
        for (int i = 0; i < n; i++) {  
            for (int j = 0; j < buckets[i].size(); j++) {  
                nums[index++] = buckets[i][j];  
            }  
        }  
    } else {  
        for (int i = n-1; i >= 0; i--) {  
            for (int j = 0; j < buckets[i].size(); j++) {  
                nums[index++] = buckets[i][j];  
            }  
        }  
    }  
}  
  
int main() {  
    int n;  
    cout << "Enter the size of the array: ";  
    cin >> n;  
    vector<int> nums(n);  
    cout << "Enter the elements of the array: ";  
    for (int i = 0; i < n; i++) {  
        cin >> nums[i];  
    }  
    cout << "Original array: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    bucketSort(nums, true);  
    cout << "Sorted array in ascending order: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    bucketSort(nums, false);  
    cout << "Sorted array in descending order: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    return 0;  
}

б)Рекурсивный.

#include <iostream>  
#include <vector>  
using namespace std;  
  
void insertionSort(vector<int>& nums, int n, bool ascending) {  
    if (n <= 1) {  
        return;  
    }  
    insertionSort(nums, n-1, ascending);  
    int last = nums[n-1];  
    int j = n-2;  
    while (j >= 0 && ((ascending && nums[j] > last) || (!ascending && nums[j] < last))) {  
        nums[j+1] = nums[j];  
        j--;  
    }  
    nums[j+1] = last;  
}  
  
void bucketSort(vector<int>& nums, bool ascending) {  
    int n = nums.size();  
    vector<vector<int>> buckets(n);  
    for (int i = 0; i < n; i++) {  
        int index = n \* nums[i];  
        buckets[index].push\_back(nums[i]);  
    }  
    for (int i = 0; i < n; i++) {  
        insertionSort(buckets[i], buckets[i].size(), ascending);  
    }  
    int index = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < buckets[i].size(); j++) {  
            nums[index++] = buckets[i][j];  
        }  
    }  
}  
  
int main() {  
    int n;  
    cout << "Enter the size of the array: ";  
    cin >> n;  
    vector<int> nums(n);  
    cout << "Enter the elements of the array: ";  
    for (int i = 0; i < n; i++) {  
        cin >> nums[i];  
    }  
    cout << "Original array: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    bucketSort(nums, true);  
    cout << "Sorted array in ascending order: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    bucketSort(nums, false);  
    cout << "Sorted array in descending order: ";  
    for (int i = 0; i < n; i++) {  
        cout << nums[i] << " ";  
    }  
    cout << endl;  
    return 0;  
}

4)

//1.4(O(n \* k))(поразрядная)

#include <iostream>

#include <vector>

#include <math.h>

using namespace std;

void radixSort(vector<int>& nums, bool ascending)

{

int n = nums.size();

int maxDigit = 0;

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

{

int digit = 0;

int num = nums[i];

while (num > 0)

{

digit++;

num /= 10;

}

maxDigit = max(maxDigit, digit);

}

vector<vector<int>> buckets(10);

for (int digit = 1; digit <= maxDigit; digit++)

{

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

{

int num = nums[i];

int index = num % (int)pow(10, digit) / (int)pow(10, digit-1);

buckets[index].push\_back(num);

}

int index = 0;

if (ascending)

{

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < buckets[i].size(); j++)

{

nums[index++] = buckets[i][j];

}

buckets[i].clear();

}

}

else

{

for (int i = 9; i >= 0; i--)

{

for (int j = 0; j < buckets[i].size(); j++)

{

nums[index++] = buckets[i][j];

}

buckets[i].clear();

}

}

}

}

int main()

{

int n;

cout << "Kol-vo elementov v massive: ";

cin >> n;

vector<int> nums(n);

cout << "Massiv: ";

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

{

cin >> nums[i];

}

cout << "Pervishniy massiv: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

radixSort(nums, true);

cout << "Po vozrastaniy: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

radixSort(nums, false);

cout << "Po ubuvaniy: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

return 0;

}

5)

//1.5(O(n \* log n))(пирамидальная)

#include <iostream>

#include <vector>

using namespace std;

void heapify(vector<int>& nums, int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && nums[left] > nums[largest])

{

largest = left;

}

if (right < n && nums[right] > nums[largest])

{

largest = right;

}

if (largest != i)

{

swap(nums[i], nums[largest]);

heapify(nums, n, largest);

}

}

void heapSort(vector<int>& nums)

{

int n = nums.size();

for (int i = n/2 - 1; i >= 0; i--)

{

heapify(nums, n, i);

}

for (int i = n-1; i >= 0; i--)

{

swap(nums[0], nums[i]);

heapify(nums, i, 0);

}

}

int main()

{

int n;

cout << "Kol-vo elementov v massive: ";

cin >> n;

vector<int> nums(n);

cout << "Massiv: ";

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

{

cin >> nums[i];

}

cout << "Pervichniy massiv: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

for (int i = n/2 - 1; i >= 0; i--)

{

heapify(nums, n, i);

}

cout << "Po ubuvaniy: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

heapSort(nums);

cout << "Po vozrastaniy: ";

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

{

cout << nums[i] << " ";

}

cout << endl;

return 0;

}

**2)**

1)

#include <iostream>  
#include <vector>  
using namespace std;  
  
const int p = 31; // простое число для хэширования  
  
bool isPalindrome(string s) {  
    int n = s.length();  
    vector<long long> h(n+1), r(n+1); // векторы для хранения прямого и обратного хэшей  
    h[0] = r[n] = 0; // начальные значения хэшей  
    for (int i = 1; i <= n; i++) {  
        h[i] = (h[i-1]\*p + s[i-1]); // вычисляем прямой хэш  
        r[n-i] = (r[n-i+1]\*p + s[n-i]); // вычисляем обратный хэш  
    }  
    for (int i = 0; i < n; i++) {  
        if (h[i+1] != r[i]) { // если хэши не равны, то строка не является палиндромом  
            return false;  
        }  
    }  
    return true; // строка является палиндромом  
}  
  
int main() {  
    string s;  
    cout << "Enter a string: ";  
    cin >> s;  
    if (isPalindrome(s)) {  
        cout << "The string is a palindrome." << endl;  
    } else {  
        cout << "The string is not a palindrome." << endl;  
    }  
    return 0;  
}

2)

#include <iostream>  
#include <vector>  
using namespace std;  
  
const int p = 31; // простое число для хэширования  
  
bool isPalindrome(string s) {  
    int n = s.length();  
    vector<long long> h(n+1), r(n+1); // векторы для хранения прямого и обратного хэшей  
    h[0] = r[n] = 0; // начальные значения хэшей  
    for (int i = 1; i <= n; i++) {  
        h[i] = (h[i-1]\*p + s[i-1]); // вычисляем прямой хэш  
        r[n-i] = (r[n-i+1]\*p + s[n-i]); // вычисляем обратный хэш  
    }  
    for (int i = 0; i < n; i++) {  
        if (h[i+1] != r[i]) { // если хэши не равны, то строка не является палиндромом  
            return false;  
        }  
    }  
    return true; // строка является палиндромом  
}  
  
int main() {  
    string s;  
    cout << "Enter a string: ";  
    cin >> s;  
    if (isPalindrome(s)) {  
        cout << "The string is a palindrome." << endl;  
    } else {  
        cout << "The string is not a palindrome." << endl;  
    }  
    return 0;  
}

3)

> ChatGPT & Midjourney | New bot: Для реализации модуля для работы с АВЛ-деревом необходимо создать класс AVLTree, который будет содержать все необходимые методы и поля для работы с деревом.  
  
Код на C++:  
  
#include <iostream>  
using namespace std;  
  
struct Node { // структура узла дерева  
int key;  
int height;  
Node\* left;  
Node\* right;  
Node(int k) {  
key = k;  
height = 1;  
left = right = nullptr;  
}  
};  
  
class AVLTree {  
private:  
Node\* root; // корень дерева  
  
int height(Node\* node) { // вычисление высоты поддерева  
if (node == nullptr) {  
return 0;  
}  
return node->height;  
}  
  
int balanceFactor(Node\* node) { // вычисление баланса поддерева  
if (node == nullptr) {  
return 0;  
}  
return height(node->right) - height(node->left);  
}  
  
void fixHeight(Node\* node) { // обновление высоты узла  
node->height = max(height(node->left), height(node->right)) + 1;  
}  
  
Node\* rotateRight(Node\* node) { // правый поворот вокруг node  
Node\* q = node->left;  
node->left = q->right;  
q->right = node;  
fixHeight(node);  
fixHeight(q);  
return q;  
}  
  
Node\* rotateLeft(Node\* node) { // левый поворот вокруг node  
Node\* p = node->right;  
node->right = p->left;  
p->left = node;  
fixHeight(node);  
fixHeight(p);  
return p;  
}  
  
Node\* balance(Node\* node) { // балансировка узла  
fixHeight(node);  
if (balanceFactor(node) == 2) {  
if (balanceFactor(node->right) < 0) {  
node->right = rotateRight(node->right);  
}  
return rotateLeft(node);  
}  
if (balanceFactor(node) == -2) {  
if (balanceFactor(node->left) > 0) {  
node->left = rotateLeft(node->left);  
}  
return rotateRight(node);  
}  
return node;  
}  
  
Node\* insert(Node\* node, int key) { // вставка ключа в дерево  
if (node == nullptr) {  
return new Node(key);  
}  
if (key < node->key) {  
node->left = insert(node->left, key);  
} else {  
node->right = insert(node->right, key);  
}  
return balance(node);  
}  
  
Node\* findMin(Node\* node) { // поиск минимального ключа в дереве  
if (node == nullptr || node->left == nullptr) {  
return node;  
}  
return findMin(node->left);  
}  
  
Node\* removeMin(Node\* node) { // удаление узла с минимальным ключом из дерева  
if (node->left == nullptr) {  
return node->right;  
}  
node->left = removeMin(node->left);  
return balance(node);  
}  
  
Node\* remove(Node\* node, int key) { // удаление ключа из дерева  
if (node == nullptr) {  
return nullptr;  
}  
if (key < node->key) {  
node->left = remove(node->left, key);  
} else if (key > node->key) {  
node->right = remove(node->right, key);  
} else {  
Node\* q = node->left;  
Node\* r = node->right;  
delete node;  
if (r == nullptr) {  
return q;  
}  
Node\* min = findMin(r);  
min->right = removeMin(r);  
min->left = q;  
return balance(min);  
}  
return balance(node);  
}  
  
public:  
AVLTree() { // конструктор  
root = nullptr;  
}  
  
~AVLTree() { // деструктор  
clear(root);  
}  
  
void clear(Node\* node) { // очистка дерева  
if (node != nullptr) {  
clear(node->left);  
clear(node->right);  
delete node;  
}  
}  
  
int size(Node\* node) { // вычисление размера дерева  
if (node == nullptr) {  
return 0;  
}  
return 1 + size(node->left) + size(node->right);  
}  
  
void insert(int key) { // вставка ключа в дерево  
r > ChatGPT & Midjourney | New bot: oot = insert(root, key);  
}  
  
void remove(int key) { // удаление ключа из дерева  
root = remove(root, key);  
}  
  
bool find(int key) { // поиск ключа в дереве  
Node\* node = root;  
while (node != nullptr) {  
if (key == node->key) {  
return true;  
} else if (key < node->key) {  
node = node->left;  
} else {  
node = node->right;  
}  
}  
return false;  
}  
  
void balance() { // балансировка дерева  
root = balance(root);  
}  
  
int size() { // вычисление размера дерева  
return size(root);  
}  
};  
  
int main() {  
AVLTree tree;  
tree.insert(5);  
tree.insert(3);  
tree.insert(7);  
tree.insert(2);  
tree.insert(4);  
tree.insert(6);  
tree.insert(8);  
cout << "Size of the tree: " << tree.size() << endl;  
cout << "Is 4 in the tree? " << (tree.find(4) ? "Yes" : "No") << endl;  
cout << "Is 9 in the tree? " << (tree.find(9) ? "Yes" : "No") << endl;  
tree.remove(3);  
cout << "Size of the tree after removing 3: " << tree.size() << endl;  
tree.balance();  
cout << "Size of the tree after balancing: " << tree.size() << endl;  
return 0;  
}

4)

#include <iostream>  
#include <vector>  
using namespace std;  
  
const int p = 31; // простое число для хэширования  
  
bool isAnagram(string s, string t) {  
    int n = s.length(), m = t.length();  
    if (n < m) { // если длина s меньше длины t, то анаграммы не может быть  
        return false;  
    }  
    vector<int> cnt(26, 0); // массив для хранения частот символов в строке t  
    for (char c : t) {  
        cnt[c-'a']++;  
    }  
    vector<long long> h(n+1), ht(m+1); // векторы для хранения хэшей строк s и t  
    h[0] = ht[0] = 0; // начальные значения хэшей  
    for (int i = 1; i <= n; i++) {  
        h[i] = (h[i-1]\*p + s[i-1]); // вычисляем хэш строки s  
        if (i <= m) {  
            ht[i] = (ht[i-1]\*p + t[i-1]); // вычисляем хэш строки t  
        }  
    }  
    for (int i = 0; i <= n-m; i++) { // перебираем все возможные подстроки s длины t  
        long long hash\_s = h[i+m] - h[i]\*pow(p, m); // вычисляем хэш текущей подстроки s  
        if (hash\_s == ht[m] && cnt == getFrequency(s.substr(i, m))) { // если хэши равны и символы в текущей подстроке s совпадают с частотами символов в строке t, то это анаграмма  
            return true;  
        }  
    }  
    return false; // анаграмма не найдена  
}  
  
vector<int> getFrequency(string s) { // функция для вычисления частот символов в строке  
    vector<int> cnt(26, 0);  
    for (char c : s) {  
        cnt[c-'a']++;  
    }  
    return cnt;  
}  
  
int main() {  
    string s, t;  
    cout << "Enter a string: ";  
    cin >> s;  
    cout << "Enter a substring: ";  
    cin >> t;  
    if (isAnagram(s, t)) {  
        cout << "The first anagram of the substring is: " << s.substr(s.find(t), t.length()) << endl;  
    } else {  
        cout << "No anagram of the substring found in the string." << endl;  
    }  
    return 0;  
}

5)

//2.5

#include <iostream>

#include <vector>

using namespace std;

vector<int> getPrefix(string s)

{ // функция для вычисления префикс-функции

int n = s.length();

vector<int> pi(n, 0);

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

{

int j = pi[i - 1];

while (j > 0 && s[i] != s[j])

{

j = pi[j - 1];

}

if (s[i] == s[j])

{

j++;

}

pi[i] = j;

}

return pi;

}

void replaceSubstring(string& s, string t, string r)

{ // функция для замены подстроки в строке

int n = s.length(), m = t.length();

vector<int> pi = getPrefix(t);

int j = 0;

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

{

while (j > 0 && s[i] != t[j])

{

j = pi[j - 1];

}

if (s[i] == t[j])

{

j++;

}

if (j == m)

{// если найдена подстрока t

s.replace(i - m + 1, m, r);

// заменяем ее на новую подстроку r

return;

}

}

cout << "Error" << endl;

// если подстрока не найдена

}

int main()

{

string s, t, r;

cout << "Stroka: ";

getline(cin, s);

cout << "Zamena v stroke ";

getline(cin, t);

cout << "Zamenit na: ";

getline(cin, r);

replaceSubstring(s, t, r);

cout << "Polushilas stroka: " << s << endl;

return 0;

}

**3)**

1)

#include <iostream>  
#include <vector>  
#include <stack>  
using namespace std;  
  
const int MAXN = 1000; // максимальное количество вершин в графе  
  
vector<int> adj[MAXN]; // список смежности графа  
int degree[MAXN]; // степени вершин  
bool visited[MAXN]; // флаги посещения вершин  
  
void dfs(int v, stack<int>& st) { // обход в глубину  
    visited[v] = true;  
    for (int u : adj[v]) {  
        if (!visited[u]) {  
            dfs(u, st);  
        }  
    }  
    st.push(v);  
}  
  
bool hasEulerCycle(int n) { // проверка наличия эйлерова цикла  
    for (int i = 0; i < n; i++) {  
        if (degree[i] % 2 != 0) {  
            return false;  
        }  
    }  
    return true;  
}  
  
int main() {  
    int n, m; // количество вершин и ребер графа  
    cin >> n >> m;  
    for (int i = 0; i < m; i++) {  
        int u, v;  
        cin >> u >> v;  
        adj[u].push\_back(v);  
        adj[v].push\_back(u);  
        degree[u]++;  
        degree[v]++;  
    }  
    if (!hasEulerCycle(n)) { // проверка наличия эйлерова цикла  
        cout << "No euler cycle\n";  
        return 0;  
    }  
    stack<int> st;  
    dfs(0, st); // обход в глубину из первой вершины  
    while (!st.empty()) { // вывод эйлерова цикла  
        cout << [st.top](http://st.top/)() << " ";  
        st.pop();  
    }  
    cout << endl;  
    return 0;  
}

2)

По рёбрам:

#include &lt;iostream&gt;  
#include &lt;vector&gt;  
  
using namespace std;  
  
vector&lt;vector&lt;int&gt;&gt; adj\_list; // список смежности графа  
vector&lt;int&gt; visited; // массив посещенных вершин  
vector&lt;vector&lt;int&gt;&gt; all\_paths; // массив всех найденных путей  
  
void dfs(int u, int end, vector&lt;int&gt;&amp; path) {  
    visited[u] = 1;  
    path.push\_back(u);  
  
    if (u == end) {  
        all\_paths.push\_back(path);  
    } else {  
        for (int v : adj\_list[u]) {  
            if (!visited[v]) {  
                dfs(v, end, path);  
            }  
        }  
    }  
  
    path.pop\_back();  
    visited[u] = 0;  
}  
  
int main() {  
    int n, m;  
    cin &gt;&gt; n &gt;&gt; m;  
  
    adj\_list.resize(n);  
    visited.resize(n);  
  
    for (int i = 0; i &lt; m; ++i) {  
        int u, v;  
        cin &gt;&gt; u &gt;&gt; v;  
  
        adj\_list[u].push\_back(v);  
        adj\_list[v].push\_back(u);  
    }  
  
    int start, end;  
    cin &gt;&gt; start &gt;&gt; end;  
  
    vector&lt;int&gt; path;  
    dfs(start, end, path);  
  
    cout &lt;&lt; "All paths from " &lt;&lt; start &lt;&lt; " to " &lt;&lt; end &lt;&lt; " without edge intersections:" &lt;&lt; endl;  
  
    for (auto&amp; p : all\_paths) {  
        for (int i = 0; i &lt; p.size(); ++i) {  
            cout &lt;&lt; p[i];  
            if (i &lt; p.size() - 1) {  
                cout &lt;&lt; " -&gt; ";  
            }  
        }  
        cout &lt;&lt; endl;  
    }  
  
    return 0;  
}

3)

#include &lt;iostream&gt;  
#include &lt;vector&gt;  
#include &lt;algorithm&gt;  
  
using namespace std;  
  
struct Edge {  
    int u, v, w;  
  
    bool operator&lt;(const Edge&amp; other) const {  
        return w &lt; other.w;  
    }  
};  
  
vector&lt;int&gt; parent;  
  
int find(int x) {  
    if (parent[x] == x) {  
        return x;  
    }  
  
    return parent[x] = find(parent[x]);  
}  
  
void merge(int x, int y) {  
    parent[find(x)] = find(y);  
}  
  
vector&lt;vector&lt;int&gt;&gt; kruskal(int n, vector&lt;Edge&gt;&amp; edges) {  
    parent.resize(n);  
    for (int i = 0; i &lt; n; ++i) {  
        parent[i] = i;  
    }  
  
    sort(edges.begin(), edges.end());  
  
    vector&lt;vector&lt;int&gt;&gt; mst(n, vector&lt;int&gt;(n));  
  
    for (auto&amp; e : edges) {  
        if (find(e.u) != find(e.v)) {  
            merge(e.u, e.v);  
            mst[e.u][e.v] = mst[e.v][e.u] = 1;  
        }  
    }  
  
    return mst;  
}  
  
int main() {  
    int n;  
    cin &gt;&gt; n;  
  
    vector&lt;Edge&gt; edges;  
    for (int i = 0; i &lt; n; ++i) {  
        for (int j = i + 1; j &lt; n; ++j) {  
            int w;  
            cin &gt;&gt; w;  
  
            edges.push\_back({i, j, w});  
        }  
    }  
  
    auto mst = kruskal(n, edges);  
  
    cout &lt;&lt; "The cheapest road system is:" &lt;&lt; endl;  
    for (int i = 0; i &lt; n; ++i) {  
        for (int j = 0; j &lt; n; ++j) {  
            cout &lt;&lt; mst[i][j] &lt;&lt; " ";  
        }  
        cout &lt;&lt; endl;  
    }  
  
    return 0;  
}

5)

#include &lt;iostream&gt;  
#include &lt;vector&gt;  
  
using namespace std;  
  
vector&lt;vector&lt;int&gt;&gt; adj\_list;  
vector&lt;int&gt; visited;  
  
void dfs(int u, vector&lt;int&gt;&amp; component) {  
    visited[u] = 1;  
    component.push\_back(u);  
  
    for (int v : adj\_list[u]) {  
        if (!visited[v]) {  
            dfs(v, component);  
        }  
    }  
}  
  
vector&lt;vector&lt;int&gt;&gt; find\_connected\_components(int n) {  
    visited.assign(n, 0);  
  
    vector&lt;vector&lt;int&gt;&gt; components;  
    for (int u = 0; u &lt; n; ++u) {  
        if (!visited[u]) {  
            vector&lt;int&gt; component;  
            dfs(u, component);  
            components.push\_back(component);  
        }  
    }  
  
    return components;  
}