**Muffins**



**Tanjid Samiul Abdullah**

**C211060 C211066 C211074**

#include <bits/stdc++.h>

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

// Go to the MingW folder and search "pb\_ds". Go to detail/resize/ go to the different file and rename it. Remove the last digits from it and save

using namespace std;

using namespace \_\_gnu\_pbds;

tree <long long, null\_type, less<long long>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> odrSet ;

**// Geometry**

#define PI acos(-1)

#define sin(a) sin(a\*PI/180)

#define cos(a) cos(a\*PI/180)

#define tan(a) tan(a\*PI/180)

#define sini(a) asin(a)/(PI/180)

#define cosi(a) acos(a)/(PI/180)

#define tani(a) atan(a)/(PI/180)

#define tanii(a,b) atan2(a,b)/(PI/180) // tan(90) = Infinity or a/0

#define t\_area(a, b, c, s) sqrt(s\*(s-a)\*(s-b)\*(s-c)) // s = (a+b+c) / 2.0

#define t\_angle(a, b, c) acos((a\*a+b\*b-c\*c)/(2\*a\*b)) // returns angle C in radian. a, b, c are anti-clockwise formatted and side a and b form angle C

#define make\_radian(x) (x\*PI) / 180

#define make\_degree(x) (x\*180) / PI

#define negmod(ans, x, m) long long y = (-1\*x) % m; if (y == 0) ans = 0; else ans = m - y; // For negative mod only i.e. when x<0. Undefined when m<0

#define dis(x1, y1, x2, y2) sqrt((x1-x2) \* (x1-x2) + (y1-y2) \* (y1-y2))

#define pointPos(x1, y1, x2, y2, x3, y3) ((x2-x1)\*(y3-y1))-((x3-x1)\*(y2-y1)); // Returns NEGATIVE if the Point P3 is on the RIGHT side of the line P1P2, otherwise returns POSITIVE in case of LEFT and ZERO when the point is on the line

#define t\_areaWithPoints(x1, y1, x2, y2, x3, y3) abs(0.5\*(x1\*(y2-y3)+x2\*(y3-y1)+x3\*(y1-y2))); // Returns the area of a triangle formed by P1, p2, p3

const long long mod = 1e9 + 7;

**// Prime Sieve**

long long range = 100; // 99998953 90000000

bool is\_prime[100] = {false}; // 99998958 90000001

vector <long long> primeList;

void sieve() {

is\_prime[0] = true;

is\_prime[1] = true;

for (long long i = 2; i <= range; i++) {

if (is\_prime[i] == false) {

primeList.push\_back(i);

for (long long j = i \* i; j <= range; j += i) {

is\_prime[j] = true;

}

}

}

}

**// Prime Factorization (Use the code inside the main function) O (log n)**

void primeFact() {

long long n;

sieve();

for (long long i = 0; i < primeList.size(); i++) {

if (primeList[i] \* primeList[i] > n) {

break;

}

while (n % primeList[i] == 0) {

n /= primeList[i];

cout << primeList[i] << " ";

}

}

// In case the input itself a prime number, because the number itself doesn't exist in i <= sqrt(n)

if (n > 1) {

cout << n << endl ;

}

}

**// Prime or Not O(sqrt(N))**

bool isPrime (long long n) {

if (n == 1)

return false;

for (long long i = 2; i <= n / i; i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

**// Nth prime number**

/\*

bool isPrime[90000005] ;

long long mx = 90000000 ;

vector <long long> primeList ;

void sieve() {

for (int i = 2; i <= mx / i; i++) {

if (isPrime[i] == false) {

for (int j = i\*i; j <= mx; j += i) {

isPrime[j] = true ;

}

}

}

for (int i = 2; i <= mx; i++) {

if (isPrime[i] == false) {

primeList.push\_back(i) ;

}

}

}

\*/

**// Square or Not (way 1)**

bool isSquare (long long x) {

long long y = sqrt(x);

return y \* y == x;

}

**// Square or Not (way 2)**

bool is\_Square (long long n) {

if (ceil((double)sqrt(n)) == floor((double)sqrt(n)))

return true;

else

return false;

}

**// Power of two or not**

bool isPowerOfTwo (long long n) {

if (n == 0)

return false;

return (ceil(log2(n)) == floor(log2(n)));

}

**// Power**

long long Pow (long long x, long long y) {

return y == 0 ? 1 : x \* pow(x, y - 1);

}

**// GCD**

long long gcd(long long a, long long b) {

return b == 0 ? a : gcd(b, a % b);

}

**// LCM**

long long lcm (long long a, long long b) {

return (a \* (b / gcd(a, b)));

}

**// Extended GCD**

long long egcd (long long a, long long b, long long &x, long long &y) {

if (b == 0) {

x = 1;

y = 0;

return a;

}

long long x1, y1;

long long gcd = egcd(b, a % b, x1, y1);

x = y1;

y = x1 - y1 \* (a / b);

return gcd;

}

**// Big Mod**

long long bigMod (long long a, long long b) {

a %= mod;

long long res = 1;

while (b > 0) {

if (b & 1)

res = (res \* a) % mod;

a = (a \* a) % mod;

b >>= 1;

}

return res;

}

**// Big Numbers Addition**

string bigSum(string str1, string str2) {

// Before proceeding further, make sure length

// of str2 is larger.

if (str1.length() > str2.length())

swap(str1, str2);

// Take an empty string for storing result

string str = "";

// Calculate length of both string

int n1 = str1.length(), n2 = str2.length();

// Reverse both of strings

reverse(str1.begin(), str1.end());

reverse(str2.begin(), str2.end());

int carry = 0;

for (int i = 0; i < n1; i++) {

// Do school mathematics, compute sum of

// current digits and carry

int sum = ((str1[i]-'0') + (str2[i]-'0') + carry);

str.push\_back(sum % 10 + '0');

// Calculate carry for next step

carry = sum / 10;

}

// Add remaining digits of larger number

for (int i = n1; i < n2; i++) {

int sum = ((str2[i] - '0') + carry);

str.push\_back(sum % 10 + '0');

carry = sum / 10;

}

// Add remaining carry

if (carry)

str.push\_back(carry + '0');

// reverse resultant string

reverse(str.begin(), str.end());

return str;

}

**// Big Numbers (Subtraction)**

// Returns true if str1 is smaller than str2.

bool isSmaller (string str1, string str2) {

// Calculate lengths of both string

int n1 = str1.length(), n2 = str2.length();

if (n1 < n2)

return true;

if (n1 > n2)

return false;

for (int i = 0; i < n1; i++)

if (str1[i] < str2[i])

return true;

else if (str1[i] > str2[i])

return false;

return false;

}

**// Function to find difference of larger numbers**

string bigSubtract(string str1, string str2) {

// Before proceeding further, make sure str1

// is not smaller

if (isSmaller(str1, str2))

swap(str1, str2);

// Take an empty string for storing result

string str = "";

// Calculate length of both string

int n1 = str1.length(), n2 = str2.length();

// Reverse both of strings

reverse(str1.begin(), str1.end());

reverse(str2.begin(), str2.end());

int carry = 0;

// Run loop till small string length

// and subtract digit of str1 to str2

for (int i = 0; i < n2; i++) {

// Do school mathematics, compute difference of

// current digits

int sub = ((str1[i] - '0') - (str2[i] - '0') - carry);

// If subtraction is less than zero

// we add then we add 10 into sub and

// take carry as 1 for calculating next step

if (sub < 0) {

sub = sub + 10;

carry = 1;

}

else

carry = 0;

str.push\_back(sub + '0');

}

// subtract remaining digits of larger number

for (int i = n2; i < n1; i++) {

int sub = ((str1[i] - '0') - carry);

// if the sub value is -ve, then make it positive

if (sub < 0) {

sub = sub + 10;

carry = 1;

}

else

carry = 0;

str.push\_back(sub + '0');

}

// reverse resultant string

reverse(str.begin(), str.end());

return str;

// Use this inside main function

// string num1, num2, ans;

// int t;

// cin >> t;

// while (t--) {

// cin >> num1 >> num2;

// ans = bigSubtract(num1, num2);

// if (isSmaller(num1, num2)) cout << "-";

// while (ans[0] == '0') ans.erase(ans.begin());

// if (ans.empty()) ans = "0";

// cout << ans << endl;

// }

}

**// Big Multiply**

string bigMultiply(string num1, string num2) {

int len1 = num1.size();

int len2 = num2.size();

if (len1 == 0 or len2 == 0)

return "0";

// will keep the result number in vector

// in reverse order

vector <int> result(len1 + len2, 0);

// Below two indexes are used to find positions

// in result.

int i\_n1 = 0;

int i\_n2 = 0;

// Go from right to left in num1

for (int i = len1 - 1; i >= 0; i--) {

int carry = 0;

int n1 = num1[i] - '0';

// To shift position to left after every

// multiplication of a digit in num2

i\_n2 = 0;

// Go from right to left in num2

for (int j=len2-1; j>=0; j--)

{

// Take current digit of second number

int n2 = num2[j] - '0';

// Multiply with current digit of first number

// and add result to previously stored result

// at current position.

int sum = n1\*n2 + result[i\_n1 + i\_n2] + carry;

// Carry for next iteration

carry = sum/10;

// Store result

result[i\_n1 + i\_n2] = sum % 10;

i\_n2++;

}

// store carry in next cell

if (carry > 0)

result[i\_n1 + i\_n2] += carry;

// To shift position to left after every

// multiplication of a digit in num1.

i\_n1++;

}

// ignore '0's from the right

int i = result.size() - 1;

while (i>=0 && result[i] == 0)

i--;

// If all were '0's - means either both or

// one of num1 or num2 were '0'

if (i == -1)

return "0";

// generate the result string

string s = "";

while (i >= 0)

s += std::to\_string(result[i--]);

return s;

// Use this inside main function

// string str1, str2;

// int t;

// cin >> t;

// while (t--) {

// cin >> str1 >> str2;

// if ((str1.at(0) == '-' or str2.at(0) == '-') and (str1.at(0) != '-' or str2.at(0) != '-' ))

// cout<<"-";

// if(str1.at(0) == '-')

// str1 = str1.substr(1);

// if(str2.at(0) == '-')

// str2 = str2.substr(1);

// cout << bigMultiply(str1, str2) << endl;

// }

}

**// Big Numbers Division**

string bigDivision(string number, int divisor) {

// As result can be very large store it in string

string ans;

// Find prefix of number that is larger

// than divisor.

int idx = 0;

int temp = number[idx] - '0';

while (temp < divisor)

temp = temp \* 10 + (number[++idx] - '0');

// Repeatedly divide divisor with temp. After

// every division, update temp to include one

// more digit.

while (number.size() > idx) {

// Store result in answer i.e. temp / divisor

ans += (temp / divisor) + '0';

// Take next digit of number

temp = (temp % divisor) \* 10 + number[++idx] - '0';

}

// If divisor is greater than number

if (ans.length() == 0)

return "0";

// else return ans

return ans;

}

inline long long MOD (long long a) { return (a % mod + mod) % mod; }

inline long long modAdd (long long a, long long b) { return MOD(MOD(a) + MOD(b)); }

inline long long modSub (long long a, long long b) { return MOD(MOD(a) - MOD(b)); }

inline long long modMul (long long a, long long b) { return MOD(MOD(a) \* MOD(b)); }

inline long long modInv (long long a) { return bigMod(a, mod - 2); }

**// To count the number of ones in binary**

long long num\_of\_ones\_in\_binary (long long x) {

return \_\_builtin\_popcountll(x);

}

**// To check whether the number of ones in binary is odd (1) or even (0)**

bool is\_number\_of\_ones\_in\_binary\_is\_odd (long long x) {

return \_\_builtin\_parity(x);

// 4 (100) returns 1

// 5 (101) returns 0

}

**// To count the leading zeros of the binary**

long long num\_of\_leadind\_zeros\_in\_binary (long long x) {

return \_\_builtin\_clzll(x);

// a = 16

// Binary form of 16 is 00000000 00000000 00000000 00010000

// Output: 27

}

**// To count the trailing zeros of the binary**

long long num\_of\_trailing\_zeros\_in\_binary (long long x) {

return \_\_builtin\_ctzll(x);

// a = 16

// Binary form of 16 is 00000000 00000000 00000000 00010000

// Output: 4

}

**// Count the digits of a number O(1)**

long long countDigits (long long n) {

return floor (log10(n) + 1);

}

**// Find the first digit O(1)**

long long firstDigit(long long n) {

// Find total number of digits - 1

int digits = (int)log10(n);

// Find first digit

n = (int)(n / pow(10, digits));

// Return first digit

return n;

}

**// Find the last digit O(1)**

long long lastDigit(long long n) {

// return the last digit

return (n % 10);

}

**// To convert a decimal number to binary number**

long long decToBin(int n) {

long long bin = 0, rem, i = 1;

while (n != 0) {

rem = n % 2;

n /= 2;

bin += rem \* i;

i \*= 10;

}

return bin;

}

**// To convert a binary number into a decimal number**

long long binToDec (long long n) {

long long dec = 0, i = 0, rem;

while (n != 0) {

rem = n % 10;

n /= 10;

dec += rem \* pow(2, i);

++i;

}

return dec;

}

**// To convert a decimal number into a binary number**

void decToHexa (int n) {

// char array to store hexadecimal number

char hexaDeciNum[100];

// counter for hexadecimal number array

int i = 0;

while (n != 0) {

// temporary variable to store remainder

int temp = 0;

// storing remainder in temp variable.

temp = n % 16;

// check if temp < 10

if (temp < 10) {

hexaDeciNum[i] = temp + 48;

i++;

}

else {

hexaDeciNum[i] = temp + 55;

i++;

}

n = n / 16;

}

// printing hexadecimal number array in reverse order

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

cout << hexaDeciNum[j];

}

**// To convert a hexadecimal number into decimal number**

int hexadecimalToDecimal(string hexa) {

int len = hexa.size();

// Initializing base value to 1, i.e 16^0

int base = 1;

int decimal = 0;

// Extracting characters as digits from last

// character

for (int i = len - 1; i >= 0; i--) {

// if character lies in '0'-'9', converting

// it to integral 0-9 by subtracting 48 from

// ASCII value

if (hexa[i] >= '0' && hexa[i] <= '9') {

decimal += (int(hexa[i]) - 48) \* base;

// incrementing base by power

base = base \* 16;

}

// if character lies in 'A'-'F' , converting

// it to integral 10 - 15 by subtracting 55

// from ASCII value

else if (hexa[i] >= 'A' && hexa[i] <= 'F') {

decimal += (int(hexa[i]) - 55) \* base;

// incrementing base by power

base = base \* 16;

}

}

return decimal;

}

**// To convert a decimal number into octal number**

void decToOctal (int n) {

// array to store octal number

int octalNum[100];

// counter for octal number array

int i = 0;

while (n != 0) {

// storing remainder in octal array

octalNum[i] = n % 8;

n = n / 8;

i++;

}

// printing octal number array in reverse order

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

cout << octalNum[j];

}

**// To convert an octal number into decimal number**

int octalToDecimal(int n) {

int num = n;

int dec\_value = 0;

// Initializing base value to 1, i.e 8^0

int base = 1;

int temp = num;

while (temp) {

// Extracting last digit

int last\_digit = temp % 10;

temp = temp / 10;

// Multiplying last digit with appropriate

// base value and adding it to dec\_value

dec\_value += last\_digit \* base;

base = base \* 8;

}

return dec\_value;

}

**// To check whether a number is palindrome or not**

bool is\_palindrome (long long x) {

long long temp = x, sum = 0;

while (x > 0) {

sum = sum \* 10 + (x % 10);

x /= 10;

}

if (temp == sum)

return true;

else

return false;

}

int dx1[] = {-1, 0, 0, 1};

int dy1[] = {0, -1, 1, 0};

int dx2[] = {-1, -1, -1, 0, 0, 0, 1, 1, 1};

int dy2[] = {-1, 0, 1, -1, 0, 1, -1, 0, 1};

**// BFS**

/\*#include <bits/stdc++.h>

using namespace std;

const long long lim = 1e5 + 5;

long long n, m;

vector <long long> adj[lim];

long long dis[lim], parent[lim];

void bfs (long long source) {

queue <long long> q;

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

dis[i] = -1;

parent[i] = -1;

}

dis[source] = 0;

q.push(source);

while (!q.empty()) {

long long u = q.front();

q.pop();

for (int i = 0; i < adj[u].size(); i++) {

long long v = adj[u][i];

if (dis[v] == -1) {

dis[v] = dis[u] + 1;

parent[v] = u;

q.push(v);

}

}

}

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

// n = number of nodes

// m = number of edges

cin >> n >> m;

for (int i = 1; i <= m; i++) {

long long x, y;

cin >> x >> y;

adj[x].push\_back(y);

adj[y].push\_back(x);

}

// source = starting node

// current = destination

long long source, current, distance;

cin >> source >> current;

bfs(source);

distance = dis[current] + 1;

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

cout << "Distance of " << source << " to " << i << " is : " << dis[i] << " " << endl;

}

cout << endl;

int comp = 0;

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

if (!dis[i]) {

bfs(i);

comp++;

}

}

vector <long long> path;

// Shortest Path of Source to Destination

while (current != -1) {

path.push\_back(current);

current = parent[current];

}

reverse(path.begin(), path.end());

cout << distance << endl;

for (int i = 0; i < path.size(); i++) {

cout << path[i] << " ";

}

cout << endl << "There are " << comp << " groups" << endl;

return 0;

}

\*/

**// Dijkstra's Shortest Path (Weight)**

/\*

const long long INF = 1e9 + 19;

const long long LIM = 1e5 + 15;

vector <vector<pair<long long, long long>>> G;

vector <bool> mark;

vector <long long> dist;

void dijkstra (long long s) {

fill(dist.begin(), dist.end(), INF);

fill(mark.begin(), mark.end(), false);

priority\_queue <

pair <long long, long long>,

vector <pair <long long, long long>>,

greater <pair <long long, long long>>

> pq;

pq.push({dist[s] = 0, s});

while (!pq.empty()) {

long long u = pq.top().second;

pq.pop();

if (mark[u]) continue;

else mark[u] = true;

for (auto v : G[u]) {

if (dist[v.first] > dist[u] + v.second) {

dist[v.first] = dist[u] + v.second;

pq.push({dist[v.first], v.first});

}

}

}

// This code goes inside main function

// long long n, m, s, t;

// cin >> n >> m;

// G.resize(n + 1);

// mark.assign(n + 1, false);

// dist.resize(n + 1);

// for (int i = 1; i <= m; ++i) {

// long long u, v, c;

// cin >> u >> v >> c;

// G[u].push\_back({v, c});

// }

// s = 1, t = n;

// dijkstra(s);

// if (dist[t] == INF)

// cout << "There is no path from " << s << " to " << t;

// else

// cout << "Min distance (" << s << " -> " << t << ") = " << dist[t];

}

\*/

// Shortest Path Print

/\*

void shortestPath() {

// This whole code goes to main function

// int n, m;

// cin >> n >> m;

// vector <vector <pair <int, int>>> graph(n + 1);

// vector <int> dist(n + 1, INT\_MAX);

// vector <bool> visited(n + 1, false);

// vector <int> parent(n + 1, -1);

// while (m--) {

// int a, b, w;

// cin >> a >> b >> w;

// graph[a].push\_back (make\_pair(b, w));

// graph[b].push\_back (make\_pair(a, w));

// // pushing back twice to make is undirected

// }

// priority\_queue <

// pair <int, int>,

// vector <pair <int, int>>, greater <pair <int, int>>

// > pq;

// pq.push (make\_pair(0, 1)); // first distance from 1 is 1 and the distance is 0

// dist[1] = 0;

// while (!pq.empty()) {

// int u = pq.top().second; // node

// int x = pq.top().first; // distance

// pq.pop();

// if (x > dist[u])

// continue;

// visited[u] = true;

// for (auto i : graph[u]) {

// int v = i.first;

// int w = i.second;

// if (!visited[v] && dist[v] > dist[u] + w) {

// dist[v] = dist[u] + w;

// pq.push(make\_pair(dist[v], v));

// parent[v] = u;

// }

// }

// }

// vector <int> path;

// for (int j = n; j >= 0; j = parent[j])

// path.push\_back(j);

// reverse(path.begin(), path.end());

// if (!visited[n]) {

// cout << -1 << endl;

// } else {

// for (auto i : path)

// cout << i << " ";

// cout << endl;

// }

}

\*/

/\*

**// DFS Traaversing**

#include <bits/stdc++.h>

using namespace std;

const int lim = 1e5 + 5;

vector <int> adj[lim];

bool visited[lim];

void dfs (int u) {

cout << u << " ";

visited[u] = 1;

for (int i = 0; i < adj[u].size(); i++) {

int v = adj[u][i];

if (visited[v] == 0) {

dfs(v);

}

}

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

// n = number of nodes

// m = number of edegs

int n, m;

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int x, y;

cin >> x >> y;

adj[x].push\_back(y);

adj[y].push\_back(x);

}

int source;

cin >> source;

dfs(source);

return 0;

}

\*/

/\*

**// DFS Shortest Path**

#include <bits/stdc++.h>

using namespace std;

vector <int> v[1001];

long long vis[1002], dis[1001];

void dfs (long long node, long long d) {

vis[node] = 1;

dis[node] = d;

for (int i = 0; i < v[node].size(); i++) {

if (vis[v[node][i]] == 0) {

dfs(v[node][i], d+1);

}

}

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

long long n, m, x, y;

cin >> n >> m;

while (m--) {

cin >> x >> y;

v[x].push\_back(y);

v[y].push\_back(x);

}

long long node;

cin >> node;

dfs(node, 0);

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

cout << node << " to " << i << " dis " << dis[i] << endl;

}

return 0;

}

\*/

**// Ordered Set Codes**

void orderedSETCodes() {

// This is how you print the ordered set

// cout << \*odrSet.find\_by\_order(i) << " " ; // {1 2 3 4 5}

// To find how many numbers are less than a number x

// cout << odrSet.order\_of\_key(5) << endl ; // 4 {1 2 3 4 <- 5}

}

**// Sorting pair using comparator**

bool comp (pair <long long, long long> p1, pair <long long, long long> p2) {

return p1.first < p2.first ;

}

**// Binary Search build in functions**

void easySearch() {

// bool found = binary\_search()(arr.begin(), arr.end(), x); -> It returns 1 0r 0

// Build In Funtion, upper\_bound()

// r = upper\_bound(arr.begin(), arr.end(), x) - arr.begin(); -> It returns the position {For Vector}

// r = upper\_bound (arr, arr+n, x) - arr ; {For Arrays}

// Build In Funtion, lower\_bound()

// r = lower\_bound(arr.begin(), arr.end(), x) - arr.begin(); // It returns the position {For Vectors}

// r = lower\_bound(arr, arr+n, x) - arr ; // {For Arrays}

}

**// Marge Sort O(NlogN)**

void Merge (long long \*arr, long long low, long long high, long long mid) { // A function to merge the two half long longo a sorted data.

// We have low to mid and mid+1 to high already sorted.

long long i, j, k, temp[high - low + 1];

i = low;

k = 0;

j = mid + 1;

// Merge the two parts long longo temp[].

while (i <= mid && j <= high) {

if (arr[i] < arr[j]) {

temp[k] = arr[i];

k++;

i++;

} else {

temp[k] = arr[j];

k++;

j++;

}

}

// Insert all the remaining values from i to mid long longo temp[].

while (i <= mid) {

temp[k] = arr[i];

k++;

i++;

}

// Insert all the remaining values from j to high long longo temp[].

while (j <= high) {

temp[k] = arr[j];

k++;

j++;

}

// Assign sorted data stored in temp[] to arr[].

for (i = low; i <= high; i++) {

arr[i] = temp[i-low];

}

}

void MergeSort (long long \*arr, long long low, long long high) { // A function to split array long longo two parts.

long long mid;

if (low < high) {

mid = (low + high) / 2;

// Split the data long longo two half.

MergeSort(arr, low, mid);

MergeSort(arr, mid+1, high);

// Merge them to get sorted output.

Merge(arr, low, high, mid);

}

// MergeSort(arr, 0, n-1); <- write this line while using merge sort in main function

}

**// Checking whether or not an array is sorted**

bool isSorted (long long arr[], long long size) {

for (long long i = 0; i < size - 1; i++) {

if (arr[i] > arr[i+1]) {

return false;

}

}

return true;

}

**// To convert a string into number**

long long stringToNum (string str) {

long long num = 0;

for (long long i = str.size()-1, j = 1; i >= 0; i--, j \*= 10) {

num += ((str[i] - '0') \* j);

}

return num;

}

**// To check whether a substring is present in a string or not and if it is, then return the starting index**

long long subStr\_in\_a\_str (string s1, string s2) {

// Find position of string s2

long long found = s1.find(s2);

// Check if position is -1 or not

if (found != string::npos) {

return found;

} else {

return -1;

}

}

**// To check whether or not a string is a substring of another string**

bool is\_it\_subString (string s1, string s2) {

// Find position of string s2

long long found = s1.find(s2);

// Check if position is -1 or not

if (found != string::npos) {

return true;

} else {

return false;

}

}

**// To sort** **multiple string lexicographically ( O(NlogN) )**

bool string\_Lexi\_Comp (string a, string b) {

if (a.compare(0, b.size(), b) == 0 || b.compare(0, a.size(), a) == 0)

return a.size() > b.size();

else

return a < b;

}

**// To check whether or not a string is Palindrome**

bool isPalindrome (string str) {

for (long long i = 0; i < str.size() / 2; i++) {

if (str[i] != str[str.size() - 1 - i]) {

return false;

}

}

return true;

}

**// To change the string into uppercase**

string upperCase (string str) {

transform( str.begin(), str.end(), str.begin(), ::toupper);

return str;

}

**// To change the string into lowercase**

string lowerCase (string str) {

transform( str.begin(), str.end(), str.begin(), ::tolower);

return str;

}

**// To check the time of executing the code**

void runTime() {

cerr << "Time Taken : " << (float)clock() / CLOCKS\_PER\_SEC << " seconds" << endl;

if ( (float)clock() / CLOCKS\_PER\_SEC >= 1.0 ) {

cerr << "Need to Optimize" << endl;

} else {

cerr << "Run Time is Ok" << endl;

}

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

return 0 ;

}