```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <math.h>
#include <map>
#include <set>
#include <sstream>
#include <cstring>
#include <climits>
#include <queue>
#include <stack>
#include <iomanip>
#include <bitset>
#include<numeric>
using namespace        std;
typedef unsigned       long long ull;
const double EPS = 1e-9;
typedef long double    ld;
#define ll             long long
#define  mod           1000000007
#define ll_min         LLONG_MIN
#define ll_max         LLONG_MAX
#define endl           "\n"
#define all(v)         v.begin(),v.end()
#define sz(s)          (int)(s.size())
#define clr(arr,x)     memset(arr,x,sizeof(arr))
const    ll            INF = 2e18;
#define format(n)      fixed<<setprecision(n)
int dx[8] = { 1, -1, 0, 0, 1, -1, 1, -1 };
int dy[8] = { 0, 0, 1, -1, 1, -1, -1, 1 };
void WEZaa() {
    //  freopen("powers.in ", "r", stdin);
    std::ios_base::sync_with_stdio(0); cin.tie(NULL); cout.tie(NULL);
}
int gcd(int a, int b)
{return b ? gcd(b, a % b) : a;}
long long lcm(int a, int b)
{return (a / gcd(a, b)) * b;}
vector<int>prime;
vector<bool>vis(3e8 + 9, 0);
void sieve(int n){
    vis[0] = vis[1] = 1;
    for (int i = 2; i <= n; i++){
        if (!vis[i]){
            prime.push_back(i);
            for (int j = 0; j < prime.size() && prime[j] * i <= n; j++)
                vis[i * prime[j]] = 1;
        }
    }
}
bool prim(int n){
    if (n <= 1) return false;
    if (n <= 3)return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i = i + 6)
```

```cpp
        if (n % i == 0 || n % (i + 2) == 0) return false;
    return true;
}
/////////////////////////////////////////////////////////////////////////////
vector<bool>vis; vector<int>prime;
void sieve(int n){
    vis[1] = vis[0] = 0;
    for (int i = 2; i < n; i++){
        if (!vis[i])
            for (int j = i * 2; j < n; j += i)
                vis[j] = 1;
        if (!vis[i]) prime.push_back(i);
    }
}
//////////////////////////////////////////////////////
int lcs(char* X, char* Y, int m, int n){
    if (m == 0 || n == 0) return 0;
    if (X[m - 1] == Y[n - 1]) return 1 + lcs(X, Y, m - 1, n - 1);
    else return max(lcs(X, Y, m, n - 1), lcs(X, Y, m - 1, n));
}
////////////////////////////////////////////////////
void printNcR(int n, int r){// p holds the value of n*(n-1)*(n-2)..., k holds the value of r*(r-1)...
    long long p = 1, k = 1;// C(n, r) == C(n, n-r),choosing the smaller value
    if (n - r < r) r = n - r;
    if (r != 0) {
        while (r) {
            p *= n;
            k *= r;// gcd of p, k
            long long m;//= __gcd(p, k);
// dividing by gcd, to simplify// product division by their gcd// saves from the overflow
            p /= m; k /= m;
            n--; r--;
        }// k should be simplified to 1// as C(n, r) is a natural number// (denominator should be 1 ) .
    }
    else p = 1;// if our approach is correct p = ans and k =1
    cout << p << endl;
}
//////////////////////////////
ll fact(int x){
    ll res = 1;
    if (x == 1 || x == 0) return 1;
    for (int i = 2; i <= x; i++)
        res *= i, res %= 7901;
    return res;
}
////////////////////// binary_search //////////////////////////
const int N = 1e6 + 4;
int a[N];
int n;//array size  //elememt to be searched in array
int k;
bool check(int dig){
    //element at dig position in array
    int ele = a[dig];
    //if k is less than element at dig position then we need to bring our higher ending to dig
    //and then continue further
    if (k <= ele) return 1;
    else return 0;
}
void binsrch(int lo, int hi){
    while (lo < hi){
        int mid = (lo + hi) / 2;
        if (check(mid)) hi = mid;
        else lo = mid + 1;
    }//if a[lo] is k
    if (a[lo] == k) cout << "Element found at index " << lo;// 0 based indexing
    else cout << "Element doesnt exist in array";//element was not in our array
}
//////////////////////////////////////////////////////////////////
int ternarySearch(int arr[], int l, int r, int x){
    if (r >= l){
        int mid1 = l + (r - l) / 3;
        int mid2 = mid1 + (r - l) / 3;// If x is present at the mid1
        if (arr[mid1] == x)  return mid1;// If x is present at the mid2
        if (arr[mid2] == x)  return mid2;// If x is present in left one-third
        if (arr[mid1] > x) return ternarySearch(arr, l, mid1 - 1, x);// If x is present in right one-third
        if (arr[mid2] < x) return ternarySearch(arr, mid2 + 1, r, x);// If x is present in middle one-third
        return ternarySearch(arr, mid1 + 1, mid2 - 1, x);
    }// We reach here when element is not present in array
    return -1;
}
////////////////////////////////////////
int randomPartition(int arr[], int l, int r);
// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method. ASSUMPTION: ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k){// If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1){
// Partition the array around a random element and// get position of pivot element in sorted array
        int pos = randomPartition(arr, l, r);// If position is same as k
        if (pos - l == k - 1) return arr[pos];
        if (pos - l > k - 1) // If position is more, recur for left subarray
            return kthSmallest(arr, l, pos - 1, k);// Else recur for right subarray
        return kthSmallest(arr, pos + 1, r, k - pos + l - 1);
    }// If k is more than the number of elements in the array
    return INT_MAX;
}
void swap(int* a, int* b){
    int temp = *a; *a = *b; *b = temp;
}
///////////////////////////////////////////////////
// Standard partition process of QuickSort(). It considers the last
// element as pivot and moves all smaller element to left of it and
// greater elements to right. This function is used by randomPartition()
int partition(int arr[], int l, int r){
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++){
        if (arr[j] <= x){
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}
// Picks a random pivot element between l and r and partitions
// arr[l..r] around the randomly picked element using partition()
int randomPartition(int arr[], int l, int r){
    int n = r - l + 1;
    int pivot = rand() % n;
    swap(&arr[l + pivot], &arr[r]);
    return partition(arr, l, r);
}
/////////////////////////////////////////////////////
// ar1[0..m-1] and ar2[0..n-1] are two given sorted arrays
// and x is given number. This function prints the pair  from
// both arrays such that the sum of the pair is closest to x.
void printClosest(int ar1[], int ar2[], int m, int n, int x){// Initialize the diff between pair sum and x.
    int diff = INT_MAX;// res_l and res_r are result indexes from ar1[] and ar2[]// respectively
    int res_l, res_r;// Start from left side of ar1[] and right side of ar2[]
    int l = 0, r = n - 1;
    while (l < m && r >= 0)
    {// If this pair is closer to x than the previously found closest, then update res_l, res_r and diff
        if (abs(ar1[l] + ar2[r] - x) < diff){
            res_l = l; res_r = r;
            diff = abs(ar1[l] + ar2[r] - x);
        }// If sum of this pair is more than x, move to smaller
        if (ar1[l] + ar2[r] > x) r--;
        else  l++;// move to the greater side
    }// Print the result
    cout << "The closest pair is [" << ar1[res_l] << ", "<< ar2[res_r] << "]\n";
}
//////////////////////////////////
void findCommon(int ar1[], int ar2[], int ar3[], int n1, int n2, int n3){// Initialize starting indexes for ar1[], ar2[] and ar3[]
    int i = 0, j = 0, k = 0;// Iterate through three arrays while all arrays have elements
    while (i < n1 && j < n2 && k < n3){
        // If x = y and y = z, print any of them and move ahead in all arrays
        if (ar1[i] == ar2[j] && ar2[j] == ar3[k]) {cout << ar1[i] << " ";   i++; j++; k++;}
        else if (ar1[i] < ar2[j])i++;
        else if (ar2[j] < ar3[k])j++;// We reach here when x > y and z < y, i.e., z is smallest
        else k++;
    }
}
/////////////////////////////////////////
int count(int S[], int m, int n){// If n is 0 then there is 1 solution (do not include any coin)
    if (n == 0) return 1;// If n is less than 0 then no// solution exists
    if (n < 0) return 0;// If there are no coins and n// is greater than 0, then no// solution exist
    if (m <= 0 && n >= 1) return 0;// count is sum of solutions (i)// including S[m-1] (ii) excluding S[m-1]
    return count(S, m - 1, n) + count(S, m, n - S[m - 1]);
}
//////////////////////////////////////////
// A utility function that returns maximum of two integers
int max(int a, int b){
    return (a > b) ? a : b;
}
// Returns the maximum value that// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n){
    int i, w;
    vector<vector<int>> K(n + 1, vector<int>(W + 1));// Build table K[][] in bottom up manner
    for (i = 0; i <= n; i++){
        for (w = 0; w <= W; w++){
            if (i == 0 || w == 0)K[i][w] = 0;
            else if (wt[i - 1] <= w) K[i][w] = max(val[i - 1] +K[i - 1][w - wt[i - 1]],K[i - 1][w]);
            else K[i][w] = K[i - 1][w];
        }
    }
    return K[n][W];
}
// A utility function that return// maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }
// Returns the maximum value that// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{// Base Case
    if (n == 0 || W == 0)return 0;
    // If weight of the nth item is more// than Knapsack capacity W, then
    // this item cannot be included// in the optimal solution
    if (wt[n - 1] > W)return knapSack(W, wt, val, n - 1);
// Return the maximum of two cases:// (1) nth item included// (2) not included
    else
        return max(val[n - 1]+ knapSack(W - wt[n - 1],wt, val, n - 1),knapSack(W, wt, val, n - 1));
}
// Print nth Ugly number// n(log(n))
int nthUglyNumber(int n){
    int pow[40] = { 1 }; // stored powers of 2 from// pow(2,0) to pow(2,30)
    for (int i = 1; i <= 30; ++i)
        pow[i] = pow[i - 1] * 2;   // Initialized low and high
    int l = 1, r = 2147483647;
    int ans = -1;           // Applying Binary Search
    while (l <= r) {        // Found mid
        int mid = l + ((r - l) / 2);
// cnt stores total numbers of ugly// number less than mid
```

```cpp
        int cnt = 0;  // Iterate from 1 to mid
        for (long long i = 1; i <= mid; i *= 5) {// Possible powers of i less than mid is i
            for (long long j = 1; j * i <= mid; j *= 3)
                cnt += upper_bound(pow, pow + 31,mid / (i * j)) - pow;
        }
// If total numbers of ugly number// less than equal// to mid is less than n we update l
        if (cnt < n) l = mid + 1;
// If total numbers of ugly number// less than qual to// mid is greater than n we update// r and ans simultaneously.
        else r = mid - 1, ans = mid;
    }
    return ans;
}
//////////////////////////////////////////////
int max(int x, int y) { return (x > y) ? x : y; }
// Returns the length of the longest palindromic subsequence in seq
int lps(char* str)
{
    int n = strlen(str);
    int i, j, cl;
    int L[n][n];  // Create a table to store results of subproblems
    // Strings of length 1 are palindrome of lentgh 1
    for (i = 0; i < n; i++)
        L[i][i] = 1;
    for (cl = 2; cl <= n; cl++) {
        for (i = 0; i < n - cl + 1; i++) {
            j = i + cl - 1;
            if (str[i] == str[j] && cl == 2) L[i][j] = 2;
            else if (str[i] == str[j])  L[i][j] = L[i + 1][j - 1] + 2;
            else  L[i][j] = max(L[i][j - 1], L[i + 1][j]);
        }
    }
    return L[0][n - 1];
}
// Check if possible subset with given sum is possible or not O(sum*n)////////
int tab[2000][2000];
int subsetSum(int a[], int n, int sum){
    // If the sum is zero it means we got our expected sum
    if (sum == 0) return 1;
    if (n <= 0) return 0;
    // If the value is not -1 it means it // already call the function
    // with the same value // it will save our from the repetation.
    if (tab[n - 1][sum] != -1) return tab[n - 1][sum];
// if the value of a[n-1] is// greater than the sum.// we call for the next value
    if (a[n - 1] > sum)
        return tab[n - 1][sum] = subsetSum(a, n - 1, sum);
    else{
// Here we do two calls because we// don't know which value is// full-fill our critaria// that's why we doing two calls
        return tab[n - 1][sum] = subsetSum(a, n - 1, sum) ||
            subsetSum(a, n - 1, sum - a[n - 1]);
    }
}
```

```
//////////////////////////////////////////////////
// A Dynamic Programming based C++ program to find minimum of coins
// to make a given change V
// m is size of coins array (number of different coins)
int minCoins(int coins[], int m,  int V){
    // table[i] will be storing the minimum number of coins
    // required for i value.  So table[V] will have result
    int table[V + 1]; // Base case (If given value V is 0)
    table[0] = 0; // Initialize all table values as Infinite
    for (int i = 1; i <= V; i++)
        table[i] = INT_MAX;
    // Compute minimum coins required for all // values from 1 to V
    for (int i = 1; i <= V; i++) {
        // Go through all coins smaller than i
        for (int j = 0; j < m; j++)
            if (coins[j] <= i){
                int sub_res = table[i - coins[j]];
                if (sub_res != INT_MAX && sub_res + 1 < table[i])
                    table[i] = sub_res + 1;
            }
    }
    if (table[V] == INT_MAX) return -1;
    return table[V];
}
///////////////////////////////////////////////////////////
// d is the number of characters in the input alphabet
//Given a text txt[0..n - 1] and a pattern pat[0..m - 1]
//write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]
//You may assume that n > m./* pat -> pattern   txt -> textq -> A prime numbe*/
#define d 256
void search(char pat[], char txt[], int q){
    int M = strlen(pat);int N = strlen(txt);int i, j;
    int p = 0;  int t = 0; int h = 1; // The value of h would be "pow(d, M-1)%q"
    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;
    // Calculate the hash value of pattern and first // window of text
    for (i = 0; i < M; i++){
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }
    // Slide the pattern over text one by one
    for (i = 0; i <= N - M; i++){
// Check the hash values of current window of text// and pattern. If the hash values match then only
        // check for characters one by one
        if (p == t){/* Check for characters one by one */
            for (j = 0; j < M; j++){
                if (txt[i + j] != pat[j]) break;
            }// if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
            if (j == M)
                cout << "Pattern found at index " << i << endl;
        }
// Calculate hash value for next window of text: Remove// leading digit, add
```

```
trailing digit
        if (i < N - M){
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;
            // We might get negative value of t, converting it// to positive
            if (t < 0) t = (t + q);
        }
    }
}
////////////////////////////////////////
// Write a program to print all permutations of a given string
// O(n*n!)
void permute(string s, string answer){
    if (s.length() == 0){
        cout << answer << "  ";
        return;
    }
    for (int i = 0; i < s.length(); i++){
        char ch = s[i];
        string left_substr = s.substr(0, i);
        string right_substr = s.substr(i + 1);
        string rest = left_substr + right_substr;
        permute(rest, answer + ch);
    }

}
/////////////////////////////////////////////////
// Maze size
#define N 4
bool solveMazeUtil(int maze[N][N], int x,int y, int sol[N][N]);
/* A utility function to print solution matrix sol[N][N] */
void printSolution(int sol[N][N]){
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", sol[i][j]);
        printf("\n");
    }
}

/* A utility function to check if x, y is valid index for N*N maze */
bool isSafe(int maze[N][N], int x, int y){ // if (x, y outside maze) return false
    if (x >= 0 && x < N && y >= 0&& y < N && maze[x][y] == 1)
        return true;
    return false;
}
/* This function solves the Maze problem using Backtracking. It mainly uses
solveMazeUtil() to solve the problem. It returns false if no path is possible,
otherwise return true and prints the path in the form of 1s. Please note that there
may be more than one solutions, this function prints one of the feasible solutions.*/
bool solveMaze(int maze[N][N]){
    int sol[N][N] = { { 0, 0, 0, 0 },{ 0, 0, 0, 0 },{ 0, 0, 0, 0 },{ 0, 0, 0, 0 } };
    if (solveMazeUtil(maze, 0, 0, sol)== false) {
        printf("Solution doesn't exist");
        return false;
    }
    printSolution(sol);
    return true;
}
/* A recursive utility function to solve Maze problem */
bool solveMazeUtil( int maze[N][N], int x, int y, int sol[N][N]){
    // if (x, y is goal) return true
    if (x == N - 1 && y == N - 1 && maze[x][y] == 1){
        sol[x][y] = 1;
        return true;
    }// Check if maze[x][y] is valid
    if (isSafe(maze, x, y) == true) {
        // Check if the current block is already part of solution path.
        if (sol[x][y] == 1) return false;
// mark x, y as part of solution path
        sol[x][y] = 1;
        /* Move forward in x direction */
        if (solveMazeUtil(maze, x + 1, y, sol)== true)
            return true;
        /* If moving in x directiondoesn't give solution then Move down in y direction */
        if (solveMazeUtil(maze, x, y + 1, sol)== true)
            return true;
        /* If moving in y direction doesn't give solution then Move back in x direction */
        if (solveMazeUtil(maze, x - 1, y, sol)== true)
            return true;
        /* If moving backwards in x direction doesn't give solution then Move upwards in y direction */
        if (solveMazeUtil( maze, x, y - 1, sol) == true)
            return true;
        /* If none of the above movement work then BACKTRACK: unmark x, y as part of solution path */
        sol[x][y] = 0;
        return false;
    }
    return false;
}
////////////////////////////////////////////////
// m Coloring Problem
class node{
    // A node class which stores the color and the edges// connected to the node
public:
    int color = 1;
    set<int> edges;
};
int canPaint(vector<node>& nodes, int n, int m){
// Create a visited array of n // nodes, initialized to zero
    vector<int> visited(n + 1, 0);
    // maxColors used till now are 1 as // all nodes are painted color 1
    int maxColors = 1;
    // Do a full BFS traversal from // all unvisited starting points
    for (int sv = 1; sv <= n; sv++){
        if (visited[sv]) continue;
```

```cpp
// If the starting point is unvisited,// mark it visited and push it in queue
        visited[sv] = 1;
        queue<int> q; q.push(sv); // BFS Travel starts here
        while (!q.empty()) {
            int top = q.front(); q.pop();
            // Checking all adjacent nodes // to "top" edge in our queue
            for (auto it = nodes[top].edges.begin();it != nodes[top].edges.end(); it++){
                // IMPORTANT: If the color of the // adjacent node is same, increase it by 1
                if (nodes[top].color == nodes[*it].color)
                    nodes[*it].color += 1;
                // If number of colors used shoots m, return // 0
                maxColors= max(maxColors, max(nodes[top].color,nodes[*it].color));
                if (maxColors > m) return 0;
                // If the adjacent node is not visited,// mark it visited and push it in queue
                if (!visited[*it]) {
                    visited[*it] = 1;
                    q.push(*it);
                }
            }
        }
    }
    return 1;
}
//////////////////////////////////////////////////////////////
//There are 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained
//after merging the above 2 arrays(i.e.array of length 2n)// The complexity should be O(log(n)).
int getMedian(int ar1[], int ar2[], int n) {
    int j = 0; int i = n - 1;
    while (ar1[i] > ar2[j] && j < n && i > -1)
        swap(ar1[i--], ar2[j++]);
    sort(ar1, ar1 + n);
    sort(ar2, ar2 + n);
    return (ar1[n - 1] + ar2[0]) / 2;
}
//////////////////////////////////////
//1) Get count of all set bits at odd positions(For 23 it’s 3).
//2) Get count of all set bits at even positions(For 23 it’s 1).
//3) If difference of above two counts is a multiple of 3 then number is also a multiple of 3.
int isMultipleOf3(int n) {
    int odd_count = 0; int even_count = 0;
    /* Make no positive if +n is multiple of 3then is -n. We are doing this to avoid
    stack overflow in recursion*/
    if (n < 0) n = -n;
    if (n == 0) return 1;
    if (n == 1) return 0;
    while (n) {/* If odd bit is set then increment odd counter */
        if (n & 1) odd_count++;
               /* If even bit is set then increment even counter */
        if (n & 2)even_count++;
        n = n >> 2;
    }
    return isMultipleOf3(abs(odd_count - even_count));
}
////////////////////////////////////
//Any number that does NOT get deleted due to above process is called “lucky”.
//Therefore, set of lucky numbers is 1, 3, 7, 13, ………
bool isLucky(int n){
    static int counter = 2;
    if (counter > n)return 1;
    if (n % counter == 0)return 0;
    /*calculate next position of input no. Variable "next_position" is just for
    readability of the program we can remove it and update in "n" only */
    int next_position = n - (n / counter);
    counter++;
    return isLucky(next_position);
}
/////////////////////////////////////////////////
/* returns count of numbers which are in range from 1 to n and don't contain 3 as a digit */
int count(int n){
    // Base cases (Assuming n is not negative)
    if (n < 3)return n;
    if (n >= 3 && n < 10) return n - 1;
    // Calculate 10^(d-1) (10 raise to the power d-1) where d is
    // number of digits in n. po will be 100 for n = 578
    int po = 1;
    while (n / po > 9)
        po = po * 10;
    // find the most significant digit (msd is 5 for 578)
    int msd = n / po;
    if (msd != 3)
        // For 578, total will be 4*count(10^2 - 1) + 4 + count(78)
        return count(msd) * count(po - 1) + count(msd) + count(n % po);
    else
        // For 35, total will be equal to count(29)
        return count(msd * po - 1);
}
void printArray(int arr[], int n){
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
/////////////////////////////////////////////////////
// Given a number, find the next smallest palindrome
// A utility function to check if num has all 9s
int AreAll9s(int* num, int n){
    int i;
    for (i = 0; i < n; ++i)
        if (num[i] != 9)
            return 0;
    return 1;
}
```

```cpp
// Returns next palindrome of a given number num[]. This function is for input type 2 and 3
void generateNextPalindromeUtil(int num[], int n){
    // Find the index of mid digit
    int mid = n / 2;
    // A bool variable to check if copy of left// side to right is sufficient or not
    bool leftsmaller = false; // End of left side is always 'mid -1'
    int i = mid - 1;
    // Beginning of right side depends // if n is odd or even
    int j = (n % 2) ? mid + 1 : mid;
    // Initially, ignore the middle same digits
    while (i >= 0 && num[i] == num[j])
        i--, j++;
    // Find if the middle digit(s) need to be// incremented or not (or copying left
    // side is not sufficient)
    if (i < 0 || num[i] < num[j]) leftsmaller = true;
    // Copy the mirror of left to tight
    while (i >= 0){
        num[j] = num[i];
        j++; i--;
    }
    // Handle the case where middle digit(s) must
    // be incremented. This part of code is for// CASE 1 and CASE 2.2
    if (leftsmaller == true){
        int carry = 1; i = mid - 1;
        // If there are odd digits, then increment // the middle digit and store the carry
        if (n % 2 == 1) {
            num[mid] += carry;
            carry = num[mid] / 10;
            num[mid] %= 10;
            j = mid + 1;
        }
        else j = mid;
        // Add 1 to the rightmost digit of the // left side, propagate the carry towards
        // MSB digit and simultaneously copying // mirror of the left side to the right side.
        while (i >= 0){
            num[i] += carry;
            carry = num[i] / 10;
            num[i] %= 10;
            num[j++] = num[i--];
        }
    }
}
// //The function that prints next palindrome////////////////
// of a given number num[] with n digits.
void generateNextPalindrome(int num[], int n){
    int i;
    printf("Next palindrome is:");
    // Input type 1: All the digits are 9, simply o/p 1// followed by n-1 0's followed by 1.
    if (AreAll9s(num, n)){
        printf("1 ");
        for (i = 1; i < n; i++)
            printf("0 ");
        printf("1");
    }
    // Input type 2 and 3
    else{
        generateNextPalindromeUtil(num, n);
        printArray(num, n);
    }
}
//////////// A Program to check whether a number is divisible by 7///////////////
int isDivisibleBy7(int num){
    // If number is negative, make it positive
    if (num < 0) return isDivisibleBy7(-num);
    if (num == 0 || num == 7) return 1;
    if (num < 10) return 0;
    // Recur for ( num / 10 - 2 * num % 10 )
    return isDivisibleBy7(num / 10 - 2 * (num - num / 10 * 10));
}
// This function puts all elements of 3 queues in the auxiliary array///////////////////
void populateAux(int aux[], queue<int> queue0, queue<int> queue1,queue<int> queue2, int* top){
    // Put all items of first queue in aux[]
    while (!queue0.empty()) {
        aux[(*top)++] = queue0.front();
        queue0.pop();
    }
    // Put all items of second queue in aux[]
    while (!queue1.empty()) {
        aux[(*top)++] = queue1.front();
        queue1.pop();
    }
    // Put all items of third queue in aux[]
    while (!queue2.empty()) {
        aux[(*top)++] = queue2.front();
        queue2.pop();
    }
}
// The main function that finds the largest possible multiple of
// 3 that can be formed by arr[] elements
int findMaxMultupleOf3(int arr[], int size){
    // Step 1: sort the array in non-decreasing order
    sort(arr, arr + size);
    // Create 3 queues to store numbers with remainder 0, 1 // and 2 respectively
    queue<int> queue0, queue1, queue2;
    // Step 2 and 3 get the sum of numbers and place them in // corresponding queues
    int i, sum;
    for (i = 0, sum = 0; i < size; ++i) {
        sum += arr[i];
        if ((arr[i] % 3) == 0) queue0.push(arr[i]);
        else if ((arr[i] % 3) == 1) queue1.push(arr[i]);
        else queue2.push(arr[i]);
    }
    // Step 4.2: The sum produces remainder 1
    if ((sum % 3) == 1) {
        // either remove one item from queue1
        if (!queue1.empty()) queue1.pop();
          // or remove two items from queue2
        else {
            if (!queue2.empty()) queue2.pop();
            else return 0;
            if (!queue2.empty()) queue2.pop();
            else return 0;
        }
    }
    // Step 4.3: The sum produces remainder 2
    else if ((sum % 3) == 2) {
        // either remove one item from queue2
        if (!queue2.empty())queue2.pop();
        // or remove two items from queue1
        else {
            if (!queue1.empty())queue1.pop();
            else return 0;
            if (!queue1.empty())queue1.pop();
            else return 0;
        }
    }
    int aux[size], top = 0;
    // Empty all the queues into an auxiliary array.
    populateAux(aux, queue0, queue1, queue2, &top);
    // sort the array in non-increasing order
    sort(aux, aux + top, greater<int>());
    // print the result
    for (int i = 0; i < top; ++i)
        cout << aux[i] << " ";
    return top;
}
int main()
{
    WEZaa();

}
```