Well paranthesized:

/\*#include <iostream>

using namespace std;

class Stack {

    char s[25];

    int top;

public:

    Stack() {

         top = -1;

         } // Initialize stack

    void push(char val) {

        if (top < 24){

            s[++toarp] = val;

        }

    }

    char pop() {

        return (top >= 0) ? s[top--] : '\*';

    }

    bool isempty() {

         return top == -1;

         }

};

class Parenthesis {

    char exp[25];

    Stack obj;

public:

    void read() {

        cout << "\nEnter the expression: ";

        cin >> exp;

    }

    void checkExpression() {

        for (int i = 0; exp[i] != '\0'; i++) {

            if (exp[i] == '{' || exp[i] == '[' || exp[i] == '(') obj.push(exp[i]);

            else if (exp[i] == '}' || exp[i] == ']' || exp[i] == ')') {

                char ch = obj.pop();

                if ((exp[i] == '}' && ch != '{') || (exp[i] == ']' && ch != '[') || (exp[i] == ')' && ch != '(')) {

                    cout << "\nExpression is not well-parenthesized.";

                    return;

                }

            }

        }

        cout << (obj.isempty() ? "\nExpression is well-parenthesized." : "\nExpression is not well-parenthesized.");

    }

};

int main() {

    Parenthesis obj;

    obj.read();

    obj.checkExpression();

    return 0;

}\*/

#include<iostream>

using namespace std;

class stack{

    char s[25];

    int top;

    public:

    stack(){

        top=-1;

    }

    bool isempty(){

        return top==-1;

    }

    void push(char val){

        if(top<24){

             s[++top]=val;

        }

        else{

            cout<<"stcak is full"<<endl;

        }

    }

    char pop(){

        if(!isempty()){

             s[top--];

        }

        else{

            cout<<"stack is emty"<<endl;

        }

    }

};

class paranthesis{

    char expn[25];

    stack obj;

    public:

    void read(){

        cout<<"enter the expression:";

        cin>>expn;

    }

    void checkexpn(){

        for(int i=0;expn[i] !='\0' ;i++){

            if(expn[i]=='{' || expn[i]=='[' ||expn[i]=='('){

                obj.push(expn[i]);

            }

            else if(expn[i]=='}' || expn[i]==']' ||expn[i]==')'){

                char ch=obj.pop();

                if((expn[i]=='}' && ch !='{') ||

                (expn[i]==']' && ch !='[') ||

                (expn[i]==')' && ch !='(')){

                    cout<<"expression is not well paranthesized"<<endl;

                }

            }

        }

        if(obj.isempty()){

            cout<<"expression is paranthesized"<<endl;

        }

        else{

            cout<<"expression not paranthesized"<<endl;

        }

    }

};

int main(){

    paranthesis pnts;

    pnts.read();

    pnts.checkexpn();

    return 0;

}

Backtracingsudoku:

public class backtrckingSudoku{

    public static boolean sudokuSolver(int sudoku[][],int row,int col){

        //base

        if(row==9){

            return true;

        }

        //recursion

        int nextRow=row,nextCol=col+1;

        if(col+1==9){

            nextRow=row+1;

            nextCol=0;

        }

        if(sudoku[row][col]!=0){

            return sudokuSolver(sudoku, nextRow, nextCol);

        }

        for(int digit=1;digit<=9;digit++){

            if(isSafe(sudoku,row,col,digit)){

                sudoku[row][col]=digit;

                if(sudokuSolver(sudoku, nextRow, nextCol)){

                    return true;

                }

                sudoku[row][col]=0;

            }

        }

        return false;

    }

    public static boolean  isSafe(int sudoku[][],int row,int col,int digit) {

        //column

        for(int i=0;i<=8;i++){

            if(sudoku[i][col]==digit){

                return false;

            }

        }

        //row

        for(int j=0;j<=8;j++){

            if(sudoku[row][j]==digit){

                return false;

            }

        }

        //grid

        int sr=(row/3)\*3;

        int sc=(col/3)\*3;

        for(int i=sr;i<sr+3;i++){

            for(int j=sc;j<sc+3;j++){

                if(sudoku[i][j]==digit){

                    return false;

                }

            }

        }

        return true;

    }

    public static void printSudoku(int sudoku[][]){

        for(int i=0;i<sudoku.length;i++){

            for(int j=0;j<sudoku[i].length;j++){

                System.out.print(sudoku[i][j]+" ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args){

        int sudoku[][] = {

            {5, 3, 0, 0, 7, 0, 0, 0, 0},

            {6, 0, 0, 1, 9, 5, 0, 0, 0},

            {0, 9, 8, 0, 0, 0, 0, 6, 0},

            {8, 0, 0, 0, 6, 0, 0, 0, 3},

            {4, 0, 0, 8, 0, 3, 0, 0, 1},

            {7, 0, 0, 0, 2, 0, 0, 0, 6},

            {0, 6, 0, 0, 0, 0, 2, 8, 0},

            {0, 0, 0, 4, 1, 9, 0, 0, 5},

            {0, 0, 0, 0, 8, 0, 0, 7, 9}

        };

    if(sudokuSolver(sudoku, 0, 0)){

        System.out.println("solution exist");

        printSudoku(sudoku);

    }

    else{

        System.out.println("solution not exist");

    }

    }

}

Backtracing java:

public class backtracking{

    public static void changeArr(int arr[],int i,int val){

        //base case

        if(i==arr.length){

            printArr(arr);

            return;

        }

        //recursion yaani kaam

        arr[i]=val;

        changeArr(arr, i+1, val+1);//fnx step

        arr[i]=arr[i]-2;//backtrcing step

    }

    public static void printArr(int arr[]){

        for(int i=0;i<arr.length;i++){

            System.out.print(arr[i]+" ");

        }

        System.out.println();

    }

    public static void findSubSet(String str,String ans,int i){

        //base case

        if(i==str.length()){

            if(ans.length()==0){

                System.out.print("null");

            }else{

                System.out.print(ans+" ,");

            }

            return;

        }

        //yes choice

        findSubSet(str, ans+str.charAt(i), i+1);

        //no choice

        findSubSet(str, ans, i+1);

    }

    //permutation

    public static void findPerm(String str,String ans){

        //base

        if(str.length()==0){

            System.out.print(ans+" ,");

            return;

        }

        //recursion

        for(int i=0;i<str.length();i++){

            char curr=str.charAt(i);

            String newStr=str.substring(0, i)+str.substring(i+1);

            findPerm(newStr, ans+curr);

        }

    }

    public static int gridWays(int i,int j,int n ,int m){

        //base

        /\*if(i==n-1 && j==m-1){

            return 1;

        }

        else if(i==n || j==m){

            return 0;

        }

        //recursion

        int w1=gridWays(i+1, j, n, m);

        int w2=gridWays(i, j+1, n, m);

        return w1+w2;\*/

        //modified for less tc above has exponential tc

        if(i==n-1 && j==m-1){

            return 1;

        }

        else if(i==n || j==m){

            return 0;

        }

        int ans=factorial(n+m-2)/factorial(n-1)\*factorial(m-1);

        return ans;

    }

    public static int factorial(int num){

        int result=1;

        for(int i=1;i<=num;i++){

            result\*=i;

        }

        return result;

    }

    public static void main(String[] args){

       // int arr[]=new int[5];

        //changeArr(arr, 0, 1);

        //printArr(arr);

        int n=3,m=3;

        System.out.println(gridWays(0, 0, n, m));

    }

}

Backtracking Nqueen:

public class backtrackingNQueen {

    /\*\*

     \* @param board

     \* @param row

     \*/

    public static boolean  isSafe(char board[][],int row,int col){

        //vertical up

        for(int i=row-1;i>=0;i--){

            if(board[i][col]=='Q'){

                return false;

            }

        }

        //diagonally left

        for(int i=row-1,j=col-1;i>=0 && j>=0;i-- , j--){

            if(board[i][j]=='Q'){

                return false;

            }

        }

        //diagonally right

        for(int i=row-1,j=col+1;i>=0&&j<board.length;i--,j++){

            if(board[i][j]=='Q'){

                return false;

            }

        }

        return true;

    }

    public static void nQueen(char board[][],int row){

        //base

        if(row==board.length){

            printBoard(board);

            return;

        }

        //column loop

        for(int j=0;j<board.length;j++){

            if(isSafe(board, row, j)){

                board[row][j]='Q';

                nQueen(board, row+1);//function call

                board[row][j]='x';//backtracking step

            }

        }

    }

    public static void printBoard(char board[][]){

        System.out.println("----------chess Board---------");

        for(int i=0;i<board.length;i++){

            for(int j=0;j<board[i].length;j++){

                System.out.print(board[i][j]+" ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        int n=6;

        char board[][]=new char[n][n];

        //initialize

        for(int i=0;i<n;i++){

            for(int j=0;j<n;j++){

                board[i][j]='x';

            }

        }

        nQueen(board, 0);

    }

}

Arraylist:

import java.util.ArrayList;

public class arrayList {

    public static void swap(ArrayList<Integer>list,int indx1,int indx2){

        int temp=list.get(indx1);

        list.set(indx1, list.get(indx2));

        list.set(indx2, temp);

    }

    public static void main(String[] args){

       /\* ArrayList<Integer>list=new ArrayList<>();

        //add function

        list.add(9);

        list.add(2);

        list.add(5);

        list.add(1);

        list.add(2,49);

        System.out.println(list);

        //get function

        int e=list.get(2);

        System.out.println(e);

        //remove function

        list.remove(2);

        System.out.println(list);

        //set function

        list.set(2, 10);

        System.out.println(list);

        //contain function tell us element is present or not

        System.out.println(list.contains(3));

        //size of the array

        System.out.println(list.size());

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

            System.out.print(list.get(i)+" ");

        }

        System.out.println();\*/

        //reverse list

        /\*ArrayList<Integer>list=new ArrayList<>();

        list.add(3);

        list.add(4);

        list.add(5);

        list.add(6);

        list.add(7);

        for(int i=list.size()-1;i>=0;i--){

            System.out.print(list.get(i)+" ");

        }

        System.out.println();

        //maximum element in arrayList

        ArrayList<Integer>list=new ArrayList<>();

        list.add(3);

        list.add(12);

        list.add(5);

        list.add(6);

        list.add(7);

        int max=Integer.MIN\_VALUE;

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

            if(max<list.get(i)){

                max=list.get(i);

            }

        }

        System.out.println(max);\*/

        //swapping numbers

        ArrayList<Integer>list=new ArrayList<>();

        list.add(3);

        list.add(12);

        list.add(5);

        list.add(6);

        list.add(7);

        int indx1=5,indx2=6;

        System.out.println(list);

        swap(list, indx1, indx2);

        System.out.println(list);

    }

}

Psofbacktrcking:

public class psofBacktrackin{

    public static void ratMaze(int maze[][],int row){

        if(row==maze.length){

            printMaze(maze);

            return;

        }

        //recuresion

        for(int j=0;j<maze.length;j++){

            if(isSafe(maze,row,j)){

                maze[row][j]='r';

                ratMaze(maze, row+1);

                maze[row][j]='x';

            }

        }

    }

    public static  boolean isSafe(int maze[][],int row,int col){

        //up

        for(int j=row-1;j>=0;j--){

            if(maze[row][j]=='r'){

                return false;

            }

        }

        //top

        for(int j=row+1;j>=0;j--){

            if(maze[row][j]=='r'){

                return false;

        }

        //left

        for(int i=col-1;i>=0;i-- ){

            if(maze[i][col]=='r'){

                return false;

            }

        }

        //right

        for(int i=col+1;i<=0;i++){

            if(maze[i][col]=='r'){

                r

            }

        }

    }

    public static void printMaze(int maze[][]){

        for(int i=0;i<maze.length;i++){

            for(int j=0;j<maze[0].length;j++){

                System.out.print(maze[i][j]+" ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        int maze[][] = { { 1, 0, 0, 0 },

        { 1, 1, 0, 1 },

        { 0, 1, 0, 0 },

        { 1, 1, 1, 1 } };

        ratMaze(maze, 0);

    }

}

Divide nd conquer:

public class DivideNdConquer {

    public static void mergeSort(int arr[],int si,int ei){

        //base case

        if(si>=ei){

            return;

        }

        //kaam

        int mid=si+(ei-si)/2;

        mergeSort(arr, si, mid);

        mergeSort(arr, mid+1, ei);

        merge(arr, si, mid, ei);

    }

    public static void merge(int arr[],int si,int mid,int ei){

        int temp[]=new int[ei-si+1];//our indexing start from 0 so we add 1

        int i=si;//iterator for left part

        int j=mid+1;//iterator for right part

        int k=0;//iterator for temp array

        while(i<=mid && j<=ei){

            if(arr[i]<arr[j]){

                temp[k]=arr[i];

                i++;

            }

            else{

                temp[k]=arr[j];

                j++;

            }

            k++;

        }

        //if some element remains then add it

        //left

        while(i<=mid){

            temp[k++]=arr[i++];

        }

        //right part

        while(j<=ei){

            temp[k++]=arr[j++];

        }

        //copy temp to original array

        for(k=0,i=si;k<temp.length;k++,i++){

            arr[i]=temp[k];

        }

    }

    public static void print(int arr[]){

        for(int i=0;i<arr.length;i++){

            System.out.print(arr[i]+" ");

        }

        System.out.println();

    }

    public static void quickSort(int arr[],int si,int ei){

        //base case

        if(si>=ei){

            return;

        }

        //kaam

        int pIndex=partition(arr,si,ei);

        quickSort(arr, si, pIndex-1);

        quickSort(arr,pIndex+1, ei);

    }

    public static int partition(int arr[],int si,int ei){

        int pivot=arr[ei];

        int i=si-1;//to make space for element smaller than pivot

        for(int j=si;j<ei;j++){

            if(arr[j]<=pivot){

                i++;

                int temp=arr[j];

                arr[j]=arr[i];

                arr[i]=temp;

            }

        }i++;

        int temp=pivot;

        arr[ei]=arr[i];

        arr[i]=temp;

        return i;

    }

    public static int searchRotatedArr(int arr[],int tar,int si,int ei ){

        //kaam

        int mid=si+(ei-si)/2;

        //case found

        if(arr[mid]<=tar){

            return mid;

        }

        //mid on L1

        if(arr[si]<=arr[mid]){

            //left of L1

            if(arr[si]<=tar && tar<=arr[mid]){

                return searchRotatedArr(arr, tar, si, mid-1);//here mid-1 bcz of we check mid

            }

            //right of L2

            else{

                return searchRotatedArr(arr, tar, mid+1, ei);

            }

        }else{

            //right of the L2

            if(arr[mid]>=tar && tar<=arr[ei]){

                return searchRotatedArr(arr, tar, mid+1, ei);

            }else{

                return searchRotatedArr(arr, tar, si, mid-1);

            }

        }

        }

    public static void main(String[] args) {

        int arr[]={6,3,9,5,2,8};

        int target=9;

        int tarIndx=searchRotatedArr(arr, target, 0, arr.length-1);

        System.out.println(tarIndx);

    }

}

Psofdnc:

public class psofDndC{

    /\*public static void mergeSort(String arr[],int si,int ei){

        //base case

        if(si>=ei){

            return;

        }

        //kaam

        int mid=si+(ei-si)/2;

        //divide

        mergeSort(arr, si, mid);

        mergeSort(arr, mid+1, ei);

        //combine

        merge(arr,si,mid,ei);

    }

    public static void merge(String arr[],int si,int mid,int ei){

        String s3[]=new String[ei-si+1];

        int i=si;

        int j=mid+1;

        int k=0;

        while(i<=mid && j<=ei){

            if(arr[i].compareTo(arr[j])<=0){

                s3[k]=arr[i];

                i++;

            }else{

                s3[k]=arr[j];

                j++;

            }

            k++;

        }

        while(i<=mid){

            s3[k++]=arr[i++];

        }

        while(j<=ei){

            s3[k++]=arr[j++];

        }

        for(k=0,i=si;k<s3.length;k++,i++){

            arr[i]=s3[k];

        }

    }

    public static void print(String arr[]){

        for(int i=0;i<arr.length;i++){

            System.out.print(arr[i]+" ");

        }

    }\*/

    //count

    public static int  mergeSort(int arr[],int si,int ei){

        int count=0;

        //base case

        if(si>=ei){

            return -1;

        }

        //kaam

        int mid=si+(ei-si)/2;

        //divide

        count+=mergeSort(arr, si, mid);

        count+=mergeSort(arr, mid+1, ei);

        //combine

        count+=merge(arr,si,mid,ei);

        return count;

    }

    public static int merge(int arr[],int si,int mid,int ei){

        int temp[]=new int[ei-si+1];

        int i=si;

        int j=mid+1;

        int k=0;

        int ans=0;

        while(i<=mid && j<=ei){

            if(arr[i]<arr[j]){

                temp[k]=arr[i];

                i++;

            }else{

                ans+=(si-i);

                temp[k]=arr[j];

                j++;

            }

            k++;

        }

        while(i<=mid){

            temp[k++]=arr[i++];

        }

        while(j<=ei){

            temp[k++]=arr[j++];

        }

        for(k=0,i=si;k<temp.length;k++,i++){

            arr[i]=temp[k];

        }

        return ans;

    }

    public static void print(int arr[]){

        for(int i=0;i<arr.length;i++){

            System.out.print(arr[i]+" ");

        }

    }

    public static void main(String[] args){

        int arr[]={4,8,3,9,3,5,3,6,3};

        mergeSort(arr, 0, arr.length-1);

        print(arr);

    }

}

Diagonal sum:

public class diagonalSum {

    public static int sumofdiagonal(int matrix[][]){

        /\* in this case tc is n^2 not good

        int sum=0;

        for(int i=0;i<matrix.length;i++){

            for(int j=0;j<matrix[0].length-1;j++){

            //pd

                if(i==j){

                    sum+=matrix[i][i];

                }

                    //sd

                else if(i+j==matrix.length-1){

                    sum+=matrix[i][matrix.length-1-i];

                }

            }

        }

        return sum;\*/

        int sum=0;

        for(int i=0;i<matrix.length;i++){

            //pd

            sum+=matrix[i][i];

            //sd

            if(i!=matrix.length-1-i)

                sum+=matrix[i][matrix.length-1-i];

        }return sum;

    }

    public static void main(String[] args) {

        int matrix[][]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};

        int result=sumofdiagonal(matrix);

        System.out.println("sum is:"+result);

    }

}