Ice-cream

```
def profit(d, p, k):
    if k == 0:
        return p[0]
    if k < 0:
        return 0

    next_shop = k - 1
    while d[next_shop] + 500 > d[k]:
        if next_shop < 0:
            return 0
        next_shop -= 1

    return max(p[k] + profit(d, p, next_shop),
profit(d, p, k - 1))
```

```
import java.util.*;

// Part of Cosmos by OpenGenus Foundation

public class Median{

public static void main(String[] args) {

int length;

int median;

System.out.println("Enter Length of array");

Scanner scanner = new Scanner(System.in);

length = scanner.nextInt();

int array[] = new int[length];

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

array[i] = scanner.nextInt();

}

Arrays.sort(array);

if (array.length % 2 == 0){

median = (((array[array.length /2]) +
(array[array.length/2 - 1])) / 2);

}

else {

median = array[array.length / 2];

}

System.out.println(median);

}

}
```

```
keys=list(map(int,input("Enter the list of
keys").split()))

freq=list(map(int,input("Enter the list of
frequencies").split()))

z=[]

n=len(keys)

for i in range(n):

z+=[[keys[i],freq[i]]]

z.sort()

cost=[[10**18 for i in range(n)] for j in range(n)]
#initialising with infinity

for i in range(n):

keys[i]=z[i][0]

freq[i]=z[i][1]

s=[freq[0]]

for i in range(n-1):

s+=[s[i]+freq[i+1]]

for i in range(n):

cost[i][i]=freq[i]

for i in range(2,n+1):

for j in range(n-i+1):

l=j

r=l+i-1

for k in range(l,r+1):

if k==l:

cost[l][r]=min(cost[l][r],cost[l+1][r])

elif k==r:

cost[l][r]=min(cost[l][r],cost[l][r-1])

else:

cost[l][r]=min(cost[l][r],cost[l][k]+cost[k+1][r])


cost[l][r]+=s[r]-s[l]+freq[l]

print ("Cost of Optimal BST is : ",cost[0][n-1])
```

```
import java.io.*;

import java.lang.*;

import java.math.*;

import java.util.*;


class BinomialCoefficient{

static int binomialCoeff(int n, int k) {

if (k>n) {

return 0;

}


int c[] = new int[k+1];

int i, j;


c[0] = 1;


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

for (j=min(i,k); j>0; j--) {

c[j] = c[j] + c[j-1];

}

}


return c[k];

}


static int min(int a, int b) {

return (a<b)? a: b;

}


//test case

public static void main(String args[]) {

int n = 5, k = 2;

System.out.println("Value of C("+n+","+k+") is
"+binomialCoeff(n, k));

}

}


class boolean_parenthesization{

public static int
boolean_parenthesization_(String symbols,
String operators) {

int noOfSymbols = symbols.length();

int[][] trueMatrix = new
int[noOfSymbols][noOfSymbols], falseMatrix =
new int[noOfSymbols][noOfSymbols];


for (int index=0; index < noOfSymbols; index++) {

if (symbols.charAt(index) == 'T') {

trueMatrix[index][index] = 1;

falseMatrix[index][index] = 0;

}else {

trueMatrix[index][index] = 0;

falseMatrix[index][index] = 1;

}

}


for (int loopVar1=1; loopVar1 < noOfSymbols;
loopVar1++) {

for (int innerLoopVar1=0,
innerLoopVar2=loopVar1; innerLoopVar2 <
noOfSymbols; innerLoopVar1++,
innerLoopVar2++) {

trueMatrix[innerLoopVar1][innerLoopVar2] = 0;

falseMatrix[innerLoopVar1][innerLoopVar2] = 0;

int b, d, e;

for (int a=0; a < loopVar1; a++){
```

```
b = innerLoopVar1 + a;

d = trueMatrix[innerLoopVar1][b] +
falseMatrix[innerLoopVar1][b];

e = trueMatrix[b+1][innerLoopVar2] +
falseMatrix[b+1][innerLoopVar2];


switch (operators.charAt(b)) {

case '|':

trueMatrix[innerLoopVar1][innerLoopVar2] += d
* e - falseMatrix[innerLoopVar1][b] *
falseMatrix[b+1][innerLoopVar2];

falseMatrix[innerLoopVar1][innerLoopVar2] +=
falseMatrix[innerLoopVar1][b] *
falseMatrix[b+1][innerLoopVar2];

break;

case '&':

trueMatrix[innerLoopVar1][innerLoopVar2] +=
trueMatrix[innerLoopVar1][b] *
trueMatrix[b+1][innerLoopVar2];

falseMatrix[innerLoopVar1][innerLoopVar2] += d
* e - trueMatrix[innerLoopVar1][b] *
trueMatrix[b+1][innerLoopVar2];

break;

case '^':

trueMatrix[innerLoopVar1][innerLoopVar2] +=
falseMatrix[innerLoopVar1][b] *
trueMatrix[b+1][innerLoopVar2] +
trueMatrix[innerLoopVar1][b] *
falseMatrix[b+1][innerLoopVar2];

falseMatrix[innerLoopVar1][innerLoopVar2] +=
trueMatrix[innerLoopVar1][b] *
trueMatrix[b+1][innerLoopVar2] +
falseMatrix[innerLoopVar1][b] *
falseMatrix[b+1][innerLoopVar2];;

break;

}

}

}

}

return trueMatrix[0][noOfSymbols - 1];

}

public static void main(String[] args){

String symbols = "TFTTF";

String operators = "|&|^";

System.out.println(boolean_parenthesization_(sy
mbols, operators));

}
}


public class WaysToCoinChange {

public static int dynamic(int[] v, int amount) {

int[][] solution = new int[v.length + 1][amount +
1];


// if amount=0 then just return empty set to
make the change

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

solution[i][0] = 1;

}


// if no coins given, 0 ways to change the amount

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

solution[0][i] = 0;

}


// now fill rest of the matrix.


for (int i = 1; i <= v.length; i++) {

for (int j = 1; j <= amount; j++) {

// check if the coin value is less than the amount
needed

if (v[i - 1] <= j) {

solution[i][j] = solution[i - 1][j]

+ solution[i][j - v[i - 1]];

} else {

// just copy the value from the top

solution[i][j] = solution[i - 1][j];

}

}

}

return solution[v.length][amount];

}


public static void main(String[] args) {

int amount = 5;

int[] v = { 1, 2, 3 };

System.out.println("By Dynamic Programming " +
dynamic(v, amount));

}

}

import java.util.Scanner;


public class edit_distance {

public static int edit_DP(String s,String t){

int l1 = s.length();

int l2 = t.length();


int dp[][] = new int[l1+1][l2+1];


for(int i=0; i<=l2; i++)

dp[0][i] = i;


for(int i=0; i<=l1; i++)

dp[i][0] = i;


for(int i=1; i<=l1; i++){

for(int j=1; j<=l2; j++){

if(s.charAt(l1-i) == t.charAt(l2-j))

dp[i][j] = dp[i-1][j-1];

else

dp[i][j] = 1 + Math.min( dp[i-1][j-1] , Math.min
(dp[i-1][j] , dp[i][j-1]) );

}

}

return dp[l1][l2];

}


public static void main(String[] args) {

Scanner sc = new Scanner(System.in);


System.out.print("Enter first string >> ");

String a = sc.next();


System.out.print("Enter second string >> ");

String b = sc.next();


System.out.println("Edit Distance : " +
edit_DP(a,b));

}
}

public class Factorial {


private static int factorial(int num) {

if (num == 0) {

return 1;

} else {

return (num * factorial(num - 1));

}

}
```

```
public static void main(String[] args) {
int number = 5;
int result;
result = factorial(number);
System.out.printf("The factorial of %d is %d", number, result);
}
}

public class Max_subarray_problem {

public static void main(String[] args) {

System.out.println(new int[] {-3, 2, -1, 4, -5}, 0, 4); // Expected output: 5
System.out.println(new int[] {-1, -2, -3, -4, -5}, 0, 4); // Expected output: -1
}

private static int findmaxsum(int[] a, int l, int h) {
int max;
if(l==h)
return a[l];
else
{
int mid = (l + h) / 2;

int leftmaxsum = findmaxsum(a, l, mid);
int rightmaxsum = findmaxsum(a , mid + 1, h);
int crossmaxsum = findcrosssum(a, l, mid, h);

max = Math.max(Math.max(leftmaxsum, rightmaxsum), crossmaxsum);
}
return max;
}

private static int findcrosssum(int[] a, int l, int mid, int h) {

int leftsum = Integer.MIN_VALUE;
int lsum = 0;
for(int i = mid; i >= l; i--)
{
lsum += a[i];
if(lsum > leftsum)
leftsum = lsum;
}
int rightsum = Integer.MIN_VALUE;
int rsum = 0;
for(int j = mid + 1; j <= h; j++)
{
rsum += a[j];
if(rsum > rightsum)
rightsum = rsum;
}
return rightsum + leftsum;
}

}

import java.util.*;
import java.lang.*;
import java.io.*;

class LBS
{
/* lbs() returns the length of the Longest Bitonic Subsequence in
arr[] of size n. The function mainly creates two temporary arrays
lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

lis[i] ==> Longest Increasing subsequence ending with arr[i]
lds[i] ==> Longest decreasing subsequence starting with arr[i]
*/
static int lbs( int arr[], int n )
{
int i, j;

/* Allocate memory for LIS[] and initialize LIS values as 1 for
all indexes */
int[] lis = new int[n];
for (i = 0; i < n; i++)
lis[i] = 1;


for (i = 1; i < n; i++)
for (j = 0; j < i; j++)
if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
lis[i] = lis[j] + 1;

/* Allocate memory for lds and initialize LDS values for
all indexes */
int[] lds = new int [n];
for (i = 0; i < n; i++)
lds[i] = 1;

/* Compute LDS values from right to left */
for (i = n-2; i >= 0; i--)
for (j = n-1; j > i; j--)
if (arr[i] > arr[j] && lds[i] < lds[j] + 1)
lds[i] = lds[j] + 1;



int max = lis[0] + lds[0] - 1;
for (i = 1; i < n; i++)
if (lis[i] + lds[i] - 1 > max)
max = lis[i] + lds[i] - 1;

return max;
}

public static void main (String[] args)
{
int arr[] = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5,
13, 3, 11, 7, 15};
int n = arr.length;
System.out.println("Length of LBS is "+ lbs( arr, n ));
}
}

class LongestCommonSubsequenceRec {
int lcs( char[] X, char[] Y, int m, int n) {
if (m == 0 || n == 0) { // base case
return 0;
}
if (X[m-1] == Y[n-1]) { // if common element is found increase lcs length by 1
return 1 + lcs(X, Y, m-1, n-1);
} else { // recursively move back on one string at a time
return Math.max(lcs(X, Y, m, n - 1), lcs(X, Y, m - 1, n));

}

}

public static void main(String[] args) {

LongestCommonSubsequenceRec lcs = new LongestCommonSubsequenceRec();

String s1 = "AAGTCGGTAB";

String s2 = "AGXTGXAYTBC";

char[] X=s1.toCharArray();

char[] Y=s2.toCharArray();

int m = X.length;

int n = Y.length;

System.out.println("Length of LCS is" + " " + lcs.lcs( X, Y, m, n ));

}

}

import java.lang.Math;

class LIS

{

// returns size of the longest increasing subsequence within the given array

// O(n^2) approach

static int lis(int arr[], int n)

{

int dp[] = new int[n];

int ans = 0;

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

{

dp[i] = 1;

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

{

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

{

dp[i] = Math.max(dp[i], 1+dp[j]);

}

}

ans = Math.max(ans, dp[i]);

}

return ans;

}

public static void main (String[] args) throws java.lang.Exception

{

int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60, 80 };

int n = arr.length;

System.out.println("Length of lis is " + lis( arr, n ) + "\n" );

}

}

class MatrixChainMultiplication

{

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n

static int MatrixChainOrder(int p[], int n)

{

int m[][] = new int[n][n];

int i, j, k, L, q;

/* m[i,j] = Minimum number of scalar multiplications needed

to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where

dimension of A[i] is p[i-1] x p[i] */

// cost is zero when multiplying one matrix.

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

m[i][i] = 0;

// L is chain length.

for (L=2; L<n; L++)

{

for (i=1; i<n-L+1; i++)

{

j = i+L-1;

if(j == n) continue;

m[i][j] = Integer.MAX_VALUE;

for (k=i; k<=j-1; k++)

{

// q = cost/scalar multiplications

q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];

if (q < m[i][j])

m[i][j] = q;

}

}

}

return m[1][n-1];

}

// Driver program to test above function

public static void main(String args[])

{

int arr[] = new int[] {1, 2, 3, 4};

int size = arr.length;

System.out.println("Minimum number of multiplications is "+

MatrixChainOrder(arr, size));

}

}

import java.util.*;

import java.lang.*;

import java.io.*;

/**

* Given a 2D array, find the maximum sum subarray in it

*/

public class MaximumSubMatrixSum

{

public static void main (String[] args) throws java.lang.Exception

{

findMaxSubMatrix(new int[][] {

{1, 2, -1, -4, -20},

{-8, -3, 4, 2, 1},

{3, 8, 10, 1, 3},

{-4, -1, 1, 7, -6}

});

}

/**

* To find maxSum in 1d array

*

* return {maxSum, left, right}

*/
```

```
public static int[] kadane(int[] a) {

//result[0] == maxSum, result[1] == start,
result[2] == end;

int[] result = new int[]{Integer.MIN_VALUE, 0, -1};

int currentSum = 0;

int localStart = 0;

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

currentSum += a[i];

if (currentSum < 0) {

currentSum = 0;

localStart = i + 1;

} else if (currentSum > result[0]) {

result[0] = currentSum;

result[1] = localStart;

result[2] = i;

}

}

//all numbers in a are negative

if (result[2] == -1) {

result[0] = 0;

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

if (a[i] > result[0]) {

result[0] = a[i];

result[1] = i;

result[2] = i;

}

}

}

return result;

}

/**

* To find and print maxSum, (left, top),(right,
bottom)

*/

public static void findMaxSubMatrix(int[][] a) {

int cols = a[0].length;

int rows = a.length;

int[] currentResult;

int maxSum = Integer.MIN_VALUE;

int left = 0;

int top = 0;

int right = 0;

int bottom = 0;

for (int leftCol = 0; leftCol < cols; leftCol++) {

int[] tmp = new int[rows];

for (int rightCol = leftCol; rightCol < cols;
rightCol++) {

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

tmp[i] += a[i][rightCol];

}

currentResult = kadane(tmp);

if (currentResult[0] > maxSum) {

maxSum = currentResult[0];

left = leftCol;

top = currentResult[1];

right = rightCol;

bottom = currentResult[2];

}

}

}

System.out.println("MaxSum: " + maxSum +

", range: [(" + left + ", " + top +

")(" + right + ", " + bottom + ")]");

}
}

mport java.io.*;

import java.lang.*;

import java.math.*;

import java.util.*;

class MinCostPath {

static int minCost(int costMatrix[][], int m, int n) {

int i,j;

int tc[][] = new int[m+1][n+1];

tc[0][0] = costMatrix[0][0];

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

tc[i][0] = tc[i-1][0] + costMatrix[i][0];

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

tc[0][j] = tc[0][j-1] + costMatrix[0][j];

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

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

tc[i][j] = min(tc[i-1][j-1],

tc[i-1][j],

tc[i][j-1]) + costMatrix[i][j];

return tc[m][n];

}

static int min(int x, int y, int z) {

if (x < y)

return (x < z)? x : z;

else

return (y < z)? y : z;

}

public static void main(String args[]) {

int cost[][] = new int[][]{

{1, 2, 3},

{4, 8, 2},

{1, 5, 3}

};

System.out.println(minCost(cost, 2, 2));

}
}

import java.util.*;

import java.lang.*;

import java.io.*;

/* A DP based program to find length

of the shortest supersequence */

public class SCS {

// Returns length of the shortest supersequence
of X and Y

static int superSequence(String X, String Y, int m,
int n)

{

int dp[][] = new int[m+1][n+1];

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

{
```

```
for (int j = 0; j <= n; j++)

{

if (i == 0)

dp[i][j] = j;

else if (j == 0)

dp[i][j] = i;

else if (X.charAt(i-1) == Y.charAt(j-1))

dp[i][j] = 1 + dp[i-1][j-1];

//Since we need to minimize the length

else

dp[i][j] = 1 + Math.min(dp[i-1][j], dp[i][j-1]);

}

}

return dp[m][n];

}

//Main function

public static void main(String args[])

{

String X = "ABCBDAB";

String Y = "BDCABA";

System.out.println("Length of the shortest
supersequence is "+ superSequence(X, Y,
X.length(),Y.length()));

}

}

class subset_sum

{

static boolean isSubsetSum(int set[], int n, int
sum)

{

boolean subset[][] = new boolean[sum+1][n+1];

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

subset[0][i] = true;

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

subset[i][0] = false;

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

{

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

{

subset[i][j] = subset[i][j-1];

if (i >= set[j-1])

subset[i][j] = subset[i][j] || subset[i - set[j-1]][j-1];

}

}

return subset[sum][n];

}

public static void main (String args[])

{

int set[] = {3, 34, 4, 12, 5, 7};

int sum = 19;

int n = set.length;

if (isSubsetSum(set, n, sum) == true)

System.out.println("Subset found");

else

System.out.println("No subset found");

}

}

from __future__ import generators

def closestpair(L):

def square(x): return x*x

def sqdist(p,q): return square(p[0]-
q[0])+square(p[1]-q[1])

best = [sqdist(L[0],L[1]), (L[0],L[1])]

d = sqdist(p,q)

if d < best[0]:

best[0] = d

best[1] = p,q

def merge(A,B):

i = 0

j = 0

while i < len(A) or j < len(B):

if j >= len(B) or (i < len(A) and A[i][1] <= B[j][1]):

yield A[i]

i += 1

else:

yield B[j]

j += 1

def recur(L):

if len(L) < 2:

return L

split = len(L)/2

splitx = L[split][0]

L = list(merge(recur(L[:split]), recur(L[split:])))

E = [p for p in L if abs(p[0]-splitx) < best[0]]

for i in range(len(E)):

for j in range(1,8):

if i+j < len(E):

testpair(E[i],E[i+j])

return L

L.sort()

recur(L)

return best[1]

import java.util.Scanner;

public class InversionCount {

public static int merge(int a[], int p, int q,int r){

int i = p ,j = q ,k = 0, count = 0;

int temp[] = new int[r-p+1];

while(i<q && j<=r){

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

temp[k++] = a[i++];

}

else{

temp[k++] = a[j++];

count += (q - i);

}

}

while(i<q){

temp[k++] = a[i++];

}

while(j<=r){

temp[k++] = a[j++];

}
```

```
k = 0;

while(p<=r)

a[p++] = temp[k++];

return count;

}

public static int mergeSort(int a[],int i, int j){

int count = 0;

if(i>=j)

return 0;

int mid = (i+j)/2;

count += mergeSort(a,i,mid);

count += mergeSort(a,mid+1,j);

count += merge(a,i,mid+1,j);

return count;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter n >> ");

int n = sc.nextInt();

int a[] = new int[n];

System.out.print("Enter elements of array >> ");

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

a[i] = sc.nextInt();

int count = mergeSort(a,0,a.length-1);

System.out.println("Number of inversions : " +
count);

 }
}

import java.lang.*;

import java.util.Scanner;

public class Multiply

{

public static String trim(String str,int n)

{

if(str.length()>n)

while(str.charAt(0)=='0' && str.length()>n)

str=str.substring(1);

else

while(str.length()!=n)

str="0"+str;

return str;

}

public static String add_str(String a,String b,int n)

{

a=trim(a,n);

b=trim(b,n);

String val="";

int i,rem=0;

char []c1=a.toCharArray();

char []c2=b.toCharArray();

int ans[]=new int[a.length()+1];

for(i=a.length();i>0;i--)

{

ans[i]=(c1[i-1]-48+c2[i-1]-48+rem)%10;

rem=(c1[i-1]-48+c2[i-1]-48+rem)/10;

}

ans[0]=rem;

for(i=0;i<ans.length;i++)

val=val+ans[i];

val=trim(val,a.length()+1);

return val;

}

public static String multiply(String s1,String s2,int
n)

{

String a,b,c,d,ac,bd,ad_bc,ad,bc;

int i;

if(n==1)

return
Integer.toString(Integer.parseInt(s1)*Integer.pars
eInt(s2));

a=s1.substring(0,n/2);

b=s1.substring(n/2,n);

c=s2.substring(0,n/2);

d=s2.substring(n/2,n);

ac=multiply(a,c,n/2);

bd=multiply(b,d,n/2);

ad=multiply(a,d,n/2);

bc=multiply(b,c,n/2);

ad_bc=add_str(ad,bc,n);

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

ac=ac+"0";

for(i=1;i<=n/2;i++)

ad_bc=ad_bc+"0";

ac=trim(ac,n*2);

ad_bc=trim(ad_bc,n*2);

bd=trim(bd,n*2);

return add_str(add_str(ac,ad_bc,n*2),bd,n*2);

}

public static void main(String args[])

{

int n;

Scanner sc=new Scanner(System.in);

System.out.print("Enter first number=");

String s1=sc.next();

System.out.print("Enter second number=");

String s2=sc.next();

n=s1.length();

String s3=multiply(s1,s2,n);

System.out.println(s3);

}
}

#include <stdio.h>

int

max(int const a, int const b, const int c)

{

if (a > b)

return (a > c ? a : c);

return (b > c ? b : c);

}

int

maximumContiguousSubsequenceSum(const int
a[], int beg, int end)

{
```

```
if (beg == end)

return (a[beg] > 0 ? a[beg] : 0);


int mid = (beg + end) / 2;

int leftSubProblem =
maximumContiguousSubsequenceSum(a, beg,
mid);

int rightSubProblem =
maximumContiguousSubsequenceSum(a, mid +
1, end);


int currentSum = 0, leftSum = 0, rightSum = 0;

int i;


for (i = mid; i >= beg; --i)

{

currentSum += a[i];


if (leftSum < currentSum)

leftSum = currentSum;


}

currentSum = 0;


for (i = mid + 1; i <= end; ++i)

{

currentSum += a[i];


if (rightSum < currentSum)

rightSum = currentSum;

}


return (max(leftSubProblem, rightSubProblem,
leftSum + rightSum));

}


int

main()

{

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);


int a[n];

printf("Enter %d Integers \n", n);

int i;

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

scanf("%d", &a[i]);


printf("Maximum Contiguous Subsequence Sum
is %d \n",
maximumContiguousSubsequenceSum(a, 0, n -
1));


return (0);
}

public class MergeSort {

int array[];

int size;


public MergeSort(int n) {

size=n;


array=new int[n];


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

array[i]=(int)
Math.round(Math.random()*89+10);

}

}


public int getSize() {

return size;

}


public void merge(int left, int mid, int right) {

int temp [] =new int[right-left+1];

int i = left;

int j = mid+1;

int k = 0;

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

if (array[i] <= array[j]) {

temp[k] = array[i];

k++;

i++;

} else { //array[i]>array[j]

temp[k] = array[j];

k++;

j++;

}

}

while(j<=right) temp[k++]=array[j++];

while(i<=mid) temp[k++]=array[i++];


for(k=0;k<temp.length;k++)
array[left+k]=temp[k];

}

public void merge_sort(int left,int right){

// Check if low is smaller then high, if not then
the array is sorted

if(left<right){

// Get the index of the element which is in the
middle

int mid=(left+right)/2;


merge_sort(left,mid);

merge_sort(mid+1,right);

// Combine them both

merge(left,mid,right);

}


}

public void print(){

System.out.println("Contents of the Array");

for(int k=0;k<15;k++) {

System.out.print(array[k]+" | ");

}

System.out.println();


}

public static void main(String args[]){

MergeSort m=new MergeSort(15);

System.out.println("Before Sort
<<<<<<<<<<<<<<<<<<<<");

m.print();

m.merge_sort(0,m.getSize()-1);

System.out.println("After Sort > > > > > > > > > >
> >");

m.print();

System.out.println("=======+============+==
=====+===========+=========");

MergeSort m2=new MergeSort(25);

System.out.println("Before Sort
<<<<<<<<<<<<<<<<<<<<");

m2.print();

m2.merge_sort(0,m2.getSize()-1);

System.out.println("After Sort > > > > > > > > > >
> >");

m2.print();
System.out.println("=======+===========+==
=====+===========+=========");

MergeSort m3=new MergeSort(30);

System.out.println("Before Sort
<<<<<<<<<<<<<<<<<<<<");

m3.print();

m3.merge_sort(0,m3.getSize()-1);

System.out.println("After Sort > > > > > > > > > >
> >");

m3.print();

System.out.println("=======+===========+==
=====+===========+=========");

}
}

public class QuickSort {

private int []v;

private int n;

public String toString() {

String result = "";

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

result += v[i] + " ";

}

return result;

}

public void quickSort(QuickSort v, int left, int
right) {

int i = left, j = right;

int aux;

int pivot = (left + right) / 2;

while(i <= j) {

while(v.v[i] < v.v[pivot]) {

i++;

}

while(v.v[j] > v.v[pivot]) {

j--;

}

if(i <= j) {

aux = v.v[i];

v.v[i] = v.v[j];

v.v[j] = aux;

i++;

j--;

}

}

if(left < j) {

quickSort(v, left, j);

}

if(i < right) {

quickSort(v, i, right);

}

}

public static void main(String []args) {

QuickSort obj = new QuickSort();

obj.n = 10;

obj.v = new int[10];

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

obj.v[i] = 10 - i;

}

System.out.println(obj);

obj.quickSort(obj, 0, obj.n - 1);

System.out.println(obj);

}

}
```

```
#include <stdio.h>

float

power(float x, int y)

{

if (y == 0)

return (1);

float temp = power(x, y / 2);

if (y % 2 == 0)

return (temp * temp);

else {

if (y > 0)

return (x * temp * temp);

else

return (temp * temp / x);

}

}

int

main()

{

float x;

printf("Enter x (float): ");

scanf("%f", &x);

int y;

printf("Enter y (int): ");

scanf("%d", &y);

printf("%f^%d == %f\n", x, y, power(x, y));

return (0);
}
```

```
import java.util.*;

public class EggDroppingPuzzle
{

private static int minTrials(int a, int b) {

int eggFloor[][] = new int[a + 1][b + 1];

int result, x;

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

eggFloor[i][0] = 0; // Zero trial for zero floor.

eggFloor[i][1] = 1; // One trial for one floor

}

// j trials for only 1 egg

for (int j = 1; j <= b; ++j) {

eggFloor[1][j] = j;

}

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

for (int j = 2; j <= b; ++j) {

eggFloor[i][j] = Integer.MAX_VALUE;

for (x = 1; x <= j; ++x) {

result = 1 + Math.max(eggFloor[i - 1][x - 1],
eggFloor[i][j - x]);
//choose min of all values for particular x

if (result < eggFloor[i][j])

eggFloor[i][j] = result;

}

}

}

return eggFloor[a][b];

}

//testing the program

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter no. of eggs");

int a = Integer.parseInt(sc.nextLine());

System.out.println("Enter no. of floors");

int b = Integer.parseInt(sc.nextLine());

//result outputs min no. of trials in worst case for
a eggs and b floors

int result = minTrials(a, b);

System.out.println("Minimum number of
attempts needed in Worst case with a eggs and b
floor are: " + result);

}
}

class Knapsack {

public static void main(String[] args) throws
Exception {

int val[] = {10, 40, 30, 50};

int wt[] = {5, 4, 6, 3};

int W = 10;

System.out.println(knapsack(val, wt, W));

}

public static int knapsack(int val[], int wt[], int W)
{

int N = wt.length; // Get the total number of
items. Could be wt.length or val.length. Doesn't
matter

int[][] V = new int[N + 1][W + 1]; //Create a
matrix. Items are in rows and weight at in
columns +1 on each side

//What if the knapsack's capacity is 0 - Set all
columns at row 0 to be 0

for (int col = 0; col <= W; col++) {

V[0][col] = 0;

}

//What if there are no items at home. Fill the first
row with 0

for (int row = 0; row <= N; row++) {

V[row][0] = 0;

}

for (int item=1;item<=N;item++){

//Let's fill the values row by row

for (int weight=1;weight<=W;weight++){

//Is the current items weight less than or equal
to running weight

if (wt[item-1]<=weight){

//Given a weight, check if the value of the
current item + value of the item that we could
afford with the remaining weight

//is greater than the value without the current
item itself

V[item][weight]=Math.max (val[item-1]+V[item-
1][weight-wt[item-1]], V[item-1][weight]);

}

else {

//If the current item's weight is more than the
running weight, just carry forward the value
without the current item

V[item][weight]=V[item-1][weight];

}

}

}

//Printing the matrix

for (int[] rows : V) {

for (int col : rows) {

System.out.format("%5d", col);

}

System.out.println();

}

return V[N][W];

}
}

import java.io.*;

class GFG {

   static int l =3;
   static int m =3;
   static int n =3;

   // A utility function that returns minimum
   // of 3 integers
   static int min(int x, int y, int z)
   {
       return (x < y)? ((x < z)? x : z) :
             ((y < z)? y : z);
   }

   // function to calculate MIN path sum of 3D
array
   static int minPathSum(int arr[][][])
   {
      int i, j, k;
      int tSum[][][] =new int[l][m][n];

      tSum[0][0][0] = arr[0][0][0];

      /* Initialize first row of tSum array */
      for (i = 1; i < l; i++)
         tSum[i][0][0] = tSum[i-1][0][0] + arr[i][0][0];

      /* Initialize first column of tSum array */
      for (j = 1; j < m; j++)
         tSum[0][j][0] = tSum[0][j-1][0] + arr[0][j][0];

      /* Initialize first width of tSum array */
      for (k = 1; k < n; k++)
         tSum[0][0][k] = tSum[0][0][k-1] +
arr[0][0][k];

      /* Initialize first row- First column of
         tSum array */
      for (i = 1; i < l; i++)
         for (j = 1; j < m; j++)
         tSum[i][j][0] = min(tSum[i-1][j][0],
                        tSum[i][j-1][0],
                        Integer.MAX_VALUE)
                      + arr[i][j][0];


      /* Initialize first row- First width of
         tSum array */
      for (i = 1; i < l; i++)
         for (k = 1; k < n; k++)
         tSum[i][0][k] = min(tSum[i-1][0][k],
                        tSum[i][0][k-1],
                        Integer.MAX_VALUE)
                      + arr[i][0][k];


      /* Initialize first width- First column of
         tSum array */
      for (k = 1; k < n; k++)
         for (j = 1; j < m; j++)
         tSum[0][j][k] = min(tSum[0][j][k-1],
                        tSum[0][j-1][k],
                        Integer.MAX_VALUE)
                      + arr[0][j][k];

      /* Construct rest of the tSum array */
      for (i = 1; i < l; i++)
         for (j = 1; j < m; j++)
         for (k = 1; k < n; k++)
            tSum[i][j][k] = min(tSum[i-1][j][k],
                           tSum[i][j-1][k],
                           tSum[i][j][k-1])
                         + arr[i][j][k];
```

```
        return tSum[l-1][m-1][n-1];

    }

    // Driver program
    public static void main (String[] args)
    {
        int arr[][][] = { { {1, 2, 4}, {3, 4, 5}, {5, 2, 1}},
                        { {4, 8, 3}, {5, 2, 1}, {3, 4, 2}},
                        { {2, 4, 1}, {3, 1, 4}, {6, 3, 8}}
                      };
        System.out.println ( minPathSum(arr));

    }
}

import java.util.*;
import java.lang.*;
import java.io.*;

class Graph
{

    class Edge {
        int src, dest, weight;
        Edge() {
            src = dest = weight = 0;
        }
    };

    int V, E;
    Edge edge[];

    // Creates a graph with V vertices and E edges
    Graph(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i=0; i<e; ++i)
            edge[i] = new Edge();
    }


    void BellmanFord(Graph graph,int src)
    {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        // Step 1: Initialize distances from src to all
other
        // vertices as INFINITE
        for (int i=0; i<V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;

        // Step 2: Relax all edges |V| - 1 times. A
simple
        // shortest path from src to any other vertex
can
        // have at-most |V| - 1 edges
        for (int i=1; i<V; ++i)
        {
            for (int j=0; j<E; ++j)
            {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u]!=Integer.MAX_VALUE &&
                    dist[u]+weight<dist[v])
                    dist[v]=dist[u]+weight;
            }
        }

        // Step 3: check for negative-weight
cycles.  The above
        // step guarantees shortest distances if graph
doesn't
        // contain negative weight cycle. If we get a
shorter
        //  path, then there is a cycle.
        for (int j=0; j<E; ++j)
        {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE &&
                dist[u]+weight < dist[v])
              System.out.println("Graph contains
negative weight cycle");
        }
        printArr(dist, V);
    }

    // A utility function used to print the solution
    void printArr(int dist[], int V)
    {
        System.out.println("Vertex   Distance from
Source");
        for (int i=0; i<V; ++i)
            System.out.println(i+"\t\t"+dist[i]);
    }

        public static void main(String[] args)
    {
        int V = 5;  // Number of vertices in graph
        int E = 8;  // Number of edges in graph

        Graph graph = new Graph(V, E);

        // add edge 0-1 (or A-B in above figure)
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = -1;

        // add edge 0-2 (or A-C in above figure)
        graph.edge[1].src = 0;
        graph.edge[1].dest = 2;
        graph.edge[1].weight = 4;

        // add edge 1-2 (or B-C in above figure)
        graph.edge[2].src = 1;
        graph.edge[2].dest = 2;
        graph.edge[2].weight = 3;

        // add edge 1-3 (or B-D in above figure)
        graph.edge[3].src = 1;
        graph.edge[3].dest = 3;
        graph.edge[3].weight = 2;

        // add edge 1-4 (or A-E in above figure)
        graph.edge[4].src = 1;
        graph.edge[4].dest = 4;
        graph.edge[4].weight = 2;

        // add edge 3-2 (or D-C in above figure)
        graph.edge[5].src = 3;
        graph.edge[5].dest = 2;
        graph.edge[5].weight = 5;

        // add edge 3-1 (or D-B in above figure)
        graph.edge[6].src = 3;
        graph.edge[6].dest = 1;
        graph.edge[6].weight = 1;

        // add edge 4-3 (or E-D in above figure)
        graph.edge[7].src = 4;
        graph.edge[7].dest = 3;
        graph.edge[7].weight = -3;

        graph.BellmanFord(graph, 0);
    }
}

public class GFG
{
    // A recursive function to calculate cost of
        // optimal binary search tree
    static int optCost(int freq[], int i, int j)
    {
      // Base cases
      if (j < i)      // no elements in this subarray
        return 0;
      if (j == i)     // one element in this subarray
        return freq[i];

      // Get sum of freq[i], freq[i+1], ... freq[j]
      int fsum = sum(freq, i, j);


      int min = Integer.MAX_VALUE;

      // One by one consider all elements as root
and
         // recursively find cost of the BST, compare
the
         // cost with min and update min if needed
      for (int r = i; r <= j; ++r)
      {
          int cost = optCost(freq, i, r-1) +
                         optCost(freq, r+1, j);
          if (cost < min)
             min = cost;
      }

      // Return minimum value
      return min + fsum;
    }

    // The main function that calculates minimum
cost of
        // a Binary Search Tree. It mainly uses
optCost() to
        // find the optimal cost.
    static int optimalSearchTree(int keys[], int freq[],
int n)
    {
         // Here array keys[] is assumed to be sorted
in
            // increasing order. If keys[] is not sorted,
then
            // add code to sort keys, and rearrange
freq[]
            // accordingly.
         return optCost(freq, 0, n-1);
    }

    // A utility function to get sum of array elements
        // freq[i] to freq[j]
    static int sum(int freq[], int i, int j)
    {
        int s = 0;
        for (int k = i; k <=j; k++)
            s += freq[k];
        return s;
    }

    public static void main(String[] args) {
        int keys[] = {10, 12, 20};
        int freq[] = {34, 8, 50};
        int n = keys.length;
        System.out.println("Cost of Optimal BST is " +
                    optimalSearchTree(keys, freq, n));
    }
}

import java.util.*;
import java.lang.*;
import java.io.*;

/**
 * Given a 2D array, find the maximum sum
subarray in it
 */
class Ideone
{
    public static void main (String[] args) throws
java.lang.Exception
    {
        findMaxSubMatrix(new int[][] {
                        {1, 2, -1, -4, -20},
                        {-8, -3, 4, 2, 1},
                        {3, 8, 10, 1, 3},
                        {-4, -1, 1, 7, -6}
                        });
    }

    /**
     * To find maxSum in 1d array
     *
     * return {maxSum, left, right}
     */
    public static int[] kadane(int[] a) {
        //result[0] == maxSum, result[1] == start,
result[2] == end;
        int[] result = new int[]{Integer.MIN_VALUE, 0,
-1};
        int currentSum = 0;
        int localStart = 0;

        for (int i = 0; i < a.length; i++) {
            currentSum += a[i];
            if (currentSum < 0) {
                currentSum = 0;
                localStart = i + 1;
            } else if (currentSum > result[0]) {
                result[0] = currentSum;
                result[1] = localStart;
                result[2] = i;
            }
        }


        if (result[2] == -1) {
            result[0] = 0;
            for (int i = 0; i < a.length; i++) {
                if (a[i] > result[0]) {
                    result[0] = a[i];
                    result[1] = i;
                    result[2] = i;
                }
            }
        }

        return result;
    }

    /**
     * To find and print maxSum, (left, top),(right,
bottom)
     */
    public static void findMaxSubMatrix(int[][] a) {
        int cols = a[0].length;
        int rows = a.length;
        int[] currentResult;
        int maxSum = Integer.MIN_VALUE;
        int left = 0;
        int top = 0;
        int right = 0;
        int bottom = 0;

        for (int leftCol = 0; leftCol < cols; leftCol++) {
            int[] tmp = new int[rows];

            for (int rightCol = leftCol; rightCol < cols;
rightCol++) {

                for (int i = 0; i < rows; i++) {
                    tmp[i] += a[i][rightCol];
                }
                currentResult = kadane(tmp);
                if (currentResult[0] > maxSum) {
                    maxSum = currentResult[0];
                    left = leftCol;
                    top = currentResult[1];
                    right = rightCol;
                    bottom = currentResult[2];
                }
            }
        }
        System.out.println("MaxSum: " + maxSum
+
                        ", range: [(" + left + ", " + top +
                          ")(" + right + ", " + bottom +
")]");
    }
}

class MSIS
{
    /* maxSumIS() returns the maximum sum of
increasing
       subsequence in arr[] of size n */
    static int maxSumIS( int arr[], int n )
    {
        int i, j, max = 0;
        int msis[] = new int[n];

        /* Initialize msis values for all indexes */
        for ( i = 0; i < n; i++ )
            msis[i] = arr[i];

        /* Compute maximum sum values in bottom
up manner */
        for ( i = 1; i < n; i++ )
            for ( j = 0; j < i; j++ )
                if ( arr[i] > arr[j] &&
                     msis[i] < msis[j] + arr[i])
                    msis[i] = msis[j] + arr[i];

        /* Pick maximum of all msis values */
        for ( i = 0; i < n; i++ )
            if ( max < msis[i] )
                max = msis[i];

        return max;
    }

    /* Driver program to test above function */
    public static void main(String args[])
    {
        int arr[] = new int[]{1, 101, 2, 3, 100, 4, 5};
        int n = arr.length;
        System.out.println("Sum of maximum sum
increasing "+
                       " subsequence is "+
        maxSumIS( arr, n ) );
    }
}

import java.util.ArrayList;
public class SubSet_sum_problem
{
    // dp[i][j] is going to store true if sum j is
    // possible with array elements from 0 to i.
    static boolean[][] dp;

    static void display(ArrayList<Integer> v)
    {
       System.out.println(v);
    }

    // A recursive function to print all subsets with
the
    // help of dp[][]. Vector p[] stores current
subset.
    static void printSubsetsRec(int arr[], int i, int
sum,
                                  ArrayList<Integer> p)
    {
        // If we reached end and sum is non-zero. We
print
        // p[] only if arr[0] is equal to sun OR
dp[0][sum]
        // is true.
        if (i == 0 && sum != 0 && dp[0][sum])
        {
            p.add(arr[i]);
            display(p);
            p.clear();
            return;
        }

        // If sum becomes 0
        if (i == 0 && sum == 0)
        {
            display(p);
            p.clear();
            return;
        }

        // If given sum can be achieved after ignoring
        // current element.
        if (dp[i-1][sum])
        {
            // Create a new vector to store path
            ArrayList<Integer> b = new ArrayList<>();
            b.addAll(p);
            printSubsetsRec(arr, i-1, sum, b);
        }

        // If given sum can be achieved after
considering
        // current element.
        if (sum >= arr[i] && dp[i-1][sum-arr[i]])
        {
            p.add(arr[i]);
            printSubsetsRec(arr, i-1, sum-arr[i], p);
        }
    }

    // Prints all subsets of arr[0..n-1] with sum 0.
    static void printAllSubsets(int arr[], int n, int
sum)
    {
        if (n == 0 || sum < 0)
           return;

        // Sum 0 can always be achieved with 0
elements
        dp = new boolean[n][sum + 1];
        for (int i=0; i<n; ++i)
        {
            dp[i][0] = true;
        }

        // Sum arr[0] can be achieved with single
element
        if (arr[0] <= sum)
           dp[0][arr[0]] = true;

        // Fill rest of the entries in dp[][]
        for (int i = 1; i < n; ++i)
            for (int j = 0; j < sum + 1; ++j)
                dp[i][j] = (arr[i] <= j) ? (dp[i-1][j] ||
                                   dp[i-1][j-arr[i]])
                                 : dp[i - 1][j];
        if (dp[n-1][sum] == false)
        {
            System.out.println("There are no subsets
with" +
                                        " sum "+ sum);
            return;
        }

        // Now recursively traverse dp[][] to find all
        // paths from dp[n-1][sum]
        ArrayList<Integer> p = new ArrayList<>();
        printSubsetsRec(arr, n-1, sum, p);
    }

    //Driver Program to test above functions
    public static void main(String args[])
    {
        int arr[] = {1, 2, 3, 4, 5};
        int n = arr.length;
        int sum = 10;
        printAllSubsets(arr, n, sum);
    }
}

import java.util.Arrays;

class GFG {

    static boolean modularSum(int arr[],
                              int n, int m)
    {
        if (n > m)
            return true;

        // This array will keep track of all
        // the possible sum (after modulo m)
        // which can be made using subsets of arr[]
        // initialising boolean array with all false
        boolean DP[]=new boolean[m];

        Arrays.fill(DP, false);

        // we'll loop through all the elements
        // of arr[]
        for (int i = 0; i < n; i++)
        {

            // anytime we encounter a sum divisible
            // by m, we are done
            if (DP[0])
                return true;

            // To store all the new encountered sum
            // (after modulo). It is used to make
            // sure that arr[i] is added only to
            // those entries for which DP[j]
            // was true before current iteration.
            boolean temp[] = new boolean[m];
            Arrays.fill(temp, false);

            // For each element of arr[], we loop
            // through all elements of DP table
            // from 1 to m and we add current
            // element i. e., arr[i] to all those
            // elements which are true in DP table
            for (int j = 0; j < m; j++)
            {

                // if an element is true in
                // DP table
                if (DP[j] == true)
                {
                    if (DP[(j + arr[i]) % m] == false)

                        // We update it in temp and update
                        // to DP once loop of j is over
                        temp[(j + arr[i]) % m] = true;
                }
            }

            // Updating all the elements of temp
            // to DP table since iteration over
            // j is over
            for (int j = 0; j < m; j++)
                if (temp[j])
                    DP[j] = true;


            // Also since arr[i] is a single
            // element subset, arr[i]%m is one
            // of the possible sum
            DP[arr[i] % m] = true;
        }

        return DP[0];
    }

    //driver code
    public static void main(String arg[])
    {
        int arr[] = {1, 7};
        int n = arr.length;
        int m = 5;

        if(modularSum(arr, n, m))
            System.out.print("YES\n");
        else
            System.out.print("NO\n");
    }
}
```