1. An automobile company manufactures both a two wheeler (TW) and a four wheeler (FW). A company manager wants to make the production of both types of vehicle according to the given data below:

* 1st data, Total number of vehicle (two-wheeler + four-wheeler)=v
* 2nd data, Total number of wheels = W

The task is to find how many two-wheelers as well as four-wheelers need to manufacture as per the given data.

**Example :**

**Input :**

* 200  -> Value of V
* 540   -> Value of W

**Output :**

TW =130 FW=70

**Explanation:**

130+70 = 200 vehicles

(70\*4)+(130\*2)= 540 wheels

**Constraints :**

* 2<=W
* W%2=0
* V<W

 Print “INVALID INPUT” , if inputs did not meet the constraints.

The input format for testing

The candidate has to write the code to accept two positive numbers separated by a new line.

* First Input line – Accept value of V.
* Second Input line- Accept value for W.

**The output format for testing**

* Written program code should generate two outputs, each separated by a single space character(see the example)
* Additional messages in the output will result in the failure of test case

import java.util.\*;

public class Solution

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int v=sc.nextInt();

int w=sc.nextInt();

float res=((4\*v)-w)/2;

if(w>=2 && (w%2==0) && v<w )

System.out.println("TW= "+(int)(res)+" FW= "+(int)(v-res));

else

System.out.println("INVALID INPUT");

}

}

1. Given a string S(input consisting) of ‘\*’ and ‘#’. The length of the string is variable. The task is to find the minimum number of ‘\*’ or ‘#’ to make it a valid string. The string is considered valid if the number of ‘\*’ and ‘#’ are equal. The ‘\*’ and ‘#’ can be at any position in the string.

Note : The output will be a positive or negative integer based on number of ‘\*’ and ‘#’ in the input string.

(\*>#): positive integer

(#>\*): negative integer

(#=\*): 0

Example 1:

Input 1:

###\*\*\* -> Value of S

Output :

* 1. → number of \* and # are equal

import java.util.\*;

public class Solution

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

String str=sc.next();

int count1=0,count2=0;

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

{

if(str.charAt(i)=='\*')

count1++;

else if(str.charAt(i)=='#')

count2++;

}

System.out.println(count1-count2);

}

}

1. Given an integer array Arr of size N the task is to find the count of elements whose value is greater than all of its prior elements.

Note : 1st element of the array should be considered in the count of the result.

For example,

Arr[]={7,4,8,2,9}

As 7 is the first element, it will consider in the result.

8 and 9 are also the elements that are greater than all of its previous elements.

Since total of 3 elements is present in the array that meets the condition.

Hence the output = 3.

Example 1:

Input

5 -> Value of N, represents size of Arr

7-> Value of Arr[0]

4 -> Value of Arr[1]

8-> Value of Arr[2]

2-> Value of Arr[3]

9-> Value of Arr[4]

Output :

3

Example 2:

5 -> Value of N, represents size of Arr

3 -> Value of Arr[0]

4 -> Value of Arr[1]

5 -> Value of Arr[2]

8 -> Value of Arr[3]

9 -> Value of Arr[4]

Output :

5

Constraints

1<=N<=20

1<=Arr[i]<=10000

import java.util.\*;

class Solution

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

int max=Integer.MIN\_VALUE;

int count=0;

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

{

if(arr[i]>max)

{

max=arr[i];

count++;

}

}

System.out.println(count);

}

}

1. A parking lot in a mall has RxC number of parking spaces. Each parking space will either be empty(0) or full(1). The status (0/1) of a parking space is represented as the element of the matrix. The task is to find index of the prpeinzta row(R) in the parking lot that has the most of the parking spaces full(1).

Note :

RxC- Size of the matrix

Elements of the matrix M should be only 0 or 1.

Example 1:

Input :

3 -> Value of R(row)

3 -> value of C(column)

[0 1 0 1 1 0 1 1 1] -> Elements of the array M[R][C] where each element is separated by new line.

Output :

3 -> Row 3 has maximum number of 1’s

Example 2:

input :

4 -> Value of R(row)

3 -> Value of C(column)

[0 1 0 1 1 0 1 0 1 1 1 1] -> Elements of the array M[R][C]

Output :

4 -> Row 4 has maximum number of 1’s

import java.util.\*;

class Solution

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int row=sc.nextInt();

int col=sc.nextInt();

int arr[][]=new int[row][col];

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

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

arr[i][j]=sc.nextInt();

int max=0,count=0,index=0;

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

{ count=0;

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

{

if(arr[i][j]==1)

count++;

}

if(count>max)

{

max=count;

index=i+1;

}

}

System.out.println(index);

}

}

1. A party has been organised on cruise. The party is organised for a limited time(T). The number of guests entering (E[i]) and leaving (L[i]) the party at every hour is represented as elements of the array. The task is to find the maximum number of guests present on the cruise at any given instance within T hours.

Example 1:

Input :

5 -> Value of T

[7,0,5,1,3] -> E[], Element of E[0] to E[N-1], where input each element is separated by new line

[1,2,1,3,4] -> L[], Element of L[0] to L[N-1], while input each element is separate by new line.

Output :

8 -> Maximum number of guests on cruise at an instance.

Explanation:

1st hour:

Entry : 7 Exit: 1

No. of guests on ship : 6

2nd hour :

Entry : 0 Exit : 2

No. of guests on ship : 6-2=4

Hour 3:

Entry: 5 Exit: 1

No. of guests on ship : 4+5-1=8

Hour 4:

Entry : 1 Exit : 3

No. of guests on ship : 8+1-3=6

Hour 5:

Entry : 3 Exit: 4

No. of guests on ship: 6+3-4=5

Hence, the maximum number of guests within 5 hours is 8.

Example 2:

Input:

4 -> Value of T

[3,5,2,0] -> E[], Element of E[0] to E[N-1], where input each element is separated by new line.

[0,2,4,4] -> L[], Element of L[0] to L[N-1], while input each element in separated by new line

Output:

6

Cruise at an instance

Explanation:

Hour 1:

Entry: 3 Exit: 0

No. of guests on ship: 3

Hour 2:

Entry : 5 Exit : 2

No. of guest on ship: 3+5-2=6

Hour 3:

Entry : 2 Exit: 4

No. of guests on ship: 6+2-4= 4

Hour 4:

Entry: 0 Exit : 4

No. of guests on ship : 4+0-4=0

Hence, the maximum number of guests within 5 hours is 6.

The input format for testing

The candidate has to write the code to accept 3 input.

First input- Accept value for number of T(Positive integer number)

Second input- Accept T number of values, where each value is separated by a new line.

Third input- Accept T number of values, where each value is separated by a new line.

The output format for testing

The output should be a positive integer number or a message as given in the problem statement(Check the output in Example 1 and Example 2)

Constraints:

1<=T<=25

0<= E[i] <=500

0<= L[i] <=500

import java.util.\*;

class Solution

{

public static void main (String[]args)

{

Scanner sc = new Scanner (System.in);

int t = sc.nextInt ();

int e[] = new int[t];

int l[] = new int[t];

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

e[i] = sc.nextInt ();

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

l[i] = sc.nextInt ();

int max = 0, sum = 0;

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

{

sum += e[i] - l[i];

max = Math.max (sum, max);

}

System.out.println (max);

}

}

1. At a fun fair, a street vendor is selling different colours of balloons. He sells N number of different colours of balloons (B[]). The task is to find the colour (odd) of the balloon which is present odd number of times in the bunch of balloons.

Note: If there is more than one colour which is odd in number, then the first colour in the array which is present odd number of times is displayed. The colours of the balloons can all be either upper case or lower case in the array. If all the inputs are even in number, display the message “All are even”.

Example 1:

7 -> Value of N

[r,g,b,b,g,y,y] -> B[] Elements B[0] to B[N-1], where each input element is sepārated by ṉew line.

Output :

r -> [r,g,b,b,g,y,y] -> “r” colour balloon is present odd number of times in the bunch.

Explanation:

From the input array above:

r: 1 balloon

g: 2 balloons

b: 2 balloons

y : 2 balloons

Hence , r is only the balloon which is odd in number.

Example 2:

Input:

10 -> Value of N

[a,b,b,b,c,c,c,a,f,c] -> B[], elements B[0] to B[N-1] where input each element is separated by new line.

Output :

b-> ‘b’ colour balloon is present odd number of times in the bunch.

Explanation:

From the input array above:

a: 2 balloons

b: 3 balloons

c: 4 balloons

f: 1 balloons

Here, both ‘b’ and ‘f’ have odd number of balloons. But ‘b’ colour balloon occurs first.

Hence , b is the output.

Input Format for testing

The candidate has to write the code to accept: 2 input

First input: Accept value for number of N(Positive integer number).

Second Input : Accept N number of character values (B[]), where each value is separated by a new line.

Output format for testing

The output should be a single literal (Check the output in example 1 and example 2)

Constraints:

3<=N<=50

B[i]={{a-z} or {A-Z}}

import java.util.\*;

class Solution

{

public static void main (String[]args)

{

Scanner sc = new Scanner (System.in);

int n = sc.nextInt ();

char arr[] = new char[n];

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

arr[i] = sc.next ().charAt (0);

int lower[] = new int[26];

int upper[] = new int[26];

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

{

if ((arr[i] >= 'A') && (arr[i] <= 'Z'))

upper[arr[i] - 'A']++;

else if ((arr[i] >= 'a') && (arr[i] <= 'z'))

lower[arr[i] - 'a']++;

}

boolean flag = false;

char ch = '\0';

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

{

if ((arr[i] >= 'A') && (arr[i] <= 'Z'))

{

if (upper[arr[i] - 'A'] % 2 == 1)

{

ch = (char) (arr[i]);

flag = true;

break;

}

}

else if ((arr[i] >= 'a') && (arr[i] <= 'z'))

{

if (lower[arr[i] - 'a'] % 2 == 1)

{

ch = (char) (arr[i]);

flag = true;

break;

}

}

}

if (flag == true)

System.out.println (ch);

else

System.out.println ("All are even");

}

}

1. There is a JAR full of candies for sale at a mall counter. JAR has the capacity N, that is JAR can contain maximum N candies when JAR is full. At any point of time. JAR can have M number of Candies where M<=N. Candies are served to the customers. JAR is never remain empty as when last k candies are left. JAR if refilled with new candies in such a way that JAR get full.

Write a code to implement above scenario. Display JAR at counter with available number of candies. Input should be the number of candies one customer can order at point of time. Update the JAR after each purchase and display JAR at Counter.

Output should give number of Candies sold and updated number of Candies in JAR.

If Input is more than candies in JAR, return: “INVALID INPUT”

Given,

N=10, where N is NUMBER OF CANDIES AVAILABLE

K =< 5, where k is number of minimum candies that must be inside JAR ever.

Example 1:(N = 10, k =< 5)

Input Value

3

Output Value

NUMBER OF CANDIES SOLD : 3

NUMBER OF CANDIES AVAILABLE : 7

Example : (N=10, k<=5)

Input Value

0

Output Value

INVALID INPUT

NUMBER OF CANDIES LEFT : 10

import java.util.Scanner;

class Main{

public static void main(String[] args) {

int n = 10, k = 5;

int num;

Scanner sc = new Scanner(System.in);

num = sc.nextInt();

if(num >= 1 && num <= 5) {

System.out.println("NUMBER OF CANDIES SOLD : " + num);

System.out.print("NUMBER OF CANDIES LEFT : " + (n - num));

} else {

System.out.println("INVALID INPUT");

System.out.print("NUMBER OF CANDIES LEFT : " + n);

}

}

}

1. Selection of MPCS exams include a fitness test which is conducted on ground. There will be a batch of 3 trainees, appearing for running test in track for 3 rounds. You need to record their oxygen level after every round. After trainee are finished with all rounds, calculate for each trainee his average oxygen level over the 3 rounds and select one with highest oxygen level as the most fit trainee. If more than one trainee attains the same highest average level, they all need to be selected.

Display the most fit trainee (or trainees) and the highest average oxygen level.

Note:

The oxygen value entered should not be accepted if it is not in the range between 1 and 100.

If the calculated maximum average oxygen value of trainees is below 70 then declare the trainees as unfit with meaningful message as “All trainees are unfit.

Average Oxygen Values should be rounded.

Example 1:

INPUT VALUES

95

92

95

92

90

92

90

92

90

OUTPUT VALUES

Trainee Number : 1

Trainee Number : 3

Note:

Input should be 9 integer values representing oxygen levels entered in order as

Round 1

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Round 2

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Round 3

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Output must be in given format as in above example. For any wrong input final output should display “INVALID INPUT”

import java.util.Scanner;

class Main {

public static void main(String[] args) {

int[][] trainee = new int[3][3];

int[] average = new int[3];

int max = 0;

Scanner sc = new Scanner(System.in);

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

trainee[i][j] = sc.nextInt();

if(trainee[i][j] < 1 || trainee[i][j] > 100) {

trainee[i][j] = 0;

}

}

}

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

average[i] = average[i] + trainee[j][i];

}

average[i] = average[i] / 3;

}

for(int i = 0; i < 3; i++) {

if(average[i] > max) {

max = average[i];

}

}

for(int i = 0; i < 3; i++) {

if(average[i] == max) {

System.out.println("Trainee Number : " + (i + 1));

}

if(average[i] <70) {

System.out.print("Trainee is Unfit");

}

}

}

}

1. A washing machine works on the principle of Fuzzy System, the weight of clothes put inside it for washing is uncertain But based on weight measured by sensors, it decides time and water level which can be changed by menus given on the machine control area.

For low level water, the time estimate is 25 minutes, where approximately weight is between 2000 grams or any nonzero positive number below that.

For medium level water, the time estimate is 35 minutes, where approximately weight is between 2001 grams and 4000 grams.

For high level water, the time estimate is 45 minutes, where approximately weight is above 4000 grams.

Assume the capacity of machine is maximum 7000 grams

Where approximately weight is zero, time estimate is 0 minutes.

Write a function which takes a numeric weight in the range [0,7000] as input and produces estimated time as output is: “OVERLOADED”, and for all other inputs, the output statement is

“INVALID INPUT”.

Input should be in the form of integer value –

Output must have the following format –

Time Estimated: Minutes

Example:

Input value

2000

Output value

Time Estimated: 25 minutes

1. The Caesar cipher is a type of substitution cipher in which each alphabet in the plaintext or messages is shifted by a number of places down the alphabet.  
   For example,with a shift of 1, P would be replaced by Q, Q would become R, and so on.  
   To pass an encrypted message from one person to another, it is first necessary that both parties have the ‘Key’ for the cipher, so that the sender may encrypt and the receiver may decrypt it.  
   Key is the number of OFFSET to shift the cipher alphabet. Key can have basic shifts from 1 to 25 positions as there are 26 total alphabets.  
   As we are designing custom Caesar Cipher, in addition to alphabets, we are considering numeric digits from 0 to 9. Digits can also be shifted by key places.  
   For Example, if a given plain text contains any digit with values 5 and keyy =2, then 5 will be replaced by 7, “-”(minus sign) will remain as it is. Key value less than 0 should result into “INVALID INPUT”

**Example 1:**Enter your PlainText: All the best  
Enter the Key: 1

The encrypted Text is: Bmm uif Cftu

Write a function CustomCaesarCipher(int key, String message) which will accept plaintext and key as input parameters and returns its cipher text as output.

1. We want to estimate the cost of painting a property. Interior wall painting cost is Rs.18 per sq.ft. and exterior wall painting cost is Rs.12 per sq.ft.

Take input as

1. Number of Interior walls

2. Number of Exterior walls

3. Surface Area of each Interior 4. Wall in units of square feet

Surface Area of each Exterior Wall in units of square feet

If a user enters zero as the number of walls then skip Surface area values as User may don’t want to paint that wall.

Calculate and display the total cost of painting the property

Example 1:

6

3

12.3

15.2

12.3

15.2

12.3

15.2

10.10

10.10

10.00

Total estimated Cost : 1847.4 INR

Note: Follow in input and output format as given in above example

import java.util.Scanner;

class Main {

public static void main(String[] args) {

int ni, ne, i = 0;

float intP = 18, extP = 12, cost = 0, temp;

Scanner sc = new Scanner(System.in);

ni = sc.nextInt();

ne = sc.nextInt();

if(ni < 0 || ne < 0) {

System.out.print("INVALID INPUT");

} else if(ni == 0 && ne == 0) {

System.out.print("Total estimated Cost : 0.0");

} else {

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

temp = sc.nextFloat();

cost += intP \* temp;

}

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

temp = sc.nextFloat();

cost += extP \* temp;

}

System.out.printf("Total estimated Cost : %.1f", cost);

}

}

}

1. A City Bus is a Ring Route Bus which runs in circular fashion.That is, Bus once starts at the Source Bus Stop, halts at each Bus Stop in its Route and at the end it reaches the Source Bus Stop again.

If there are n number of Stops and if the bus starts at Bus Stop 1, then after nth Bus Stop, the next stop in the Route will be Bus Stop number 1 always.

If there are n stops, there will be n paths.One path connects two stops. Distances (in meters) for all paths in Ring Route is given in array Path[] as given below:

Path = [800, 600, 750, 900, 1400, 1200, 1100, 1500]

Fare is determined based on the distance covered from source to destination stop as Distance between Input Source and Destination Stops can be measured by looking at values in array Path[] and fare can be calculated as per following criteria:

If d =1000 metres, then fare=5 INR

(When calculating fare for others, the calculated fare containing any fraction value should be ceiled. For example, for distance 900n when fare initially calculated is 4.5 which must be ceiled to 5)

Path is circular in function. Value at each index indicates distance till current stop from the previous one. And each index position can be mapped with values at same index in BusStops [] array, which is a string array holding abbreviation of names for all stops as-

“THANERAILWAYSTN” = ”TH”, “GAONDEVI” = “GA”, “ICEFACTROY” = “IC”, “HARINIWASCIRCLE” = “HA”, “TEENHATHNAKA” = “TE”, “LUISWADI” = “LU”, “NITINCOMPANYJUNCTION” = “NI”, “CADBURRYJUNCTION” = “CA”

Given, n=8, where n is number of total BusStops.

BusStops = [ “TH”, ”GA”, ”IC”, ”HA”, ”TE”, ”LU”, ”NI”,”CA” ]

Write a code with function getFare(String Source, String Destination) which take Input as source and destination stops(in the format containing first two characters of the Name of the Bus Stop) and calculate and return travel fare.

Example 1:

Input Values

ca

Ca

Output Values

INVALID OUTPUT

Example 2:

Input Values

NI

HA

Output Values

23.0 INR

Note: Input and Output should be in format given in example.

Input should not be case sensitive and output should be in the format INR

1. There are total n number of Monkeys sitting on the branches of a huge Tree. As travelers offer Bananas and Peanuts, the Monkeys jump down the Tree. If every Monkey can eat k Bananas and j Peanuts. If total m number of Bananas and p number of Peanuts are offered by travelers, calculate how many Monkeys remain on the Tree after some of them jumped down to eat.

At a time one Monkeys gets down and finishes eating and go to the other side of the road. The Monkey who climbed down does not climb up again after eating until the other Monkeys finish eating.

Monkey can either eat k Bananas or j Peanuts. If for last Monkey there are less than k Bananas left on the ground or less than j Peanuts left on the ground, only that Monkey can eat Bananas(<k) along with the Peanuts(<j).

Write code to take inputs as n, m, p, k, j and return the number of Monkeys left on the Tree.

Where, n= Total no of Monkeys

k= Number of eatable Bananas by Single Monkey (Monkey that jumped down last may get less than k Bananas)

j = Number of eatable Peanuts by single Monkey(Monkey that jumped down last may get less than j Peanuts)

m = Total number of Bananas

p = Total number of Peanuts

Remember that the Monkeys always eat Bananas and Peanuts, so there is no possibility of k and j having a value zero

Example 1:

Input Values

20

2

3

12

12

Output Values

Number of Monkeys left on the tree:10

Note: Kindly follow the order of inputs as n,k,j,m,p as given in the above example. And output must include the same format as in above example(Number of Monkeys left on the Tree:)

For any wrong input display INVALID INPUT

import java.util.\*;

class Monkeys

{

public static void main(String []args)

{

Scanner sc = new Scanner (System.in);

int n = sc.nextInt();

int k = sc.nextInt();

int j = sc.nextInt();

int m = sc.nextInt();

int p = sc.nextInt();

int atebanana=0 ,atepeanut=0;

if( n<0 && k<0 || j<0 || m<0 || p<0)

{

System.out.println("Invalid Input");

}

else

{

if(k>0)

{

atebanana =m/k;

m=m%k;

}

if(j>0)

{

atepeanut = p/j;

p=p%j;

}

n=n-atebanana-atepeanut;

if((m!=0) || (p!=0))

n=n-1;

System.out.println("Number of Monkeys left on the Tree: "+n);

}

}

}

1. Chain Marketing Organization has has a scheme for income generation, through which its members generate income for themselves. The scheme is such that suppose A joins the scheme and makes R and V to join this scheme then A is Parent Member of R and V who are child Members. When any member joins the scheme then the parent gets total commission of 10% from each of its child members.

Child members receive commission of 5% respectively. If a Parent member does not have any member joined under him, then he gets commission of 5%.

Take name of the members joining the scheme as input.

Display how many members joined the scheme including parent member.Calculate the Total commission gained by each members in the scheme. The fixed amount for joining the scheme is Rs.5000 on which commission will be generated

SchemeAmount = 5000

Example 1: When there are more than one child members

Input : (Do not give input prompts.Accept values as follows. )

Amit //Enter parent Member as this

Y //Enter Y if Parent member has child members otherwise enter N

Rajesh,Virat //Enter names of child members of Amit in comma separated

Output:(Final Output must be in format given below.)

TOTAL MEMBERS:3

COMISSION DETAILS

Amit: 1000 INR

Rajesh :250 INR

Virat: 250 INR

Example 2: When there is only one child member in the hierarchy

Input :

Amit

Y

Rajesh

Output:

Total Members: 2

Comission Details

Amit: 500 INR

Rajesh: 250 INR

1. FULLY AUTOMATIC VENDING MACHINE – dispenses your cuppa on just press of button. A vending machine can serve range of products as follows:

Coffee

Espresso Coffee

Cappuccino Coffee

Latte Coffee

Tea

Plain Tea

Assam Tea

Ginger Tea

Cardamom Tea

Masala Tea

Lemon Tea

Green Tea

Organic Darjeeling Tea

Soups

Hot and Sour Soup

Veg Corn Soup

Tomato Soup

Spicy Tomato Soup

Beverages

Hot Chocolate Drink

Badam Drink

Badam-Pista Drink

Write a program to take input for main menu & sub menu and display the name of sub menu selected in the following format (enter the first letter to select main menu):

Welcome to CCD

Enjoy your

Example 1:

Input:

c

1

Output

Welcome to CCD!

Enjoy your Espresso Coffee!

Example 2:

Input

t

9

Output

INVALID OUTPUT!

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Question

{

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

{

String[] c = {"Espresso Coffee", "Cappuccino Coffee", "Latte Coffee"};

String[] t = {"Plain Tea", "Assam Tea", "Ginger Tea", "Cardamom Tea", "Masala Tea", "Lemon Tea", "Green Tea", "Organic Darjeeling Tea"};

String[] s = {"Hot and Sour Soup", "Veg Corn Soup", "Tomato Soup", "Spicy Tomato Soup"};

String[] b = {"Hot Chocolate Drink", "Badam Drink", "Badam-Pista Drink"};

String str = "Welcome to CCD!\nEnjoy your ";

Scanner sc = new Scanner(System.in);

char ch = sc.next().charAt(0);

int item = sc.nextInt();

if(ch != 'C' &amp;&amp; ch != 'c' &amp;&amp; ch != 'T' &amp;&amp; ch != 't' &amp;&amp; ch != 'B' &amp;&amp; ch != 'b' &amp;&amp; ch != 'S' &amp;&amp; ch != 's')

System.out.println("INVALID OPTION!");

else if(ch == 'C' || ch == 'c') {

if(item 3)

System.out.println("INVALID OPTION!");

else {

System.out.println(str + c[item-1] + "!");

}

}

else if(ch == 'T' || ch == 't') {

if(item 8)

System.out.println("INVALID OPTION!");

else {

System.out.println(str + t[item-1] + "!");

}

}

else if(ch == 'S' || ch == 's') {

if(item 4)

System.out.println("INVALID OPTION!");

else {

System.out.println(str + s[item-1] + "!");

}

}

else if(ch == 'B' || ch == 'b') {

if(item 3)

System.out.println("INVALID OPTION!");

else {

System.out.println(str + b[item-1] + "!");

}

}

}

}\_\_

1. A doctor has a clinic where he serves his patients. The doctor’s consultation fees are different for different groups of patients depending on their age. If the patient’s age is below 17, fees is 200 INR. If the patient’s age is between 17 and 40, fees is 400 INR. If patient’s age is above 40, fees is 300 INR. Write a code to calculate earnings in a day for which one array/List of values representing age of patients visited on that day is passed as input.

Note:

Age should not be zero or less than zero or above 120

Doctor consults a maximum of 20 patients a day

Enter age value (press Enter without a value to stop):

Example 1:

Input

20

30

40

50

2

3

14

Output

Total Income 2000 INR

Note: Input and Output Format should be same as given in the above example.

For any wrong input display INVALID INPUT

Output Format

Total Income 2100 INR

1. To check whether a year is leap or not

Step 1:

We first divide the year by 4.

If it is not divisible by 4 then it is not a leap year.

If it is divisible by 4 leaving remainder 0

Step 2:

We divide the year by 100

If it is not divisible by 100 then it is a leap year.

If it is divisible by 100 leaving remainder 0

Step 3:

We divide the year by 400

If it is not divisible by 400 then it is a leap year.

If it is divisible by 400 leaving remainder 0

Then it is a leap year

/\*Java program to check whether a year entered by user is a leap year or not and a leap year is a year

which is completely divisible by 4,but the year should not be a century year except it is divisible by 400\*/

import java.util.Scanner;

public class Main

{

public static void main(String[] args)

{

//scanner class declaration

Scanner sc=new Scanner(System.in);

//input year from user

System.out.println("Enter a Year");

int year = sc.nextInt();

//condition for checking year entered by user is a leap year or not

if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)

System.out.println(year + " is a leap year.");

else

System.out.println(year + " is not a leap year.");

}

}

1. Prime Numbers with a Twist

Ques. Write a code to check whether no is prime or not. Condition use function check() to find whether entered no is positive or negative ,if negative then enter the no, And if yes pas no as a parameter to prime() and check whether no is prime or not?

Whether the number is positive or not, if it is negative then print the message “please enter the positive number”

It is positive then call the function prime and check whether the take positive number is prime or not.

/\*Java program to check whether a number entered by user is prime or not for only positive numbers,

if the number is negative then ask the user to re-enter the number\*/

//Prime number is a number which is divisible by 1 and another by itself only.

import java.util.Scanner;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

//input a number from user

System.out.println("Enter the number to be checked : ");

int n = sc.nextInt();

//create object of class CheckPrime

Main ob=new Main();

//calling function with value n, as parameter

ob.check(n);

}

//function for checking number is positive or negative

void check(int n)

{

if(n<0)

System.out.println("Please enter a positive integer");

else

prime(n);

}

//function for checking number is prime or not

void prime(int n)

{

int c=0;

for(int i=2;i<n;i++)

{

if(n%i==0)

++c;

}

if(c>=1)

System.out.println("Entered number is not a prime number");

else

System.out.println("Entered number is a prime number");

}

}

1. Number Series with a Twist – 1

Find the 15th term of the series?

0,0,7,6,14,12,21,18, 28

Explanation : In this series the odd term is increment of 7 {0, 7, 14, 21, 28, 35 – – – – – – }

And even term is a increment of 6 {0, 6, 12, 18, 24, 30 – – – – – – }

//Java program to find 15th element of the series

class Main

{

public static void main(String[] args)

{

int a = 7, b = 0,c;

System.out.println("Series :");

for(int i = 1 ; i < 8 ; i++)

{

c = a \* b;

System.out.print(c+" "+(c-b)+" ");

b++;

}

c = a \* b;

System.out.println(c);

System.out.print("15th element of the series is = "+c);

}

}

Output :

Series :

0 0 7 6 14 12 21 18 28 24 35 30 42 36 49

15th element of the series is = 49

1. Number Series with a Twist 2

Link to this Question

Consider the following series: 1, 1, 2, 3, 4, 9, 8, 27, 16, 81, 32, 243, 64, 729, 128, 2187 …

This series is a mixture of 2 series – all the odd terms in this series form a geometric series and all the even terms form yet another geometric series. Write a program to find the Nth term in the series.

The value N in a positive integer that should be read from STDIN. The Nth term that is calculated by the program should be written to STDOUT. Other than value of n th term,no other character / string or message should be written to STDOUT. For example , if N=16, the 16th term in the series is 2187, so only value 2187 should be printed to STDOUT.

You can assume that N will not exceed 30.

//Java program to find nth element of the series

import java.util.Scanner;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

//input value of n

System.out.print("Enter the value of n : ");

int n = sc.nextInt();

int a = 1, b = 1;

//statement for even value of n

if(n % 2 == 0)

{

for(int i = 1 ; i <= (n-2) ; i = i+2)

{

a = a \* 2;

b = b \* 3;

}

System.out.print(n+" element of the series is = "+b);

}

//statement for odd value of n

else

{

for(int i = 1 ; i < (n-2) ; i = i+2)

{

a = a \* 2;

b = b \* 3;

}

a = a \* 2;

System.out.print(n+" element of the series is = "+a);

}

}

}

Output :

Enter the value of n : 14

14 element of the series is = 729

1. Number Series with a Twist 3

Link to this Question –

Consider the below series :

0, 0, 2, 1, 4, 2, 6, 3, 8, 4, 10, 5, 12, 6, 14, 7, 16, 8

This series is a mixture of 2 series all the odd terms in this series form even numbers in ascending order and every even terms is derived from the previous term using the formula (x/2)

Write a program to find the nth term in this series.

The value n in a positive integer that should be read from STDIN the nth term that is calculated by the program should be written to STDOUT. Other than the value of the nth term no other characters /strings or message should be written to STDOUT.

For example if n=10,the 10 th term in the series is to be derived from the 9th term in the series. The 9th term is 8 so the 10th term is (8/2)=4. Only the value 4 should be printed to STDOUT.

You can assume that the n will not exceed 20,000.

//Java program to find nth element of the series

import java.util.Scanner;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int a = 0, b = 0;

if(n % 2 == 0)

{

for(int i = 1 ; i <= (n-2) ; i = i+2)

{

a = a + 2;

b = a / 2;

}

System.out.print(b);

}

else

{

for(int i = 1 ; i < (n-2) ; i = i+2)

{

a = a + 2;

b = a / 2;

}

a = a + 2;

System.out.print(a);

}

}

}

1. String with a Twist

Link to this Questions

1. The program will recieve 3 English words inputs from STDIN

These three words will be read one at a time, in three separate line

The first word should be changed like all vowels should be replaced by %

The second word should be changed like all consonants should be replaced by #

The third word should be changed like all char should be converted to upper case

Then concatenate the three words and print them

Other than these concatenated word, no other characters/string should or message should be written to STDOUT

For example if you print how are you then output should be h%wa#eYOU.

You can assume that input of each word will not exceed more than 5 chars

import java.util.\*;

public class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter three words : ");

String s1 = sc.next();

String s2 = sc.next();

String s3 = sc.next();

int l1 = s1.length();

int l2 = s2.length();

String str1 = "";

String str2 = "";

String str3 = "";

char c;

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

{

c = s1.charAt(i);

if(c == 'A' || c == 'a' || c == 'E' ||

c == 'e' || c == 'I' || c == 'i' || c == 'O' || c == 'o' || c == 'U' || c == 'u')

str1 = str1 + "%";

else

str1 = str1 + c;

}

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

{

c = s2.charAt(i);

if((c >= 'A' && c <= 'Z')||(c >= 'a' && c <= 'z'))

{

if(c == 'A' || c == 'a' || c == 'E' || c == 'e' ||

c == 'I' || c == 'i' || c == 'O' || c == 'o' || c == 'U' || c == 'u')

str2 = str2 + c;

else

str2 = str2 + "#";

}

else

str2 = str2 + c;

}

str3 = s3.toUpperCase();

System.out.println(str1+str2+str3);

}

}

23. Addition of two numbers a Twist

1. Using a method, pass two variables and find the sum of two numbers.

Test case:

Number 1 – 20

Number 2 – 20.38

Sum = 40.38

There were a total of 4 test cases. Once you compile 3 of them will be shown to you and 1 will be a hidden one. You have to display error message if numbers are not numeric.

import java.util.Scanner;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Number 1 : ");

int num1 = sc.nextInt();

System.out.print("Number 2 : ");

float num2 = sc.nextFloat();

float sum = num1 + num2;

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

}

}

1. Consider the below series :

0, 0, 2, 1, 4, 2, 6, 3, 8, 4, 10, 5, 12, 6, 14, 7, 16, 8

This series is a mixture of 2 series all the odd terms in this series form even numbers in ascending order and every even terms is derived from the previous term using the formula (x/2)

Write a program to find the nth term in this series.

The value n in a positive integer that should be read from STDIN the nth term that is calculated by the program should be written to STDOUT. Other than the value of the nth term no other characters /strings or message should be written to STDOUT.

For example if n=10,the 10 th term in the series is to be derived from the 9th term in the series. The 9th term is 8 so the 10th term is (8/2)=4. Only the value 4 should be printed to STDOUT.

You can assume that the n will not exceed 20,000.

1. Given a maximum of four digit to the base 17(10 -> A, 11 -> B, 12 -> C, 16 -> G) as input, output its decimal value.

Input:

23GF

import java.util.\*;

class Main

{

public static void main(String[] args) {

HashMap<Character,Integer> hmap = new HashMap<Character,Integer>();

hmap.put('A',10);

hmap.put('B',11);

hmap.put('C',12);

hmap.put('D',13);

hmap.put('E',14);

hmap.put('F',15);

hmap.put('G',16);

hmap.put('a',10);

hmap.put('b',11);

hmap.put('c',12);

hmap.put('d',13);

hmap.put('e',14);

hmap.put('f',15);

hmap.put('g',16);

Scanner sin = new Scanner(System.in);

String s = sin.nextLine();

long num=0;

int k=0;

for(int i=s.length()-1;i>=0;i--)

{

if((s.charAt(i)>='A'&&s.charAt(i)<='Z')||(s.charAt(i)>='a' &&s.charAt(i)<='z'))

{

num = num + hmap.get(s.charAt(i))\*(int)Math.pow(17,k++);

}

else

{

num = num+((s.charAt(i)-'0')\*(int)Math.pow(17,k++));

}

}

System.out.println(num);

}

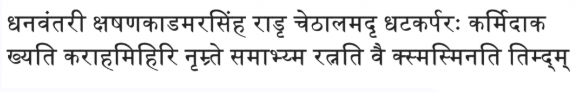
}

Output

10980

**A Sober Walk**

1. Our hoary culture had several great persons since time immemorial and king vikramaditya’s nava ratnas (nine gems) belongs to this ilk.They are named in the following shloka:



Among these, Varahamihira was an astrologer of eminence and his book Brihat Jataak is recokened as the ultimate authority in astrology. He was once talking with Amarasimha,another gem among the nava ratnas and the author of Sanskrit thesaurus, Amarakosha. Amarasimha wanted to know the final position of a person, who starts from the origin 0 0 and travels per following scheme.

* He first turns and travels 10 units of distance
* His second turn is upward for 20 units
* Third turn is to the left for 30 units
* Fourth turn is the downward for 40 units
* Fifth turn is to the right(again) for 50 units

… And thus he travels, every time increasing the travel distance by 10 units.

**Constraints:**

2<=n<=1000

**Input:**

3

import java.util.\*;

import java.lang.\*;

class Main {

public static void main (String[] args) {

Scanner sc = new Scanner(System.in);

int n=sc.nextInt();

char c = 'R';

int x = 0, y = 0;

while(n>0){

switch(c){

case 'R':

x = Math.abs(x) + 10;

y = Math.abs(y);

c ='U';

break;

case 'U':

y = y + 20;

c = 'L';

break;

case 'L':

x = -(x + 10);

c = 'D';

break;

case 'D':

y = -(y);

c = 'R';

break;

}

n--;

}

System.out.println(x+" "+y);

}

}

1. Word is the key

One programming language has the following keywords that cannot be used as identifiers:

break, case, continue, default, defer, else, for, func, goto, if, map, range, return, struct, type, var

Write a program to find if the given word is a keyword or not

Input #1:

defer

Output:

defer is a keyword

Input #2:

While

import java.util.Scanner;

class Main

{

public static void main(String args[])

{

String str[]= {"break", "case", "continue", "default", "defer", "else","for", "func", "goto",

"if", "map", "range", "return", "struct", "type", "var"};

int flag = 0;

Scanner sc = new Scanner(System.in);

String input=sc.nextLine();

for(int i = 0; i<16;i++){

if(str[i].equals(input)){

flag = 1;

break;

}

}

if(flag==1){

System.out.println(input+" is a keyword");

}

else{

System.out.println(input+" is not a keyword");

}

}

}

Output

while is not a keyword

1. Jar of Candies

There is a jar full of candies for sale at a mall counter. The jar has the capacity N, that is JAR can contain maximum N Candies when a JAR is full. At any point in time, JAR can have an M number of candies where M<=N. Candies are served to the customers. JAR is never remaining empty as when the last K candidates are left, JAR is refilled with new candidates in such a way that JAR gets full.

Write the code to implement the above scenario. Display JAR at the counter with the available number of candies.

Input should be the number of candies one customer orders at a point in time. Update the JAR after every purchase and display JAR at the counter. The output should give the number of candies sold and the updated number of candies in the JAR. If the input is more than the number of candies in JAR, return “INVALID INPUT”.

Given,

N=10, Where N is the number of candies available, K<=5, Where K is the number of minimum candies that must be inside JAR ever.

Example1: (N=10,K=<5)

Input #1:

3

Output :

Number of Candies Sold: 3

Number of Candies available:7

Input #2:

4

import java.util.\*;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int n=10,k = sc.nextInt();

if(k==0)

{

System.out.println("INVALID INPUT");

System.out.println("NUMBER OF CANDIES AVAILABLE: "+n);

}

else

{

System.out.println("NUMBER OF CANDIES SOLD: "+k);

System.out.println("NUMBER OF CANDIES AVAILABLE: "+(n-k));

}

sc.close();

}

}

Output

NUMBER OF CANDIES SOLD: 4\n NUMBER OF CANDIES AVAILABLE: 6

1. Oxygen Value

The selection of MPCS exams includes a fitness test which is conducted on the ground. There will be a batch of 3 trainees, appearing for a running test on track for 3 rounds.

You need to record their oxygen level after every round. After trainees are finished with all rounds, calculate for each trainee his average oxygen level over the 3 rounds and select the one with the highest average oxygen level as the fittest trainee. If more than one trainee attains the same highest average level, they all need to be selected. Display the fittest trainee(or trainers) and the highest average oxygen level.

Note:

1.The oxygen value entered should not be accepted if it is not in the range between 1 and 100.

2.If the calculated maximum average oxygen value of the trainees is below 70 then declare the trainees as unfit with a meaningful message as “All trainees are unfit”

3.Average oxygen values should be rounded

Example 1:

Input #1:

95

92

95

92

90

92

90

92

90

Output:

Trainee Number: 1

Trainee Number: 3

Note: Input should be 9 integer values representing oxygen levels entered in order as

Round 1:

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Round 2:

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Round 3:

Oxygen value of trainee 1

Oxygen value of trainee 2

Oxygen value of trainee 3

Oxygen must be in the given format as in the above example. For any wrong input, the final output should display “INVALID INPUT”

Input #2:

91

92

45

92

80

90

90

92

90

import java.util.\*;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int i, x, T1 = 0, T2 = 0, T3 = 0, count = 1;

double A1, A2, A3;

while(count<=9)

{

x=sc.nextInt();

if(x >= 1 && x <= 100)

{

if(count % 3 == 1)

{

T1 = T1 + x;

}

else if(count % 3 == 2)

{

T2 = T2 + x;

}

else

{

T3 = T3 + x;

}

count++;

}

else

{

System.out.println("INVALID INPUT");

count++;

return;

}

}

A1 = Math.round(T1/3);

A2 = Math.round(T2/3);

A3 = Math.round(T3/3);

if(A1 <= 70 && A2 <= 70 && A3 <= 70)

{

System.out.println("All trainees are unfit");

return;

}

if(A1 >= A2 && A1>= A3)

System.out.println("Trainee Number: 1");

if(A2 >= A1 && A2 >= A3)

System.out.println("Trainee Number: 2");

if(A3 >= A1 && A3 >= A2)

System.out.println("Trainee Number: 3");

return;

}

}

Output

Trainee Number: 1

1. To Zero or Not Zero

Given a pair of positive integers m and n (m < n; 0 < m < 999; 1 < n < = 999), write a program to smartly affix zeroes, while printing the numbers from m to n.

Example-1

Input

5 10

Expected output

05 06 07 08 09 10

Example-2

Input

9 100

Expected output

009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063 064 065 067 068 069 070 071 072 073 074 075 076 077 078 079 080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095 096 097 098 099 100

Example-3

Input

1 9

import java.util.\*;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int low=sc.nextInt();

int up=sc.nextInt();

for(int i=low;i<=up;i++)

{

if(up>=100)

System.out.printf("%03d ",i);

else if(up>=10)

System.out.printf("%02d ",i);

else

System.out.printf("%d ",i);

}

}

}

Output

* 1. 2 3 4 5 6 7 8 9

1. Oddly Even

Given a maximum of 100 digit numbers as input, find the difference between the sum of odd and even position digits.

Input 1:

4567

Expected output:

2

Explanation

The Sum of odd position digits 4 and 6 is 10. The Sum of even position digits 5 and 7 is 12. The difference is 12-10=2.

Input #2:

9834698765123

import java.util.\*;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

String num = sc.nextLine();

int Osum=0,Esum=0;

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

{

int n = (int)(num.charAt(i)-'0');

if(i%2==0)

Esum+=n;

else

Osum+=n;

}

System.out.println(Math.abs(Esum-Osum));

}

}

Output

1

1. Minting Mints

Problem statement:

It was one of the places, where people need to get their provisions only through fair price (“ration”) shops. As the elder had domestic and official work to attend to, their wards were asked to buy the items from these shops. Needless to say, there was a long queue of boys and girls. To minimize the tedium of standing in the serpentine queue, the kids were given mints. I went to the last boy in the queue and asked him how many mints he has. He said that the number of mints he has is one less than the sum of all the mints of kids standing before him in the queue. So I went to the penultimate kid to know how many mints she has.

She said that if I add all the mints of kids before her and subtract one from it, the result equals the mints she has. It seemed to be a uniform response from everyone. So, I went to the boy at the head of the queue consoling myself that he would not give the same response as others. He said, “I have four mints”.

Given the number of first kid’s mints (n) and the length (len) of the queue as input, write a program to display the total number of mints with all the kids.

constraints:

2<n<10

1<len<20

Input#1:

4 2

Output:

7

Input#2:

14 4

import java.util.\*;

class Main

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int s = sc.nextInt();

int n = sc.nextInt();

int sum=s,prev;

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

{

prev=sum-1;

sum+=prev;

}

System.out.println(sum);

}

}

Output

105

1. At an exam center, M number of students are allocated for one classroom as per the University Rules. The Examination staff has made sitting arrangements where the classroom contains N number of benches arranged in columns separated by a suitable distance occupying room space from the left to the right wall. Given, M = 10 , N=5

Students from class “TY” can sit one after the other from the 1st bench starting at the left wall in the order of their Roll Numbers. And students from class “SY” are allowed to sit beside the students from class ‘TY” in the order of their Roll Numbers one after the other. All students enter the classroom in a random order as input in String Array Students[ ]. Few students may remain absent. Assume the Roll Numbers are in continuous range with no drop, and Class Name(“TY”,”SY”) should be prefixed for every Roll Number. For example, you can pass Input

with values like:

Students = [“TY01”, “TY02”, “SY01”, “SY05”, “SY04”, “TY03”, “SY02”, “TY04”, “SY03”, “TY05”]

Display the sitting arrangement status at the exam time. “ABSENT” should be marked at the place of the Roll Numbers of missing or absent students.

Example 1:

Input Values(Input format should be same as below)

TY01

TY02

SY01

SY05

SY04

TY03

SY02

TY04

SY03

Output Values

[TY01][SY01]

[TY02][SY02]

[TY03][SY03]

[TY04][SY04]

[ABSENT][SY05]

Note: Output should be in the format given in above example. If input values are more than M, display INVALID INPUT. If input value contains other than class SY or TY display INVALID INPUT.

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Question

{

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

{

// your code goes here

Scanner sc = new Scanner(System.in);

String[][] arr = new String[5][2];

for(int i=0; i&lt;5; i++)

for(int j=0; j5) {

System.out.println("INVALID INPUT");

return;

}

if(s.charAt(0) != 'T' &amp;&amp; s.charAt(0) != 'S'){

System.out.println("INVALID INPUT");

return;

}

r = s.charAt(3) - '1';

if(s.charAt(0) == 'T')

c = 0;

else

c = 1;

arr[r] = s;

try {

s = sc.nextLine();

} catch(Exception e) {

break;

}

}

for(int i=0; i&lt;5; i++)

System.out.println(&quot;[&quot; + arr[i][0]+ &quot;][&quot;+arr[i][1]+&quot;]&quot;);

}

}

<https://www.geeksforgeeks.org/competitive-programming-a-complete-guide/>

1. Given an array arr[] of size n, its prefix sum array is another array prefixSum[] of the same size, such that the value of prefixSum[i] is arr[0] + arr[1] + arr[2] … arr[i].

Input : arr[] = {10, 20, 10, 5, 15}

Output : prefixSum[] = {10, 30, 40, 45, 60}

Explanation : While traversing the array, update

the element by adding it with its previous element.

prefixSum[0] = 10,

prefixSum[1] = prefixSum[0] + arr[1] = 30,

prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.

// Java Program for Implementing

// prefix sum arrayclass

class Prefix {

// Fills prefix sum array

static void fillPrefixSum(int arr[], int n,

int prefixSum[])

{

prefixSum[0] = arr[0];

// Adding present element

// with previous element

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

prefixSum[i] = prefixSum[i - 1] + arr[i];

}

// Driver code

public static void main(String[] args)

{

int arr[] = { 10, 4, 16, 20 };

int n = arr.length;

int prefixSum[] = new int[n];

fillPrefixSum(arr, n, prefixSum);

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

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

System.out.println("");

}

}

10 14 30 50

2. Given an array arr[] of size n. Given Q queries and in each query given L and R, print sum of array elements from index L to R.

**Example:**

**Input :** n = 6

a[ ] = {3, 6, 2, 8, 9, 2}

q = 4

l = 2, r = 3.

l = 4, r = 6.

l = 1, r = 5.

l = 3, r = 6.

**Output :** 8

19

28

21

**Time Complexity:** O(n)

import java.util.\*;

class GFG {

public static void main(String[] args)

{

int n = 6;

int[] a = { 3, 6, 2, 8, 9, 2 };

int[] pf = new int[n + 2];

pf[0] = 0;

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

pf[i + 1] = pf[i] + a[i];

}

int[][] q

= { { 2, 3 }, { 4, 6 }, { 1, 5 }, { 3, 6 } };

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

int l = q[i][0];

int r = q[i][1];

// Calculating sum from r to l.

System.out.print(pf[r] - pf[l - 1] + "\n");

}

}

}

3. Equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in an array A:

***Input****: A[] = {-7, 1, 5, 2, -4, 3, 0}****Output****: 3   
3 is an equilibrium index, because:   
A[0] + A[1] + A[2] = A[4] + A[5] + A[6]*

***Input****: A[] = {1, 2, 3}****Output****: -1*

*To handle all the testcase, we can use binary search algorithm.*

*1.calculate the mid and then create left sum and right sum around mid*

*2.if left sum is greater than right sum, move to left until it become equal or less than right sum*

*3. else if right sum is greater than left, move right until it become equal or less than left sum.*

*4. finally we compare two sums if they are equal we got mid as index else its -1*

// Java program for the above approach

import java.io.\*;

import java.util.\*;

class GFG{

static void find(int arr[], int n)

{

int mid = n / 2;

int leftSum = 0, rightSum = 0;

//calculation sum to left of mid

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

{

leftSum += arr[i];

}

//calculating sum to right of mid

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

{

rightSum += arr[i];

}

//if rightsum > leftsum

if (rightSum > leftSum)

{

//we keep moving right until rightSum become equal or less than leftSum

while (rightSum > leftSum && mid < n - 1)

{

rightSum -= arr[mid + 1];

leftSum += arr[mid];

mid++;

}

}

else

{

//we keep moving right until leftSum become equal or less than RightSum

while (leftSum > rightSum && mid > 0)

{

rightSum += arr[mid];

leftSum -= arr[mid - 1];

mid--;

}

}

//check if both sum become equal

if (rightSum == leftSum)

{

System.out.print("First Point of equilibrium is at index ="+ mid);

return;

}

System.out.print("First Point of equilibrium is at index =" + -1);

}

// Driver code

public static void main(String args[])

{

int arr[] = { 1,1,1,-1,1,1,1 };

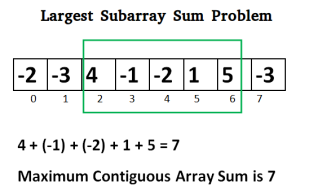
int n = arr.length;

find(arr, n);

}

}

4. Write an efficient program to find the sum of contiguous subarray within a one-dimensional array of numbers that has the largest sum.



The simple idea of Kadane’s algorithm is to look for all positive contiguous segments of the array (max\_ending\_here is used for this). And keep track of maximum sum contiguous segment among all positive segments (max\_so\_far is used for this). Each time we get a positive-sum compare it with max\_so\_far and update max\_so\_far if it is greater than max\_so\_far

Lets take the example:

{-2, -3, 4, -1, -2, 1, 5, -3}

max\_so\_far = max\_ending\_here = 0

for i=0, a[0] = -2

max\_ending\_here = max\_ending\_here + (-2)

Set max\_ending\_here = 0 because max\_ending\_here < 0

for i=1, a[1] = -3

max\_ending\_here = max\_ending\_here + (-3)

Set max\_ending\_here = 0 because max\_ending\_here < 0

for i=2, a[2] = 4

max\_ending\_here = max\_ending\_here + (4)

max\_ending\_here = 4

max\_so\_far is updated to 4 because max\_ending\_here greater

than max\_so\_far which was 0 till now

for i=3, a[3] = -1

max\_ending\_here = max\_ending\_here + (-1)

max\_ending\_here = 3

for i=4, a[4] = -2

max\_ending\_here = max\_ending\_here + (-2)

max\_ending\_here = 1

for i=5, a[5] = 1

max\_ending\_here = max\_ending\_here + (1)

max\_ending\_here = 2

for i=6, a[6] = 5

max\_ending\_here = max\_ending\_here + (5)

max\_ending\_here = 7

max\_so\_far is updated to 7 because max\_ending\_here is

greater than max\_so\_far

for i=7, a[7] = -3

max\_ending\_here = max\_ending\_here + (-3)

max\_ending\_here = 4

import java.io.\*;

// Java program to print largest contiguous array sum

import java.util.\*;

class Kadane

{

public static void main (String[] args)

{

int [] a = {-2, -3, 4, -1, -2, 1, 5, -3};

System.out.println("Maximum contiguous sum is " +

maxSubArraySum(a));

}

static int maxSubArraySum(int a[])

{

int size = a.length;

int max\_so\_far = Integer.MIN\_VALUE, max\_ending\_here = 0;

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

{

max\_ending\_here = max\_ending\_here + a[i];

if (max\_so\_far < max\_ending\_here)

max\_so\_far = max\_ending\_here;

if (max\_ending\_here < 0)

max\_ending\_here = 0;

}

return max\_so\_far;

}

}

**Output:**

Maximum contiguous sum is 7

5. Job Sequencing Problem

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

**Examples:**

Input: Five Jobs with following

deadlines and profits

JobID Deadline Profit

a 2 100

b 1 19

c 2 27

d 1 25

e 3 15

Output: Following is maximum

profit sequence of jobs

c, a, e

*1) Sort all jobs in decreasing order of profit.   
2) Iterate on jobs in decreasing order of profit.For each job , do the following :   
a)Find a time slot i, such that slot is empty and i < deadline and i is greatest.Put the job in   
this slot and mark this slot filled.   
b)If no such i exists, then ignore the job.*

import java.util.\*;

class Job

{

    // Each job has a unique-id,

    // profit and deadline

    char id;

    int deadline, profit;

    // Constructors

    public Job() {}

    public Job(char id, int deadline, int profit)

    {

        this.id = id;

        this.deadline = deadline;

        this.profit = profit;

    }

    // Function to schedule the jobs take 2

    // arguments arraylist and no of jobs to schedule

    void printJobScheduling(ArrayList<Job> arr, int t)

    {

        // Length of array

        int n = arr.size();

        // Sort all jobs according to

        // decreasing order of profit

        Collections.sort(arr,

                         (a, b) -> b.profit - a.profit);

        // To keep track of free time slots

        boolean result[] = new boolean[t];

        // To store result (Sequence of jobs)

        char job[] = new char[t];

        // Iterate through all given jobs

        for (int i = 0; i < n; i++)

        {

            // Find a free slot for this job

            // (Note that we start from the

            // last possible slot)

            for (int j

                 = Math.min(t - 1, arr.get(i).deadline - 1);

                 j >= 0; j--) {

                // Free slot found

                if (result[j] == false)

                {

                    result[j] = true;

                    job[j] = arr.get(i).id;

                    break;

                }

            }

        }

        // Print the sequence

        for (char jb : job)

        {

            System.out.print(jb + " ");

        }

        System.out.println();

    }

    // Driver code

    public static void main(String args[])

    {

        ArrayList<Job> arr = new ArrayList<Job>();

        arr.add(new Job('a', 2, 100));

        arr.add(new Job('b', 1, 19));

        arr.add(new Job('c', 2, 27));

        arr.add(new Job('d', 1, 25));

        arr.add(new Job('e', 3, 15));

        // Function call

        System.out.println("Following is maximum "

                           + "profit sequence of jobs");

        Job job = new Job();

        // Calling function

        job.printJobScheduling(arr, 3);

    }

}

6. Least prime factor of numbers till n

Given a number n, print least prime factors of all numbers from 1 to n. The least prime factor of an integer n is the smallest prime number that divides the number. The least prime factor of all even numbers is 2. A prime number is its own least prime factor (as well as its own greatest prime factor).

Note: We need to print 1 for 1.

Example :

Input : 6

Output : Least Prime factor of 1: 1

Least Prime factor of 2: 2

Least Prime factor of 3: 3

Least Prime factor of 4: 2

Least Prime factor of 5: 5

Least Prime factor of 6: 2

Create a list of consecutive integers from 2 through n: (2, 3, 4, …, n).

Initially, let i equal 2, the smallest prime number.

Enumerate the multiples of i by counting to n from 2i in increments of i, and mark them as having least prime factor as i (if not already marked). Also mark i as least prime factor of i (i itself is a prime number).

Find the first number greater than i in the list that is not marked. If there was no such number, stop. Otherwise, let i now equal this new number (which is the next prime), and repeat from step 3.

// Java program to print the least prime factors

// of numbers less than or equal to

// n using modified Sieve of Eratosthenes

import java.io.\*;

import java.util.\*;

class GFG

{

public static void leastPrimeFactor(int n)

{

// Create a vector to store least primes.

// Initialize all entries as 0.

int[] least\_prime = new int[n+1];

// We need to print 1 for 1.

least\_prime[1] = 1;

for (int i = 2; i <= n; i++)

{

// least\_prime[i] == 0

// means it i is prime

if (least\_prime[i] == 0)

{

// marking the prime number

// as its own lpf

least\_prime[i] = i;

// mark it as a divisor for all its

// multiples if not already marked

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

if (least\_prime[j] == 0)

least\_prime[j] = i;

}

}

// print least prime factor of

// of numbers till n

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

System.out.println("Least Prime factor of " +

+ i + ": " + least\_prime[i]);

}

public static void main (String[] args)

{

int n = 10;

leastPrimeFactor(n);

}

}

// Code Contributed by Mohit Gupta\_OMG <(0\_o)>

Output

Least Prime factor of 1: 1

Least Prime factor of 2: 2

Least Prime factor of 3: 3

Least Prime factor of 4: 2

Least Prime factor of 5: 5

Least Prime factor of 6: 2

Least Prime factor of 7: 7

Least Prime factor of 8: 2

Least Prime factor of 9: 3

Least Prime factor of 10: 2

Time Complexity: O(nloglog(n))

Auxiliary Space: O(n)

7. Partition problem

Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is the same.

Examples:

arr[] = {1, 5, 11, 5}

Output: true

The array can be partitioned as {1, 5, 5} and {11}

arr[] = {1, 5, 3}

Output: false

The array cannot be partitioned into equal sum sets.

Following are the two main steps to solve this problem:

1) Calculate sum of the array. If sum is odd, there can not be two subsets with equal sum, so return false.

2) If sum of array elements is even, calculate sum/2 and find a subset of array with sum equal to sum/2.

The first step is simple. The second step is crucial, it can be solved either using recursion or Dynamic Programming.

import java.io.\*;

class Partition {

// A utility function that returns true if there is a

// subset of arr[] with sun equal to given sum

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

{

// Base Cases

if (sum == 0)

return true;

if (n == 0 && sum != 0)

return false;

// If last element is greater than sum, then ignore

// it

if (arr[n - 1] > sum)

return isSubsetSum(arr, n - 1, sum);

/\* else, check if sum can be obtained by any of

the following

(a) including the last element

(b) excluding the last element

\*/

return isSubsetSum(arr, n - 1, sum)

|| isSubsetSum(arr, n - 1, sum - arr[n - 1]);

}

// Returns true if arr[] can be partitioned in two

// subsets of equal sum, otherwise false

static boolean findPartition(int arr[], int n)

{

// Calculate sum of the elements in array

int sum = 0;

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

sum += arr[i];

// If sum is odd, there cannot be two subsets

// with equal sum

if (sum % 2 != 0)

return false;

// Find if there is subset with sum equal to half

// of total sum

return isSubsetSum(arr, n, sum / 2);

}

// Driver code

public static void main(String[] args)

{

int arr[] = { 3, 1, 5, 9, 12 };

int n = arr.length;

// Function call

if (findPartition(arr, n) == true)

System.out.println("Can be divided into two "

+ "subsets of equal sum");

else

System.out.println(

"Can not be divided into "

+ "two subsets of equal sum");

}

}

Can be divided into two subsets of equal sum

8. Shortest Common Supersequence

Given two strings str1 and str2, the task is to find the length of the shortest string that has both str1 and str2 as subsequences.

Examples :

Input: str1 = "geek", str2 = "eke"

Output: 5

Explanation:

String "geeke" has both string "geek"

and "eke" as subsequences.

Input: str1 = "AGGTAB", str2 = "GXTXAYB"

Output: 9

Explanation:

String "AGXGTXAYB" has both string

"AGGTAB" and "GXTXAYB" as subsequences.

A simple analysis yields below simple recursive solution.

Let X[0..m - 1] and Y[0..n - 1] be two

strings and m and n be respective

lengths.

if (m == 0) return n;

if (n == 0) return m;

// If last characters are same, then

// add 1 to result and

// recur for X[]

if (X[m - 1] == Y[n - 1])

return 1 + SCS(X, Y, m - 1, n - 1);

// Else find shortest of following two

// a) Remove last character from X and recur

// b) Remove last character from Y and recur

else

return 1 + min( SCS(X, Y, m - 1, n), SCS(X, Y, m, n - 1) );

Below is simple naive recursive solution based on above recursive formula.

// A dynamic programming based Java program to

// find length of the shortest supersequence

class GFG {

// Returns length of the shortest

// supersequence of X and Y

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

{

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

// Fill table in bottom up manner

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

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

// Below steps follow above recurrence

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];

else

dp[i][j] = 1

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

dp[i][j - 1]);

}

}

return dp[m][n];

}

// Driver Code

public static void main(String args[])

{

String X = "AGGTAB";

String Y = "GXTXAYB";

System.out.println(

"Length of the shortest supersequence is "

+ superSeq(X, Y, X.length(), Y.length()));

}

}

// This article is contributed by Sumit Ghosh

Output:

Length of the shortest supersequence is 9

9. Check whether a given point lies inside a triangle or not

Given three corner points of a triangle, and one more point P. Write a function to check whether P lies within the triangle or not.

For example, consider the following program, the function should return true for P(10, 15) and false for P'(30, 15)

Let the coordinates of three corners be (x1, y1), (x2, y2) and (x3, y3). And coordinates of the given point P be (x, y)

1) Calculate area of the given triangle, i.e., area of the triangle ABC in the above diagram. Area A = [ x1(y2 – y3) + x2(y3 – y1) + x3(y1-y2)]/2

2) Calculate area of the triangle PAB. We can use the same formula for this. Let this area be A1.

3) Calculate area of the triangle PBC. Let this area be A2.

4) Calculate area of the triangle PAC. Let this area be A3.

5) If P lies inside the triangle, then A1 + A2 + A3 must be equal to A.

import java.util.\*;

class GFG {

/\* A utility function to calculate area of triangle

formed by (x1, y1) (x2, y2) and (x3, y3) \*/

static double area(int x1, int y1, int x2, int y2,

int x3, int y3)

{

return Math.abs((x1\*(y2-y3) + x2\*(y3-y1)+

x3\*(y1-y2))/2.0);

}

/\* A function to check whether point P(x, y) lies

inside the triangle formed by A(x1, y1),

B(x2, y2) and C(x3, y3) \*/

static boolean isInside(int x1, int y1, int x2,

int y2, int x3, int y3, int x, int y)

{

/\* Calculate area of triangle ABC \*/

double A = area (x1, y1, x2, y2, x3, y3);

/\* Calculate area of triangle PBC \*/

double A1 = area (x, y, x2, y2, x3, y3);

/\* Calculate area of triangle PAC \*/

double A2 = area (x1, y1, x, y, x3, y3);

/\* Calculate area of triangle PAB \*/

double A3 = area (x1, y1, x2, y2, x, y);

/\* Check if sum of A1, A2 and A3 is same as A \*/

return (A == A1 + A2 + A3);

}

/\* Driver program to test above function \*/

public static void main(String[] args)

{

/\* Let us check whether the point P(10, 15)

lies inside the triangle formed by

A(0, 0), B(20, 0) and C(10, 30) \*/

if (isInside(0, 0, 20, 0, 10, 30, 10, 15))

System.out.println("Inside");

else

System.out.println("Not Inside");

}

}

Inside

Time Complexity: O(1)

10. Program for nth Catalan Number

Catalan numbers are a sequence of natural numbers that occurs in many interesting counting problems like following.

Count the number of expressions containing n pairs of parentheses which are correctly matched. For n = 3, possible expressions are ((())), ()(()), ()()(), (())(), (()()).

Count the number of possible Binary Search Trees with n keys (See this)

Count the number of full binary trees (A rooted binary tree is full if every vertex has either two children or no children) with n+1 leaves.

Given a number n, return the number of ways you can draw n chords in a circle with 2 x n points such that no 2 chords intersect.

See this for more applications.

The first few Catalan numbers for n = 0, 1, 2, 3, … are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, …

class GFG {

// A dynamic programming based function to find nth

// Catalan number

static int catalanDP(int n)

{

// Table to store results of subproblems

int catalan[] = new int[n + 2];

// Initialize first two values in table

catalan[0] = 1;

catalan[1] = 1;

// Fill entries in catalan[]

// using recursive formula

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

catalan[i] = 0;

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

catalan[i]

+= catalan[j] \* catalan[i - j - 1];

}

}

// Return last entry

return catalan[n];

}

// Driver code

public static void main(String[] args)

{

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

System.out.print(catalanDP(i) + " ");

}

}

}

// This code contributed by Rajput-Ji

Output

1 1 2 5 14 42 132 429 1430 4862

Time Complexity: Time complexity of above implementation is O(n2)

11. Unbounded Binary Search Example (Find the point where a monotonically increasing function becomes positive first time)

Given a function ‘int f(unsigned int x)’ which takes a non-negative integer ‘x’ as input and returns an integer as output. The function is monotonically increasing with respect to the value of x, i.e., the value of f(x+1) is greater than f(x) for every input x. Find the value ‘n’ where f() becomes positive for the first time. Since f() is monotonically increasing, values of f(n+1), f(n+2),… must be positive and values of f(n-2), f(n-3), … must be negative.

Find n in O(logn) time, you may assume that f(x) can be evaluated in O(1) time for any input x.

A simple solution is to start from i equals to 0 and one by one calculate the value of f(i) for 1, 2, 3, 4 … etc until we find a positive f(i). This works but takes O(n) time.

Can we apply Binary Search to find n in O(Logn) time? We can’t directly apply Binary Search as we don’t have an upper limit or high index. The idea is to do repeated doubling until we find a positive value, i.e., check values of f() for following values until f(i) becomes positive.

f(0)

f(1)

f(2)

f(4)

f(8)

f(16)

f(32)

....

....

f(high)

Let 'high' be the value of i when f() becomes positive for first time.

Can we apply Binary Search to find n after finding ‘high’? We can apply Binary Search now, we can use ‘high/2’ as low and ‘high’ as high indexes in binary search. The result n must lie between ‘high/2’ and ‘high’.

The number of steps for finding ‘high’ is O(Logn). So we can find ‘high’ in O(Logn) time. What about the time taken by Binary Search between high/2 and high? The value of ‘high’ must be less than 2\*n. The number of elements between high/2 and high must be O(n). Therefore, the time complexity of Binary Search is O(Logn) and the overall time complexity is 2\*O(Logn) which is O(Logn).

import java.util.\*;

class Binary

{

public static int f(int x)

{ return (x\*x - 10\*x - 20); }

// Returns the value x where above

// function f() becomes positive

// first time.

public static int findFirstPositive()

{

// When first value itself is positive

if (f(0) > 0)

return 0;

// Find 'high' for binary search

// by repeated doubling

int i = 1;

while (f(i) <= 0)

i = i \* 2;

// Call binary search

return binarySearch(i / 2, i);

}

// Searches first positive value of

// f(i) where low <= i <= high

public static int binarySearch(int low, int high)

{

if (high >= low)

{

/\* mid = (low + high)/2 \*/

int mid = low + (high - low)/2;

// If f(mid) is greater than 0 and

// one of the following two

// conditions is true:

// a) mid is equal to low

// b) f(mid-1) is negative

if (f(mid) > 0 && (mid == low || f(mid-1) <= 0))

return mid;

// If f(mid) is smaller than or equal to 0

if (f(mid) <= 0)

return binarySearch((mid + 1), high);

else // f(mid) > 0

return binarySearch(low, (mid -1));

}

/\* Return -1 if there is no positive

value in given range \*/

return -1;

}

// driver code

public static void main(String[] args)

{

System.out.print ("The value n where f() "+

"becomes positive first is "+

findFirstPositive());

}

}

The value n where f() becomes positive first is 12

12. Program to check if a given number is Lucky (all digits are different)

Difficulty Level : Basic

Last Updated : 24 Mar, 2021

A number is lucky if all digits of the number are different. How to check if a given number is lucky or not.

Examples:

Input: n = 983

Output: true

All digits are different

Input: n = 9838

Output: false

8 appears twice

class GFG

{

// This function returns true if n is lucky

static boolean isLucky(int n)

{

// Create an array of size 10 and initialize all

// elements as false. This array is used to check

// if a digit is already seen or not.

boolean arr[]=new boolean[10];

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

arr[i] = false;

// Traverse through all digits

// of given number

while (n > 0)

{

// Find the last digit

int digit = n % 10;

// If digit is already seen,

// return false

if (arr[digit])

return false;

// Mark this digit as seen

arr[digit] = true;

// Remove the last digit from number

n = n / 10;

}

return true;

}

// Driver code

public static void main (String[] args)

{

int arr[] = {1291, 897, 4566, 1232, 80, 700};

int n = arr.length;

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

if(isLucky(arr[i]))

System.out.print(arr[i] + " is Lucky \n");

else

System.out.print(arr[i] + " is not Lucky \n");

}

}

// This code is contributed by Anant Agarwal.

Output:

1291 is not Lucky

897 is Lucky

4566 is not Lucky

1232 is not Lucky

80 is Lucky

700 is not Lucky

Time Complexity: O(d) where d is a number of digits in the input number.

13. Count Inversions in an array

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in the reverse order, the inversion count is the maximum.

Formally speaking, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

Example:

Input: arr[] = {8, 4, 2, 1}

Output: 6

Explanation: Given array has six inversions:

(8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1).

Input: arr[] = {3, 1, 2}

Output: 2

Explanation: Given array has two inversions:

(3, 1), (3, 2)

Approach: Traverse through the array, and for every index, find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all index in the array and print the sum.

Algorithm:

Traverse through the array from start to end

For every element, find the count of elements smaller than the current number up to that index using another loop.

Sum up the count of inversion for every index.

Print the count of inversions.

// Java program to count

// inversions in an array

class Test {

static int arr[] = new int[] { 1, 20, 6, 4, 5 };

static int getInvCount(int n)

{

int inv\_count = 0;

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

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

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

inv\_count++;

return inv\_count;

}

// Driver method to test the above function

public static void main(String[] args)

{

System.out.println("Number of inversions are "

+ getInvCount(arr.length));

}

}

Output

Number of inversions are 5

Complexity Analysis:

Time Complexity: O(n^2), Two nested loops are needed to traverse the array from start to end, so the Time complexity is O(n^2)

14. Print the first N terms of the series 6, 28, 66, 120, 190, 276, …

Given a number N, the task is to print the first N terms of the series 6, 28, 66, 120, 190, 276, and so on.

Examples:

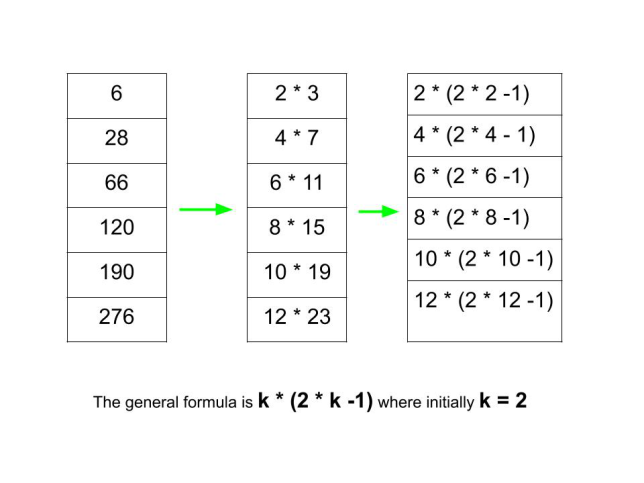
Input: N = 10

Output: 6 28 66 120 190 276 378 496 630 780

Input: N = 4

Output: 6 28 66 120

**Approach:** To solve the problem mentioned above, we have to observe the below pattern:

[](https://media.geeksforgeeks.org/wp-content/uploads/20200623201023/GFG1.jpg)

// Java program for the above approach

class GFG{

// Function to print the series

static void printSeries(int n)

{

// Initialise the value of k with 2

int k = 2;

// Iterate from 1 to n

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

{

// Print each number

System.out.print(k \* (2 \* k - 1) + " ");

// Increment the value of

// K by 2 for next number

k += 2;

}

System.out.println();

}

// Driver code

public static void main(String args[])

{

// Given number N

int N = 12;

// Function Call

printSeries(N);

}

}

// This code is contributed by shivaniisnghss2110

Output:

6 28 66 120 190 276 378 496 630 780 946 1128

Time Complexity: O(N)

15. Number of quadruples where the first three terms are in AP and last three terms are in GP

Given an array arr[] of N integers. The task is to find the number of index quadruples (i, j, k, l) such that a[i], a[j] and a[k] are in AP and a[j], a[k] and a[l] are in GP. All the quadruples have to be distinct.

Examples:

Input: arr[] = {2, 6, 4, 9, 2}

Output: 2

Indexes of elements in the quadruples are (0, 2, 1, 3) and (4, 2, 1, 3) and corresponding quadruples are (2, 4, 6, 9) and (2, 4, 6, 9)

Input: arr[] = {1, 1, 1, 1}

Output: 24

An efficient approach is to use combinatorics to solve the above problem. Initially keep a count of the number of occurrences of every array element. Run two nested loops, and consider both elements to be the second and third number. Hence the first element will be a[j] – (a[k] – a[j]) and the fourth element will be a[k] \* a[k] / a[j] if it is an integer value. Hence the number of quadruples using these two index j and k will be count of first number \* count of fourth number with the second and third element being fixed.

Below is the implementation of the above approach:

import java.util.\*;

class GFG

{

// Function to return the count of quadruples

static int countQuadruples(int a[], int n)

{

// Hash table to count the number of occurrences

HashMap<Integer, Integer> mp = new HashMap<Integer, Integer>();

// Traverse and increment the count

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

if (mp.containsKey(a[i]))

{

mp.put(a[i], mp.get(a[i]) + 1);

}

else

{

mp.put(a[i], 1);

}

int count = 0;

// Run two nested loop for second and third element

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

{

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

{

// If they are same

if (j == k)

continue;

// Initially decrease the count

mp.put(a[j], mp.get(a[j]) - 1);

mp.put(a[k], mp.get(a[k]) - 1);

// Find the first element using common difference

int first = a[j] - (a[k] - a[j]);

// Find the fourth element using GP

// y^2 = x \* z property

int fourth = (a[k] \* a[k]) / a[j];

// If it is an integer

if ((a[k] \* a[k]) % a[j] == 0)

{

// If not equal

if (a[j] != a[k])

{

if (mp.containsKey(first) && mp.containsKey(fourth))

count += mp.get(first) \* mp.get(fourth);

}

// Same elements

else if (mp.containsKey(first) && mp.containsKey(fourth))

count += mp.get(first) \* (mp.get(fourth) - 1);

}

// Later increase the value for

// future calculations

if (mp.containsKey(a[j]))

{

mp.put(a[j], mp.get(a[j]) + 1);

}

else

{

mp.put(a[j], 1);

}

if (mp.containsKey(a[k]))

{

mp.put(a[k], mp.get(a[k]) + 1);

}

else

{

mp.put(a[k], 1);

}

}

}

return count;

}

// Driver code

public static void main(String[] args)

{

int a[] = { 2, 6, 4, 9, 2 };

int n = a.length;

System.out.print(countQuadruples(a, n));

}

}

Output:

2

16. Sum of P terms of an AP if Mth and Nth terms are given

Given Mth and Nth terms of arithmetic progression. The task is to find the sum of its first p terms.

Examples:

Input: m = 6, n = 10, mth = 12, nth = 20, p = 5

Output:30

Input:m = 10, n = 20, mth = 70, nth = 140, p = 4

Output:70

mth term = a + (m-1)d and

nth term = a + (n-1)d

From these two equations, find the value of a and d. Now use the formula of sum of p terms of an AP.

Sum of p terms =

( p \* ( 2\*a + (p-1) \* d ) ) / 2;

Below is the implementation of the above approach:

import java.util.\*;

class GFG

{

// Function to calculate the value of the

static ArrayList<Integer> findingValues(int m, int n,

int mth, int nth)

{

// Calculate value of d using formula

int d = (Math.abs(mth - nth)) /

Math.abs((m - 1) - (n - 1));

// Calculate value of a using formula

int a = mth - ((m - 1) \* d);

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

res.add(a);

res.add(d);

// Return pair

return res;

}

// Function to calculate value sum

// of first p numbers of the series

static int findSum(int m, int n, int mth,

int nth, int p)

{

// First calculate value of a and d

ArrayList<Integer> ad = findingValues(m, n, mth, nth);

int a = ad.get(0);

int d = ad.get(1);

// Calculate the sum by using formula

int sum = (p \* (2 \* a + (p - 1) \* d)) / 2;

// Return the sum

return sum;

}

// Driver Code

public static void main (String[] args)

{

int m = 6, n = 10, mTerm = 12, nTerm = 20, p = 5;

System.out.println(findSum(m, n, mTerm, nTerm, p));

}

}

Output:

30

17. Find the sum of first N terms of the series 2\*3\*5, 3\*5\*7, 4\*7\*9, …

Given an integer N, the task is to find the sum of first N terms of the series:

(2 \* 3 \* 5), (3 \* 5 \* 7), (4 \* 7 \* 9), …

Input: N = 3

Output: 387

S3 = (2 \* 3 \* 5) + (3 \* 5 \* 7) + (4 \* 7 \* 9) = 30 + 105 + 252 = 387

Input: N = 5

Output: 1740

Approach: Let the Nth term of the series be Tn. Sum of the series can be easily found by observing the Nth term of the series:

Tn = {nth term of 2, 3, 4, …} \* {nth term of 3, 5, 7, …} \* {nth term of 5, 7, 9, …}

Tn = (n + 1) \* (2 \* n + 1) \* (2\* n + 3)

Tn = 4n3 + 12n2 + 11n + 3

Sum(Sn) of first n terms can be found by

Sn = ΣTn

Sn = Σ[4n3 + 12n2 + 11n + 3]

Sn = (n / 2) \* [2n3 + 12n2 + 25n + 21]

class GFG {

// Function to return the sum of the

// first n terms of the given series

static int calSum(int n)

{

// As described in the approach

return (n \* (2 \* n \* n \* n + 12 \* n \* n + 25 \* n + 21)) / 2;

}

// Driver Code

public static void main(String args[])

{

int n = 3;

System.out.println(calSum(n));

}

}

Output:

387

18. Chinese Remainder Theorem

We are given two arrays num[0..k-1] and rem[0..k-1]. In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],

x % num[1] = rem[1],

.......................

x % num[k-1] = rem[k-1]

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them. We need to find the minimum possible value of x that produces given remainders.

Examples :

Input: num[] = {5, 7}, rem[] = {1, 3}

Output: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get remainder 1.

(2) When we divide it by 7, we get remainder 3.

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}

Output: 11

Explanation:

11 is the smallest number such that:

(1) When we divide it by 3, we get remainder 2.

(2) When we divide it by 4, we get remainder 3.

(3) When we divide it by 5, we get remainder 1.

The first part is clear that there exists an x. The second part basically states that all solutions (including the minimum one) produce the same remainder when divided by-product of n[0], num[1], .. num[k-1]. In the above example, the product is 3\*4\*5 = 60. And 11 is one solution, other solutions are 71, 131, .. etc. All these solutions produce the same remainder when divided by 60, i.e., they are of form 11 + m\*60 where m >= 0.

A Naive Approach to find x is to start with 1 and one by one increment it and check if dividing it with given elements in num[] produces corresponding remainders in rem[]. Once we find such an x, we return it.

Below is the implementation of Naive Approach.

import java.io.\*;

class GFG {

// k is size of num[] and rem[]. Returns the smallest

// number x such that:

// x % num[0] = rem[0],

// x % num[1] = rem[1],

// ..................

// x % num[k-2] = rem[k-1]

// Assumption: Numbers in num[] are pairwise coprime

// (gcd for every pair is 1)

static int findMinX(int num[], int rem[], int k)

{

int x = 1; // Initialize result

// As per the Chinese remainder theorem,

// this loop will always break.

while (true)

{

// Check if remainder of x % num[j] is

// rem[j] or not (for all j from 0 to k-1)

int j;

for (j=0; j<k; j++ )

if (x%num[j] != rem[j])

break;

// If all remainders matched, we found x

if (j == k)

return x;

// Else try next number

x++;

}

}

// Driver method

public static void main(String args[])

{

int num[] = {3, 4, 5};

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

int k = num.length;

System.out.println("x is " + findMinX(num, rem, k));

}

}

/\*This code is contributed by Nikita Tiwari.\*/

Output :

x is 11

Time Complexity : O(M), M is the product of all elements of num[] array.

19. Sum of all the factors of a number

Given a number n, the task is to find the sum of all the factors.

Examples :

Input : n = 30

Output : 72

Dividers sum 1 + 2 + 3 + 5 + 6 +

10 + 15 + 30 = 72

Input : n = 15

Output : 24

Dividers sum 1 + 3 + 5 + 15 = 24

// Simple Java program to

// find sum of all divisors

// of a natural number

import java.io.\*;

class GFG {

// Function to calculate sum of all

//divisors of a given number

static int divSum(int n)

{

if(n == 1)

return 1;

// Final result of summation

// of divisors

int result = 0;

// find all divisors which divides 'num'

for (int i = 2; i <= Math.sqrt(n); i++)

{

// if 'i' is divisor of 'n'

if (n % i == 0)

{

// if both divisors are same

// then add it once else add

// both

if (i == (n / i))

result += i;

else

result += (i + n / i);

}

}

// Add 1 and n to result as above loop

// considers proper divisors greater

// than 1.

return (result + n + 1);

}

// Driver program to run the case

public static void main(String[] args)

{

int n = 30;

System.out.println(divSum(n));

}

}

20. Longest Increasing Subsequence

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.

Input: arr[] = {3, 10, 2, 1, 20}

Output: Length of LIS = 3

The longest increasing subsequence is 3, 10, 20

Input: arr[] = {3, 2}

Output: Length of LIS = 1

The longest increasing subsequences are {3} and {2}

Input: arr[] = {50, 3, 10, 7, 40, 80}

Output: Length of LIS = 4

The longest increasing subsequence is {3, 7, 40, 80}

Method 1: Recursion.

Optimal Substructure: Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.

Then, L(i) can be recursively written as:

L(i) = 1 + max( L(j) ) where 0 < j < i and arr[j] < arr[i]; or

L(i) = 1, if no such j exists.

To find the LIS for a given array, we need to return max(L(i)) where 0 < i < n.

Formally, the length of the longest increasing subsequence ending at index i, will be 1 greater than the maximum of lengths of all longest increasing subsequences ending at indices before i, where arr[j] < arr[i] (j < i).

Thus, we see the LIS problem satisfies the optimal substructure property as the main problem can be solved using solutions to subproblems.

Method 2: Dynamic Programming.

We can see that there are many subproblems in the above recursive solution which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation.

The simulation of approach will make things clear:

Input : arr[] = {3, 10, 2, 11}

LIS[] = {1, 1, 1, 1} (initially)

Iteration-wise simulation :

arr[2] > arr[1] {LIS[2] = max(LIS [2], LIS[1]+1)=2}

arr[3] < arr[1] {No change}

arr[3] < arr[2] {No change}

arr[4] > arr[1] {LIS[4] = max(LIS [4], LIS[1]+1)=2}

arr[4] > arr[2] {LIS[4] = max(LIS [4], LIS[2]+1)=3}

arr[4] > arr[3] {LIS[4] = max(LIS [4], LIS[3]+1)=3}

lass LIS {

/\* lis() returns the length of the longest

increasing subsequence in arr[] of size n \*/

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

{

int lis[] = new int[n];

int i, j, max = 0;

/\* Initialize LIS values for all indexes \*/

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

lis[i] = 1;

/\* Compute optimized LIS values in

bottom up manner \*/

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;

/\* Pick maximum of all LIS values \*/

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

if (max < lis[i])

max = lis[i];

return max;

}

public static void main(String args[])

{

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

int n = arr.length;

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

+ "\n");

}

}

/\*This code is contributed by Rajat Mishra\*/

Output

Length of lis is 5

Complexity Analysis:

Time Complexity: O(n2).

As nested loop is used.