# Prime Generator

## Problem code: PRIME1

Peter wants to generate some prime numbers for his cryptosystem. Help him! Your task is to generate all prime numbers between two given numbers!

### Input

The input begins with the number t of test cases in a single line (t<=10). In each of the next t lines there are two numbers m and n (1 <= m <= n <= 1000000000, n-m<=100000) separated by a space.

### Output

For every test case print all prime numbers p such that m <= p <= n, one number per line, test cases separated by an empty line.

### Example

**Input:**

```
2
1 10
3 5
```

**Output:**

```
2
3
5
7

3
5
```

CODIGO

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
using namespace std;
int main()
{
   int casen;
   scanf("%d", &casen);
   while(casen--)
   {
      int n,m;
      scanf("%d %d", &n, &m);
      int * primes = new int[m-n+1];
      for(int i=0;i<m-n+1;++i)
         primes[i] = 0;
      for(int p=2;p*p<=m;++p)
      {
         int less = n / p;
         less *= p; // first number <= N && p divides N
         for(int j=less;j<=m;j+=p)
            if(j != p && j >= n)
            primes[j - n] = 1;
      }
      for(int i=0;i<m-n+1;++i)
      {
         if(primes[i] == 0 && n+i != 1) // We don't want to print if it's 1
         printf("%d\n",n+i);
      }
      if(casen)
         printf("\n");
      delete [] primes;
      }
}
```

# Adding Reversed Numbers

## Problem code: ADDREV

The Antique Comedians of Malidinesia prefer comedies to tragedies. Unfortunately, most of the ancient plays are tragedies. Therefore the dramatic advisor of ACM has decided to transfigure some tragedies into comedies. Obviously, this work is very hard because the basic sense of the play must be kept intact, although all the things change to their opposites. For example the numbers: if any number appears in the tragedy, it must be converted to its reversed form before being accepted into the comedy play.

Reversed number is a number written in arabic numerals but the order of digits is reversed. The first digit becomes last and vice versa. For example, if the main hero had 1245 strawberries in the tragedy, he has 5421 of them now. Note that all the leading zeros are omitted. That means if the number ends with a zero, the zero is lost by reversing (e.g. 1200 gives 21). Also note that the reversed number never has any trailing zeros.

ACM needs to calculate with reversed numbers. Your task is to add two reversed numbers and output their reversed sum. Of course, the result is not unique because any particular number is a reversed form of several numbers (e.g. 21 could be 12, 120 or 1200 before reversing). Thus we must assume that no zeros were lost by reversing (e.g. assume that the original number was 12).

### Input

The input consists of *N* cases (equal to about 10000). The first line of the input contains only positive integer *N*. Then follow the cases. Each case consists of exactly one line with two positive integers separated by space. These are the reversed numbers you are to add.

### Output

For each case, print exactly one line containing only one integer - the reversed sum of two reversed numbers. Omit any leading zeros in the output.

### Example

```
Sample input:

3
24 1
4358 754
305 794

Sample output:

34
1998
1
```

```
import java.util.*;

public class Main
{
   public static void main(String[] args)
   {
          Scanner entrada = new Scanner(System.in);
          int n = entrada.nextInt();
          while(n-->0)
          {
                    String a = entrada.next();
          String b = entrada.next();
          a = new StringBuilder(a).reverse().toString();
          b = new StringBuilder(b).reverse().toString();
          String c = "";
        int n1 = Integer.parseInt(a);
        int n2 = Integer.parseInt(b);
        int suma = n1 + n2;
        a = "";
        a = Integer.toString(suma);
        a = new StringBuilder(a).reverse().toString();
        suma = Integer.parseInt(a);
        System.out.println(suma);
          }
   }
}
```

# Maximo Total

Empezando en la cima del siguiente triángulo y moviéndose a los números adyacentes de la fila inferior, el máximo total de la cima a la base es 23.

3
7 4
2 4 6
8 5 9 3

3 + 7 + 4 + 9 = 23.

Encuentre el máximo total de la cima a la base de un triángulo de un máximo de 100 filas.

Empezando en la cima del siguiente triángulo y moviéndose a los números adyacentes de la fila inferior, el máximo total de la cima a la base es 23.

3
7 4
2 4 6
8 5 9 3

3 + 7 + 4 + 9 = 23.

Encuentre el máximo total de la cima a la base de un triángulo de un máximo de 100 filas.

## Input

La entrada consiste en varios casos de prueba. La primera línea de cada caso de prueba coresponde a n, número de filas del triángulo. La siguientes n líneas son las filas correspondientes del triángulo, su número irá incrementando gradualmente de uno en uno de acuerdo al número de fila correspondiente, (vea el ejemplo de entrada)

## Output

Por cada caso de prueba la salida debe contener un único número que representa el máximo total.

## Example

**Input:**

```
43
7 4
2 4 6
8 5 9 3
```

**Output:**

```
23
```

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<cmath>
using namespace std;

int main()
{
   int n;
   while(scanf("%d", &n) != EOF)
   {
      int m = n + 1;
      int a[n][m];
      for(int i = 0; i < n; i++)
      {
         for(int j = 0; j < m; j++)
         {
            a[i][j] = 0;
         }
      }
      for(int i = 0; i < n; i++)
      {
         for(int j = 1; j <= i+1; j++)
         {
            cin>>a[i][j];
         }
      }
      /*for(int i = 0; i < n; i++)
      {
         for(int j = 0; j < m; j++)
         {
            cout<<a[i][j]<<" ";
         }
         cout<<endl;
      }*/
      for(int i = 1; i < n; i++)
      {
         int maxi = 0;
         for(int j = 1; j <= i+1; j++)
         {
            maxi = max((a[i-1][j-1]+a[i][j]),(a[i-1][j]+a[i][j]));
            a[i][j] = maxi;
         }
      }
      int maxi = 0;
      for(int i = 0; i < n; i++)
      {
         if(a[n-1][i] > maxi)
```

```
            maxi = a[n-1][i];
        }
        cout<<maxi<<endl;
    }
    return 0;
}
```

# Pecas

## Problem code: EMI2

En un episodio del show The Dick Van Dyke, el pequeño Richie conecta las pecas en la espalda de su padre para formar la imagen de la Campana de la Libertad. Por desgracia, una
de las pecas resulta ser una cicatriz, por lo que esto quedara para otra vez.
Considere la posibilidad de dibujar un avión en la espalda de Dick con pecas en varias localidades (x, y). Tu trabajo es decirle a Richie cómo conectar los puntos con el fin de minimizar la cantidad utilizada de tinta. Richie conecta los puntos trazando líneas rectas entre
los pares, es posible levantar la pluma entre las líneas. Cuando Richie ha terminado debe haber una secuencia de líneas conectadas de tal forma que se pueda ir de cualquier peca a cualquier otra peca.

En un episodio del show The Dick Van Dyke, el pequeño Richie conecta las pecas en la espalda de su padre para formar la imagen de la Campana de la Libertad. Por desgracia, una de las pecas resulta ser una cicatriz, por lo que esto quedara para otra vez.

Considere la posibilidad de dibujar un avión en la espalda de Dick con pecas en varias localidades (x, y). Tu trabajo es decirle a Richie cómo conectar los puntos con el fin de minimizar la cantidad utilizada de tinta. Richie conecta los puntos trazando líneas rectas entre los pares, es posible levantar la pluma entre las líneas. Cuando Richie ha terminado debe haber una secuencia de líneas conectadas de tal forma que se pueda ir de cualquier peca a cualquier otra peca.

### Input

La entrada comienza con un solo número entero positivo en una línea, que indica el número de los casos siguientes, cada uno de ellos como se describe a continuación. Esta línea es seguida por una línea en blanco, y también hay una línea en blanco entre dos entradas consecutivas.

Por cada caso, la primera línea contiene 0 <n <= 100, el número de pecas en la espalda de Dick. Para cada peca sigue una línea, cada línea contiene dos números reales que indican las coordenadas (x, y) de la peca.

## Output

Para cada caso de prueba, la salida debe seguir la siguiente descripción. Las salidas de dos casos consecutivos estarán separados por una línea en blanco.

Tu programa de imprimir un número real con dos cifras decimales. (La cantidad mínima total de líneas de tinta que se necesita para conectar todas las pecas).

## Example

**Input:**

```
1
3
1.0 1.0
2.0 2.0
2.0 4.0
```

**Output:**

```
3.41
```

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <vector>

using namespace std;

int pi[110];
int uf_rank[110];

void uf_makeset(int x) {
   pi[x] = x;
   uf_rank[x] = 0;
}

int uf_find(int x) {
   while (x != pi[x]) {
      x = pi[x];
```

```
    }
    return x;
}

void uf_union(int x, int y) {
    int rx = uf_find(x);
    int ry = uf_find(y);

    if (rx == ry) return;
    if (rx > ry) pi[ry] = rx;
    else {
        pi[rx] = ry;
        if (uf_rank[rx] == uf_rank[ry])
            uf_rank[ry] = uf_rank[ry] + 1;
    }
}

float euclidiano(pair<float, float> a, pair<float, float> b) {
    float x = a.first - b.first;
    float y = a.second - b.second;
    return sqrt(x * x + y * y);
}

bool compare(pair<float, pair<int, int> > a, pair<float, pair<int, int> > b) {
    return a.first < b.first;
}

int main() {
    int T, N;
    float x, y;
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &N);

        vector<pair<float, float> > points;
        float matrix[N][N];
        float visited[N], cost[N], prev[N];

        for (int i = 0; i < N; i++) {
            scanf("%f %f", &x, &y);
            points.push_back(make_pair(x, y));
        }

        for (int i = 0; i < N; i++) uf_makeset(i);

        vector<pair<float, pair<int, int> > > E;
        E.clear();
        vector<pair<float, pair<int, int> > > X;
```

```
      X.clear();
      for (int i = 0; i < N; i++) {
         for (int j = i + 1; j < N; j++) {
            E.push_back(make_pair(euclidiano(points[i], points[j]), make_pair(i, j)));
         }
      }
      sort(E.begin(), E.end(), compare);

      for (int i = 0; i < E.size(); i++) {
         if (uf_find(E[i].second.first) != uf_find(E[i].second.second)) {
            X.push_back(E[i]);
            uf_union(E[i].second.first, E[i].second.second);
         }
      }

      float total = 0;
      for (int i = 0; i < X.size(); i++) total += X[i].first;
      printf("%.2f\n", total);

      if(T > 0) printf("\n");
   }
}
```

# Trinomios Triangulares

## Problem code: EMI3

Considere la siguiente expresión:
Dado un valor de n se quiere conocer cual es el valor de los coeficientes de la expresión. Por ejemplo. si n toma el valor de 1 los términos de la expresión son 1, 1, 1. Cuando n=2 tenemos
por lo que la respuesta es 1, 2, 3, 2, 1.

Considere la siguiente expresión:

$$(1+x+x^2)^n$$

Dado un valor de n se quiere conocer cual es el valor de los coeficientes de la expresión. Por ejemplo. si n toma el valor de 1 los términos de la expresión son 1, 1, 1. Cuando n=2 tenemos $1+2x+3x^2+2x^3+x^4$ por lo que la respuesta es 1, 2, 3, 2, 1.

## Input

La primera línea contiene un número T que indica el número de casos de prueba. Cada caso de prueba viene en una línea que contiene el número $0 \leq n \leq 40$ que es el exponente de la expresión.

## Output

Por cada caso de prueba su programa debe escribir en una línea y separados por un espacio los coeficientes del polinomio resultante.

## Example

**Input:**

```
3
1
2
0
```

**Output:**

```
1 1 1
1 2 3 2 1
1
```

```
#include<cstdio>
#include<iostream>
#include<cstring>

using namespace std;
void backtrack()
{

}
int main()
{
   int p = 5;
   long long T[41][82];
   for(int i = 0; i < 41; i++)
      for(int j =0; j < 82; j++)
         T[i][j] = 0;
```

```
T[0][0] = 1;
T[1][0] = 1;
T[1][1] = 1;
T[1][2] = 1;

for(int i = 2; i < 41; i++, p+=2)
{
   for(int j = 0; j <= p/2; j++)
   {
      if(j == 0)
      {
         T[i][j] = 1;
      }
      else
      {
         if(j == 1)
         {
            T[i][j] = T[i-1][j-1]+T[i-1][j];
         }
         else
         {
            T[i][j] = T[i-1][j-2] + T[i-1][j-1] + T[i-1][j];
         }
      }
   }
   for(int j = (p/2)+1, k =2; j < p; j++,k+=2)
   {
      if(j == (p-1))
      {
         T[i][j] = 1;
      }
      else
      {
         T[i][j] = T[i][j-k];
      }
   }
}
/*for(int i = 0; i < 41; i++)
{
   for(int j = 0; T[i][j] != 0; j++)
   {
      cout<<T[i][j]<<" ";
   }
   cout<<endl;
}*/
int n;
cin>>n;
while(n--)
```

```
  {
    int y;
    cin>>y;
    for(int i = 0; T[y][i] != 0; i++)
    {
      cout<<T[y][i]<<" ";
    }
    cout<<endl;
  }
  /*for(int i = 0; i <= 40; i++, j+=2)
  {
    cout<<i<<": "<<j<<endl;
  }
*/
  return 0;
}
```

# Código Binario

## Problem code: EMI5

Supongamos que se tiene una cadena binaria como la siguiente:
011100011
Una forma de encriptar esta cadena es añadir a cada dígito la suma de sus dígitos adyacentes.
Por ejemplo, la cadena de arriba se convertiría en:
123210122
En particular, si P es la cadena original, y Q es la cadena encriptada, entonces Q[i] = P[i-1] +
P[i] + P[i+1] para todos los dígitos en la posición i. Los caracteres de fuera hacia la derecha e
izquierda son tratados como ceros.
Una cadena encriptada dada en este formato, puede ser decodificada como sigue (usando 123210122 como en el ejemplo):
Asumimos que P[0] = 0.
Como Q[0] = P[0] + P[1] = 0 + P[1] = 1, sabemos que P[1] = 1.
Como Q[1] = P[0] + P[1] + P[2] = 0 + 1 + P[2] = 2, sabemos que P[2] = 1.
Como Q[2] = P[1] + P[2] + P[3] = 1 + 1 + P[3] = 3, sabemos que P[3] = 1.
Repitiendo estos pasos nos da que P[4] = 0, P[5] = 0, P[6] = 0, P[7] = 1, y P[8] = 1.
Verificamos nuestro trabajo notando que Q[8] = P[7] + P[8] = 1 + 1 = 2. Ya que la ecuación
funciona, entonces hemos terminado, y hemos recuperado una de las posibles cadenas originales.
Ahora repetimos el proceso, asumiendo el opuesto acerca de P[0]:
Asumimos que P[0] = 1.
Como Q[0] = P[0] + P[1] = 1 + P[1] = 0, sabemos que P[1] = 0.

Como Q[1] = P[0] + P[1] + P[2] = 1 + 0 + P[2] = 2, sabemos que P[2] = 1.
Ahora notamos que Q[2] = P[1] + P[2] + P[3] = 0 + 1 + P[3] = 3, lo que nos lleva a la conclusión
de que P[3] = 2. En este caso, esto va en contra del hecho de que cada caracter de la cadena
original debe ser '0' o '1'. Entonces, no existe tal cadena original P donde el primer dígito sea '1'.
Se nota que este algoritmo produce al menos dos decodificaciones para cualquier cadena encriptada. No puede existir más de una forma posible de decodificar una cadena una vez que
el primer dígito es elegido.

Supongamos que se tiene una cadena binaria como la siguiente:

011100011

Una forma de encriptar esta cadena es añadir a cada dígito la suma de sus dígitos adyacentes.

Por ejemplo, la cadena de arriba se convertiría en:

123210122

En particular, si P es la cadena original, y Q es la cadena encriptada, entonces Q[i] = P[i-1] + P[i] + P[i+1] para todos los dígitos en la posición i. Los caracteres de fuera hacia la derecha e izquierda son tratados como ceros.

Una cadena encriptada dada en este formato, puede ser decodificada como sigue (usando 123210122 como en el ejemplo):

Asumimos que P[0] = 0.

Como Q[0] = P[0] + P[1] = 0 + P[1] = 1, sabemos que P[1] = 1.

Como Q[1] = P[0] + P[1] + P[2] = 0 + 1 + P[2] = 2, sabemos que P[2] = 1.

Como Q[2] = P[1] + P[2] + P[3] = 1 + 1 + P[3] = 3, sabemos que P[3] = 1.

Repitiendo estos pasos nos da que P[4] = 0, P[5] = 0, P[6] = 0, P[7] = 1, y P[8] = 1.

Verificamos nuestro trabajo notando que Q[8] = P[7] + P[8] = 1 + 1 = 2. Ya que la ecuación funciona, entonces hemos terminado, y hemos recuperado una de las posibles cadenas originales.

Ahora repetimos el proceso, asumiendo el opuesto acerca de P[0]:

Asumimos que P[0] = 1.

Como Q[0] = P[0] + P[1] = 1 + P[1] = 0, sabemos que P[1] = 0.

Como Q[1] = P[0] + P[1] + P[2] = 1 + 0 + P[2] = 2, sabemos que P[2] = 1.

Ahora notamos que Q[2] = P[1] + P[2] + P[3] = 0 + 1 + P[3] = 3, lo que nos lleva a la conclusión de que P[3] = 2. En este caso, esto va en contra del hecho de que cada caracter de la cadena original debe ser '0' o '1'. Entonces, no existe tal cadena original P donde el primer dígito sea '1'.

Se nota que este algoritmo produce al menos dos decodificaciones para cualquier cadena encriptada. No puede existir más de una forma posible de decodificar una cadena una vez que el primer dígito es elegido.

## Input

Cada línea contiene una cadena encriptada de tamaño $0 < l \leq 50$. Esta cadena consta de únicamente de los caracteres '0', '1', '2' y '3'. La entrada es finalizada por el fin de archivo

## Output

Por cada línea de entrada, generar dos líneas de salida. La primera línea debe contener la cadena desencriptada asumiendo que el primer caracter es '0' y para la segunda línea se debe asumir que el primer caracter es '1'. Si en alguno de estos casos no es posible regenerar la cadena original, la respuesta es el mensaje "NINGUNA" (sin comillas).

## Example

**Input:**

```
123210122
11
22111
123210120
3
12221112222211122211111111112221111
```

**Output:**

```
011100011
NINGUNA
01
10
NINGUNA
```

```
11001
NINGUNA
NINGUNA
NINGUNA
NINGUNA
01101001101101001101001001001101001
10110010110110010110010010010110010
```

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<vector>
using namespace std;

int main()
{
   string entrada;
   while(getline(cin, entrada))
   {
      int a[entrada.length()+1];
      int enc1[entrada.length() + 2];
      int enc2[entrada.length() + 2];
      for(int i=0;i<entrada.length()+1;a[i]=0,i++);
      for(int i=0;i<entrada.length()+2;enc1[i]=0,i++);
      for(int i=0;i<entrada.length()+2;enc2[i]=0,i++);
      enc1[0] = 0, enc1[entrada.length() + 1] = 0, enc2[0] = 0, enc2[entrada.length() + 1] = 0;
      for(int i = 1; i <= entrada.length(); i++)
      {
         a[i] = entrada[i-1] - 48;
      }
      //asumiendo p[0] = 0
      enc1[1] = 0;
      enc2[1] = 1;
      bool flag1 = false, flag2 = false;
      if(entrada.length() == 1)
      {
         enc1[1] = a[1] - enc1[1];
         enc2[1] = a[1] - enc2[1];

         if(enc1[1] > 1 || enc1[1] < 0)
         {
            flag1 = true;
         }
         if(enc2[1] > 1 || enc2[1] < 0)
         {
            flag2 = true;
         }
```

```
    }
    for(int i = 2; i <= entrada.length(); i++)
    {
        if(i == entrada.length())
        {
            enc1[i] = a[i] - enc1[i-1];
            enc2[i] = a[i] - enc2[i-1];
        }
        else
        {
            enc1[i] = a[i-1] - enc1[i-1] - enc1[i-2];
            enc2[i] = a[i-1] - enc2[i-1] - enc2[i-2];
        }
        if(enc1[i] > 1 || enc1[i] < 0)
        {
            flag1 = true;
        }
        if(enc2[i] > 1 || enc2[i] < 0)
        {
            flag2 = true;
        }
    }
    if(!flag1)
    {
        for(int i = 1; i <= entrada.length(); i++)
        {
            cout<<enc1[i];
        }
    }
    else
    {
        cout<<"NINGUNA";
    }
    cout<<endl;
    if(!flag2)
    {
        for(int i = 1; i <= entrada.length(); i++)
        {
            cout<<enc2[i];
        }
        cout<<endl;
    }
    else
    {
        cout<<"NINGUNA"<<endl;
    }
}
return 0; }
```

COMPRANDO BARATO

Problem code: BARATO

Steve le gustaría comprar un coche nuevo. No es rico, por lo que preferiría un coche razonablemente barato. El único problema es que la calidad de los coches más baratos es ... digamos que cuestionable.
Así, Steve decidió hacer una lista de precios de los automóviles y para comprar un coche con el tercer precio más bajo.
Se le dará un int [] precios. El mismo precio puede aparecer varias veces en los precios, pero debe contar sólo una vez en el ordenamiento de los precios disponibles. Véase el ejemplo 1 para mayor aclaración.
Escriba una función que devuelve el tercer precio más bajo en esta lista. Si hay menos de tres precios diferentes de coches en los precios, el método debe devolver -1.
A Steve le gustaría comprar un coche nuevo. No es rico, por lo que preferiría un coche razonablemente barato. El único problema es que la calidad de los coches más baratos es ... digamos que cuestionable.
Así, Steve decidió hacer una lista de precios de los automóviles y comprar el coche con el tercer precio más bajo.
Se le dará un int [] precios. El mismo precio puede aparecer varias veces en los precios, pero debe contar sólo una vez en el ordenamiento de los precios disponibles. Véase el ejemplo 1 para mayor aclaración.
Escriba una función que devuelve el tercer precio más bajo en esta lista. Si hay menos de tres precios diferentes de coches en los precios, el método debe devolver -1.
Restricciones:
- Los precios contienen entre 1 y 50 elementos.
- Cada elemento de los precios será de entre 1 y 1000, ambos inclusive.
Primeramente se debe leer el numero de casos de prueba y posteriormente "n" casos de prueba.
A continuacion en cada caso de prueba se debe leer un numero "m" que es el tamaño del vector que contiene los elementos y seguido debe leer "m" elementos correspondientes a ese vector.

## Ejemplo

**Entrada:**

```
3

9

10 40 50 20 70 80 30 90 60

10

10 10 10 10 10 20 20 30 30 40 40

1

10
```

**Salida:**

```
30
30
-1
```

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include <set>
using namespace std;

int main()
{
   int n;
   cin>>n;
   while(n--)
   {
      int t;
      cin>>t;
      set<int> p;
      set<int>::iterator i = p.begin();
      for(int j = 0; j < t; j++)
      {
         int k;
         cin>>k;
         p.insert(k);
      }
      if(p.size() < 3)
      {
         cout<<"-1"<<endl;
      }
      else
      {
         i = p.begin();
         ++i;
         ++i;
         cout<<*i<<endl;
      }

   }
   return 0;
}
```

# ORDENANDO VECTORES

Problem code: EMI19

Una palabra se dice pal´ındromo si la misma palabra es exactamente id´entica al derecho y al rev´es
como por ejemplo ABA, AA, A y CASAC. Por otra parte cualquier secuencia de caracteres puede ser vista como 1 o m´as secuencias de pal´ındromos concatenadas. Por ejemplo guapa es la
concatenaci´on de los pal´ındromos g, u y apa; o casacolorada es la concatenaci´on de casac, olo,
r, ada. El problema consiste dado una palabra de a lo m´aximo 2000 letras min´usculas, imprima
el n´umero m´ınimo de palabras pal´ındromas en que se puede dividir.

Dados dos arreglos de numeros enteros A, B donde cada uno contiene ($1 \leq N \leq 100$) numeros. definimos la funcion

$$S = \sum_{i=0}^{i=N} a_i b_i$$

Se pide reordenar el arreglo A de tal forma que la funcion S de el valor mınimo.

La entrada consiste de varios casos de prueba. Cada caso de prueba consiste de tres lineas. La primera linea tiene el numero N de elementos de los vectores A, B. La segunda linea tiene los elementos del vector A separados por un espacio. La tercera linea los elementos del vector B separados por un espacio. La entrada termina cuando no hay mas datos.

En la salida escriba en una linea el valor minimo de S.

**Ejemplo**

| Ejemplo de entrada | Ejemplo de salida |
|---|---|
| 3<br>1 1 3<br>10 30 20<br>5<br>1 1 1 6 0<br>2 7 8 3 1<br>9<br>5 15 100 31 39 0 0 3 26<br>11 12 13 2 3 4 5 9 1 | 80<br>18<br>528 |

```
import java.util.*;
public class Main
{
	public static void main(String[] args)
	{
		Scanner entrada = new Scanner(System.in);
		int s = 0;
		while(entrada.hasNext())
		{
			int tamaño = entrada.nextInt();
			int A[] = new int [tamaño];
			int B[] = new int [tamaño];

			Arrays.fill(A, 1);
			Arrays.fill(B, 1);

			for(int i = 0; i < tamaño; i++)
			{
				A[i] = entrada.nextInt();
			}
			for(int i = 0; i < tamaño; i++)
			{
				B[i] = entrada.nextInt();
			}
			Arrays.sort(A);
			Arrays.sort(B);
			int j = (tamaño - 1);

			for(int i = 0; i < tamaño; i++)
			{
				s = s + (A[i] * B[j]);
				j--;
			}

			System.out.println(s);
```

```
                    s = 0;
                }
            }
}
```

# Subconjuntos Primos

## Problem code: EMI24

Sean A y B dos conjuntos talque todos elemento de B esta en A, entonces decimos que B es subconjunto de A.

Sea A el conjunto A = {1, 2, 3}. Todos los subconjuntos que podemos formar con los elementos

de A son: {1}, {2}, {1, 2}, {3}, {1, 3}, {2, 3}, {1, 2, 3}

Lo que nos interesa es obtener todos los subconjuntos cuya suma sea un numero primo.

En el ejemplo tenemos {1} = 1,{2} = 2,{1, 2} = 1 + 2 = 3,{3} = 3,{1, 3} = 1 + 3 = 4,{2, 3} =

2 + 3 = 5,{1, 2, 3} = 1 + 2 + 3 = 7

Los conjuntos que se deber´ıan imprimir son los que suman 2, 3, 5 y 7 Hay que hacer notar que un conjunto no tiene valores repetidos. El numero de elementos del conjunto se denomina

cardinalidad.

Sean A y B dos conjuntos talque todos elemento de B esta en A, entonces decimos que B es

subconjunto de A.

Sea A el conjunto A = {1, 2, 3}. Todos los subconjuntos que podemos formar con los elementos

de A son: {1}, {2}, {1, 2}, {3}, {1, 3}, {2, 3}, {1, 2, 3}

Lo que nos interesa es obtener todos los subconjuntos cuya suma sea un numero primo.

En el ejemplo tenemos {1} = 1,{2} = 2,{1, 2} = 1 + 2 = 3,{3} = 3,{1, 3} = 1 + 3 = 4,{2, 3} =

2 + 3 = 5,{1, 2, 3} = 1 + 2 + 3 = 7

Los conjuntos que se deber´ıan imprimir son los que suman 2, 3, 5 y 7 Hay que hacer notar

que un conjunto no tiene valores repetidos. El numero de elementos del conjunto se denomina

cardinalidad.

## Input

Cada linea de entrada contiene un conjunto de A de números donde cada Ai < $10^5$ y el numero de elementos del conjunto n □<= 20. Se termina cuando no hay mas datos en la entrada.

## Output

Por cada linea de entrada imprima los subconjuntos, cada uno en una linea, cuya suma es un numero primo. Tome en cuenta el orden en el que se imprimen los subconjuntos, primero se imprime el primer elemento, con sus subconjuntos en forma ordenada de acuerdo al orden de

aparición.

Después de cada caso de prueba escriba una linea con diez guiones como se muestra en el ejemplo.

## Example

**Input:**

```
1 2 3
3 2 1
4 6 8
1 10 100 10000 100000
```

**Output:**

```
2
1 2
3
2 3
----------
3
2
3 2
2 1
----------
no existe
----------
1 10
1 100
1 10 100 10000
----------
```

```
import java.util.*;
public class Main{

	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		while(in.hasNext())
		{
			String a = in.nextLine();
			String[] b = a.split(" ");
			int [] A = new int[b.length];
			for (int i = 0; i < b.length; i++) {
				A[i] = Integer.parseInt(b[i]);
			}
			int nbits = A.length;
			int max = 1 << A.length;
			boolean flag = false;
			for(int i = 0; i < max ;i++)
			{
				int s = 0;
				String N = "";
				for (int j = 0; j<nbits; j++)
			  {
					if ((i & (1<<j)) > 0){
						s += A[j];
						N = N + A[j];
						N += " ";
				  }
			  }
				int cont = 0;
				for(int j = 1; j <= s; j++)
				{
					if(s % j == 0)
					{
						cont++;
					}
				}
				if(cont == 2)
				{
					flag = true;
					System.out.println(N);
				}

			}
			if(!flag)
				System.out.println("no existe");
			System.out.println("----------");
		}
	}}
```

# Palindromes

## Problem code: EMI20

Una palabra se dice palındromo si la misma palabra es exactamente id´ntica al derecho y al reves como por ejemplo ABA, AA, A y CASAC. Por otra parte cualquier secuencia de caracteres puede ser vista como 1 o mas secuencias de palındromos concatenadas. Por ejemplo guapa es la concatenacion de los palındromos g, u y apa; o casacolorada es la concatenacion de casac, olo,r,ada. El problema consiste dado una palabra de a lo maximo 2000 letras minusculas, imprima el numero mınimo de palabras palındromas en que se puede dividir.

**Input**

La entrada consiste de una lınea por caso, cada lınea contiene una cadena s de entre 1 y 2000 caracteres. La entrada termina con EOF. Puede estar seguro que la entrada no contiene mas de 1000 casos.

**Output**

Por cada caso imprime el numero minimo de palabras palındromas en que se puede dividir.

**Ejemplo de Entrada y de Salida**

**Entrada**

casacolorada

casac

hola

**Salida**

4

1

4

```
#include<cstdio>
#include<iostream>
#include<cstring>

using namespace std;

int main()
{
   string s;
```

```
  while(getline(cin, s))
  {

    int l = 0, i = 0, c = 0, j = 0, k = s.length();
    while(j < s.length())
    {
      string cad = s.substr(j, s.length());
      for(l = cad.length(); l > 0; l--)
      {
        string subcad = cad.substr(0, l);
        for(i = 0; i < subcad.length() / 2; i++)
        {
          if(subcad.at(i) != subcad.at(subcad.length()- 1 - i))
          {
            break;
          }
        }
        if(i < subcad.length() / 2)
        {
          continue;
        }
        else
        {
          c++;
          j += l;
          break;
        }
      }
    }
    cout<<c<<endl;
  }
  return 0;
}
```

# Factoriales

## Problem code: EMI22

A usted le dan un n´mero entero N . El factorial de N se define como N (N − 1)(N − 2)..,1.
u
Calcule el factorial de N , quite todos los ceros de la derecha. Si el resultado tiene m´s de K
a
d´
ıgitos, devuelva los ultimos K d´
´
ıgitos, en los otros casos devuelva el resultado completo.

Por ejemplo el n´mero 10 tiene el factorial $10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3628800$. Quitamos
u
los ceros de la derecha para obtener 36288 finalmente nos piden 3 d´
ıgitos imprimimos 288.

A usted le dan un n´mero entero N . El factorial de N se define como $N (N - 1)(N - 2)..,1$.

Calcule el factorial de N , quite todos los ceros de la derecha. Si el resultado tiene m´s de K dıgitos, devuelva los ultimos K dıgitos, en los otros casos devuelva el resultado completo.

Por ejemplo el n´mero 10 tiene el factorial $10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3628800$. Quitamos los ceros de la derecha para obtener 36288 finalmente nos piden 3 dıgitos imprimimos 288.

## Input

Los datos de entrada son los n´meros N y K separados por un espacio donde $1 \leq N \leq 20$ y $1 \leq K \leq 9$. La entrada termina cuando no hay mas datos.

## Output

Por cada dato de entrada escriba los dıgitos especificados por K en una lınea.

## Example

**Input:**
```
10 3
6 1
6 3
7 2
20 9
1 1
```
**Output:**
```
288
2
72
04
200817664
1
```

```
#include<cstdio>
#include<cstring>
#include<iostream>

using namespace std;

int main()
{
```

```
  int a, b;
  while(scanf("%d %d", &a, &b) != EOF)
  {
     long long f = 1;
     int r;
     string A;
     for(int i = 1; i <= a; f *= i, i++);
     while(f > 0)
     {
        r = f % 10;
        f /= 10;
        r = r + 48;
        A = (char)r + A;
     }
     r = A.length();
     while(A.at(r - 1) == '0')
     {
        r--;
        A = A.substr(0, r);
     }
     if(b < A.length())
     {
        A = A.substr(A.length() - b , A.length());
     }
     cout<<A<<endl;

  }
}
```

# Casi Primos

Problem code: EMI21

# CASI PRIMOS

Los n´umeros casi primos son n´umeros no-primos que son divisibles por solo un n´umero primo.
En este problema tu trabajo es escribir un programa que encuentre la cantidad de n´umero casi
primos dentro de cierto rango.
No se consideran casi primos los n´umeros primos.
Veamos un ejemplo, si tomamos el rango entre 2 y 10 solo hay 3 n´umeros Casi primos:
El 4 solo divisible por el 2, por lo que es casi primo
El 6 es divisible por 2 y 3, por lo que no es casi primo
El 8 solo divisible por el 2, por lo que es casi primo
El 9 solo divisible por el 3, por lo que es casi primo
El 10 es divisible por 2 y 5, por lo que no es casi primo
Los n´umeros 2, 3, 5, 7 no son casi primo son primos.

Los números casi primos son numeros no-primos que son divisibles por solo un número primo.

En este problema tu trabajo es escribir un programa que encuentre la cantidad de n´umero casi

primos dentro de cierto rango.

No se consideran casi primos los números primos.

Veamos un ejemplo, si tomamos el rango entre 2 y 10 solo hay 3 números Casi primos:

* El 4 solo divisible por el 2, por lo que es casi primo
* El 6 es divisible por 2 y 3, por lo que no es casi primo
* El 8 solo divisible por el 2, por lo que es casi primo
* El 9 solo divisible por el 3, por lo que es casi primo
* El 10 es divisible por 2 y 5, por lo que no es casi primo

Los números 2, 3, 5, 7 no son casi primo son primos.

## Input

La primera línea de la entrada contiene un entero N (N ☐<= 600) que indica cuantos conjuntos de

datos siguen. Cada una de las siguientes N líneas son los conjuntos de entrada. Cada conjunto

contiene dos número enteros low y high (0 ☐ <= low <=☐ high <=☐ $10^7$).

## Output

Por cada línea de entrada excepto la primera usted debe producir una línea de salida. Esta línea

contendrá un entero, que indica cuantos números casi primos hay dentro del rango(incluyendo)

low...high.

## Example

**Input:**

```
6
2 10
10 20
2 100
2 1000000
```

```
500000 5000000
2 10000000

Output:
3
1
10
236
241
555
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
static char* primes;

void cachePrimes(int n)
{
   int i,j,k;
   primes = (char*) malloc((n+1)*sizeof(char));
   for(i = 0;i<=n;i++)
      primes[i] = 0;
   primes[0]=1;
   primes[1] = 1;
   k = sqrt(n);
   for(i = 2;i<=k;i++)
      if(!primes[i])
         for(j=i*i;j<=n;j+=i)
            primes[j] = 1;
}
int main()
{
   cachePrimes(1000000);
   int c ;
   register long long int i;
   register long long int j;
   long long int almost[80070];
   long long int n,m;
   int top = 0;
   int count = 0;
   for(i=2;i<=1000000;i++)
      if(!primes[i])
         for(j=i*i;j<=1000000000000;j*=i)
            almost[top++] = j;
   scanf("%d",&c);
   while(c>0)
   {
      scanf("%lld %lld",&n,&m);
      count = 0;
```

```
    for(i=0;i<80070;i++)
      if(almost[i]>=n&&almost[i]<=m)
        count++;
    printf("%d\n",count);
    c--;
  }
  return 0;
}
```

# Caballerosidad

## Problem code: EMI23

Dos colas de personas a menudo tienen que fusionarse en una sola cola. Pero, la caballerosidad no ha muerto cuando un hombre y una mujer est´n a punto de entrar a una sola cola, el hombre a siempre sede el lugar a la mujer.

Se tienen dos colas donde hay hombres y mujeres, escribir un programa que ordene seg´n el u g´nero de las personas en ambas colas. Si dos mujeres se encuentran al frente de ambas colas, e la mujer de la primera l´ ınea va primero. Del mismo modo, si dos hombres est´n en la parte a delantera de ambas colas, el hombre de la primera cola va primero.

Entonces, las personas en la parte delantera de ambas l´ ıneas se comparan de nuevo. Cada cola de entrada es una cadena de letras, cada letra representa a un hombre o una mujer. Cada hombre ser´ representada por una M y cada mujer por una W . La salida debe ser de la misma a forma.

La parte izquierda de una cola representa a la cabeza de la cola.

Dos colas de personas a menudo tienen que fusionarse en una sola cola. Pero, la caballerosidad no ha muerto cuando un hombre y una mujer est´n a punto de entrar a una sola cola, el hombre siempre sede el lugar a la mujer.

Se tienen dos colas donde hay hombres y mujeres, escribir un programa que ordene segun el genero de las personas en ambas colas. Si dos mujeres se encuentran al frente de ambas colas, la mujer de la primera lınea va primero. Del mismo modo, si dos hombres est´n en la parte delantera de ambas colas, el hombre de la primera cola va primero.

Entonces, las personas en la parte delantera de ambas lıneas se comparan de nuevo. Cada cola de entrada es una cadena de letras, cada letra representa a un hombre o una mujer.

Cada hombre ser  representada por una M y cada mujer por una W . La salida debe ser de la misma forma.

La parte izquierda de una cola representa a la cabeza de la cola.

## Input

La entrada de datos consiste de dos cadenas cada una representa una cola, ambas colas tienen entre [1 − 50] caracteres, los unicos caracteres que tendran estas dos cadenas son M y W , la entrada termina cuando no haya m´s casos de prueba

## Output

Para cada caso de entrada mostrar la cadena que representa la cola ordenada de acuerdo a los requerimientos presentados al principio.

## Example

**Input:**

```
M
W
MM
MW
MMMM
W
M
WWW
MMMMMMM
W
WWWWWW
M
```

**Output:**

```
WM
MMMW
WMMMM
WWWM
WMMMMMMM
WWWWWWM
```

```
import java.util.*;
public class Main{
	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		while(in.hasNext())
		{
			String a, b, c;
			a = in.nextLine();
			b = in.nextLine();
```

```
                        c = "";
                        int p;
                        if(a.length() <= b.length())
                                p = a.length();
                        else
                                p = b.length();
                        int ind1 = 0, ind2 = 0;
                        while(ind1 < a.length() && ind2 < b.length())
                        {
                                if(a.charAt(ind1) >= b.charAt(ind2))
                                {
                                        c = c + a.charAt(ind1);
                                        ind1++;
                                }
                                else
                                {
                                        c = c + b.charAt(ind2);
                                        ind2++;
                                }
                        }
                        if(ind1 < a.length())
                                c = c + a.substring(ind1);
                        if(ind2 < b.length())
                                c = c + b.substring(ind2);
                        System.out.println(c);
                }
        }
}
```