Određivanje delioca, test primalnosti, merenje vremena, Eratostenovo sito,

1. Određivanje svih delioca celog broja n

```
n=int(input())
for delilac in range(1, n+1):
    if n%delilac ==0:
        print(delilac)
```

TEST PRIMERI

ULAZ
18
IZLAZ
1
2
3
6
9
18

ULAZ
1000000007
IZLAZ
1
1000000007

2. **Određivanje svih delioca celog broja n (test kandidata i, n/i)**

```
import math
n=int(input())
for delilac in range(1, int(math.sqrt(n)+1)):
    if n%delilac ==0:
        if (delilac != n//delilac):
            print(delilac, n//delilac, sep=" ")
        else:
            print(delilac)
```

ULAZ
100
IZLAZ
1 100
2 50
4 25
5 20
10

ULAZ
1000000007
IZLAZ
1 1000000007

UPOREDITE vreme rada prethodna dva programa. Šta zaključujete?

3. **Test primalnosti prirodnog broja n**

```
def prostBroj(n):
    prost=True
    for delilac in range (2,n):
        if n%delilac ==0:
            prost=False
    if n!=1 and prost:
        print("DA")
    else:
        print("NE")

n=int(input())
prostBroj(n)
```

Predajte resenje na evaluaciju
https://petlja.org/biblioteka/r/problemi/Zbirka/prost_broj

Sta zakljucujete?

4. **Test primalnosti prirodnog broja n**

```
import math
def prostBroj(n):
    prost=True
    for delilac in range (2,int(math.sqrt(n))+1):
        if n%delilac ==0:
            prost=False
    if n!=1 and prost:
        print("DA")
    else:
        print("NE")

n=int(input())
prostBroj(n)
```

Predajte resenje na evaluaciju
https://petlja.org/biblioteka/r/problemi/Zbirka/prost_broj

Sta zakljucujete?

5. Implementirajte u Python-u sledeci test primalnosti

Program se još malo može ubrzati ako se primeti da su svi prosti brojevi veći od 2 i 3 oblika $6k-1$ ili $6k+1$, za $k \geq 1$ (stav 1)

Da li važi obratno: Brojevi oblika 6k−1 ili 6k+1 (za k≥1) su prosti,

Pojašnjenje stava 1: Zaista, brojevi oblika 6k, 6k+2 i 6k+4 su sigurno parni tj. deljivi sa 2, brojevi oblika 6k+3 su deljivi sa 3, tako da su jedini preostali 6k+1 i 6k+5, pri čemu su ovi drugi sigurno oblika 6k′−1 (za k′=k+1). Dakle, umesto da proveravamo deljivost sa svim neparnim brojevima manjim od korena, možemo proveravati deljivost sa svim brojevima oblika 6k−1 ili 6k+1, čime izbegavamo proveru sa jednim na svaka tri neparna broja i program ubrzamo shodno tome.

```
#include <iostream>
using namespace std;

bool prost(int n) {
 if (n == 1 ||
    (n % 2 == 0 && n != 2) ||
    (n % 3 == 0 && n != 3))
  return false;
 for (int k = 1; (6*k - 1) * (6*k - 1) <= n; k++)
  if (n % (6 * k + 1) == 0 || n % (6 * k - 1) == 0)
   return false;
 return true;
}
```

6. Funkcija time u Python-u

import time

```
#time je modul u kom se nalaze funkcije za merenje proteklih sekundi
# u odnosu na 1.1.1970.
import math
def prostBroj(n):
    prost=True
    for delilac in range (2,int(math.sqrt(n))+1):
        if n%delilac ==0:
            prost=False
    if n!=1 and prost:
        print("DA")
    else:
        print("NE")

n=int(input())
t1=time.time()
prostBroj(n)
t2=time.time()
print('Vreme ', t2-t1, ' sekundi.')
```

7. Izmerite vreme za funkcije iz zadatka 4,5, 6 na primeru ulaza 1000000009

8. Eksperiment: C++ vs Python (vreme vs memorija vs implementacija)

https://petlja.org/biblioteka/r/problemi/Zbirka/eratostenovo_sito

Eratostenovo sito

Vreme	memorija	ulaz	izlaz
1 s	64 Mb	standardni ulaz	standardni izlaz

Napiši program koji određuje broj prostih brojeva u intervalu [a,b] i njihov zbir (ako zbir ima više od 6 cifara, ispisati samo poslednjih 6).

Ulaz

Sa standardnog ulaza unose se brojevi a i b ($1 \leq a \leq b \leq 10^7$), svaki u posebnoj liniji.

Izlaz

Na standardnom izlazu prikazati u jednoj liniji, odvojeni jednim blanko znakom, broj prostih brojeva is intervala [a,b] i traženi zbir.

Primer

Ulaz
1
1000
Izlaz
168 76127

Resenje (C++):

```
#include <iostream>
#include <vector>

using namespace std;

// funkcija koja popunjava logicki niz podacima o prostim brojevima iz
// intervala [0, n]
void Eratosten(vector<bool>& prost, int n) {
  // alociramo potreban prostor
  prost.resize(n + 1, true);
  prost[0] = prost[1] = false; // 0 i 1 po definiciji nisu prosti
  // brojevi ciji se umnosci precrtavaju
  for (int i = 2; i * i <= n; i++)
    // nema potrebe precrtavati umnoske slozenih brojeva
    if (prost[i]) {
      // precrtavamo umnoske broja i i to krenuvsi od i*i
      for (int j = i * i; j <= n; j += i)
          prost[j] = false;
    }
}

int main() {
  // ucitavamo granice intervala
  int a, b;
  cin >> a >> b;
```

```
// odredjujemo proste brojeve u intervalu [0, b]
vector<bool> prost;
Eratosten(prost, b);

// odredjujemo broj i zbir po modulu 1000000 prostih brojeva iz
// intervala [a, b]
int zbir = 0, broj = 0;
for (int i = a; i <= b; i++)
  if (prost[i]) {
    zbir = (zbir + i) % 1000000;
    broj++;
  }
```

Osnovna ideja algoritma je da se prvo napišu svi brojevi od 1 do datog broja n, zatim da se precrta broj 1 (jer on po definiciji nije prost), nakon njega svi umnošci broja 2 (nisu prosti zato što su deljivi sa 2, dok broj 2 ostaje neprecrtan jer je on prost), zatim umnošci broja 3 (nisu prosti jer su deljivi brojem 3), zatim umnošci broja 5 (nisu prosti zato što su deljivi brojem 5) i tako dalje. Umnoške složenih brojeva nema potrebe posebno precrtavati jer su oni već precrtani tokom precrtavanja umnožaka nekog od njihovih prostih faktora (na primer, nema potrebe posebno precrtavati umnoške broja 4 jer su oni već precrtani tokom precrtavanja umnožaka broja 2). Takođe, prilikom precrtavanja umnožaka broja ii dovoljno je krenuti od i·i jer su manji umnošci već precrtani ranije (imaju prave faktore manje od i). Potrebno je da se postupak ponavlja sve dok se ne precrtaju umnošci svih prostih brojeva koji nisu veći od korena broja nn. Brojevi koji su ostali neprecrtani su prosti (jer znamo da nemaju pravih delilaca manjih ili jednakih korenu od nn, pa samim tim i manjih ili jednakih svom korenu, a pošto nemaju delilaca ispod vrednosti korena, na osnovu teoreme koju smo dokazali u zadatku Prost broj, nemaju pravih delilaca ni iznad vrednosti korena). Precrtavanje brojeva modelovaćemo nizom (ili vektorom) koji sadrži logičke vrednosti (vrednosti tipa bool) i precrtane brojeve označavaćemo sa false, a neprecrtane sa true.

ANALIZA SLOŽENOSTI

Analiza složenosti je komplikovanija i zahteva određeno (doduše veoma elementarno) poznavanje teorije brojeva. Procenimo broj izvršavanja tela unutrašnje petlje. U početnom koraku spoljne petlje precrtava se oko $\frac{n}{2}$ elemenata. U narednom, oko $\frac{n}{3}$. U narednom koraku je broj 4 već precrtan, pa se ne precrtava ništa. U narednom se precrtava oko $\frac{n}{5}$, nakon toga opet ništa, zatim $\frac{n}{7}$ itd. U poslednjem koraku se precrtava oko $\frac{n}{\sqrt{n}}$ elemenata. Dakle, broj precrtavanja je

$$\frac{n}{2}+\frac{n}{3}+\frac{n}{5}+\ldots+\frac{n}{\sqrt{n}}=n\cdot\left(\sum_{\substack{d \text{ prost},\\ d\leq\sqrt{n}}}\frac{1}{d}\right)$$

Još je veliki Ojler otkrio da je zbir $H(m)=1+1/2+1/3+\ldots+1/m=\sum_{d\leq m}\frac{1}{d}$ (takozvani harmonijski zbir) asimptotski jednak $\log m$ (razlika između ove dve funkcije teži takozvanoj Ojler-Maskeronijevoj konstanti $\gamma\approx 0.5772156649$) - samim tim znamo da taj zbir divergira. Takođe, otkrio je da kada se sabiranje vrši samo po prostim brojevima, tada se zbir ponaša kao logaritam harmonijskog zbira, tj. kao $\log\log m$ (pa je i on divergentan). Dakle, u našem primeru možemo zaključiti da je broj precrtavanja jednak $n\cdot\log\log\sqrt{n}$. Pošto je $\log\log\sqrt{n}=\log\log n^{\frac{1}{2}}=\log\left(\frac{1}{2}\log n\right)=\log\frac{1}{2}+\log\log n$, važi da je složenost Eratostenovog sita $O(n\cdot\log\log n)$. Iako nije linearna, funkcija $\log\log n$ toliko sporo raste, da se za sve praktične potrebe Eratostanovo sito može smatrati linearnim u odnosu na n (što je opet dosta sporije samo od ispitivanja da li je broj n prost, što ima složenost $O(\sqrt{n})$).

Brojanje prostih brojeva manjih od 10^7 (ima ih 664579) korišćenjem zasebne provere svakog od brojeva i osnovne implementacije provere da li je broj prost traje oko $4,718$ sekundi, a korišćenjem ubrzane implementacije koja proverava

JEDAN MANJE EFIKASAN PYTHON CODE (poboljsati!!!)

```
a=int(input())
b=int(input())
prime = [True for i in range(b+1)]
prime[1]=False
p = 2
while (p * p <= b):
    # If prime[p] is not changed, then it is a prime
    if (prime[p] == True):

        # Update all multiples of p
        for i in range(p * p, b+1, p):
            prime[i] = False
    p += 1
```

```
zbir=0
broj=0
for i in range(a,b+1):
        if prime[i]:
            zbir=(zbir +i)%1000000
            broj+=1
print(broj, zbir, sep=" ")
```

Zadaci za vezbu:

Osnovni nivo

1. https://petlja.org/biblioteka/r/problemi/Zbirka/najblizi_prost_broj
2. https://petlja.org/biblioteka/r/problemi/Zbirka/rastavljanje_na_proste_cinioce
3. https://csacademy.com/contest/archive/task/counting-quacks/

Napredni nivo

4. https://csacademy.com/contest/archive/task/num-cube-sets/
5. https://csacademy.com/contest/archive/task/farey_sequence/