# Р1 Т05.05. Прост број

## Р1 Т05.05. Прост број (увод: тема 5.5)

### Задатак 1: Да ли је број прост?

За дати цео број **n** проверити да ли је прост (исписати **DA** ако јесте, а **NE** ако није)

### Задатак 1: Да ли је број прост? – Поступак решавања задатка

#### Корак 1: Математика: Подсетник

**Математика:**

Прост број је природан број већи од 1, дељив само бројем 1 и самим собом ([Википедија](https://sr.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82_%D0%B1%D1%80%D0%BE%D1%98)).

Природан број је прост ако је дељив само бројем 1 и самим собом.

Природан број се зове прост број, ако је већи од 1 и ако се не може записати као производ два природна броја која су мања од њега.

Прости бројеви су, на пример:   
**2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113**.

**Алгоритам (прва верзија) - Математика:**

То значи да проверу да ли је број **n** прост (тест прималности броја **n**) можемо да обавимо тако што:  
за сваки број **од 2 до n−1**   
проверимо да ли је **n** **дељив тим бројем**.

Ако **n** није дељив ниједним од тих бројева, то значи да је **прост**.

Овакво проверавање ради за све бројеве осим за број **1**, па тај случај разматрамо посебно.

С обзиром да је било који природан број или прост или није (уколико није онда је сложен), као и   
да је довољно да природан број има бар један делилац, који је мањи од њега, а већи од 1 да би био прост,  
увешћемо локалну логичку променљиву **prost**, којој ћемо пре почетка петље (односно провере за сваки број од 2 до n-1) доделити вредност тачно (што значи да је број прост).

Уколико проласком кроз петљу бар за један број **n** буде дељив тим бројем логичка променљива **prost** добиће вредност **нетачно**.

У супротном, уколико ниједном не прође кроз ову грану, остаће иницијална вредност **тачно**, што значи да је број **n** прост.

**Алгоритам (прва верзија) – Псеудо-код:**

1. Учитај вредност природног броја **n**.
2. Постави подразумевану вредност **тачно** променљивој **prost**.
3. За сваку вредност променљиве **d** почев од **2** до **n-1** понови, односно провери:
   1. Да ли је **n** дељив са **d** (односно да ли је остатак при целобројном дељењу једнак нули)?
   2. Уколико јесте постави вредност променљивој **prost** на **нетачно**.
4. Прикажи вредност променљиве **prost**.

**Алгоритам (прва верзија) – Дијаграм тока:**

За записивање алгоритма дијаграмом тока могу нам бити корисни неки постојећи алати, односно апликације, попут **[FlowGorithm](http://www.flowgorithm.org/)**-а:

|  |  |  |
| --- | --- | --- |
| Слика 1: Алгоритам: [Дијаграм тока](https://github.com/draganilicnis/R3_T01_01_Algebarski_algoritmi_01_Test_primalnosti_Prost_broj/blob/main/R1_T05_05_Prost_Ver_000.fprg) | * Након израде алгоритма потребно је да тестирамо његову коректност и исправност. * Тестирати за неколико примера, односно различих вредности датог природног броја **N**. * Најпре за мале бројеве, за које можемо и „напамет“ да проверимо тачност, односно да ли су прости или нису, а затим за неколико већих бројева. | |
| Улаз | Излаз |
| **N** | **Prost** |
| **2** | **DA** |
| **3** | **DA** |
| 4 | NE |
| **5** | **DA** |
| 6 | NE |
| **7** | **DA** |
| 9 | NE |
| **19** | **DA** |
| 25 | NE |
| 100 | NE |
| **101** | **DA** |
| **113** | **DA** |

**Алгоритам (прва верзија) – Програмски језик:**

За записивање алгоритма у неком од програмских језика можемо користити било који текст едитор, као на пример Бележницу ([Notepad](https://sr.wikipedia.org/wiki/Notepad) или [Notepad++](https://notepad-plus-plus.org/downloads/)), али је због провере коректности и исправности много боље да користимо неко од одговарајућих развојних окружења, почев од доступних и бесплатних онлајн развојних окружења, до интегрисаних развојних окружења исправно инсталираних на локалном (нашем) рачунару, као што је на пример [Visual Studio](https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/prirucnik-csharp/instalacija-razvojnog-okruzenja).

Прва верзија програмског кода у програмском језику **C#** би изгледала попут следећег кода:

|  |
| --- |
| using System;  public class R1\_T05\_05\_Prost  {  public static void Main()  {  int n = int.Parse(Console.ReadLine());  bool prost = true;  for (int d = 2; d <= n - 1; d++)  if (n % d == 0) prost = false;  Console.WriteLine(prost ? "DA" : "NE");  }  } |

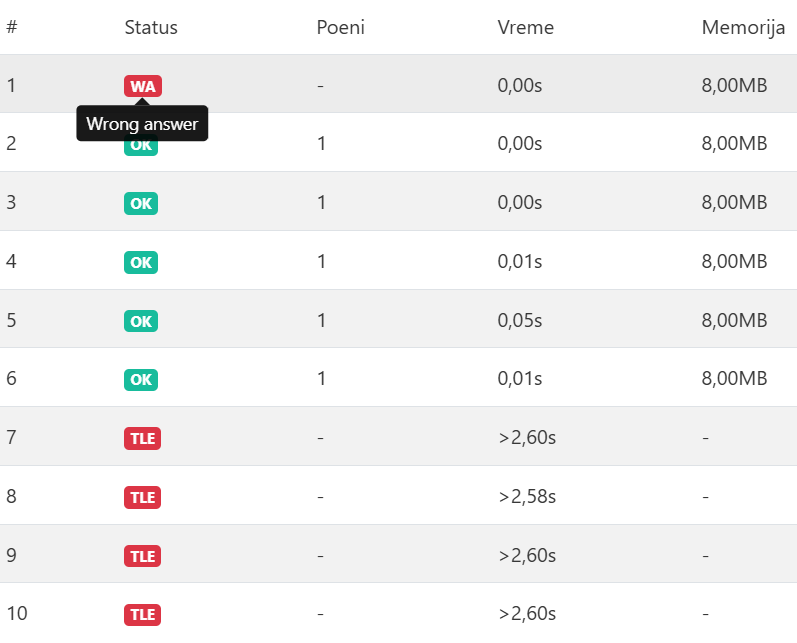
Тестирајте програм и уверите се да исправно ради. Испробајте га и за неке велике бројеве, на пример за  
**n = 1 000 000 000**.

Приметићете да на одговор треба сачекати. То није ништа чудно, јер програм обавља **скоро милијарду** провера. Питање је да ли је било неопходно да се обави толико провера.

**Аутоматско тестирање (провера) програмског кода на специјализованим системима за тестирање**

Постоје системи за аутоматско тестирање (проверу) исправности нашег програмског кода на којем су унапред припремљени неколико различитих и пажљиво припремљени тест примери, који могу, али не морају да буду нама (ауторима програмског кода) доступни. У настави најчешће користимо Арену на порталу Фондације Петља.

Искорпирајте претходни програмски код на   
<https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-05-prost-broj#tab_132331> и покрените тестирање. Видећемо да ће наш програмски код проћи 5 од 10 тест примера:



Што се тиче првог тест примера, порука о грешци је **погрешан одговор** (WA). Могуће је да је у том тест примеру вредност улазне променљиве **n = 1**, па ћемо исправити (допунити) наш програмски код:

|  |
| --- |
| using System;  public class R1\_T05\_05\_Prost  {  public static void Main()  {  int n = int.Parse(Console.ReadLine());  bool prost = true;  **if (n == 1) prost = false;** // 1 nije prost broj po definiciji  for (int d = 2; d <= n - 1; d++)  if (n % d == 0) prost = false;  Console.WriteLine(prost ? "DA" : "NE");  }  } |

Сада нам пролази првих 6 тест примера.

Међутим, поново за последња 4 тест примера (од 7 до 10) са система добијамо поруку (извештај, повратне информације) да је дошло до **прекорачења временског ограничања** (TLE), јер је потребно време за извршавање програмског кода веће од 2,5 секинде, а у услову задатка је предвиђено највише **1,5** секунда.  
Можемо да претпоставимо да се у ова 4 тест примера налазе велики природни бројеви (можда близу **109**, јер је тако и наведено у условима задатка).

Тестирајте програм и уверите се да исправно ради. Испробајте га и за неке велике бројеве, на пример за  
**n = 1 000 000 000 (=109)**.

Приметићете да на одговор треба сачекати. То није ништа чудно, јер програм обавља **скоро милијарду** провера. Питање је да ли је било неопходно да се обави толико провера. То значи да је потребна временска оптимизација нашег програмског кода.

### Временска оптимизација програмског кода

Једна ствар коју можемо да учинимо да бисмо смањили број операција је да прекинемо петљу чим нађемо један делилац, јер тада већ знамо одговор. Тако, уместо

**if** (n % d == 0)

prost = **false**;

можемо да пишемо

**if** (n % d == 0)

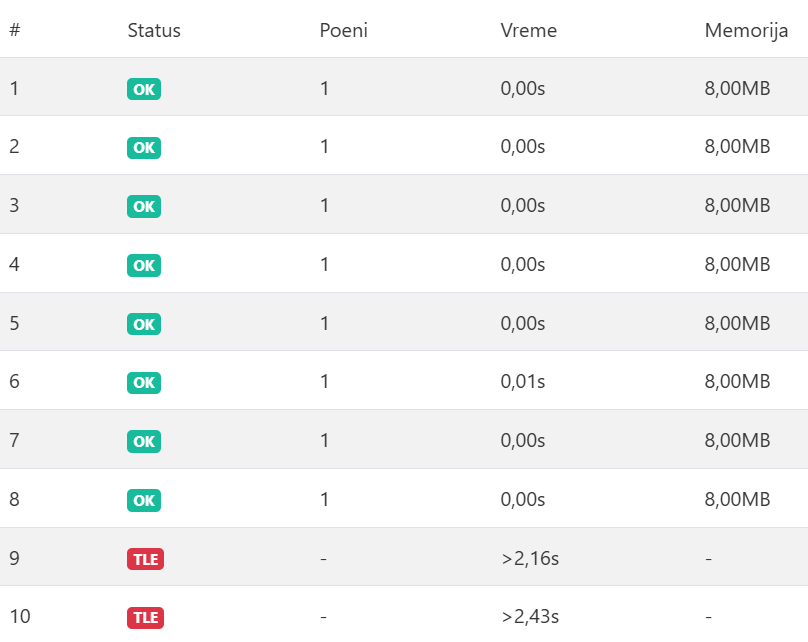
{

prost = **false**;

**break**;

}

Сада нам пролази првих 8 тест примера:



Програм сада ради много брже за **n = 1 000 000 000 (=109)**.

То је наравно зато што је ово **n** дељиво са **2**, а то значи да из петље излазимо већ у првој итерацији.   
Чак и да број није дељив баш са **2**, број провера ће бити много мањи него у првој верзији.

Често, али не увек. Испробајте измењен програм за **n = 1 000 000 007**.

Програм се поново дуго извршава. То је зато што је овај број прост, па нам наредба **break** ништа не помаже. **Тело петље се поново извршава око милијарду пута**.

Временска сложеност алгоритма је још увек **O(N)**.

Детаљније информације о временској сложености алгоритма могу се пронаћи на:   
<https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/malecki-uvod/maleckiuvod_02_slozenost>.

Можемо ли још нешто да учинимо?

### Друга верзија: Оптимизација: Одсецање

Приметимо да ако је број **n** сложен, не могу сви његови делиоци бити већи од .

Заиста, када би то било могуће, онда би за било који делилац **d** броја **n** бројеви **d** и **n/d** оба били делиоци од **n** који су већи од па би њихов производ морао да буде већи од **=n**, али **d⋅n/d = n**, то јест није веће од **n**, што је контрадикција!

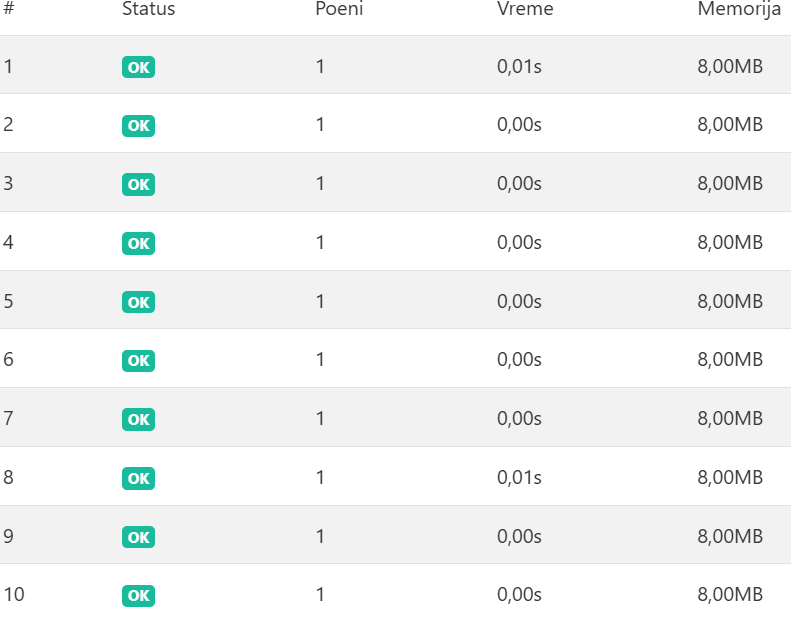
Ово значи да број **n**, ако је сложен, мора имати делилац који је мањи или једнак .

Другим речима, ако број **n** који је већи од **2** нема делилац мањи или једнак од , онда је тај број прост.

Захваљујући овом разматрању, број итерација можемо значајно да смањимо. Провере дељивости са **d** могу да почну са **d=2**, а да се прекину када је **d\*d>n** (ако до тада није нађен делилац онда га и нема).

|  |
| --- |
| // test primalnosti - druga verzija  using System;  public class R1\_T05\_05\_Prost  {  public static void Main()  {  int n = int.Parse(Console.ReadLine());  bool prost = true;  if (n == 1) prost = false; // 1 nije prost broj po definiciji  else if (n == 2) prost = true;  else  for (int d = 2; d\*d <= n; d++)  if (n % d == 0)  {  prost = false;  break;  }  Console.WriteLine(prost ? "DA" : "NE");  }  } |

Сада програмски код пролази свих 10 тест примера:



Сада, чак и за **n = 1 000 000 007** програм даје резултат без застоја. То смо и очекивали, јер број операција је овај пут приближно  **< 32 000**. Дакле, програм је сада пар десетина хиљада пута бржи!

Временска сложеност алгоритма је још увек **O(**.**)**.

Детаљније информације о временској сложености алгоритма могу се пронаћи на:   
<https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/malecki-uvod/maleckiuvod_02_slozenost>.

Програм се може и даље убрзавати, али не више овако драстично. На пример, можемо да искористимо чињеницу да је број **2** једини паран прост број, па према томе и једини паран број за који треба проверити да ли је делилац од **n** (ако **n** није дељив са **2**, онда није дељив ниједним парним бројем).

Зато можемо проверу дељивости са **2** да обавимо пре петље, а у петљи да почнемо од 3 па да повећавамо **d** за по **2**, проверавајући на тај начин само непарне **d**. Тиме би програм постао још двоструко бржи.

|  |
| --- |
| // test primalnosti - treca verzija  using System;  public class R1\_T05\_05\_Prost  {  public static void Main()  {  int n = int.Parse(Console.ReadLine());  bool prost = true;  if (n == 1) prost = false; // 1 nije prost broj po definiciji  else if (n == 2) prost = true;  else if (n % 2 == 0) prost = false;  else  {  prost = true;  for (int d = 3; d\*d <= n; d = d+2)  if (n % d == 0)  {  prost = false;  break;  }  }  Console.WriteLine(prost ? "DA" : "NE");  }  } |

Даља убрзавања програма би била све мања, а овде нам и нису од значаја, па ћемо се на овоме зауставити. Покушајте да решите задатке на тему дељивости, који следе у наставку ([Р1 Т05 05](https://petlja.org/sr-Latn-RS/kurs/17862/23/2756)).

### Шта је са бројевима који су већи од милијарде?

Тестирајте сада програмски код за бројеве који су већи од **109** (а мањи од 2\***1019**), као на пример за број

**n=1 111 111 111 111 111 111 ≈ 1.1 \* 1019**

Овај број има 19 јединица (≈**1019**). Шта примећујеш?

Исправићемо програмски код тако да све променљиве типа **int** заменимо типом **ulong**.

|  |
| --- |
| // test primalnosti - treca verzija ulong = 10^19  using System;  public class R1\_T05\_05\_Prost  {  public static void Main()  {  ulong n = ulong.Parse(Console.ReadLine());  bool prost = true;  if (n == 1) prost = false; // 1 nije prost broj po definiciji  else if (n == 2) prost = true;  else if (n % 2 == 0) prost = false;  else  {  prost = true;  for (ulong d = 3; d\*d <= n; d = d+2)  if (n % d == 0)  {  prost = false;  break;  }  }  Console.WriteLine(prost ? "DA" : "NE");  }  } |

Програм се поново дуго извршава (неколико секунди).

Програмски код пролази свих 10 тест примера на Петљи, али сада желимо да прецизније утврдимо потребно време за извршавање програмског кода и како вредност **N** утиче на брзину извршавања кода.

## Мерење потребног времена за извршавање програмског кода

Постоји неколико начина да прецизније израчунамо потребно време за извршавање програмског кода.

У овом примеру користићемо интегрисано развојно окружење (IDE) **Visual Studio**, на следећи начин, хронолошки корак по корак:

### Корак 1: Нов пројекат (конзолна апликација)

Покрени интегрисано развојно окружење (IDE) **Visual Studio** и креирај конзолну апликацију као нов пројекат у **C#**.

Како би лакше упоредили неколико различитих алгоритама за проверу да ли је број прост (од најспоријих до најбржих), сваки од тих алгоритама имплементираћемо као методе (потпрограме) и именоваћемо их редом **Prost\_Ver\_01** до **Prost\_Ver\_04**, од најспоријег до најбржег.

Користићемо и штоперицу за прецизније мерење потребног времена за извршавање делова програмског кода. Да би користили штоперицу неопходно је да користимо библиотеку **System.Diagnostics**.

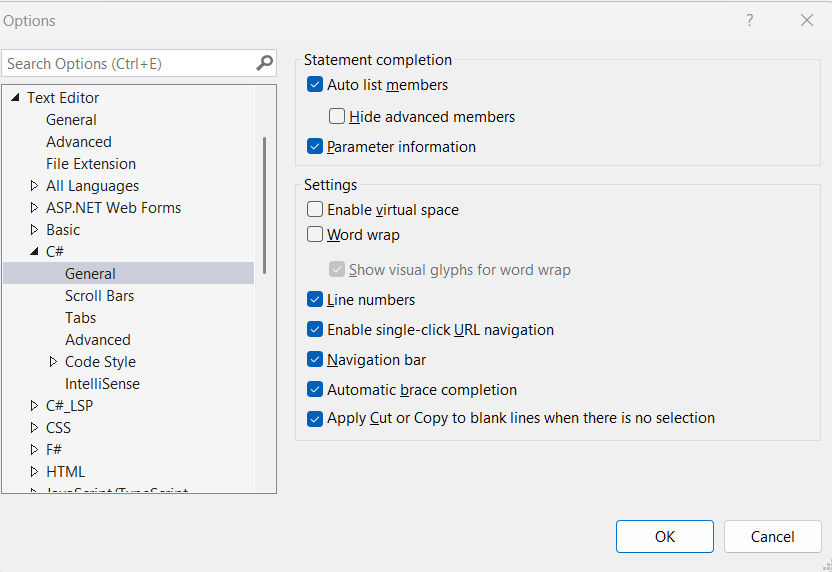
За све четири методе мерићемо потребно време за два „најспорија“ тест примера, када је:

**n = 1 000 000 007 = 109 + 7** и   
**n = 1 111 111 111 111 111 111 ≈ 1.1 \* 1019**

Да не би стално (као корисници) уносили са тастатуре ова две вредности, поставићемо их у самом коду.

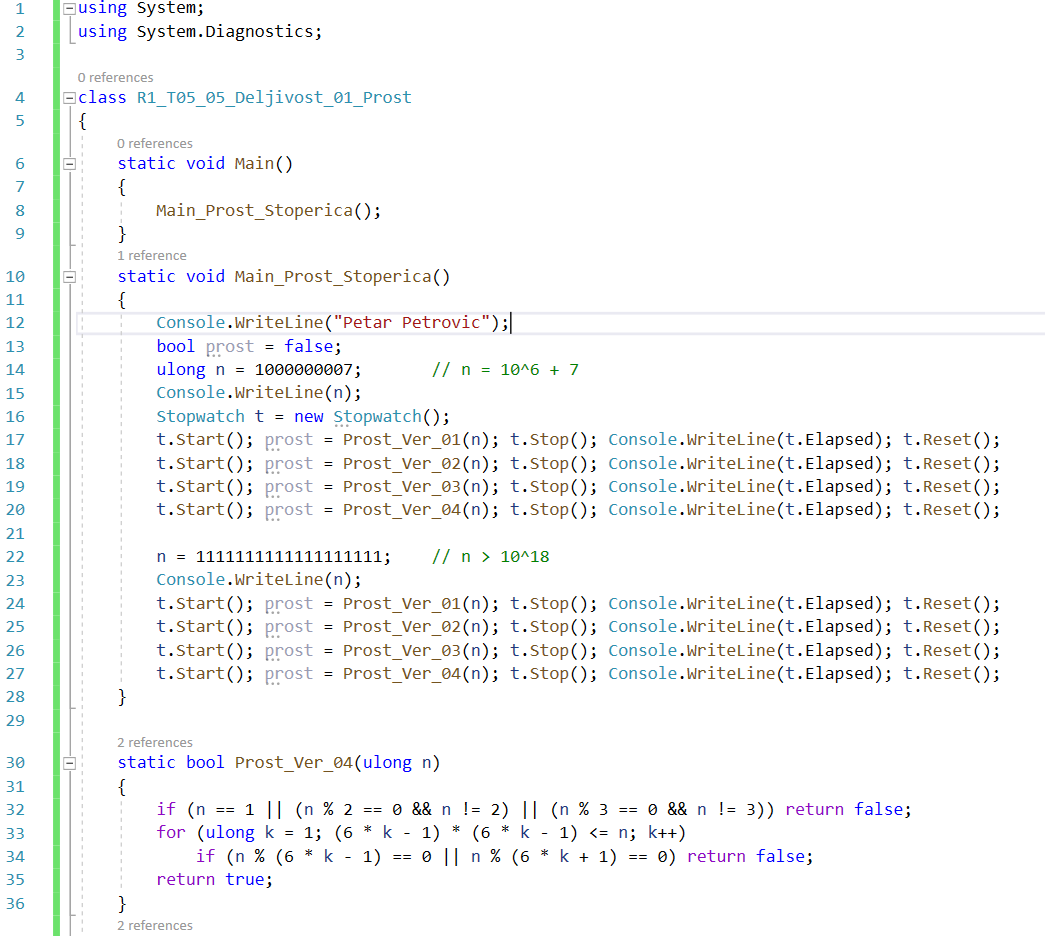
### Корак 2: Подеси приказивање броја линија

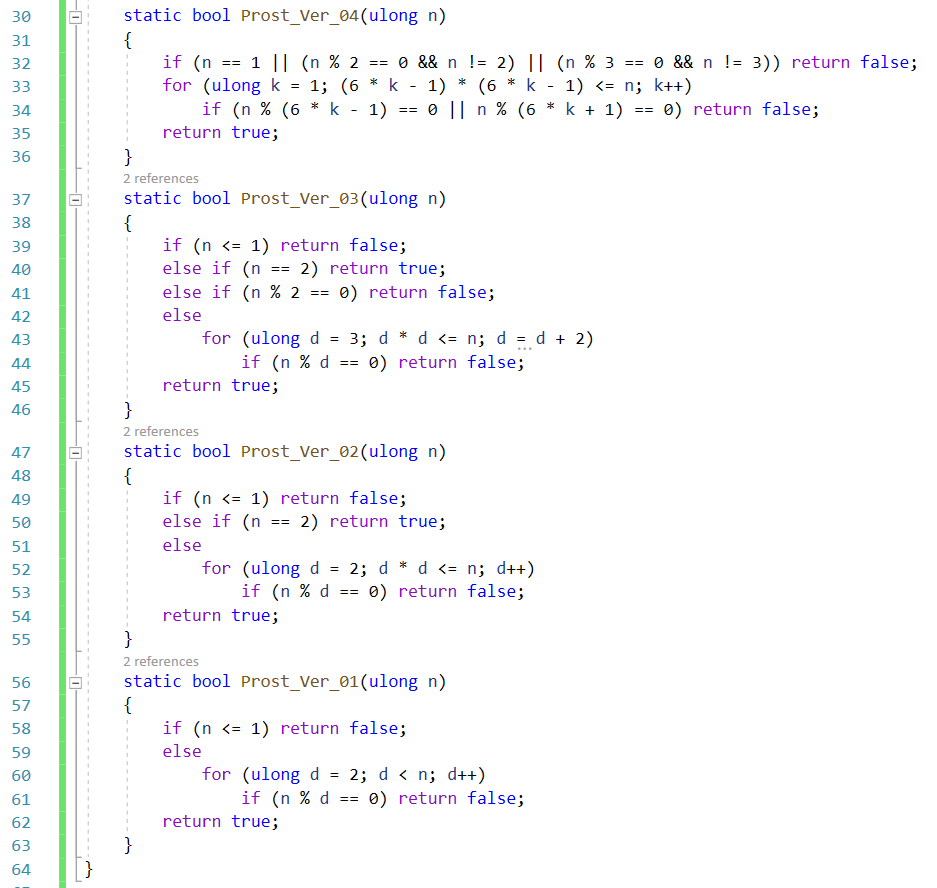
С обзирим да ће програмски код имати неколико десетина линија било би добро да се приказују редни бројеви линија са леве стране едитора, што се може подесити избором опције из менија:  
**Tools – Options -Text Editor – C# - General – Line Numbers**, као на слици:



### Корак 3: Напиши (прекуцај) следећи програмски код

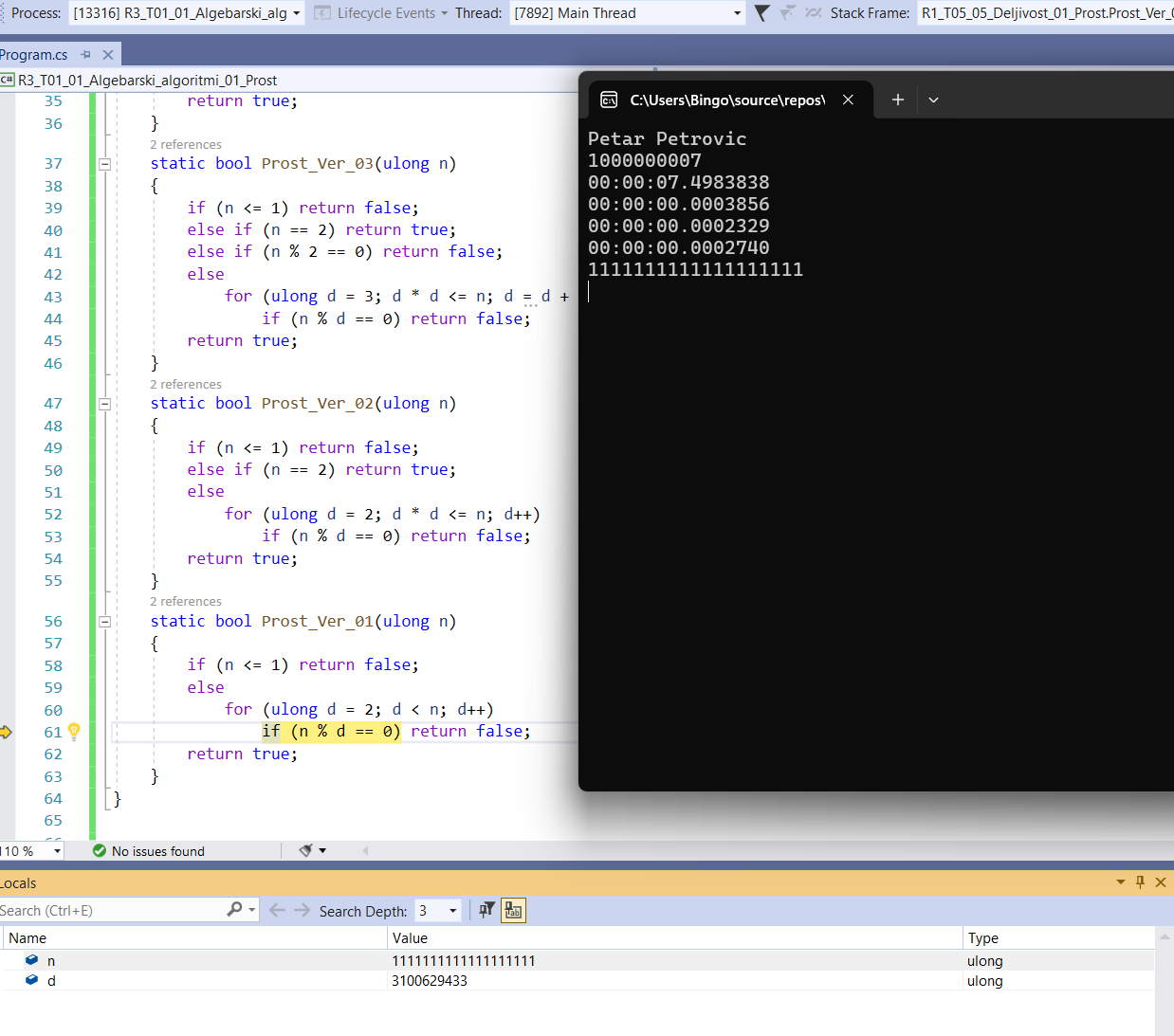
Покрени интегрисано развојно окружење (IDE) **Visual Studio** и креирај конзолну апликацију као нов пројекат у C# и прекуцај програмски код тако да буде као на слици:





### Корак 4: Покрени извршавање програмског кода (и паузирај)

Покрени извршавање програмског кода чиме ће се појавити прозор као на слици:



Приметићемо да се први тест пример (**109+7**) за најспорији алгоритам извршава неколико (око 7) секунди, а да се за остале извршава за мање од милисекунде.

Међутим, за други тест пример (19 јединица) програмски код се извршава доста дуго (сасвим сигурно много више од минута).

**Шта мислиш колико дуго ће се извршавати (ред величине)?**

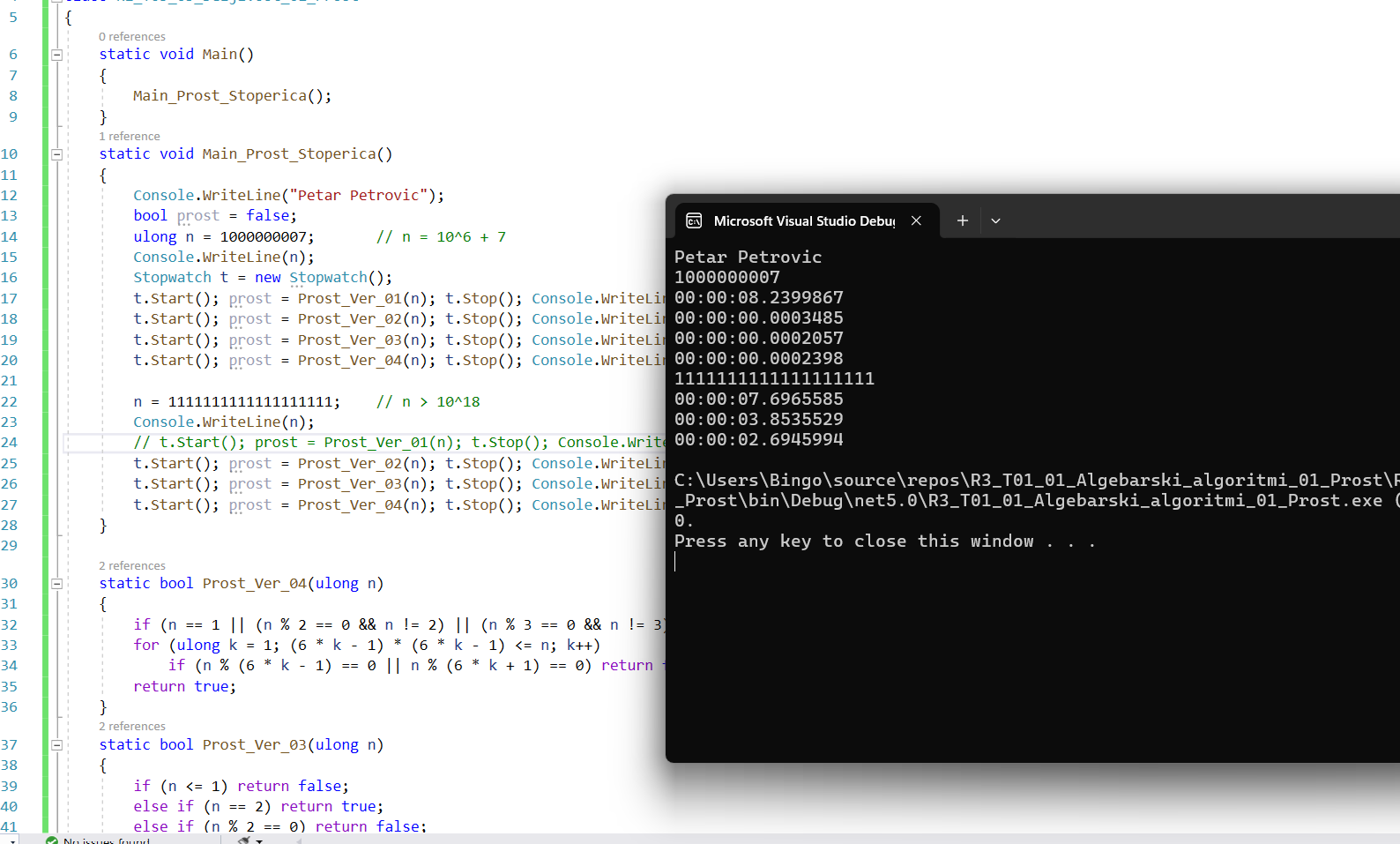
Да ли имамо времена да чекамо да се заврши. Шта да радимо?

Да ли да зауставимо извршавање програма?

Можемо да паузирамо извршавање програма из интегрисаног развојног окружења (као на слици горе), а можемо и да зауставимо извршавање програма комбинацијом тастера **CTRL-C**.

С обзиром да се најспорија метода превише дуго извршава избацићемо је из извршавања, тако што ћемо закоментарисати ту линију кода (у линији 24) и покренути програм комбинацијом тастера **CTRL-F5**.

Након неколико секунди добићемо резултате сличне као на следећој слици:

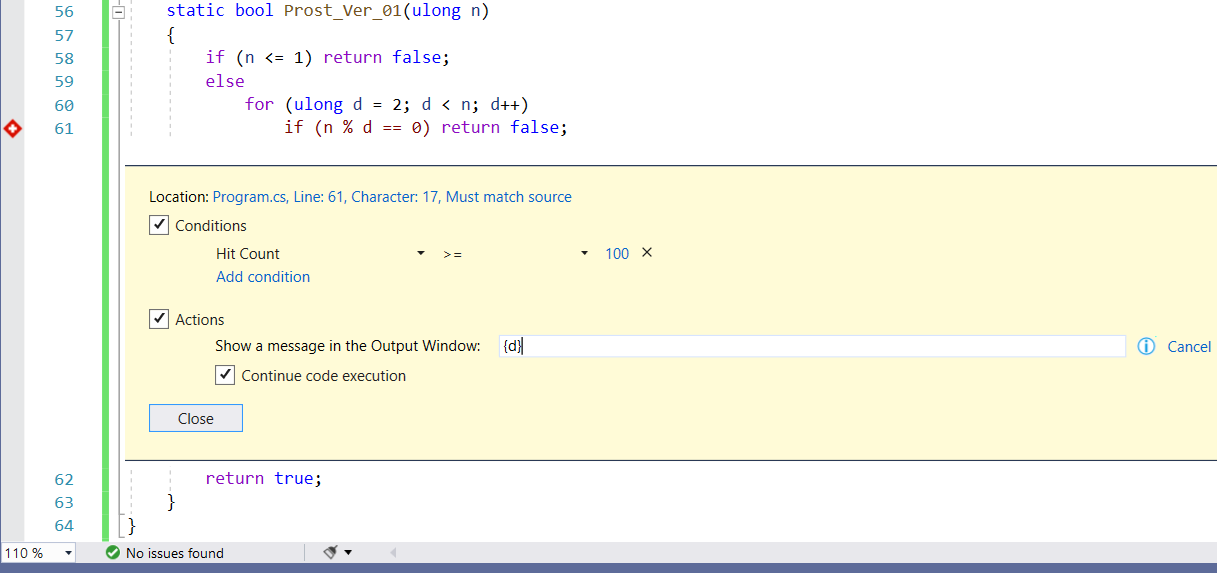


Шта закључујеш на основу ових резултата?

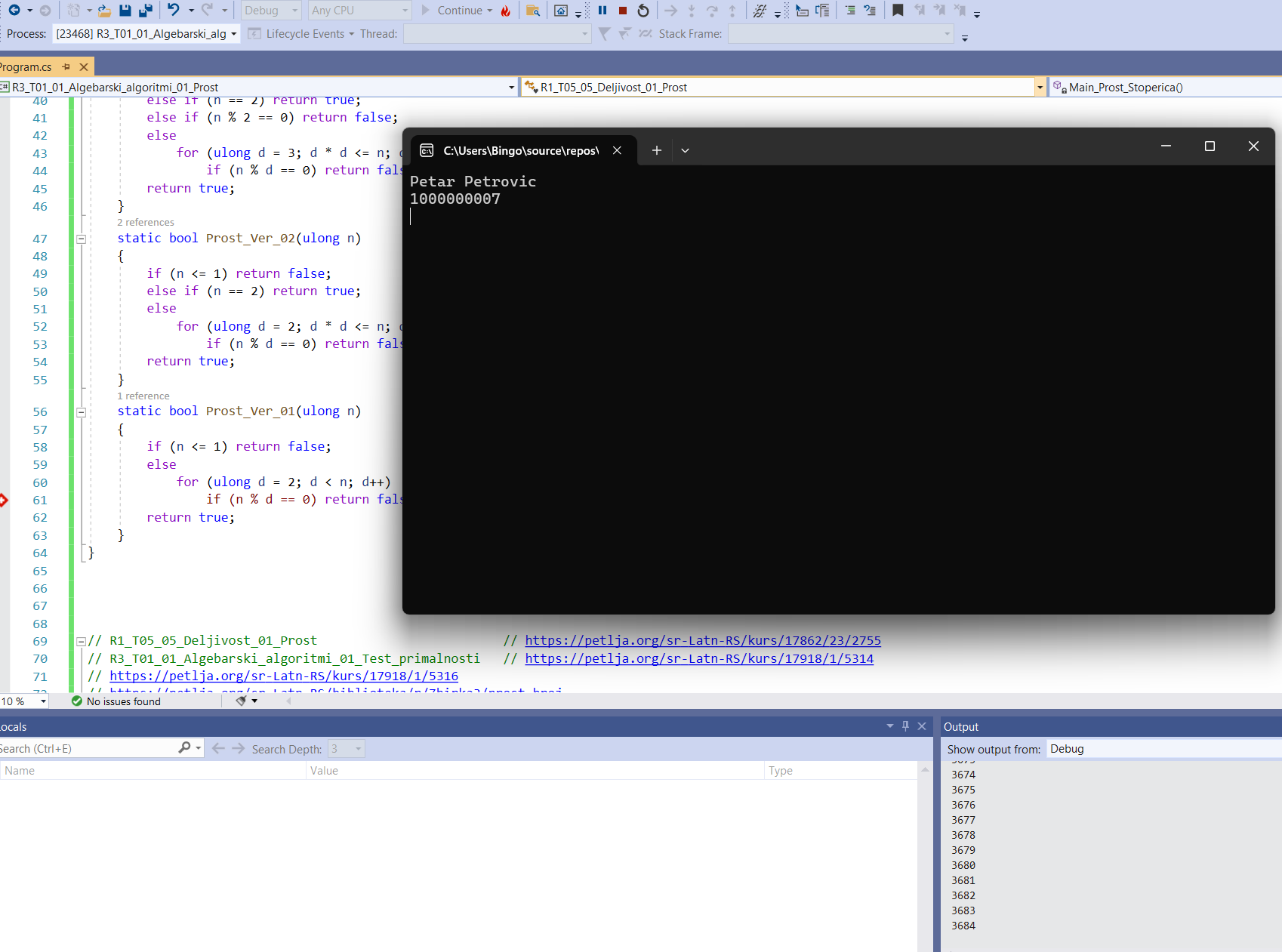
### Корак 5: Прекидне тачке

Прекидне тачке (у интегрисаном развојном окрижењу) су веома корисне програмерима током развоја, израде, корекције, провере и тестирања програмског кода.

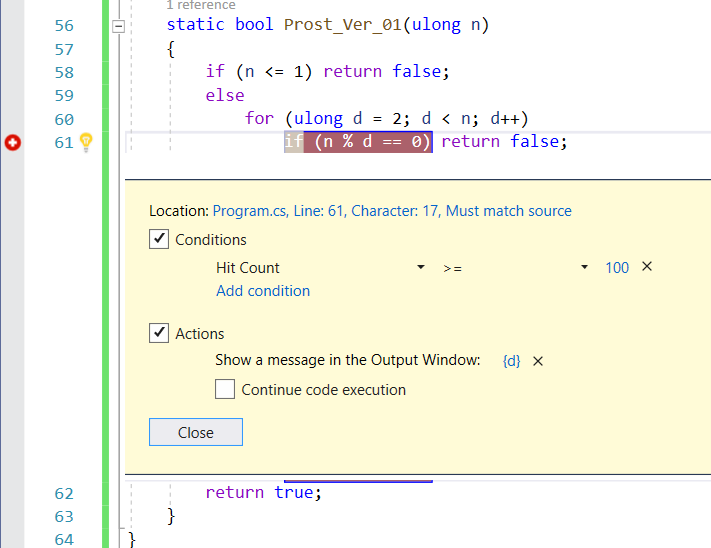
Поставићемо прекидну тачку (F9) у најспоријој методи (у петљи) у линији 61 и подесићемо услове тако да буду као као на следећој слици:

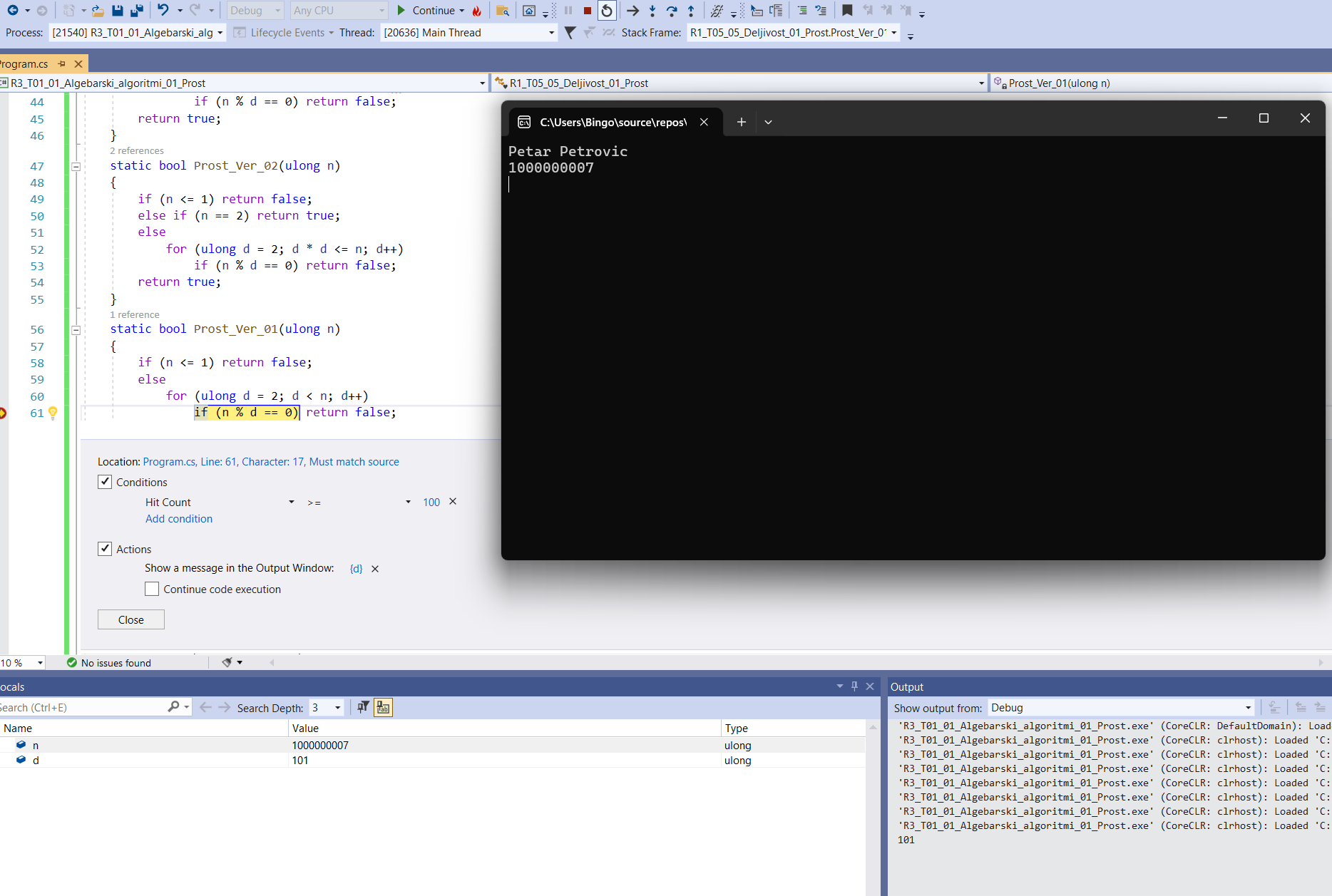


Покренућемо извршавање програмског кода (F5) чиме ће се појавити прозор као на слици:



Променићемо услове прекидне тачке тако да буду као на слици и поново ћемо покренути извршавање програма (F5):

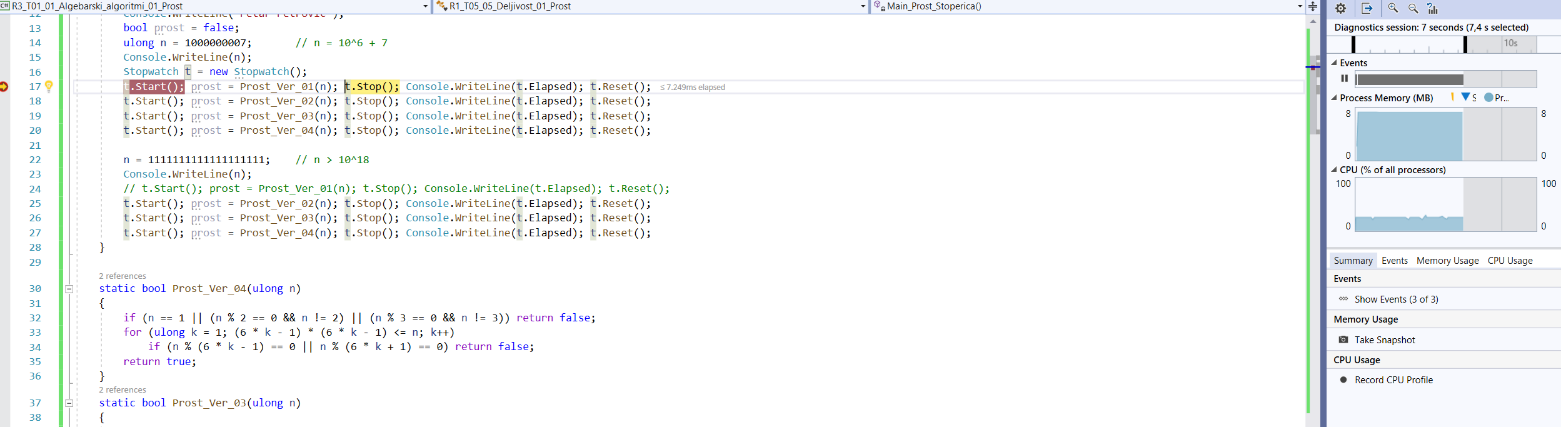




Шта примећујеш? Обрати пажњу на текуће вредности променљивих (у доњем левом углу прозора). Извшравање програмског кода можемо наставити на неколико начина.

### **Корак 6: Постепено извршавање програма корак по корак (F10) и/или (F11**)

Сада постави прекидну тачку у линији 17, поново покрени програм (F5), након заустављања извршавања програмског кода у прекидној тачки настави [постепено извршавање програма](https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/prirucnik-csharp/postepeno-izvrsavanje-programa) корак по корак (тастерима F10 и/или F11), као на слици:



## Упоређивање спорог и брзог алгоритма

Овде ћемо табеларно приказати и упоредити **[временску сложеност](https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/malecki-uvod/maleckiuvod_02_slozenost)** два алгоритма,   
**споријег** (из прве верзије) и **брзог** (из последње верзије):

|  |  |  |
| --- | --- | --- |
| **Ставка** | **Спорији** | **Бржи** |
| Верзија | Прва (интуитивна) | Четврта (временски оптимизована) |
| Кључна разлика | Проверава све делиоце од **2**  **до N-1** | Проверава све делиоце од **2**  **до** |
| Временска сложеност | Линеарна | Корен од **N** |
| Ознака | **O(N)** | **O()** |
| Потребно време за: **N = 109 + 7** (милијарда) | неколико **секунди**: **00:00:07.0573240** **N = 109** милијарду операција ≈ неколико секунди | мање од **милисекунде 00:00:00.0002418** 32 хиљада операција < милисекунде **Алгоритам је око 32 хиљаде пута бржи од линеарног!** |
| Потребно време за: **N ≈ 1019**  10 милијарди милијарди  = 10 [трилиона](https://bs.wikipedia.org/wiki/Duga_i_kratka_skala) | неколико стотина **година** што је неколико десетина милијарди секунди **N ≈ 1019** 10 милијарди милилијарди операција ≈ **1010 секунди** што је око 10 милијарди секунди, односно око 167 милиона минута, или скоро 3 милиона сати, или око 115 хиљада дана, или  око **317 година** | неколико **секунди 00:00:02.6228750** око 3 милијарди операција ≈ 3 секунди    **Алгоритам је око 3 милијарде пута бржи од линеарног!** |
| **Алгоритам (дијаграм тока)**  напомена: груба верзија алгоритма како би се лакше уочила кључна разлика |  |  |
| **Алгоритам (програмски код)** |  |  |
| **Кључна разлика у програмском коду** | **for (ulong d=2; d < n; d++)** | **for (ulong d=3; d\*d <= n; d=d+2)** |
| **Петља се извшрава све док је** | **d < n** | **d \* d <= n** |
| **делилац мањи од** | d **< N** | d **<=** |

### Колико је потребно времена

Слично претходној табели, можемо да формирамо и обрнуту табелу, која показује колико је времена (оквирно, приближно, ред вечичине) потребно алгоритмима различите сложености да обраде улазе датих величина, ако претпоставимо да процесеор може да изврши око милијарду операција у секунди:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Спор**  **O(N)** | | | | | **Брз**  **O()** | | | | |
| **N** | у секундама | ознака | опис | детаљније |  | у секундама | ознака | опис | детаљније |
| **100** | **10-9** | **1 ns** | 1 наносекунда | милијардити део секунде | **1\*100** | **10-9** | **1 ns** | 1 наносекунда | милијардити део секунде |
| **101** | **10-8** | **10 ns** | 10 наносекунда | стомилионити део секунде | **3\*100** | **3\*10-9** | **3 ns** | 3 наносеунде | 300 милионити део секунде |
| **102** | **10-7** | **100 ns** | 100 наносекунда | десетмилионити део секунде | **1\*101** | **10-8** | **10 ns** | 10 наносекунди | стомилионити део секунде |
| **103** | **10-6** | **1 µs** | 1 миркосекунда | милионити део секунде | **3\*101** | **3\*10-8** | **30 ns** | 30 наносекунди | 30 милионити део секунде |
| **106** | **10-3** | **1 ms** | 1 милисекунда | хиљадити  део секунде | **1\*103** | **10-6** | **1 µs** | 1 миркосекунда | милионити део секунде |
| **109** | **100** | **1 s** | 1 секунда | један секунд | **3\*104** | **3\*10-5** | **30 µs** | 30 миркосекунди | 300 хиљадити део секунде |
| **1012** | **103** | **103 s ≈16 min** | хиљаду секунди | хиљаду секунди | **1\*106** | **10-3** | **1 ms** | 1 милисекунда | хиљадити  део секунде |
| **1015** | **106** | **106 s ≈11 дана** | милон секунди | милон секунди | **3\*107** | **3\*10-2** | **30 ms** | 30 милисекунди | 1/30 (део) секунде |
| **1018** | **109** | **109 s ≈32 године** | милијарду секунди | милијарду секунди | **1\*109** | **100** | **1 s** | 1 секунда | један секунд |
| **1019** | **1010** | **1010 s ≈3 века** | 10 милијарди секунди. | 10 милијарди секунди | **3\*109** | **3\*100** | **3 s** | 3 секунде | три секунде |

У наставку је табеларни преглед потребног времена израженог у секундама и одговарајућим јединицама за мерење времена:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Степен** | у секундама | **ознака** | **префикс** | **назив** | **напомена** |
| **-9** | **10-9** | **1 ns** | нано | наносекунда | милијардити део секунде |
| **-6** | **10-6** | **1 µs** | микро | миркосекунда | милионити део секунде |
| **-3** | **10-3** | **1 ms** | мили | милисекунда | хиљадити део секунде |
| **0** | **100** | **1 s** |  | секунда | један секунд |
| **3** | **103** | **103 s** | хиљаду (кило) | хиљаду секунди | **≈16 минута** |
| **6** | **106** | **106 s** | милион (мега) | милон секунди | **≈11 дана** |
| **9** | **109** | **109 s** | милијарда (гига) | милијарду секунди | **≈32 године** |
| **Степен** | у секундама | **ознака** | **префикс** | **назив** | **напомена** |
| **0** | **100** | **1 s** |  | 1 секунда | један секунд |
| **1** | **101** | **10 s** |  | 10 секунди | 10 секунди |
| **2** | **102** | **100 s** |  | 100 секунди | **≈1 минут** |
| **3** | **103** | **103 s** | хиљаду (кило) | хиљаду секунди | **≈16 минута** |
| **4** | **104** | **104 s** | хиљаду (кило) | 10 хиљада секунди | **≈2,5 сати** |
| **5** | **105** | **105 s** | хиљаду (кило) | 100 хиљада секунди | **≈25 сати ≈ 1 дан** |
| **6** | **106** | **106 s** | милион (мега) | милон секунди | **≈11 дана** |
| **7** | **107** | **107 s** | милион (мега) | 10 милона секунди | **≈110 дана** |
| **8** | **108** | **108 s** | милион (мега) | 100 милона секунди | **≈1100 дана ≈ 3 године** |
| **9** | **109** | **109 s** | милијарда (гига) | милијарду секунди | **≈32 године** |

## Питања за размишљање

Размисли и покушај да припремиш одговоре на следећа питања (редом):

1. Шта је потребно изменити у програмском коду уколико је потребно да одредимо и испишемо **најближи прост број** датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109**?  
   <https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-05-prost-broj#tab_133476>
2. Шта је потребно изменити у програмском коду уколико је потребно да **одредимо број простих чинилаца** датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109**?
3. Шта је потребно изменити у програмском коду уколико је потребно да **испишемо све просте чиниоце** датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109**?
4. Шта је потребно изменити у програмском коду уколико је потребно да **одредимо број делилаца** датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109**?
5. Шта је потребно изменити у програмском коду уколико је потребно да **испишемо све делиоце** датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109+7**?
6. Шта је потребно изменити у програмском коду уколико је потребно да **одредимо колико има простих бројева** који су мањи од датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109+7**?
7. Шта је потребно изменити у програмском коду уколико је потребно да **испишемо све просте бројеве** који су мањи од датог природног броја **N**?  
   Да ли можеш да процениш колико ће бити потребно времена за извршавање тог програма за **N=109+7**?
8. Своја решења провери на Арени на адреси:   
   <https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-05-prost-broj>

## Литература

* <https://petlja.org/sr-Latn-RS/kurs/17862/23/2755> Курс на Петљи за Р1 5.5. Прост број
* <https://petlja.org/sr-Latn-RS/kurs/17918/1/5314> Курс на Петљи за Р3 1.2. Тест прималности (0.1s)
* <https://petlja.org/sr-Latn-RS/biblioteka/r/Zbirka2/prost_broj> Збирка 2: Одсецање (1.5s)
* <https://petlja.org/sr-Latn-RS/biblioteka/r/Zbirka3/prost_broj> Збирка 3: Дељивост и прости бројеви (0.1s)
* <https://petlja.org/sr-Latn-RS/biblioteka/r/problemi/Zbirka-stara/prost_broj> (1.0s)
* <https://petlja.org/sr-Latn-RS/biblioteka/r/Problems/PRIME1> (1.0s)
* <https://petlja.org/sr-Latn-RS/biblioteka/r/Lectures/matematicki-algoritmi-1>
* <https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/prirucnik-csharp/postepeno-izvrsavanje-programa>
* <https://petlja.org/sr-Latn-RS/biblioteka/r/lekcije/malecki-uvod/maleckiuvod_02_slozenost>
* <https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-05-prost-broj#tab_132331> (1.5s)
* <https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-05-prost-broj#tab_133475> (0.1s)
* <https://arena.petlja.org/sr-Latn-RS/competition/r1-t05-deljivost-prosti-tle-001-2024#tab_132331> (1.5s)
* <https://arena.petlja.org/sr-Latn-RS/competition/r1-03-petlje-04-deljivost#tab_91511> (1.0s)
* <https://arena.petlja.org/sr-Latn-RS/competition/tkm001#tab_91511> (1.0s)
* <https://www.onlinegdb.com/JeTNwk0_R>
* <https://onlinegdb.com/Enxnz_kWT>
* <https://github.com/draganilicnis/R1_T05_05_Deljivost_Prost_TLE>
* <https://github.com/draganilicnis/R3_T01_01_Algebarski_algoritmi_01_Test_primalnosti_Prost_broj>
* <https://petlja.org/sr-Latn-RS/kurs/17918/1/5314> Курс на Петљи за 3. разред 1.2. Тест прималности