Доклад

по Алгоритми и структури от данни

Тема – Динамично програмиране

1.Увод в динамичното програмиране. Разбиване на задача на подзадачи. Състояние. Мемоизация

2.Опишете следните задачата и решението им (кода да се компилира)

2.1.Задача за раницата - опишете задачата и решението и

2.2.Най-дълга нарастваща подредица

2.3.Триъгълник на Паскал

3.Двумерни задачи за динамично програмиране. Примери

4.Приложение на рекурсията в динамичното програмиране. Примери

1.Увод в динамичното програмиране. Разбиване на задача на подзадачи. Състояние. Мемоизация

Динамичното оптимиране е математически метод за решаване на оптимизационни задачи. Негов аналог, известен като динамично програмиране, служи за решаване на комбинаторни задачи (преброяване на обекти от определен вид). Поради сходството на двата метода имената им често са взаимозаменяеми.

Динамичното оптимиране се използва в математиката, икономиката, информатиката и биоинформатиката. Същността му се състои в решаването на задача чрез разделянето ѝ на набор от по-прости подзадачи, всяка от които се решава само веднъж, а решенията им се запазват в подходяща структура от данни. Когато решението на някоя подзадача потрябва пак, то се взима наготово. Така се пести време, но за сметка на изразходването на повече памет. Техниката за запомняне на решенията на подзадачите (вместо повторното им решаване) се нарича „мемоизация“.

Динамичното оптимиране е коректно, но по-бавно и изразходва повече памет, тъй като пази не едно възможно решение, а няколко – толкова, колкото е нужно. От друга страна, динамичното оптимиране е много по-бързо от пълното изчерпване, защото разглежда не всички възможни случаи, а само най-малкия необходим брой.

Мемоизация е пример за използване на кеш при разработка на софтуер, при програмиране, съхраняване на резултатите от изпълнение на функции за предотвратяване на повтарящи се изчисления. Това е един от методите за оптимизация , използван за увеличаване на скоростта на изпълнение на компютърни програми . Преди извикване на функция се проверява дали функцията е била извиквана преди:

* ако не е извикана, тогава функцията се извиква и резултатът от нейното изпълнение се записва;
* ако се извика, се използва запазеният резултат.

Мемоизацията може да се използва не само за увеличаване на скоростта на дадена програма. Например, той се използва при взаимно рекурсивен анализ отгоре надолу в обобщения алгоритъм за анализ отгоре надолу [1] .

2.Опишете следните задачата и решението им (кода да се компилира)

2.1.Задача за раницата

Дадено:

Като се имат предвид теглата и стойностите на N артикула, поставете тези артикули в раница с капацитет W , за да получите максималната обща стойност в раницата. Тук можем да разбиваме предмети за максимизиране на общата стойност на раницата.

Вход:

Елементи като двойки (стойност, тегло)

arr[] = {{60, 10}, {100, 20}, {120, 30}}

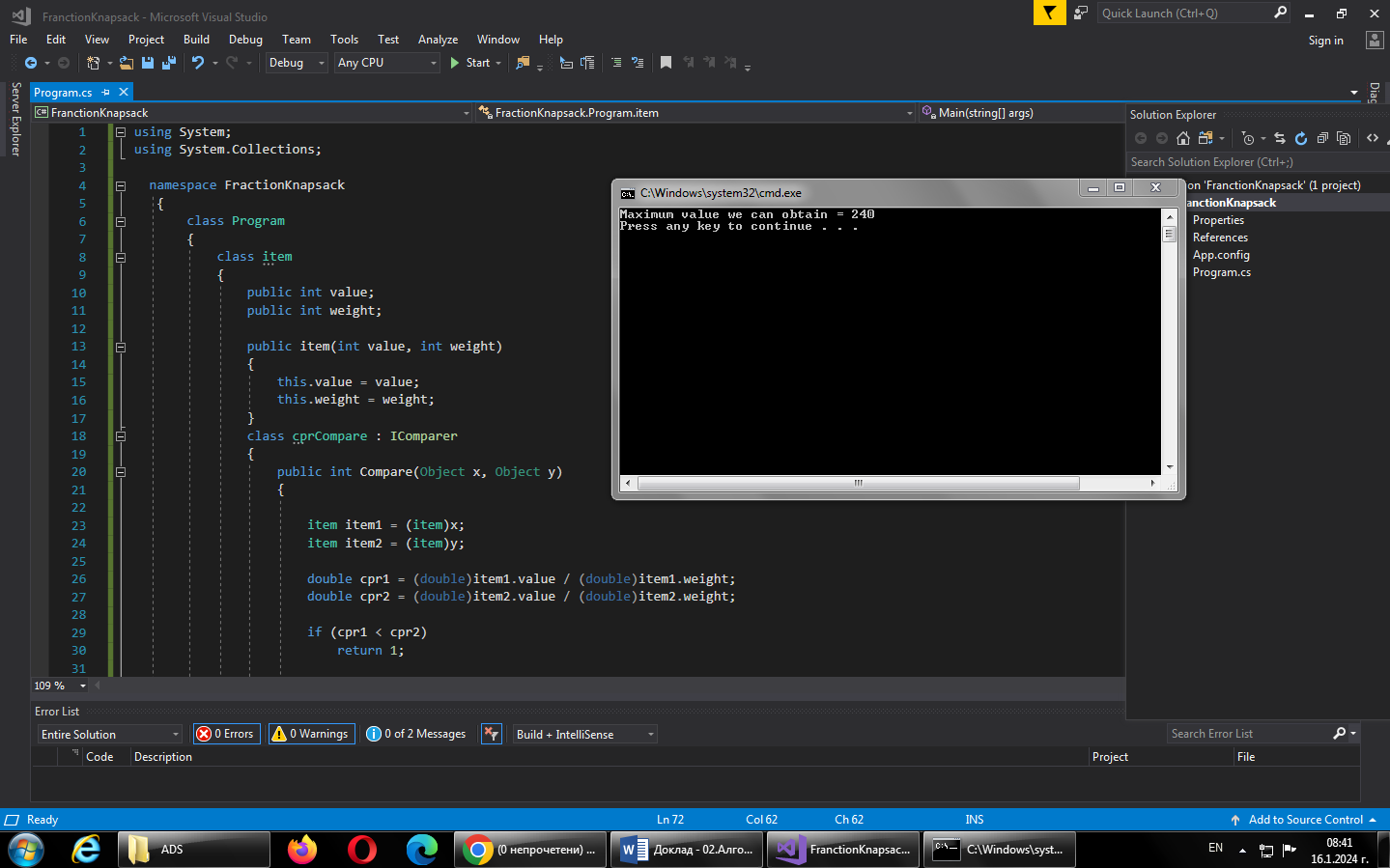
Капацитет на раницата, W = 50

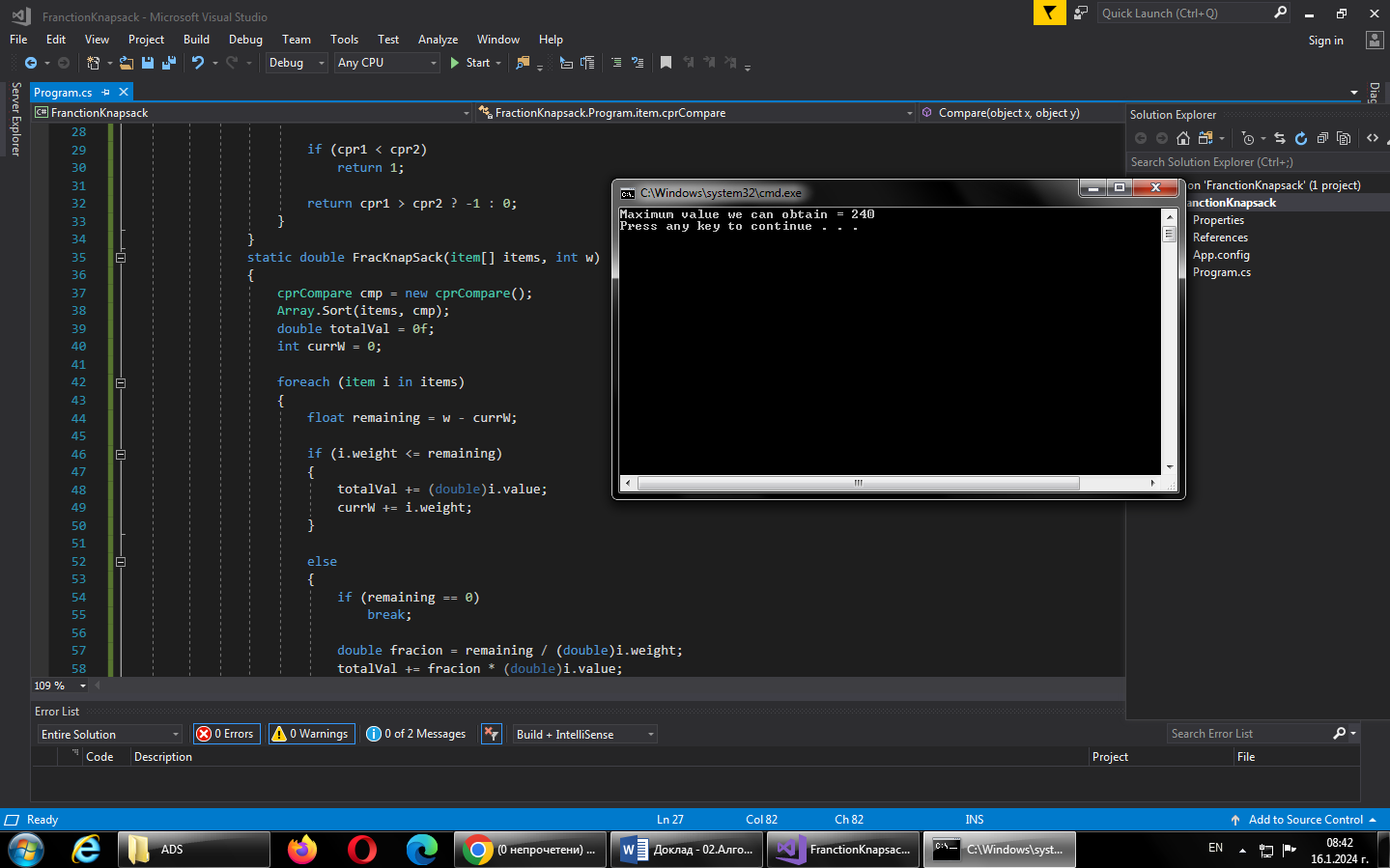
Изход: Максимална възможна стойност = 240

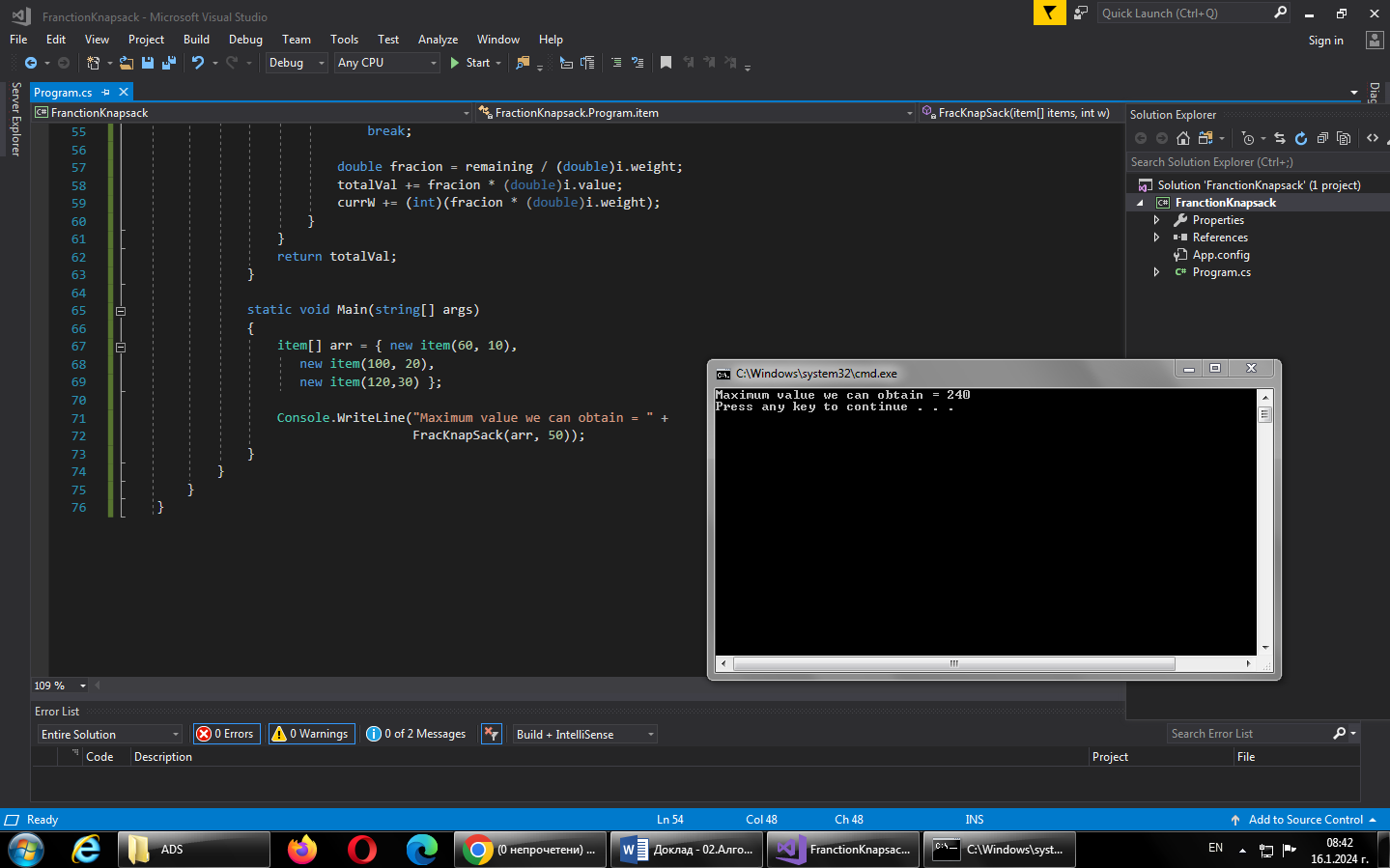
Обяснение: като се вземат предмети с тегло 10 и 20 kg и 2/3 част от 30 kg.

Следователно общата цена ще бъде 60+100+(2/3)(120) = 240

Решение:







2.2.Най-дълга нарастваща подредица

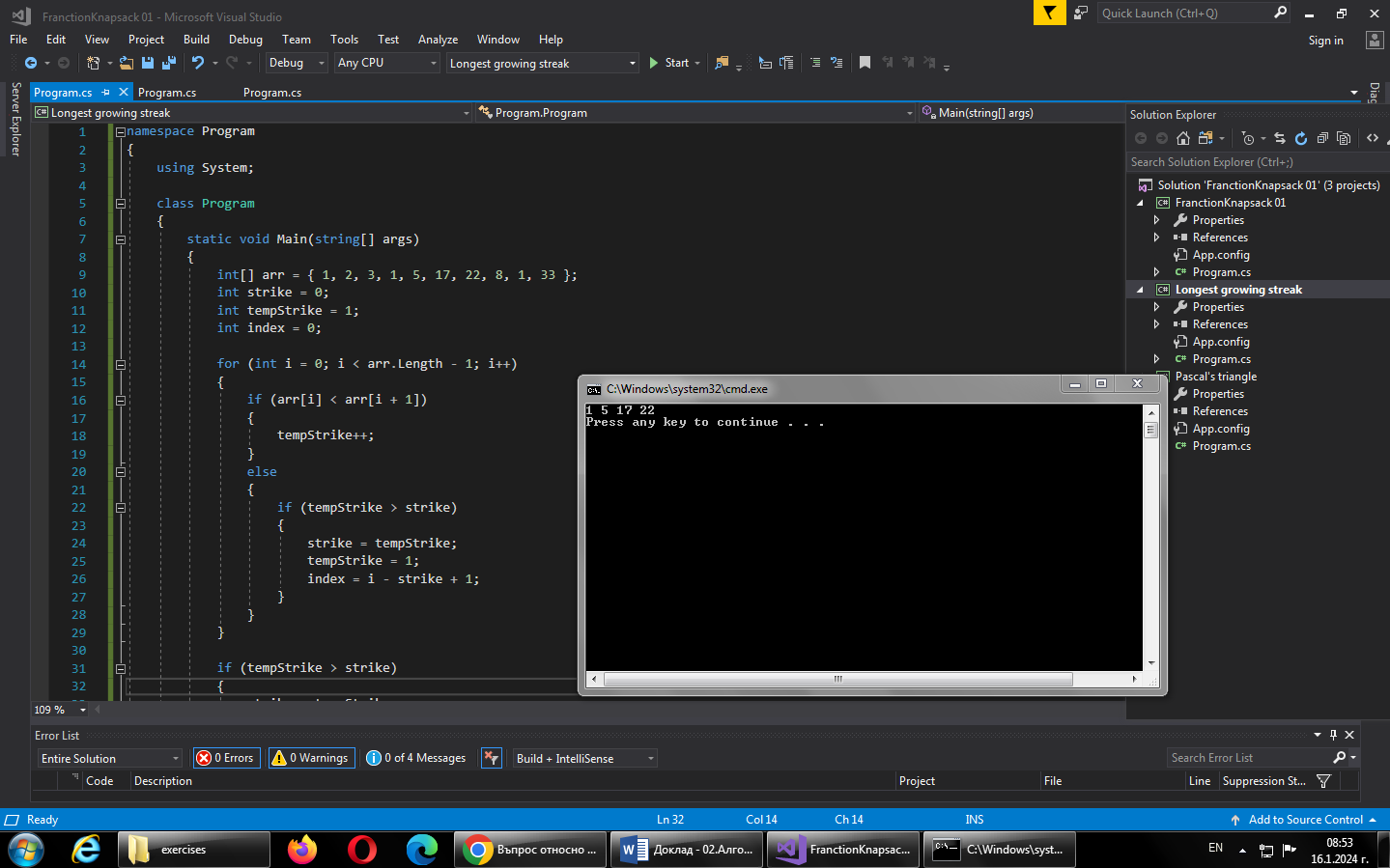
Дадено:

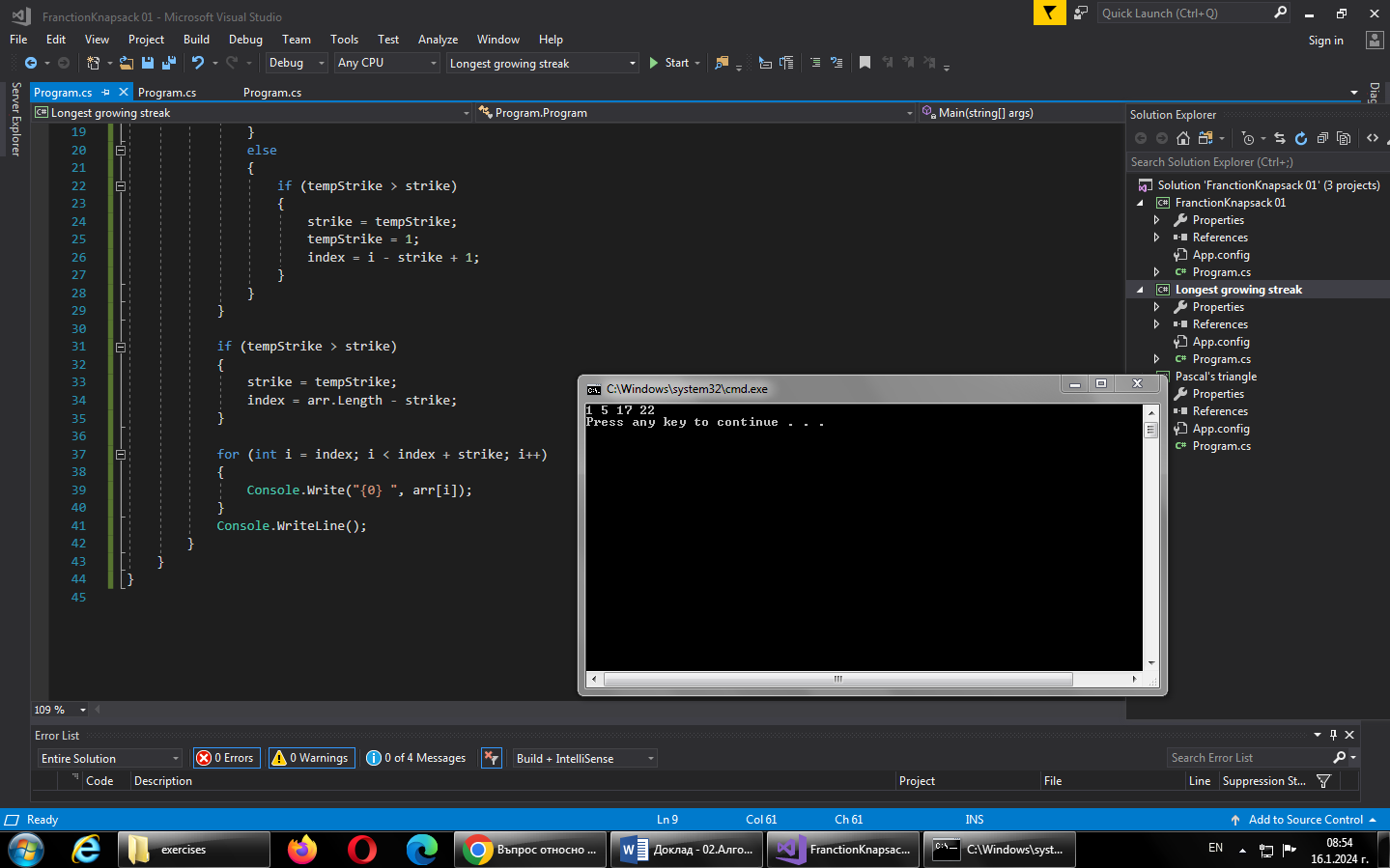
Даден ни е целочислен масив, примерно:

int[] matrix = { 1, 2, 3, 1, 5, 17, 22, 8, 1, 33 };

Намерете най-дългата поредица от нарастващи елементи и да я запишете примерно в масив. В случая тя е: 1, 5, 17, 22.

Решение:





3.Двумерни задачи за динамично програмиране. Примери

Зад.01 Дадено:

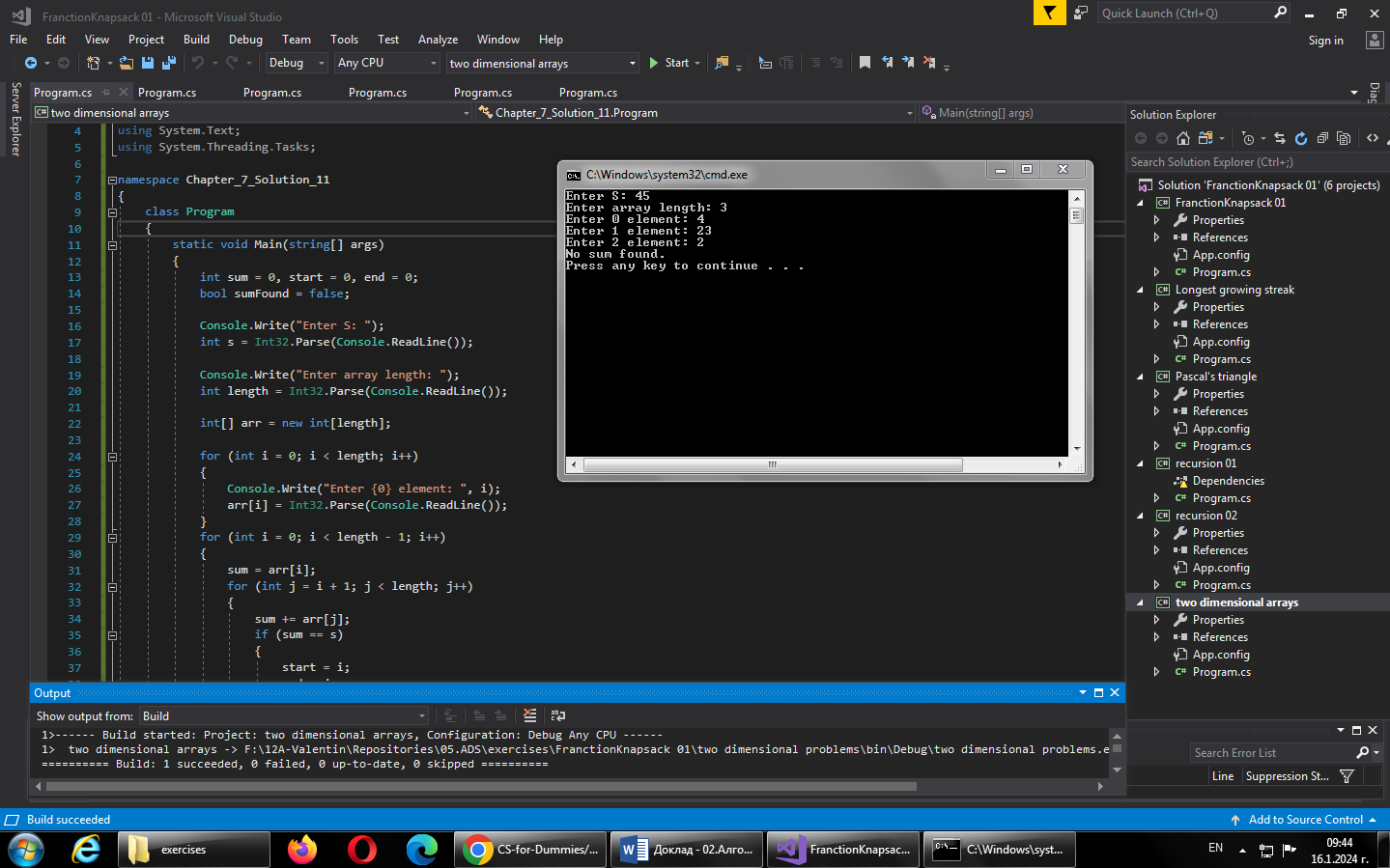
Напишете програма, която намира максималната подредица от нараст­ващи елементи в масив arr[n]. Елементите може и да не са последо­вателни.

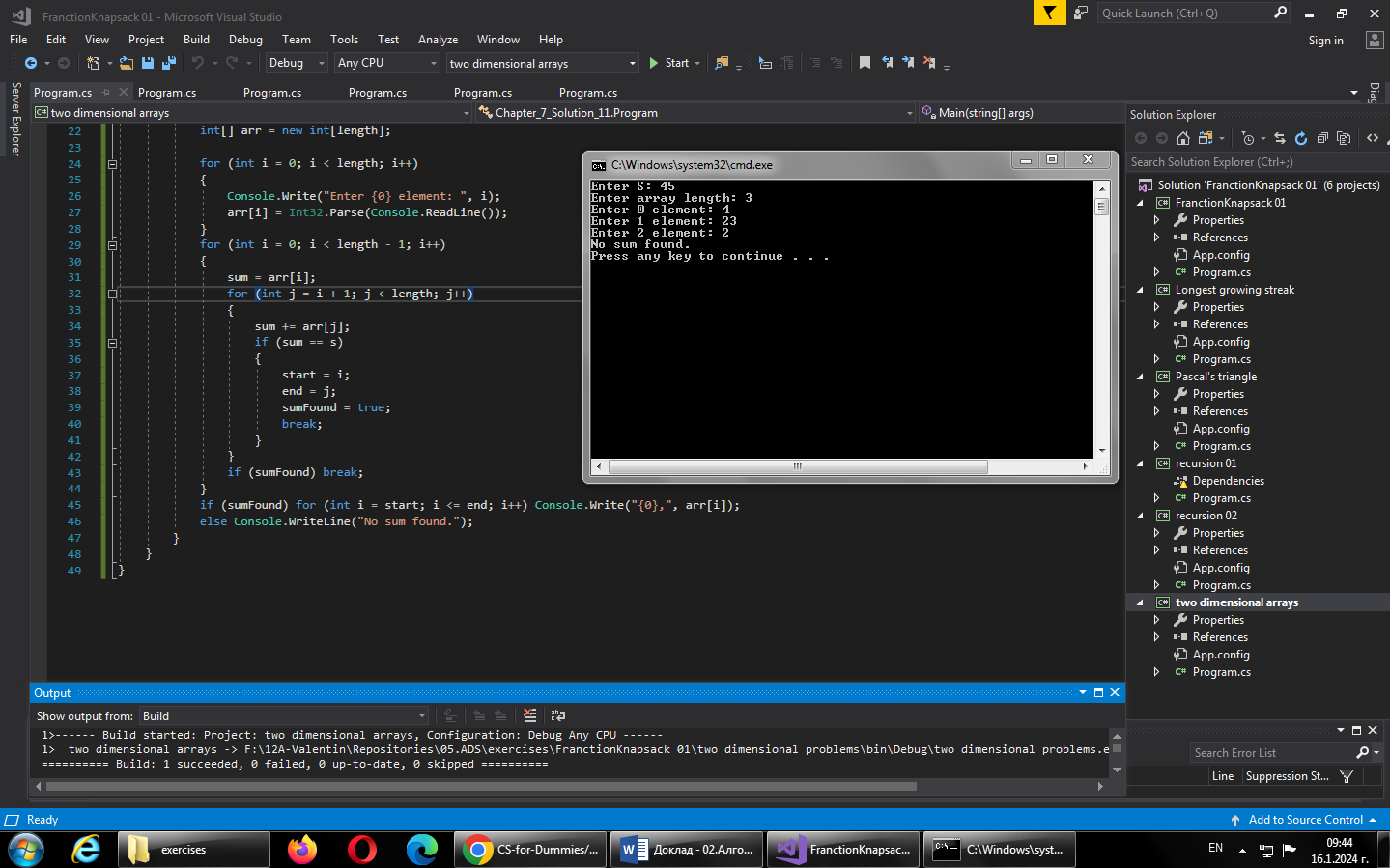
Пример: {9, 6, 2, 7, 4, 7, 6, 5, 8, 4} à {2, 4, 6, 8}.

Упътване: Задачата може да се реши с два вложени цикъла и допълнителен масив len[0..n-1] . Нека в стойността len[i] пазим дължината на най-дългата нарастваща подредица, която започва някъде в масива (не е важно къде) и завършва с елемента arr[i] . Тогава len[0]=1, a len[x] е максималната сума max(1+len[prev]) , където prev < x и arr[prev] < arr[x] . Следвайки дефиницията len[0..n-1] може да се пресметне с два вложени цикъла по следния начин: първият цикъл обхожда масива последователно отляво надясно с водеща променлива x. Вторият цикъл (който е вложен в първия) обхожда масива от началото до позиция x-1 и търси елемент prev с максимална стойност на len[prev], за който arr[prev] < arr[x] . След приключване на търсенето len[x] се инициализира с 1 + най-голямата намерена стойност на len[prev] или с 1, ако такава не е намерена.

Описаният алгоритъм намира дължините на всички максимални нарастващи подредици, завършващи във всеки негов елемент. Най-голямата от тези стойности е дължината на най-дългата нарастваща подредица. Ако трябва да намерим самите елементи съставящи тази максимална нарастваща подредица, можем да започнем от елемента, в който тя завършва (нека той е на индекс x), да го отпечатаме и да търсим предходния елемент (prev). За него е в сила, че prev < x и len[x]=1+len[prev] . Така намирайки и отпечатвайки предходния елемент докато има такъв, можем да намерим елементите съставящи най-дългата нарастваща подредица в обратен ред (от последния към първия).

Решение:





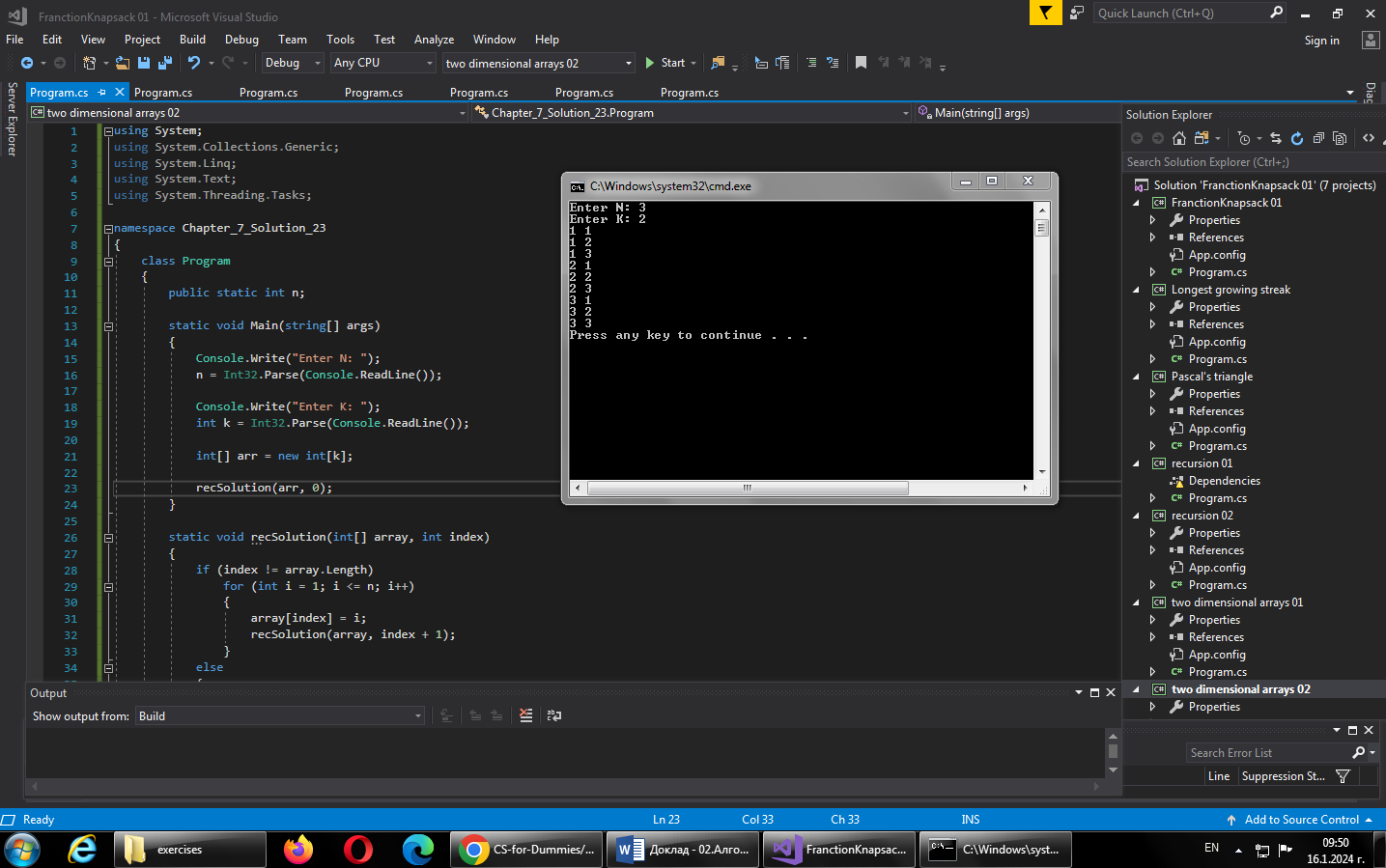
зад. 02 Дадено:

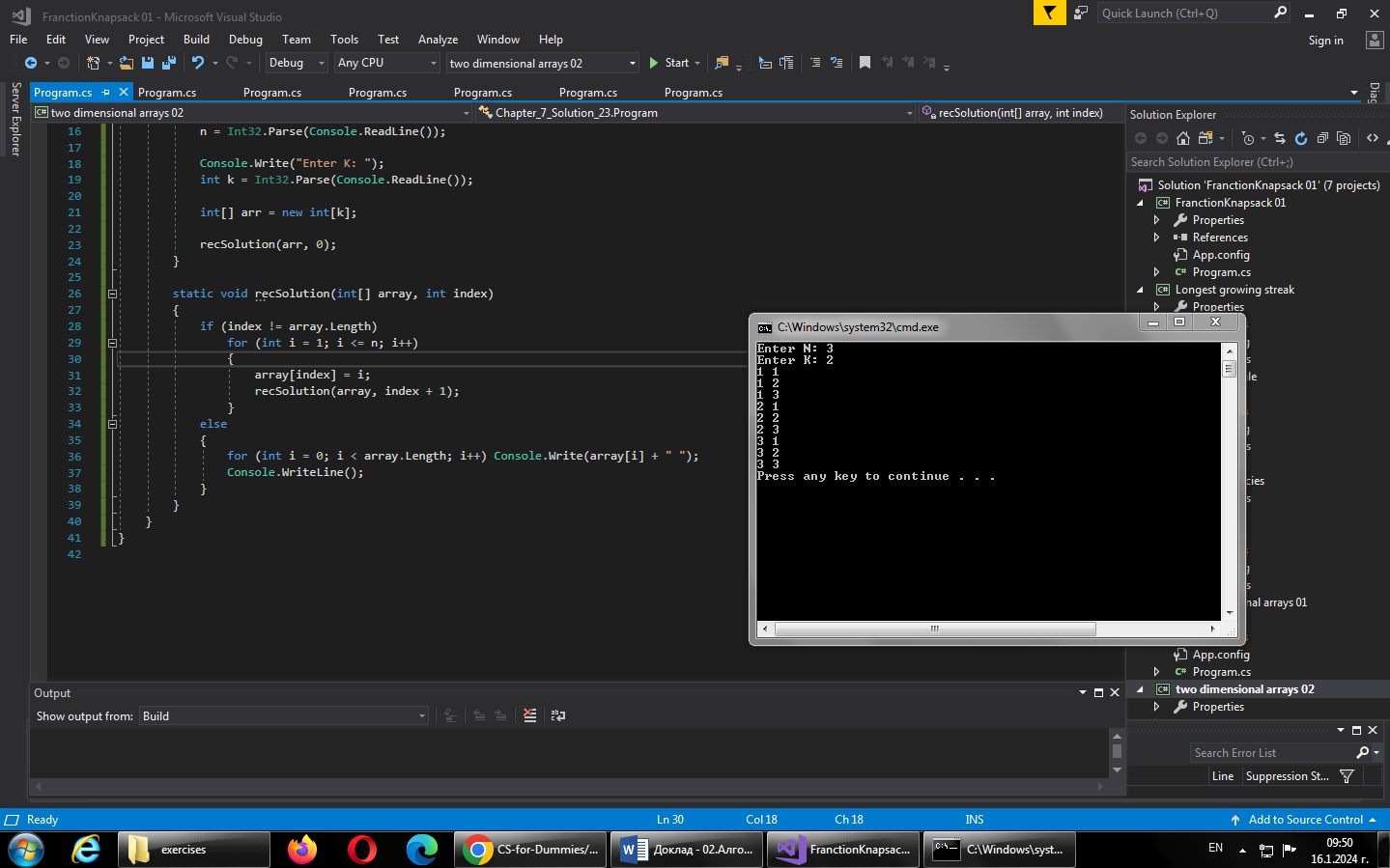
Напишете програма, която прочита цяло число N от конзолата и отпечатва всички пермутации на числата [1…N].

Пример: N = 3 à {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}

Упътване: Задачата може да се реши с рекурсия. Напишете подходяща рекурсия и всеки път променяме позицията на всеки елемент.

Решение:





4.Приложение на рекурсията в динамичното програмиране

Рекурсия, която съдържа само едно извикване се нарича единична рекурсия, а рекурсия, при която е налице многократно извикване се нарича множествена рекурсия. Стандартни примери за единична рекурсия са обхождане на списък, например при линейно търсене, или при изчисляване на факториел, а такива за множествена рекурсия са: обхождане на дърво или при намиране членовете на редица на Фибоначи.

В повечето случаи единичната рекурсия е по-ефективна от множествената и може да бъде заменена с итеративно изчисление, изпълняващо се за линейно време, заделяйки постоянно количество памет. Множествената рекурсия, от друга страна, може да изисква експоненциално нарастващо процесорно време и памет и е по-фундаментално рекурсивна, като не може да бъде заменена от итерация без изрични стекови операции.

Множествената рекурсия може да бъде сведена до единична (респективно преобразувана в итерация). Например, при изчисление на поредицата на Фибоначи, действието може да бъде сведено до единична рекурсия чрез подаването на две последователни стойности като параметри. Това се нарича двойна рекурсия, при която на всяка стъпка се подават два параметъра.

Принципи при приложението на рекурсия

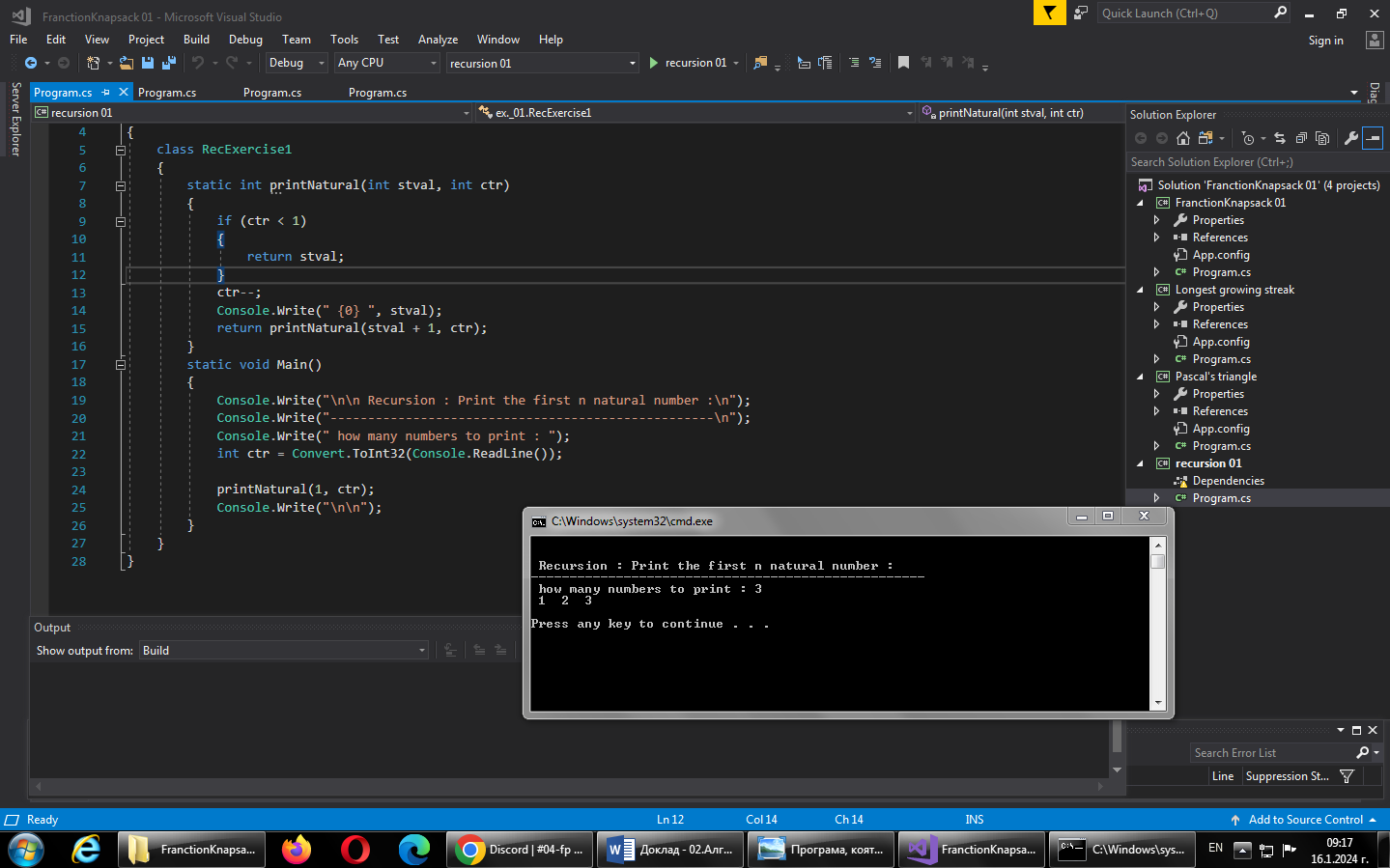
В опростен сценарий при извикване на рекурсивен метод, методът връща резултат, който представлява базовия случай. Ако методът трябва да се справи с по-сложен проблем, то проблемът бива разделен на две условни части: част, с която методът може да се справи и опростена версия на първоначалното състояние. Въпреки че опростената версия е нова за метода, поради приликите ѝ с оригиналната версия, налице е рекурсивно извикване, с което методът решава новата версия.

За да приключи рекурсивното извикване, методът извиква себе си с опростена версия на първоначалното състояние, като всяко последващо опростяване трябва да произхожда от базовия случай. Когато методът разпознае базовия сценарий (най-простата версия), резултатът се подава обратно към предишното извикване и чрез последователност от връщания се достига до първоначалното извикване на метода, което връща крайния резултат.

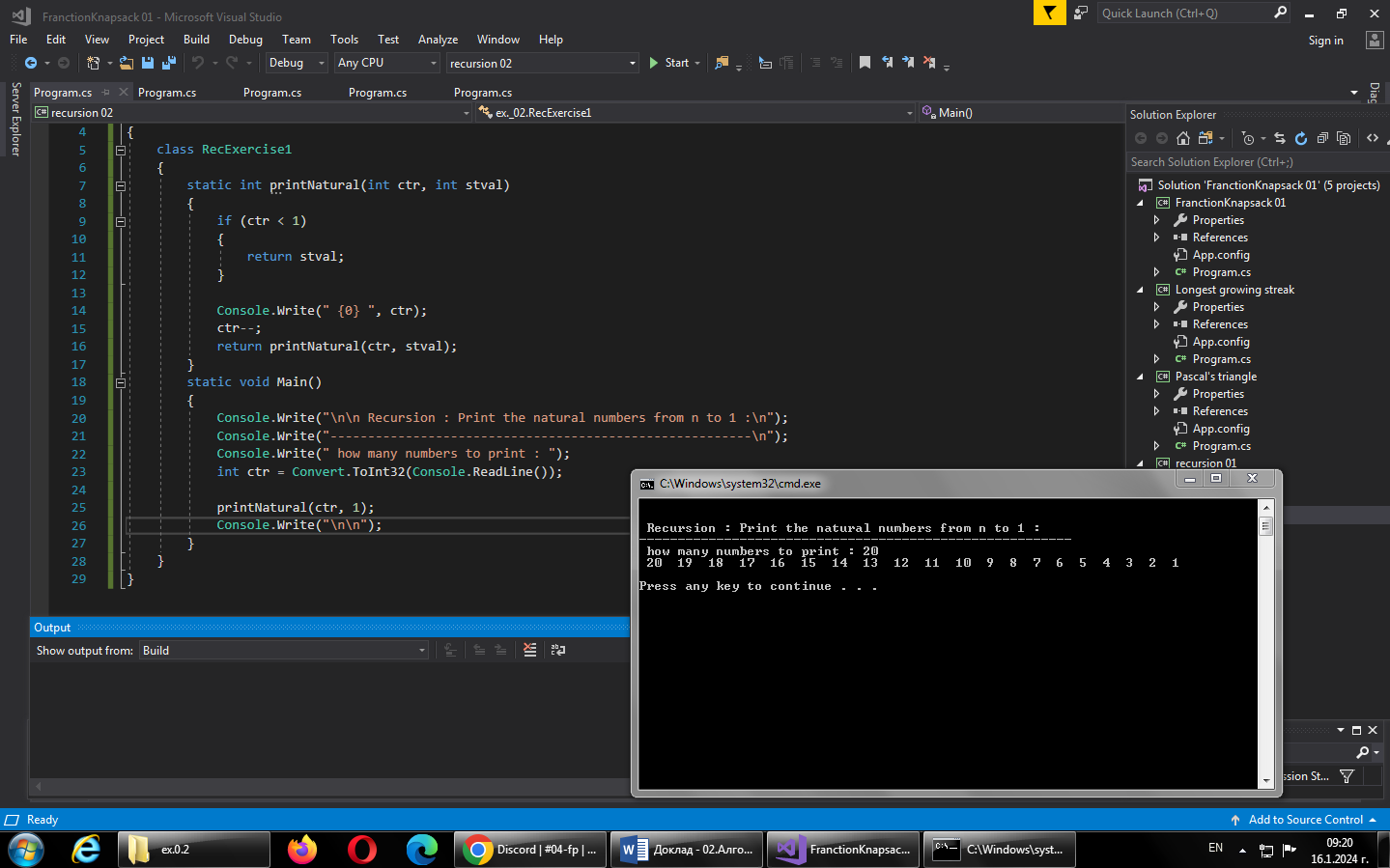
Реализацията на рекурсия изисква голям разход на процесорно време и памет, поради множественото извикване на даден метод.

Примери:

Програма, която отпечатва на екрана n на брой числа рекурсивно



Програма, която отпечатва на екрана рекурсивно всички числата от n до 1



Източници:

<https://github.com/BG-IT-Edu/School-Programming/blob/main/Courses/Applied-Programmer/Algo-and-Data-Structures-Advanced/04-dynamic-programming.pdf>

<https://softuni.bg/forum/29295/more-exercises-arrays-5-longest-increasing-subsequence-lis>

https://softuni.bg/forum/5228/vypros-otnosno-namiraneto-na-nay-dylga-poredica-ot-narastvashti-elementi-v-celochislen-masiv

https://github.com/ivanpop/CS-for-Dummies/blob/master/Chapter%207%20Solution%206/Program.cs

<https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F>

<https://ivanpop.azurewebsites.net/bg/Projects/CSharpForDummies/7>

<https://github.com/ivanpop/CS-for-Dummies/blob/master/Chapter%207%20Solution%2023/Program.cs>

Изготвил: Валентин Кьосев 12„а“ клас