دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۲*

ملیکا نوبختیان
هادی شیخی
سید صالح اعتمادی

نیم‌سال اول ۰۰-۹۹

ایمیل/تیمز	m_nobakhtian@comp.iust.ac.ir ha_sheikhi@comp.iust.ac.ir
نام شاخه	fb_A2
نام پروژه/پوشه/پول ریکوست	A2
مهلت تحویل	۹۹/۷/۱۹

*تشکر ویژه از خانم مریم سادات هاشمی که در نیم‌سال اول سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند. همچنین از اساتید حل‌تمرین نیم‌سال اول سال تحصیلی ۹۹-۹۸ سارا کدیری، محمد مهدی عبداللهپور، مهدی مقدمی، مهسا قادران، علیرضا مرادی، پریسا یل‌سوار، غزاله محمودی و محمدجواد میرشکاری که مستند این مجموعه تمرین‌ها را بهبود بخشیدند، متشکرم.

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A2 بسازید.

۲. کلاس هر سوال را به پروژهی خود اضافه کنید و در قسمت مربوطه کد خود را بنویسید. هر کلاس شامل دو متد اصلی است:

- متد اول: تابع Solve است که شما باید الگوریتم خود را برای حل سوال در این متد پیاده سازی کنید.
- متد دوم: تابع Process است که مانند تمرین های قبلی در TestCommon پیاده سازی شده است. بنابراین با خیال راحت سوال را حل کنید و نگران تابع Process نباشید! زیرا تمامی پیاده سازی ها برای شما انجام شده است و نیازی نیست که شما کدی برای آن بزنید.

۳. اگر برای حل سوالی نیاز به تابع های کمکی دارید؛ می توانید درکلاس مربوط به همان سوال تابع تان را اضافه کنید.

اکنون که پیاده سازی شما به پایان رسیده است، نوبت به تست برنامه می رسد. مراحل زیر را انجام دهید.

۱. یک UnitTest برای پروژهی خود بسازید.

۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژهی تست خود اضافه کنید.

۳. فایل GradedTests.cs را به پروژهی تستی که ساخته اید اضافه کنید.

توجه:
برای اینکه تست شما از بهینه سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست ها را روی بیلد Release امتحان کنید، درغیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Collections.Generic;
using TestCommon;

namespace A2.Tests
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod()]
        public void SolveTest_Q1NaiveMaxPairWise()
        {
            RunTest(new Q1NaiveMaxPairWise("TD1"));
        }

        [TestMethod(), Timeout(1500)]
        public void SolveTest_Q2FastMaxPairWise()
        {
            RunTest(new Q2FastMaxPairWise("TD2"));
        }

        [TestMethod()]
        public void SolveTest_StressTest()
        {
            Assert.Inconclusive();
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A2", p.Process, p.TestDataName, p.Verifier);
        }

    }
}
```

Maximum Pairwise Product

در این تمرین شما باید حداکثر حاصل ضرب دو عدد متمایز را در یک دنباله از اعداد صحیح غیر منفی پیدا کنید.

ورودی: دنباله ای از اعداد صحیح غیر منفی.
خروجی: حداکثر مقدار که می توان با ضرب دو عنصر مختلف از دنباله به دست آورد.

محدودیت ها: $0 \leq a_1, ..., a_n \leq 2 * 10^5$; $2 \leq n \leq 2 * 10^5$.

- محدودیت زمانی : **۱۵۰۰ میلی ثانیه**
- محدودیت حافظه: **۵۱۲ مگابایت**

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		7	4
7	35	42	14		28
4	20	24	8	28	

۱ Naive Algorithm

ساده ترین روش حل کردن مسئله‌ی Maximum Pair Wise Product این است که تمام دودویی های ممکن را چک کرده و دو عنصر با بزرگترین خروجی را پیدا کنیم:

```
MaxPairwiseProductNaive(A[1 . . . n]):
product ← 0
for i from 1 to n:
    for j from 1 to n:
        if i ≠ j:
            if product < A[i] · A[j]:
                product ← A[i] · A[j]
return product
```

الگوریتم بالا را در تابع Solve کلاس Q1NaiveMaxPairWise پیاده سازی کنید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A2
{
    public class Q1NaiveMaxPairWise : Processor
    {
        public Q1NaiveMaxPairWise(string testDataName) : base(testDataName) { }
        public override string Process(string inStr) =>
            Solve(inStr.Split(new char[] { '\n', '\r', ' ' },
                StringSplitOptions.RemoveEmptyEntries)
                 .Select(s => long.Parse(s))
                 .ToArray()).ToString();

        public virtual long Solve(long[] numbers)
        {
            throw new NotImplementedExceptoin();
        }
    }
}
```

حال تست SolveTest_Q1NaiveMaxPairWise را اجرا کنید. زمان زیادی طول می کشد تا تست فوق پاس شود.(چرا؟) در سوال بعد الگوریتم بهینه تری برای حل این سوال پیاده سازی خواهیم کرد.

تذکر: شما می توانید متد های تست دیگری به غیر از دو متد هایی که در بالا ازشما خواسته شده است، در پروژه ی تست خود داشته باشید و داده های دلخواه خود را امتحان کنید.

۲ Fast Algorithm

در این بخش برای حل مشکل Naive Algorithm راهی مطرح شده است. از آنجا که ما فقط به دو عنصر بزرگ موجود نیاز داریم، تنها دو لوپ کافی است، در لوپ اول بزرگترین عنصر و در لوپ دوم بزرگترین عنصر از بین عناصر باقیمانده را پیدا میکنیم:

```
MaxPairwiseProductFast(A[1 . . . n]):
index_1 ← 1
for i from 2 to n:
    if A[i] > A[index_1]:
        index_1 ← i
index_2 ← 1
for i from 2 to n:
    if A[i] ≠ A[index_1] and A[i] > A[index_2]:
        index_2 ← i
return A[index_1] · A[index_2]
```

الگوریتم بالا را در تابع Solve کلاس Q2MaxPairWiseFast پیاده سازی کنید.
توجه کنید که الگوریتم شما باید تمامی TestCase ها را در ۱۵۰۰ میلی ثانیه پاس کند.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using TestCommon;
7
8  namespace A2
9  {
10     public class Q2FastMaxPairWise : Processor
11     {
12         public Q2FastMaxPairWise(string testDataName) : base(testDataName) { }
13         public override string Process(string inStr) =>
14             Solve(inStr.Split(new char[] { '\n', '\r', ' ' },
15                 StringSplitOptions.RemoveEmptyEntries)
16                  .Select(s => long.Parse(s))
17                  .ToArray()).ToString();
18
19         public virtual long Solve(long[] numbers)
20         {
21             throw new NotImplementedExceptoin();
22         }
23     }
24 }
```

تست SolveTest_Q2FastMaxPairWise را اجرا کنید. مشکل کجاست؟
برای اینکه متوجه شوید که دلیل این مشکل چیست و در چه حالتی این اتفاق رخ می دهد، از Stress Testing استفاده می کنیم.

۳ Stress Testing

اکنون Stress Testing را معرفی می کنیم. یک روش برای تولید هزاران Test با هدف پیدا کردن یک TestCase که راه حل شما در آن ناکام است.

Stress Testing شامل چهار بخش است:

۱. اجرای الگوریتم شما.

۲. یک الگوریتم با آهسته از نظر زمانی اما با ارایه پاسخ صحیح برای یک مشکل مشابه.

۳. یک مولد تست تصادفی.

۴. یک حلقه بی نهایت که در آن تست جدید تولید می شود و در هر دو پیاده سازی الگوریتم به مقایسه نتایج می پردازد. اگر نتایج آنها متفاوت باشد، تست و هر پاسخ هر دو پیاده سازی خروجی هستند، و برنامه متوقف می شود، در غیر این صورت حلقه تکرار می شود.

ایده Stress Testing این است که دو پیاده سازی صحیح برای یک مسئله باید برای هر TestCase یک جواب بدهد (در صورتی که پاسخ به این مشکل منحصر به فرد باشد). اگر، با این حال، یکی از پیاده سازی ها نادرست باشد، پس تستی وجود دارد که پاسخ های دو پیاده سازی با هم متفاوت هستند. تنها در یک حالت این طوری نیست و آن زمانی است که برای هر دو پیاده سازی یک اشتباه مشابه وجود داشته باشد که چنین حالتی بعید است (مگر اینکه اشتباه جایی در دستورات ورودی / خروجی است که برای هر دو راه حل مشترک است). در واقع، اگر یک پیاده سازی صحیح باشد و دیگری اشتباه، حتما یک TestCase وجود دارد که پاسخ این دو پیاده سازی با هم متفاوت باشند. اگر هر دو پاسخ اشتباه بدهند، احتمالا یک تست وجود دارد که دو پیاده سازی نتایج متفاوتی را ارائه می دهند.

```
MaxPairwiseProductFast(A[1 . . . n]):
index_1 ← 1
for i from 2 to n:
    if A[i] > A[index_1]:
        index_1 ← i
index_2 ← 1
for i from 2 to n:
    if A[i] ≠ A[index_1] and A[i] > A[index_2]:
        index_2 ← i
return A[index_1] · A[index_2]
```

اکنون که با Stress Testing آشنا شدید، با استفاده از توضیحات بالا، برای دو الگوریتم Naive و Fast که در بخش های قبل پیاده سازی کردید؛ یک Stress Test بنویسید تا متوجه شوید که دنباله‌ی ورودی به چه صورت که باشد الگوریتم Fast جوابی متفاوت از الگوریتم Naive می‌دهد و فکر کنید که برای حل این مشکل چه تغییری باید در الگوریتم Fast خود ایجاد کنید تا جواب هر دو الگوریتم یکسان و صحیح باشد. Stress Test را در فایل GradedTests.cs و در متود زیر پیاده سازی کنید. توجه کنید که پس از پیاده سازی Assert.Inconclusive را حذف کنید.

```
[TestMethod()]
0 references
public void SolveTest_StressTest()
{
    Assert.Inconclusive();
}
```

۴ Even Faster Algorithm

اکنون شما توانسته اید، به کمک Stress Testing الگوریتم Fast خود را درست کنید و تمامی TestCase ها رو با موفقیت پشت سر بگذارید. توجه کنید چون حلقه ی While ای که در Stress Test نوشته اید یک حلقه ی بی نهایت است، این تست تا ابد تمام نخواهد شد. زیرا الگوریتم Fast شما درست شده است و دیگر در تمامی Test ها جواب یکسانی با الگوریتم Naive خواهد داشت. پس برای جلوگیری از این کار شرط حلقه را به گونه ای بگذارید که حلقه برای ۵ ثانیه اجرا شود.

خسته نباشید، اکنون شما توانستید تمامی مراحل را با موفقیت بگذرانید.

آیا می توانید بگویید مرتبه ی زمانی هر یک از الگوریتم های Naive و Fast چیست؟