```
public static class PrimeClass
{
    public static bool IsPrime(int number)
    {
        if (number <= 1) return false;
        if (number == 2) return true;
        if (number % 2 == 0) return false;

        var boundary = (int)Math.Floor(Math.Sqrt(number));

        for (int i = 3; i <= boundary; i += 2)
            if (number % i == 0)
                return false;

        return true;
    }
    public static IEnumerable<T> TakePrime<T>(this IEnumerable<T> s, int count)
    {
        IEnumerable<T> TakePrime_Aux()
        {
            using (var it = s.GetEnumerator())
            {
                int pos = 0;
                int ris = 0;
                while (it.MoveNext())
                {
                    if (ris == count) break;
                    if (IsPrime(pos))
                    {
                        ris++;
                        yield return it.Current;
                    }
                    pos++;
                }
            }
        }
        if (null == s) throw new ArgumentNullException(nameof(s));
        if (count <= 0) throw new ArgumentOutOfRangeException(nameof(count));
        return TakePrime_Aux();
    }
}

public class TestPrime
{
    [Test]
    public void test1()
    {
        IEnumerable<int> GenIntSeq()
        {
            for (int i = 0; i < 20; i++)
            {
                yield return i;
            }
        }
        Assert.That(GenIntSeq().TakePrime(6), Is.EqualTo(
         new[]{ 2, 3, 5, 7, 11, 13}));
    }

    [TestCase(2)]
    public void test2(int b)
    {
        IEnumerable<int> bPower()
        {
            int i = 0;
```

```
            while (true)
            {
                yield return (int)Math.Pow(b, i);
                i++;
            }
        }
        Assert.That(()=> bPower().TakePrime(0),
         Throws.TypeOf<ArgumentOutOfRangeException>());
    }

    [TestCase(50)]
    public void test3(int size)
    {
        IEnumerable<int> GenSeq()
        {
            int i = 0;
            while (true)
            {
                yield return i;
                i++;
            }
        }

        var ris = GenSeq().TakePrime(size).Take(100).ToArray();
        bool testTrue = true;
        for (int i = 0; i < ris.Length -1; i++)
        {
            if (ris[i] > ris[i + 1]) testTrue = false;
        }
        Assert.That(testTrue, Is.True);
    }
}
```