Exercises: Data Structures, Algorithms and Complexity

This document defines the **in-class exercises** assignments for the "Data Structures" course @ Software University.

Problem 1. Check Prime – Calculate the Complexity (Worst Case)

Calculate the expected running time **O(f(n))** in the **worst case** for the following C# function:

```
static bool IsPrime(long num)
{
    for (int i = 2; i < num; i++)
    {
        if (num % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Problem 2. Check Prime – Calculate the Complexity (Best Case)

Calculate the expected running time **O(f(n))** of the above C# function in the **best case**.

Problem 3. Fast Check Prime – Calculate the Complexity

Calculate the expected running time **O(f(n))** in the **worst case** for the following C# function:

```
static bool IsPrimeFast(long num)
{
    int maxDivisor = (int)Math.Sqrt(num);
    for (int i = 2; i <= maxDivisor; i++)
    {
        if (num % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Problem 4. First N Primes – Calculate the Complexity

Calculate the expected running time **O(f(n))** in the **worst case** for the following C# function:

```
static IList<int> FindFirstNPrimes(int n)
{
    var primes = new List<int>(n);
    int p = 2;
    while (primes.Count < n)
    {
        if (IsPrimeFast(p))
        {
            primes.Add(p);
        }
        p++;
    }
    return primes;
}
```

Problem 5. First N Primes – Calculate the Memory Consumption

Calculate the expected memory consumption `O(f(n))` in the **average case** for the following C# function:

```
static IList<int> FindFirstNPrimes(int n)
{
    var primes = new List<int>(n);
    int p = 2;
    while (primes.Count < n)
    {
        if (IsPrimeFast(p))
        {
            primes.Add(p);
        }
        p++;
    }
    return primes;
}
```

Problem 6. Primes in Range – Calculate the Complexity

Calculate the expected running time `O(f(n))` in the **worst case** for the following C# function:

```
static IList<int> FindPrimesInRange(int start, int end)
{
    var primes = new List<int>();
    for (int p = start; p <= end; p++)
    {
        if (IsPrimeFast(p))
        {
            primes.Add(p);
        }
    }
    return primes;
}
```

Problem 7. Compare Execution Speed

Write a program to **compare the execution speed** of the functions **`IsPrime(p)`** and **`IsPrimeFast(p)`**, e.g.

```
var startTime = DateTime.Now;
for (int i = 0; i < 50000; i++)
{
    IsPrime(i);
}
var executionTime =
    DateTime.Now - startTime;
```

```
var startTime = DateTime.Now;
for (int i = 0; i < 50000; i++)
{
    IsPrimeFast(i);
}
var executionTime =
    DateTime.Now - startTime;
```

Console.WriteLine("Execution time: {0}", executionTime);	Console.WriteLine("Execution time: {0}", executionTime);

Fill the following table to compare the execution time (in seconds):

	p = 1 000	p = 10 000	p = 50 000	p = 100 000	p = 1 000 000
IsPrime(p)					
IsPrimeFast(p)					

Fill "**hangs**" if the execution time is more than a minute.