# Exercises: Data Structures, Algorithms and Complexity

This document defines the **in-class exercises** assignments for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1147/Data-Structures-June-2015).

## 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;  } |
| O(2\*n) |

## Check Prime – Calculate the Complexity (Best Case)

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

O(1)

## 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;  } |
| O(2\*Math.Sqrt(n)) |

## 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;  }  O(Math.Sqrt(p) \* n) |

## 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;  } |
| O(n \* Math.Sqrt(p)) |

## 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;  } |
| O(end \* Math.Sqrt(p)) |

## 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;  Console.WriteLine("Execution time: {0}",  executionTime); | var startTime = DateTime.Now;  for (int i = 0; i < 50000; i++)  {  IsPrimeFast(i);  }  var executionTime =  DateTime.Now - startTime;  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) | 00:00:00.0030001 | 00:00:00.0700040 | 00:00:01.4290817 | 00:00:05.4253103 | hangs |
| IsPrimeFast(p) | 00:00:00.0010001 | 00:00:00.0040002 | 00:00:00.0190011 | 00:00:00.0450026 | 00:00:00.9850564 |

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