ДОДАТОК А

```
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.Emit;
using System.Reflection;
using System.Diagnostics;

namespace ConsoleApp3
{
    internal class Program
    {
        static void Main(string[] args)
        {
            var syntaxTree = CSharpSyntaxTree.ParseText(@"
using System;
using System.IO;

namespace CompileSample
{
    public class Process
    {
        public void Execute()
        {
            string[] lines = new string[1000000];

            for (int i = 0; i < 1000000; i++)

                lines[i] = i.ToString();



            File.WriteAllLines(""c:\\test.txt"", lines);

        }

    }

}");



            var assemblyName = Path.GetRandomFileName();

            var references = new MetadataReference[]

            {

                MetadataReference.CreateFromFile(typeof(object).Assembly.Location),

                MetadataReference.CreateFromFile(typeof(Enumerable).Assembly.Location)

            };



            var compilation = CSharpCompilation.Create(

                assemblyName,

                syntaxTrees: new[] { syntaxTree },

                references: references,

                options: new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary)

            );
            using var ms = new MemoryStream();

            var result = compilation.Emit(ms);

            if (!result.Success)
            {
                IEnumerable<Diagnostic> failures = result.Diagnostics.Where(diagnostic =>
                    diagnostic.IsWarningAsError ||
                    diagnostic.Severity == DiagnosticSeverity.Error);

                foreach (Diagnostic diagnostic in failures)
                {
                    Console.Error.WriteLine("{0}: {1}", diagnostic.Id, diagnostic.GetMessage());
                }
            }
            else
            {
                ms.Seek(0, SeekOrigin.Begin);
                var assembly = Assembly.Load(ms.ToArray());
                var type = assembly.GetType("CompileSample.Process");
                var obj = Activator.CreateInstance(type);

                var cpu = new PerformanceCounter("Processor Information", "% Processor Time",
    "_Total");
}
                var memory = new PerformanceCounter("Memory", "% Committed Bytes In Use");


                var cpuBefore = cpu.NextValue();

                var memoryBefore = memory.NextValue();

                var time = Stopwatch.StartNew();


                type.InvokeMember("Execute",

                    BindingFlags.Default | BindingFlags.InvokeMethod,

                    null,

                    obj,

                    null);


                Console.WriteLine($"CPU cycles usage - from {cpuBefore} to
{cpu.NextValue()}");

                Console.WriteLine($"Memory usage - {memory.NextValue() - memoryBefore}
Bytes");

                time.Stop();

                Console.WriteLine("Execution time - " + time.Elapsed);

            }

        }

    }

}
```