Homework

Task 1. `memoryleak` app (preparation)

Were you missing **`memoryleak`**? So, here is the setup again:

- clone **`git clone https://github.com/sebastienros/memoryleak.git`** (great testing app by Sébastien Ros)
- run it (in **`.\src\MemoryLeak\MemoryLeak`**):

```
dotnet run -c Release
```

- **`https://localhost:5001/`** should present a nice introspective graph about memory usage etc.
- let's make a simple load test agains **`bistring`** endpoint using **`https://github.com/aliostad/SuperBenchmarker`** command-line tool (just download single EXE file from the repository). Run the following command to confirm it is working correctly:

```
.\sb.exe -y 100 -n 10000000 -c 64 -u http://localhost:5000/api/bigstring
```

  As you see we use **`http`** endpoint to avoid unnecessary https handshake overhead.

Task 1.

- Let's observe... `sb.exe` under Performance Monitor, because it is .NET Framework app. Use whatever counters you want to use from `.NET CLR Memory` group, but make sure you add also `# of Pinned Objects`. What is behavior of this counter?
- let's investigate **what's pinned** by using PerfView *.NET* option only (make sure *GC Collect Only* is NOT selected!) for around 10 seconds. In the resulting session observe *Events* table and look for `PinObjectAtGCTime`. Are there any from `sb`? What are the `Typename`s of pinned objects? Look at the *Pinning At GC Time Stacks* view from *Advanced Group* as a nice summary of this data. Look at *Pinned Obj* column in *GC Stats* report.
- as PerfView documentation is saying: *"if you turn on the `'clrPrivate'` provider with stacks (`clrPrivate:@StacksEnabled=true`), it will give additional information on the **exact stack where the pinning took place** for each such pinned object"*. So, let's type `clrPrivate:@StacksEnabled=true` in *Additional Providers* textbox in *Collect* dialog and re-run the session! A new *Pinning Stacks* view in *Advanced Group* should be visible now!
- additionally, if we enable `.NET Alloc` in *Collect* dialog (the one that inject CLR profiler library), we will be able to see **stacks where the pinned object was allocated** included in the *Pinning At GC Time Stacks* view. Just remember that `.NET Alloc` is very expensive and **requires the session to be started BEFORE observed application start**.

Task 1.

- now let's observe .NET 5 application - **`memoryleak`** with the help of CLI Diagnostic Tools. First, observe it under **`dotnet-counters`** to confirm, there is unfortunatelly no counter about pinning...
- then, record session with the help of **`dotnet-trace`**, but using both **`gc-collect`** or **`gc-verbose`** profiles won't be enough. They do not enable providers/keywords necesary for recording **`PinObjectAtGCTime`** events. Try to configure it have all keywords (**`0xffffffffffffffff`**) at *Verbose* (**`5`**) level both from **`Microsoft-Windows-DotNETRuntime`** and **`Microsoft-Windows-DotNETRuntimePrivate`**.