In this project, you are going to jump into using the Go language by implementing a simple sorting program. This course is a “learn by doing” course where you will need to write quite a bit of code. Although it does not have any specific programming language prerequisite, you will need to be able to quickly learn a new language (Go).

This project serves as a diagnostic for how prepared you are for the remainder of the course:

- **Green light**: If you are able to implement a working sort without any particular difficulty in a couple of hours, then you should be well-prepared for the remaining projects
- **Yellow light**: If you were able to implement a working sort, but took more than a couple hours or required lots of extra help, then you may find this course to be quite time intensive, especially the later projects, so plan your schedule accordingly
- **Red light**: If you were not able to implement a working sort program then it is not recommended that you take this course as the projects only get considerably more difficult over time.

GitHub Link

- You can accept the GitHub invitation for the starter code via https://classroom.github.com/a/Hx2zS6xe

Sort specification

This project will read, write, and sort files consisting of zero or more records. A record is a 100 byte binary key-value pair, consisting of a 10-byte key and a 90-byte value. Your sort should be ascending based on key, meaning that the output should have the record with the smallest key first, then the second-smallest, etc.

Project objective

You are to write a sort program that reads in an input file and produces a sorted version of the output file. Use src/sort.go as a starting point. Your program should support the following interface:

Usage:

```
$ bin/sort inputfile outputfile
```

It is OK to read the input file into memory, however you might consider how you could implement sort so that the memory usage of your program does not exceed a pre-defined value (e.g. a few megabytes).

Once the data is in memory, you can use Go's in-built sort function, but to do so you will need to implement a custom comparison function that understands the 100 byte record format.

To aid you in generating input files and validating the output of your program, we've provided you with a few utilities in binary format. Currently there are utilities for x86_64-based Linux and the Intel- and Apple M1 mac.

Utilities

Gensort

Gensort generates random input. If the 'randseed' parameter is provided, the given seed is used to ensure deterministic output.

'size' can be provided as a non-negative integer to generate that many bytes of output. However human-readable strings can be used as well, such as "10 mb" for 10 megabytes, "1 gb" for one gigabyte", "256 kb" for 256 kilobytes, etc. You can specify the input size in any format supported by the https://github.com/c2h5oh/datasize package.

If the specified size is not a multiple of 100 bytes, the requested size will be rounded up to the next multiple of 100.

Usage:

```
$ bin/gensort outputfile size
  -randseed int
       Random seed
```

Showsort

Showsort shows the records in the provided file in a human-readable format, with the key followed by a space followed by an abbreviated version of the value.

Usage:

```
$ bin/showsort inputfile
```

Example output:

```
$ bin/showsort inp
A2842BE0CFA8289159A4    E5D6F869CD25EBF7744F...A7FD7B7C32A28679C819
ADE2AE42C52C75494D89    3BA06FCF049EADAEAC6A...526623739ED465CE8F19
E5C0A05368684A5E8860    405FC5E9EEF2D72952FF...AF051F8877A7D2E86D7D
E0CC8B91522C59D7170C    0B28D7FB9DD4B381D7B2...2FF318B4D4D8A045C747
34EB87BDF72F52CDE0CD    355F1DE83FC363A05175...B9B03AD7CD166BE93036
```

```
38FD9D0ADFB19FE3CFFB    3EAA51C168AC2AFF2FCF...4A4A19C12EB5B40A1C64
8EB6FB642FA3A268FC6B    55B4D40B14FC8892A480...52CD5CD054A29A67DBCD
1FE20F1E67D5E2913DE0    66467F78464536C64BD1...D511B782DADDF770222F
```

Valsort

Valsort scans the provided input file to check if it is sorted. Note that valsort does not verify that the set of key/value pairs in the provided file matches the set from gensort–it only checks that the key-value pairs in the provided file are sorted.

Usage:

```
$ bin/valsort inputfile
```

Building

To build your sort program:

```
$ go build -o bin/sort src/sort.go
```

Verifying your sort implementation

A simple way to verify the correctness of your implementation of sort is to run the standard unix sort command on the output of 'showsort'. For example, to generate, sort, and verify a 1 megabyte file:

```
$ bin/gensort example1.dat "1 mb"
No random seed provided, using current timestamp
Initializing the random seed to 1641144051385376000
Requested 1 mb (= 1048576) bytes
Increasing size to 1048600 to be divisible by 100

$ bin/sort example1.dat example1-sorted.dat

$ bin/valsort example1-sorted.dat
File is sorted

$ bin/showsort example1.dat | sort > example1-chk.txt
$ bin/showsort example1-sorted.dat > example1-sorted-chk.txt
$ diff example1-chk.txt example1-sorted-chk.txt
```

This last 'diff' should simply return to the command prompt. If it indicates any differences that means that there is an error in your sort routine.

You should verify your solution for different file sizes, including 0 bytes and 100 bytes. Your program should support input sizes of at least 10s to 100s of megabytes.

Expected effort level

To set expectations for how much work is involved, we can say that our reference solution of sort.go was 87 lines of code.

###