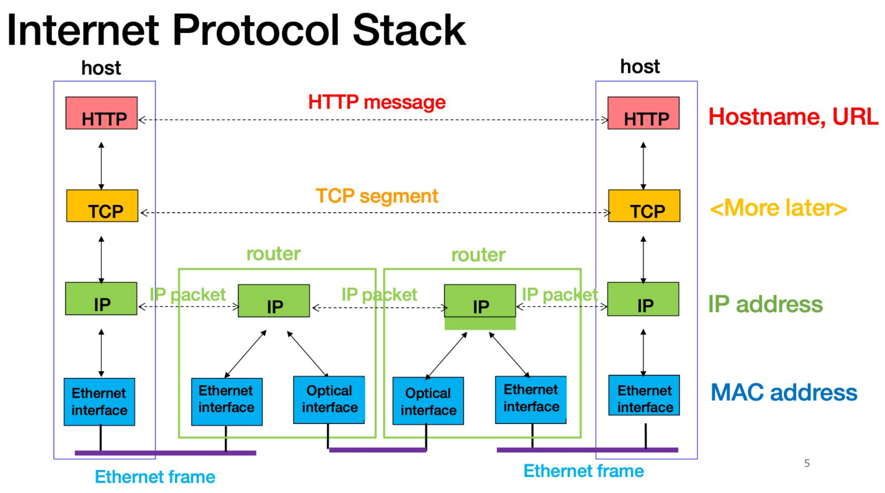
COS 316 Precept #5

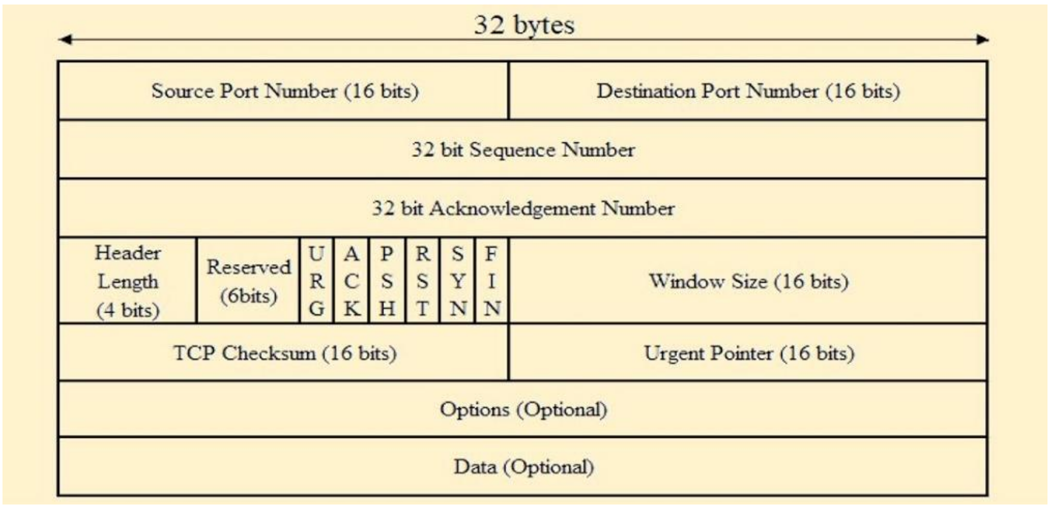
Testing & Benchmarking

TCP is about streams



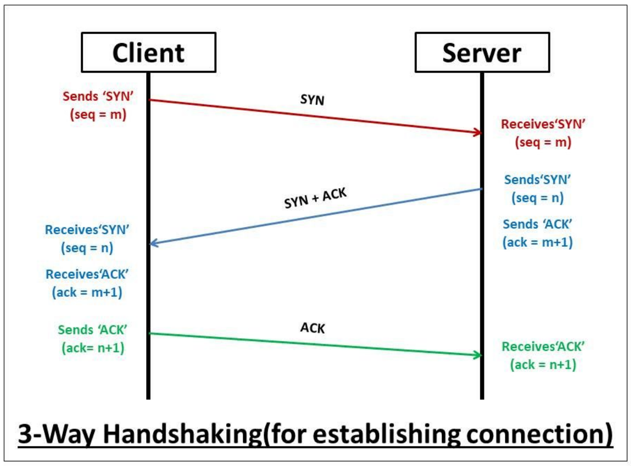
- TCP is a stream-oriented transport layer protocol
- This means that it is meant to send long streams of data, even if that data has to be split across multiple packets
- On this slide, we’ve included a figure of the network layers that were discussed in lecture
- What we want you to take away from this figure is the following:
    - This figure represents two hosts that are communicating over a network connection
    - When data is sent from one host to another, you can think of the data as going down the layers on the sender, then upon receipt, going back up the through the layers on the receiving side
    - Each host has code that implements each layer of the network stack
    - They each have code that handles the IP protocol, TCP or UDP, HTTP, etc.
    - As data makes its way down through the network layers, the code on the sender appends layer-specific information to the packet and continues to pass it down
    - This layer-specific information is called a header and there’s a different header for each layer
    - On the receiver side, the code at each layer will consume the layer specific information that is on the received packet and act on it

The TCP header



- Let's take a look at the TCP header
- We see the source and destination port numbers
- Remember that those are used by the hosts to multiplex their IP connections
- Below that, we have the 32-bit sequence number. Remember that TCP is for transmitting long byte-streams. The sequence number on each packet tells the receiver that the first byte of the data contained in the packet starts at offset "sequence number" of the whole data stream.
  - For example, if the first data packet's seqno is m, and assume that each subsequent data packet contains 10 bytes of (non-TCP-header) data,
  - then the 4th, 5th, 10th packet sent by the client would have seq numbers m+40, m+50, and m+100 respectively.
- The acknowledgment number is the **next sequence number** that the sender of this packet expects from the receiver (detailed in next example)
- Most of the rest of this we can gloss over, but let's point out a few things.
  - In the fourth row, there are several vertical rectangles; these are flags
  - For the next slide, we care about the SYN and ACK flags, and we'll elaborate on them there
- For now, let's move on to how TCP connections are initiated

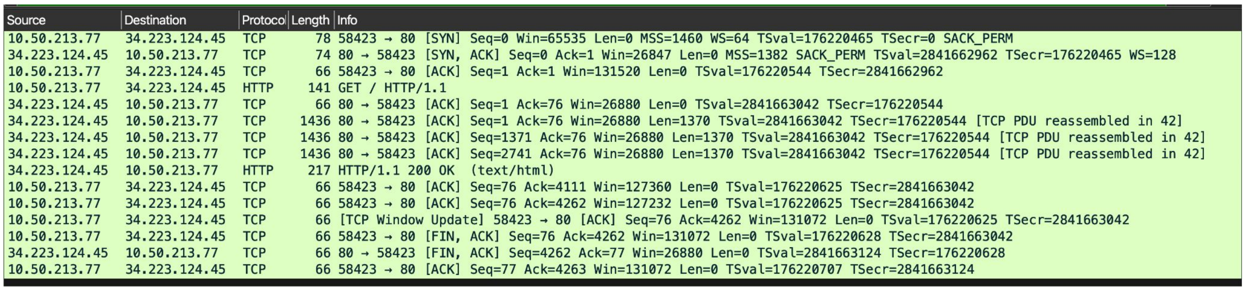
3-way handshake



- In order to initiate a new TCP connection, something called a 3-way handshake takes place
- Usually, a new TCP connection is initiated by a client that is attempting to connect to a server, so for the rest of this discussion, we'll assume that is the case
- For clarity, TCP can be used between any two hosts, regardless of the client and server language that we use. It just happens that this is both a useful teaching tool and the most common way that TCP connections are established
- Also, it's called a 3-way handshake but in reality it's just 3 messages. Not sure why "way" is used instead of "message", but let's just go with it
- The first part of the handshake is a message from the client to the server
- It is a SYN (synchronize) packet; this means that the SYN flag in the TCP header is set so that the receiver knows that this is an attempt at a new connection
- Notice in the diagram that it says "seq = m"
- That is the sequence number for the packet
- Remember that the sequence number is essentially a byte offset and it tells the receiver of the packet that the start of the data in this packet is byte m of the stream
- When the server receives the SYN packet, it will respond with a SYN-ACK packet
- This is a packet with both the SYN and ACK flags set
- Looking at the figure, you'll notice that on the server side, it says seq = n (the sequence number used by the server) and ack = m + 1 (the server

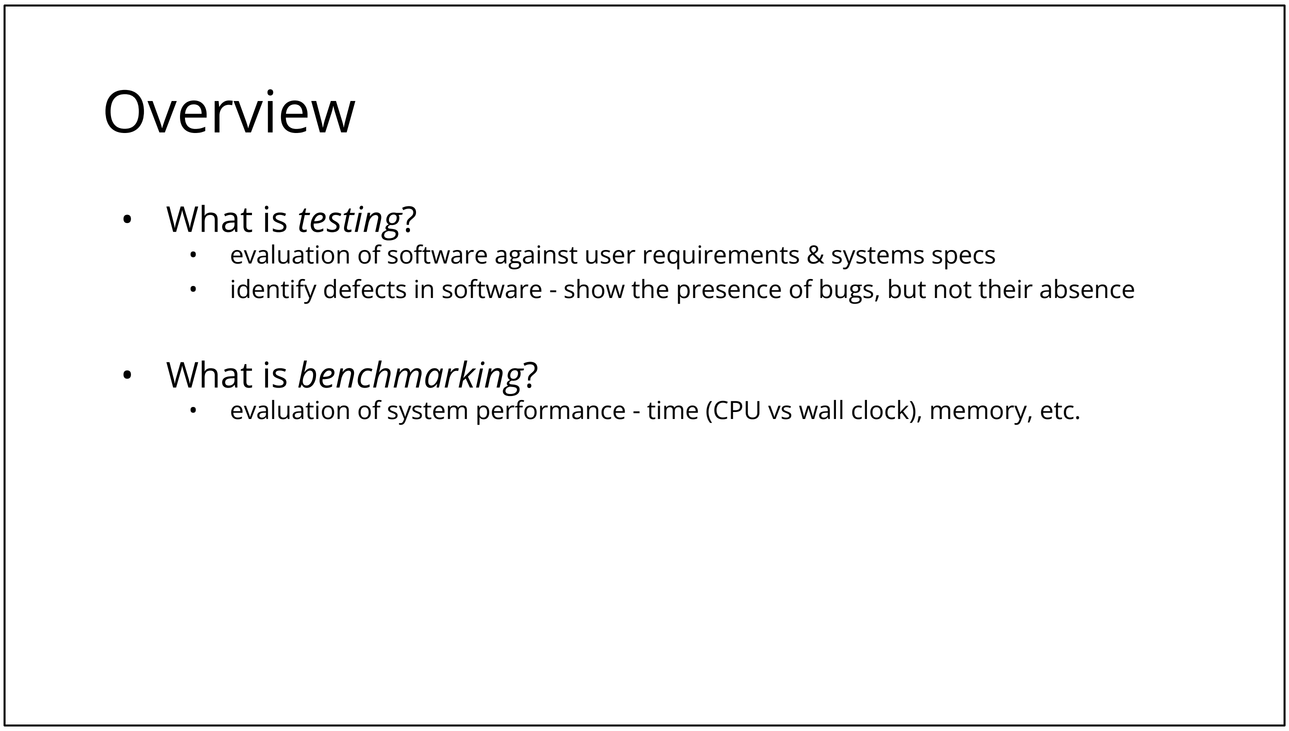
- acknowledges the client’s sequence number m, and expects the next one to be m+1)
- Each host in a TCP connection has its own sequence number that it keeps track of
- The ack number is always the sequence number that the receiver expects to receive next
- We’ll walk through an example later so that things are more clear
- Finally, the client responds with a final ACK packet and the connection is established
- These two hosts can continue to communicate indefinitely until one of them shuts the connection down!

Example



```
Source          Destination     Protocol Length Info
10.50.213.77    34.223.124.45   TCP       78 58423 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=176220465 TSecr=0 SACK_PERM
34.223.124.45   10.50.213.77    TCP       74 80 → 58423 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1382 SACK_PERM TSval=2841662962 TSecr=176220465 WS=128
10.50.213.77    34.223.124.45   TCP       66 58423 → 80 [ACK] Seq=1 Ack=1 Win=131520 Len=0 TSval=176220544 TSecr=2841662962
10.50.213.77    34.223.124.45   HTTP     141 GET / HTTP/1.1
34.223.124.45   10.50.213.77    TCP       66 80 → 58423 [ACK] Seq=1 Ack=76 Win=26880 Len=0 TSval=2841663042 TSecr=176220544
34.223.124.45   10.50.213.77    TCP     1436 80 → 58423 [ACK] Seq=1 Ack=76 Win=26880 Len=1370 TSval=2841663042 TSecr=176220544 [TCP PDU reassembled in 42]
34.223.124.45   10.50.213.77    TCP     1436 80 → 58423 [ACK] Seq=1371 Ack=76 Win=26880 Len=1370 TSval=2841663042 TSecr=176220544 [TCP PDU reassembled in 42]
34.223.124.45   10.50.213.77    TCP     1436 80 → 58423 [ACK] Seq=2741 Ack=76 Win=26880 Len=1370 TSval=2841663042 TSecr=176220544 [TCP PDU reassembled in 42]
34.223.124.45   10.50.213.77    HTTP     217 HTTP/1.1 200 OK  (text/html)
10.50.213.77    34.223.124.45   TCP       66 58423 → 80 [ACK] Seq=76 Ack=4111 Win=127360 Len=0 TSval=176220625 TSecr=2841663042
10.50.213.77    34.223.124.45   TCP       66 58423 → 80 [ACK] Seq=76 Ack=4262 Win=127232 Len=0 TSval=176220625 TSecr=2841663042
10.50.213.77    34.223.124.45   TCP       66 [TCP Window Update] 58423 → 80 [ACK] Seq=76 Ack=4262 Win=131072 Len=0 TSval=176220625 TSecr=2841663042
10.50.213.77    34.223.124.45   TCP       66 58423 → 80 [FIN, ACK] Seq=76 Ack=4262 Win=131072 Len=0 TSval=176220628 TSecr=2841663042
34.223.124.45   10.50.213.77    TCP       66 80 → 58423 [FIN, ACK] Seq=4262 Ack=77 Win=26880 Len=0 TSval=2841663124 TSecr=176220628
10.50.213.77    34.223.124.45   TCP       66 58423 → 80 [ACK] Seq=77 Ack=4263 Win=131072 Len=0 TSval=176220707 TSecr=2841663124
```

- Here's a screenshot of some network traffic results from wireshark; you can see an example of what the establishment of a TCP connection and data exchange looks like
- We use a terminal to request a webpage from neverssl.com
- Each of these rows represents a packet that my machine has either sent or received
- The first three packets are the 3-way handshake
- The next packet is my terminal sending an HTTP Get request
- The next few packets are the web page being sent back in chunks, followed by the HTTP Response with a status message
- And finally, the teardown of the TCP connection is started by the host



Overview

- What is *testing*?
  - evaluation of software against user requirements & systems specs
  - identify defects in software - show the presence of bugs, but not their absence
- What is *benchmarking*?
  - evaluation of system performance - time (CPU vs wall clock), memory, etc.

What is Testing?

- Evaluation of software to ensure it meets user requirements and system specifications.
- Identify defects in the software, ensuring that bugs are detected. It can show the presence of bugs, but it's important to note that it cannot guarantee the absence of bugs.

What is Benchmarking?

- Benchmarking is the process of evaluating system performance by measuring various parameters.
- Common metrics include processing time such as CPU time, wall-clock time (real world time) and resource usage like memory.
- Difference of CPU time and wall-clock time?
  a. CPU time only measures the time CPU is used to process
  b. Wall-clock time is the whole time user should wait. It can include some other time such as I/O
  c. Question? Is it possible that CPU time is longer than wall clock
     i. YES! Multi-threaded Systems!

Testing - Basic Approach in Go

- Source files and associated test files are placed in the same package/folder
- The name of the test file for any given source file is **`_test.go`**
  - E.g., **`router.go`** and **`router_test.go`**
- Import "**`testing`**"
- Test functions need to have the "**`Test`**" prefix, and the next character in the function name should be capitalized

Testing - Exercises

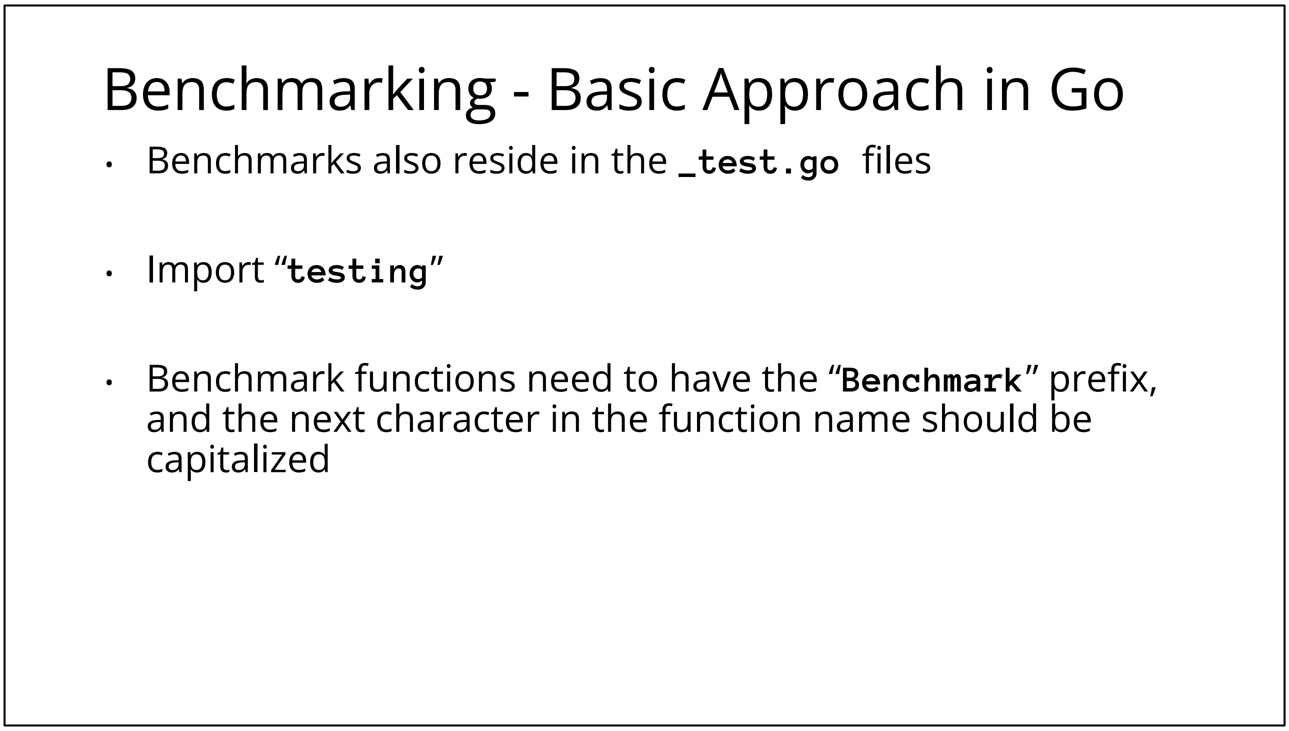
```
› cd precept4/mysort

# run test framework
› go test -v

# fix the bug and demonstrate tests pass
```

The bug is in the mysort_test.go. The MergeSort function has a return value but it’s verifying the old list instead of the returned list.

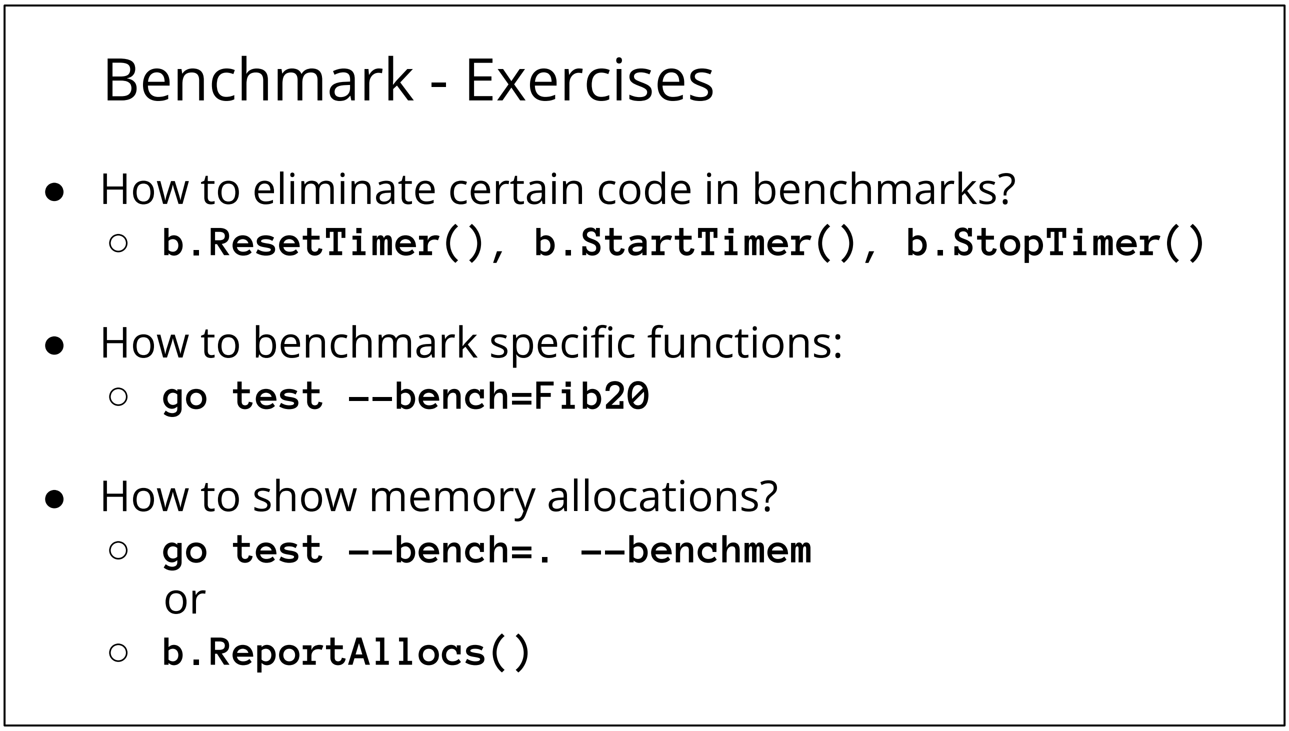
```
slice = MergeSort(slice)
```



Benchmarking - Basic Approach in Go

- Benchmarks also reside in the **`_test.go`** files
- Import “**`testing`**”
- Benchmark functions need to have the “**`Benchmark`**” prefix, and the next character in the function name should be capitalized

- Benchmarks are written within _test.go files, the same files where you define test functions.
- Also need to import “testing”
- Recall that the test functions have “Test” prefix. Benchmark functions have “Benchmark” prefix.



Benchmark - Exercises

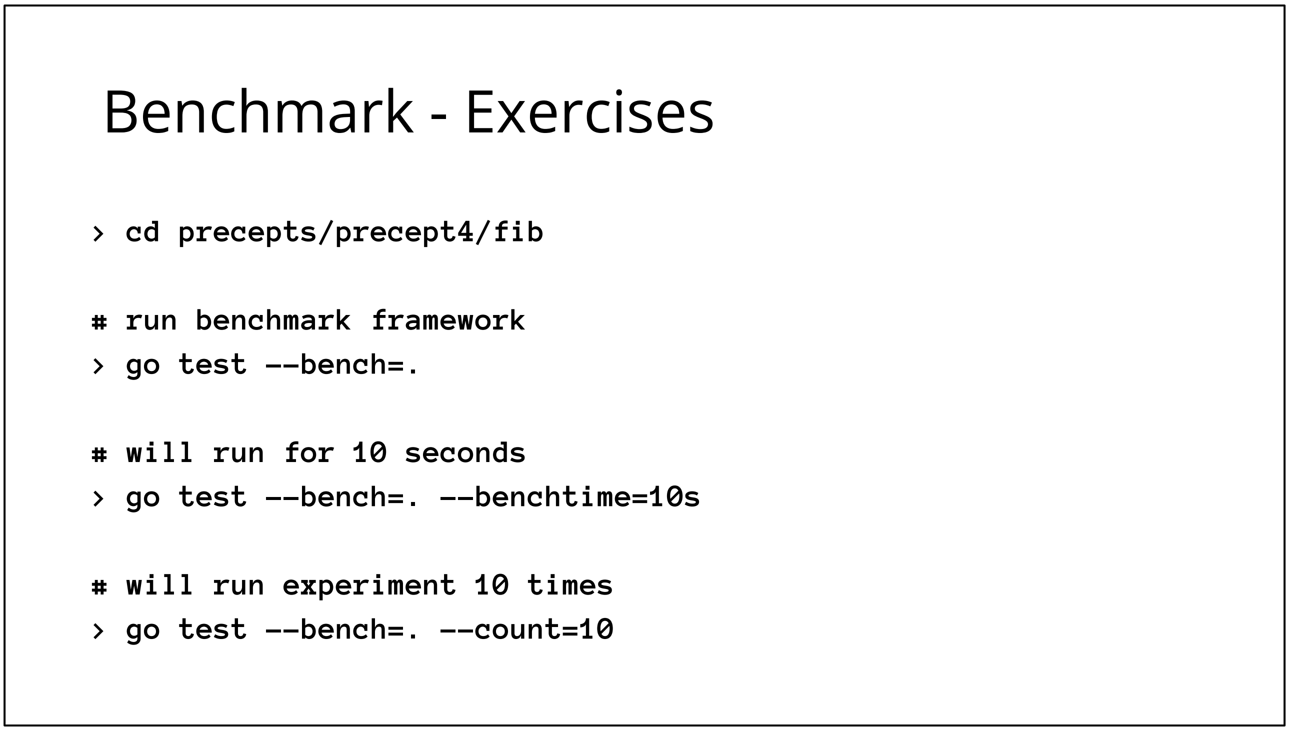
- How to eliminate certain code in benchmarks?
  - `b.ResetTimer(), b.StartTimer(), b.StopTimer()`
- How to benchmark specific functions:
  - `go test --bench=Fib20`
- How to show memory allocations?
  - `go test --bench=. --benchmem`
    or
  - `b.ReportAllocs()`

To exclude certain parts of code (like setup tasks that shouldn't be measured in the benchmark), use:

- **b.ResetTimer():** ResetTimer zeroes the elapsed benchmark time and memory allocation counters and deletes user-reported metrics. It does not affect whether the timer is running.
- **b.StartTimer():** StartTimer starts timing a test. This function is called automatically before a benchmark starts, but it can also be used to resume timing after a call to B.StopTimer
- **b.StopTimer():** StopTimer stops timing a test. This can be used to pause the timer while performing complex initialization that you don't want to measure.

benchmark specific functions by running by specify a function as an argument.

You can also show memory allocations by specify “benchmem”
go test --bench=. --benchmem This command means to run all benchmarks and includes memory allocation statistics.



Benchmark - Exercises

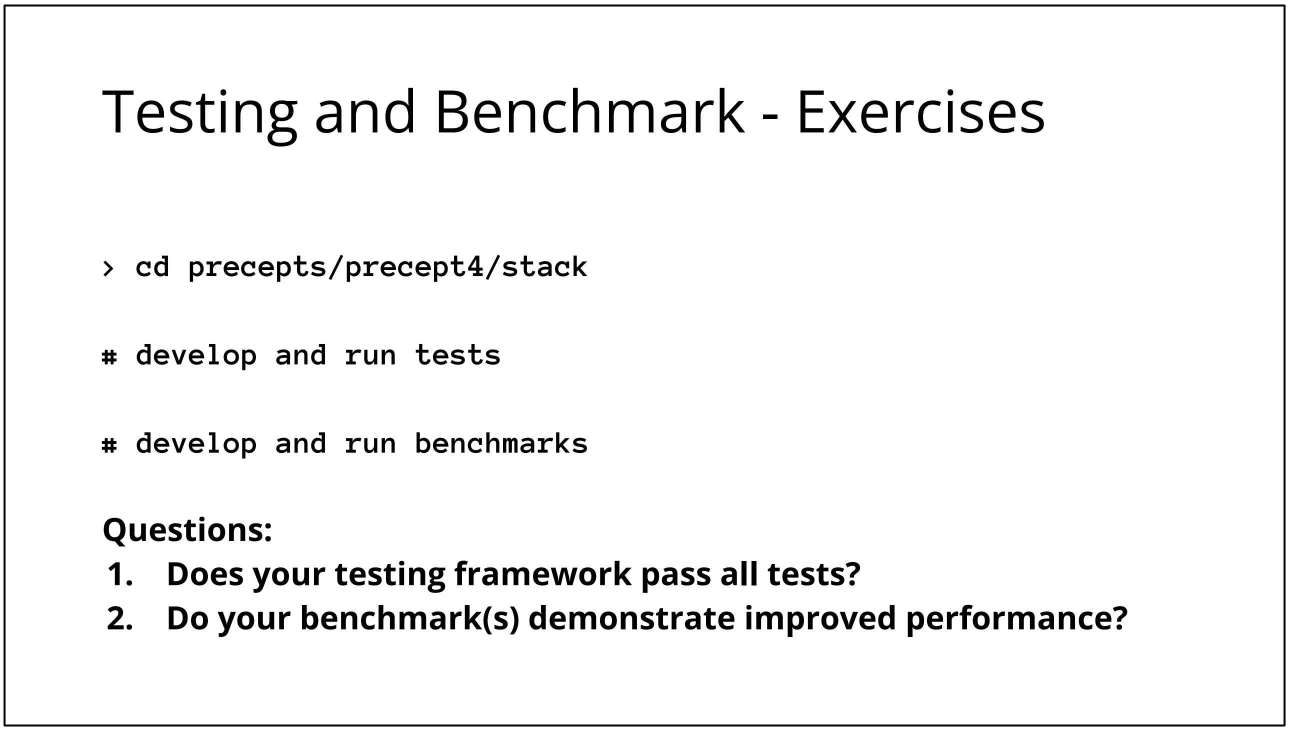
```
> cd precepts/precept4/fib

# run benchmark framework
> go test --bench=.

# will run for 10 seconds
> go test --bench=. --benchtime=10s

# will run experiment 10 times
> go test --bench=. --count=10
```

- –benchtime=10s means that for each benchmark, go will run the benchmark for a **minimum** of 10 seconds.
- If running the benchmark takes less than 10 seconds, golang will run with multiple iterations to reach 10s. If longer than 10s, golang will not terminate it but wait it to finish with only one iteration.



Testing and Benchmark - Exercises

```
> cd precepts/precept4/stack

# develop and run tests

# develop and run benchmarks
```

Questions:

1. **Does your testing framework pass all tests?**
2. **Do your benchmark(s) demonstrate improved performance?**

The bug is in the function Pop(); the `s.n = s.n - 1` line is commented out but is needed.