Go's benchmark uses a struct called "highPrecisionTime" to measure time duration of a benchmark.

for non-windows

```go
//go:build !windows

package testing

import "time"

// isWindowsRetryable reports whether err is a Windows error code
// that may be fixed by retrying a failed filesystem operation.
func isWindowsRetryable(err error) bool { return false }

// highPrecisionTime represents a single point in time.
// On all systems except Windows, using time.Time is fine.
Implement interface
type highPrecisionTime struct {
    now time.Time
}

// highPrecisionTimeNow returns high precision time for benchmarking.
func highPrecisionTimeNow() highPrecisionTime {
    return highPrecisionTime{now: time.Now()}
}

// highPrecisionTimeSince returns duration since b.
func highPrecisionTimeSince(b highPrecisionTime) time.Duration {
    return time.Since(b.now)
}
```

On most operating systems

'time.Now()' relies on Os's

wall clock. The wall clocks

are designed for general

time-keeping, hence their granularity

is very coarse.

Historically, Windows system clock

has had a granularity of 10-16 ms,

this means the clock updates

its value 100 to 62 times per

second.

So, calling time.NOW frequently will return the same value multiple times, making it impossible to accurately measure sub-millisecond latency.

Eg: If a benchmark function takes 50 microseconds, & the system wall clock updates its value every 10-16 ms, the measurement will likely show 0 ms or 10-16 ms which

is inaccurate. Hence, Go creates highPrecisionTime which on WINDOWS relies on "QueryPerformanceCounter"

provides high resolution time.

```
// TODO: If Windows runtime implements high resolution timing then highPrecisionTime
// can be removed.
Implement interface
type highPrecisionTime struct {
    now int64
}

// highPrecisionTimeNow returns high precision time for benchmarking.
func highPrecisionTimeNow() highPrecisionTime {
    var t highPrecisionTime
    // This should always succeed for Windows XP and above.
    t.now = windows.QueryPerformanceCounter()
    return t
}
```