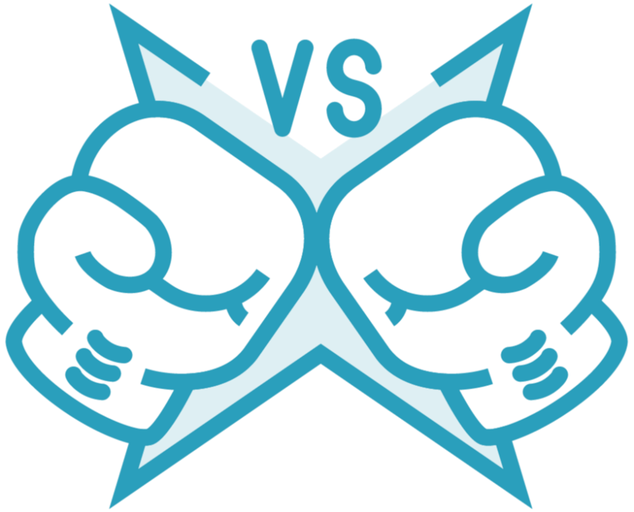
Measuring and Optimizing Performance

Zoran Horvat
CEO at Coding Helmet

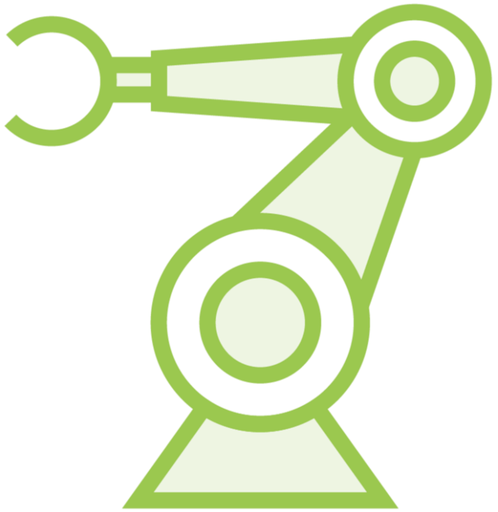
@zoranh75 https://codinghelmet.com

Managing Performance

Performance issues require measurement



Measuring requires an alternative solution to compare to



Use measurement to navigate development

Investigating Performance Improvements

Measuring is only the first step in a longer journey

Some experiments will point in direction of an improvement

Even an unsuccessful experiment can help change perspective

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate).First()
```

Returns the smallest element

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate).First()
```

~~$O(n \log n)$~~ sorting time

$O(1)$ selection time

~~$O(n \log n)$~~ overall time!

Unless...

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate).First()
```

$O(n \log n)$ time

$O(n)$ space

$O(1)$ time

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate).First()
```

$O(n)$ time

$O(1)$ space

Hi there, what's up?

What do you mean?

You mean **min()**?

Hey, sorted!

Nothing, just... don't sort.

Give me the first output item.

Yeah, I mean... min.

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Minimum item that satisfies the predicate

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Full sort

* Not optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Full sort

* Not optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Apply the predicate until it returns `true`

* Not optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Apply the predicate until it returns `true`

Expression result

* Not optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

* Optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Apply the predicate to all elements

Selected minimum

* Optimized

Thinking Outside the Box

```
workers.OrderBy(worker => worker.Rate)
       .First(worker => worker.Name.Contains("Joe"))
```

Not optimized

— Applies predicate until `true`

Optimized

——— Applies predicate N times

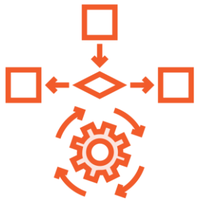
The predicate could be ***expensive***, or have ***side effects***

Important Considerations

Avoid changes with adverse effects, including non-functional ones

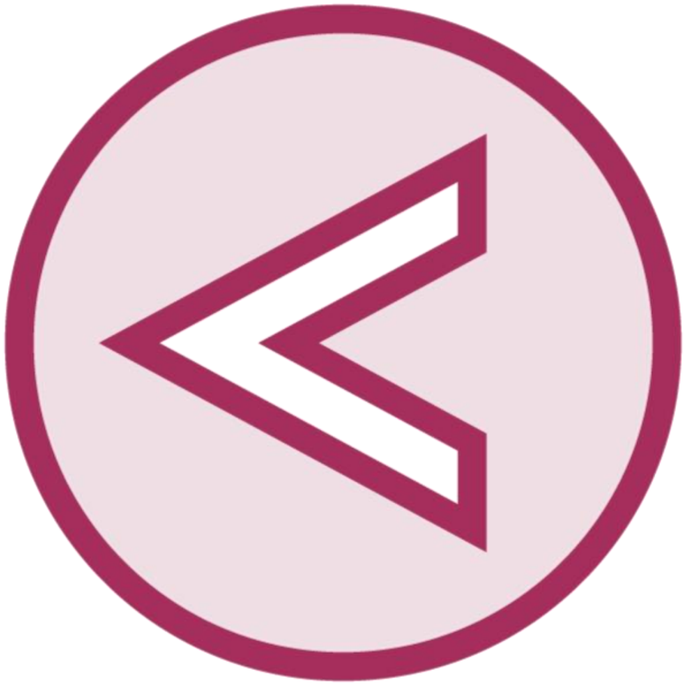
Think if the optimization only applies under constrained conditions

Expose implementations that are best under different conditions



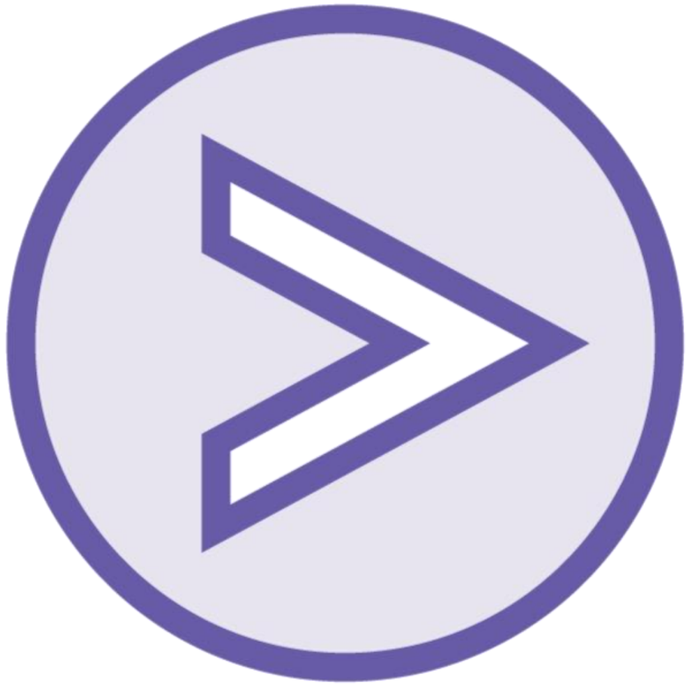
Let the caller decide whether to call the alternative implementation

Working with Collections in .NET



Algorithms Theory

Learn as much as you can



.NET BCL Collections

Learn as much as you can, too

The Role of BCL Collections

Wrong: Using collections in domain modeling

Right: Build a domain model around collections

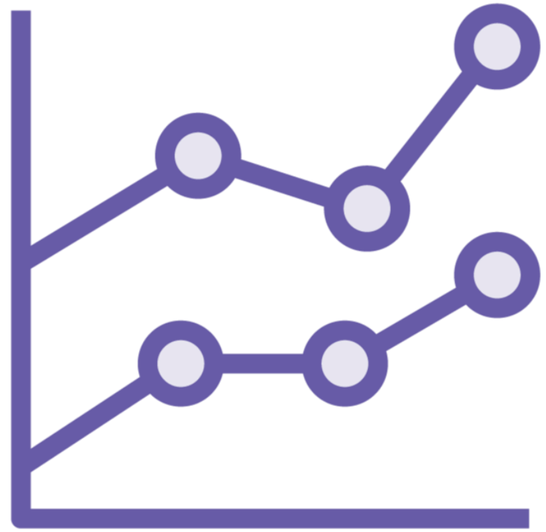
A BCL collection is used as a data store

Example: Optimizing performance of the list paginator

Aftermath

Successful in reducing performance penalty

No imminent need to develop a more complex algorithm



Performance improvement in line with benchmark's prediction

Benchmarking is a reliable tool when running experiments

Summary

Measured using BenchmarkDotNet

- Collections are usually costly objects
- Often cause bottlenecks

Benchmarking a piece of logic

- Construct methods for critical operations
- Construct measurement tables

Summary

Using benchmarks in design

- Validate isolated design ideas
- Measure impact of local optimizations

Using benchmarks in planning

- Insert measured code into the design
- Improvements will be on a par with what benchmarks predict

Up Next: Generics Fundamentals