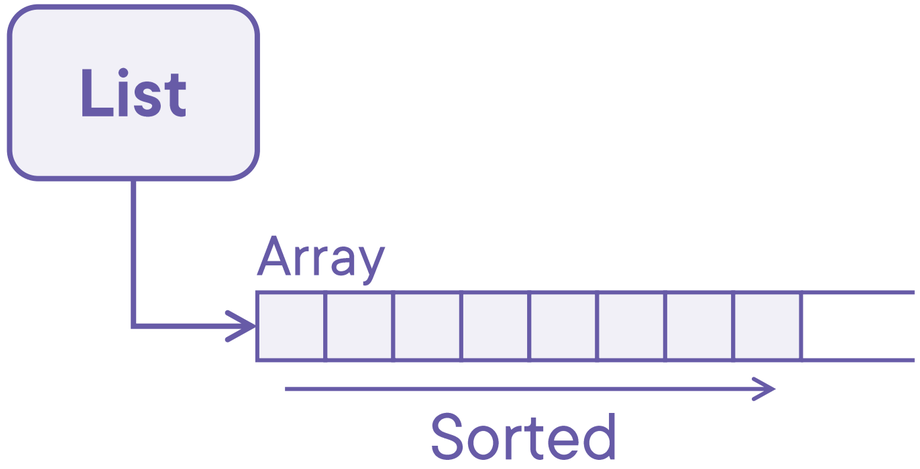
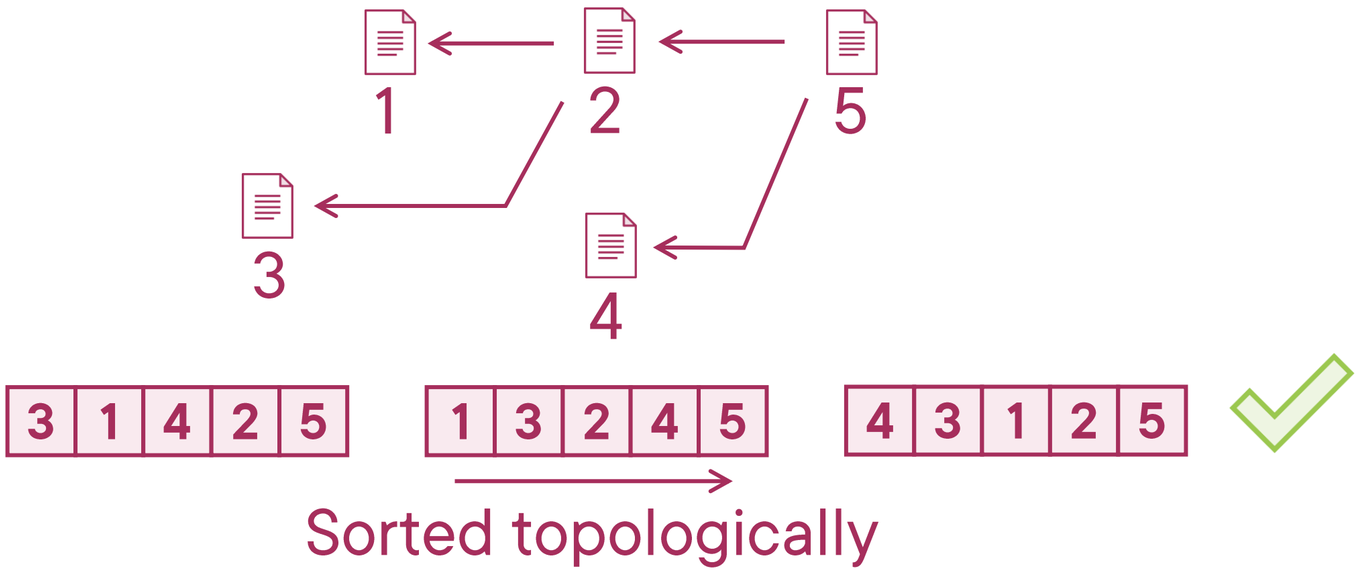
Engineering Custom Linear Collections

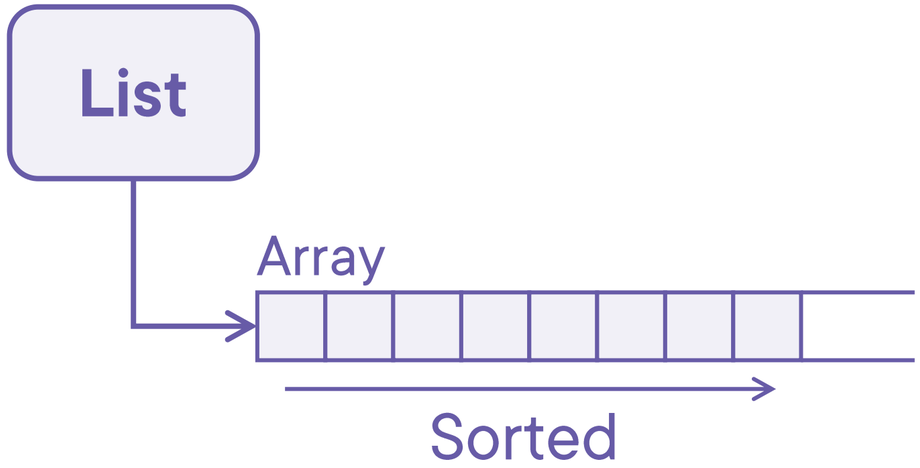
Zoran Horvat

CEO at Coding Helmet

@zoranh75 https://codinghelmet.com

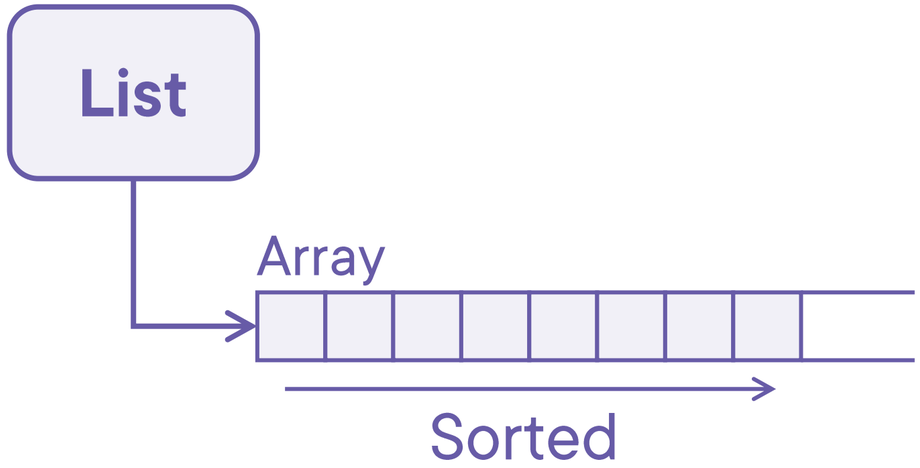

List
Array
Sorted


1
2
5
3
4
3 1 4 2 5
1 3 2 4 5
4 3 1 2 5
Sorted topologically



Use the `IOrderedList<T>` to calculate percentiles of a sequence

Use the `IOrderedList<T>` to remove outliers from a sequence



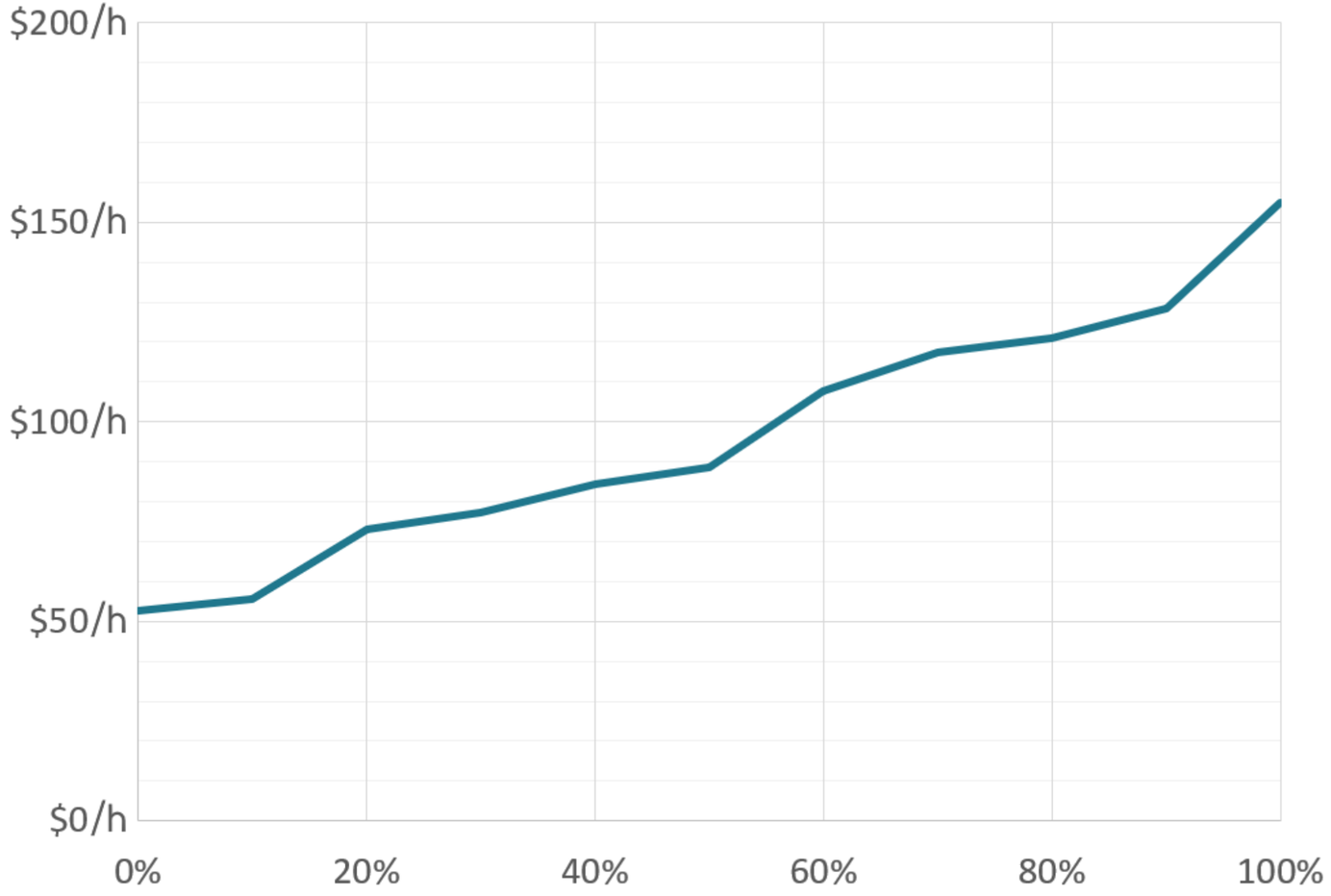
Customer's requirements:

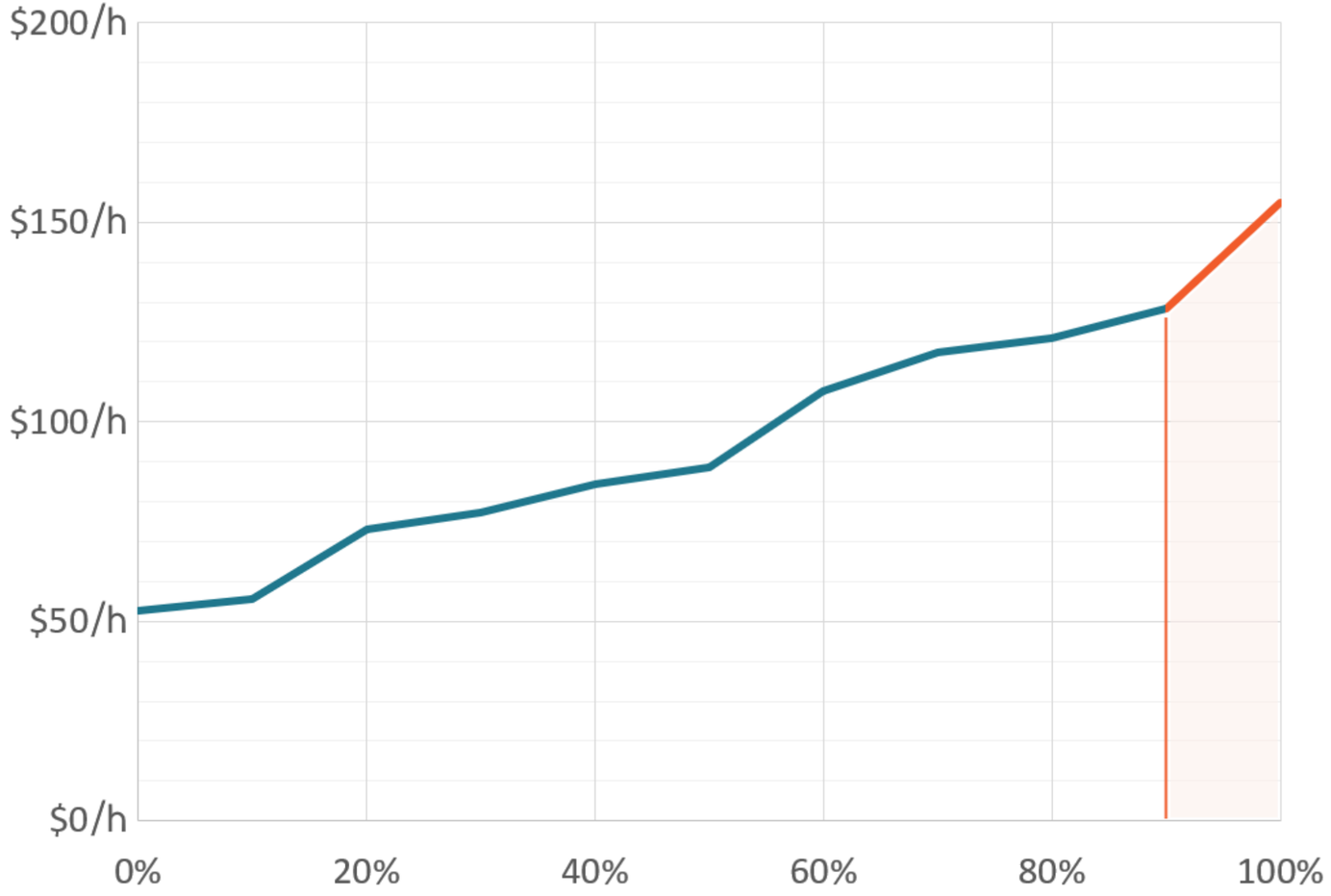
Given: Sequence of objects
Sorting criterion
Percentage value

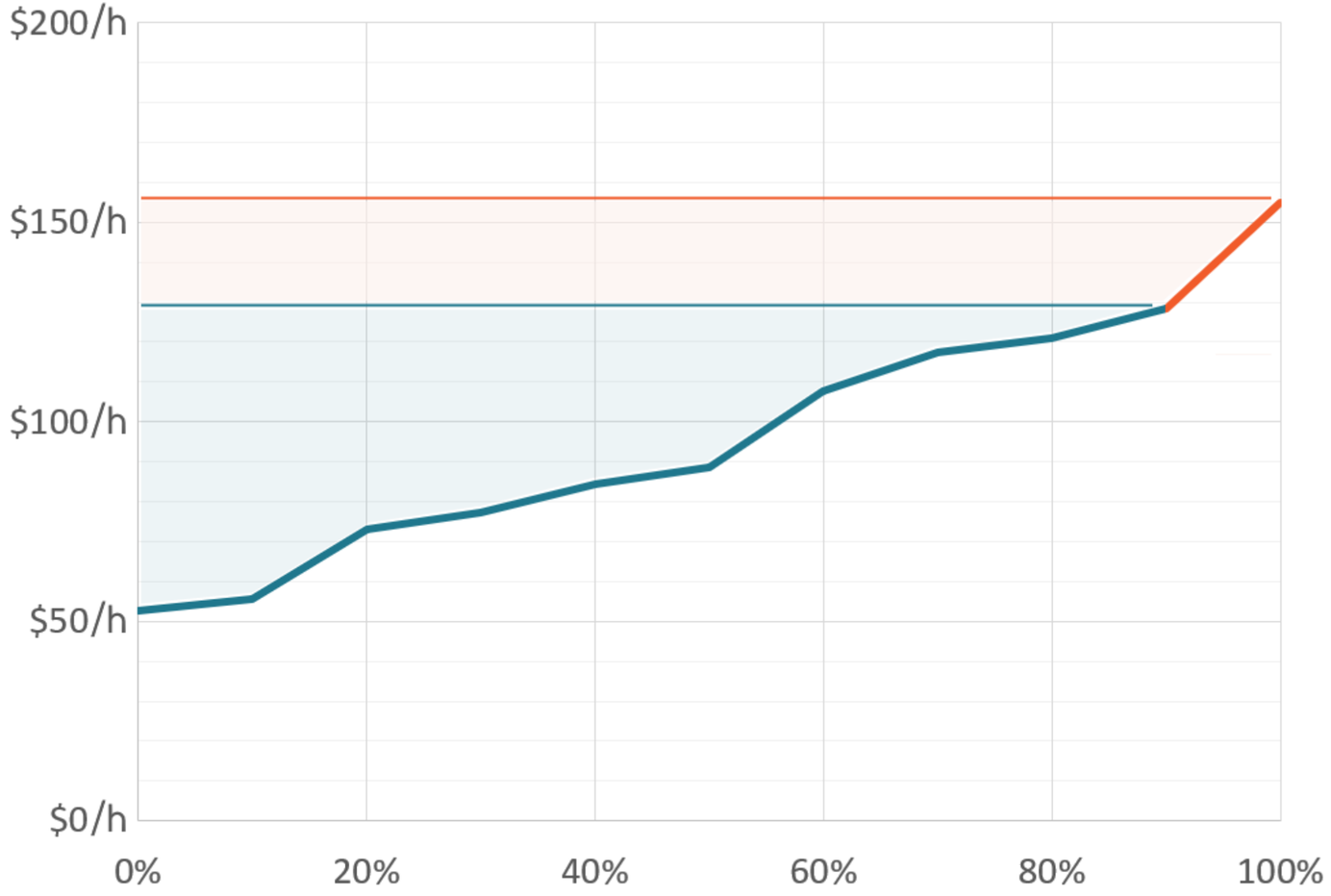
Calculate: Return the requested percentile object
(that many of other objects are less or equal to it)

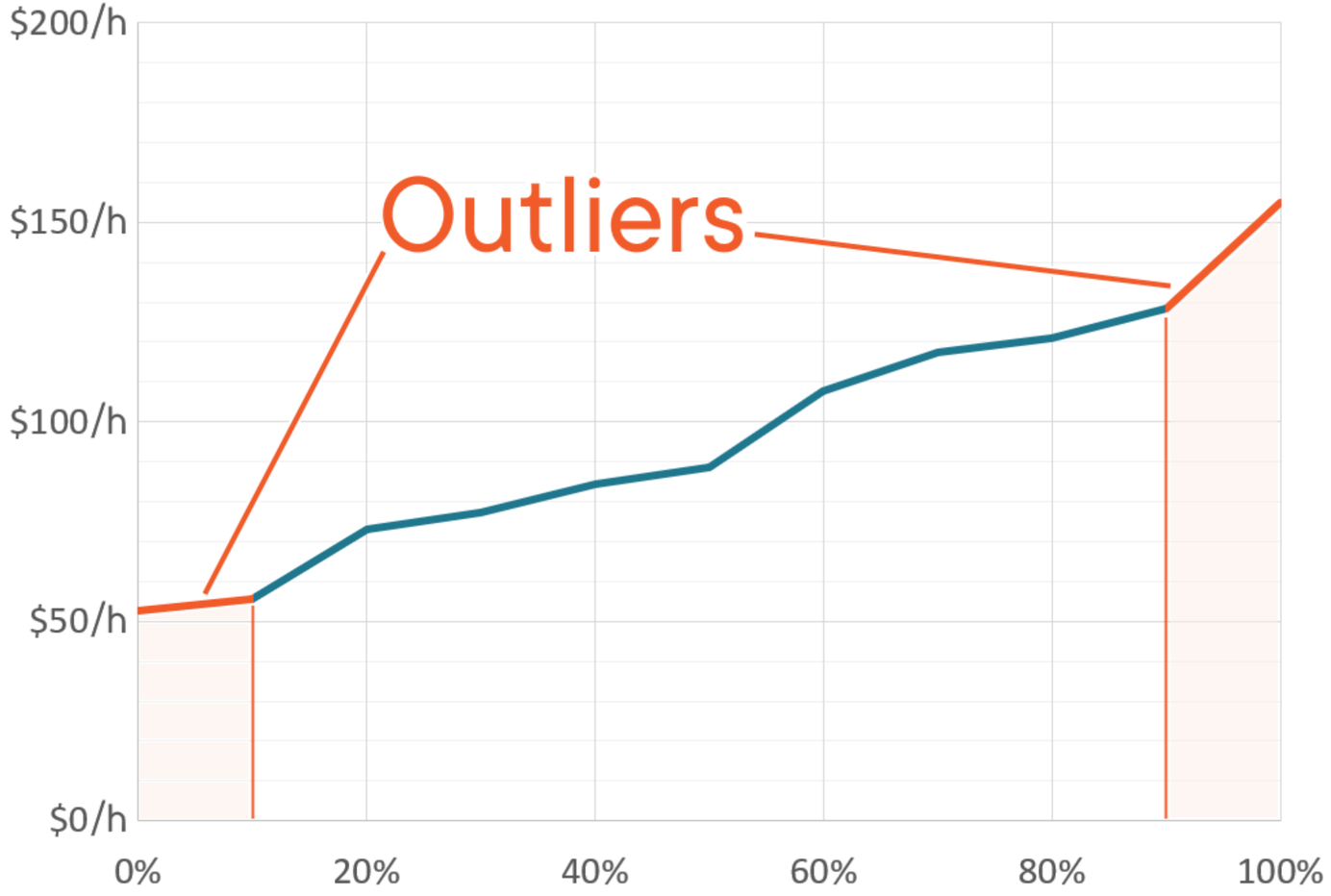
“Favor object composition
over class inheritance.”

Gang of Four, 1994


$200/h
$150/h
$100/h
$50/h
$0/h
0%
20%
40%
60%
80%
100%


$200/h
$150/h
$100/h
$50/h
$0/h
0%
20%
40%
60%
80%
100%


$200/h
$150/h
$100/h
$50/h
$0/h
0%
20%
40%
60%
80%
100%


$200/h
$150/h
$100/h
$50/h
$0/h
Outliers
0%
20%
40%
60%
80%
100%

Windows PowerShell

```
  393 KB |
|              |                |               |                |               |        |         |           |           |          |
         |
|  PlainCreate |         100000 |   6,483.43 us |      74.638 us |     69.816 us |   1.00 |    0.00 |  492.1875 |  242.1875 | 242.1875 |
3,908 KB |
| CachedCreate |         100000 |   8,563.27 us |     101.076 us |     89.601 us |   1.32 |    0.02 |  484.3750 |  234.3750 | 234.3750 |
3,909 KB |
|              |                |               |                |               |        |         |           |           |          |
         |
|  PlainCreate |        1000000 | 101,362.43 us | 1,482.499 us | 1,237.954 us |   1.00 |    0.00 | 4400.0000 | 2400.0000 | 600.0000 | 3
9,066 KB |
| CachedCreate |        1000000 |  80,267.79 us |   686.799 us |   608.829 us |   0.79 |    0.01 | 4000.0000 | 1285.7143 | 285.7143 | 3
9,066 KB |

// * Hints *
Outliers
  CurrencyCreationBenchmark.CachedCreate: Default -> 1 outlier  was  removed (657.53 us)
  CurrencyCreationBenchmark.CachedCreate: Default -> 1 outlier  was  removed (8.95 ms)
  CurrencyCreationBenchmark.PlainCreate: Default  -> 2 outliers were removed (106.21 ms, 106.66 ms)
  CurrencyCreationBenchmark.CachedCreate: Default -> 1 outlier  was  removed (84.33 ms)

// * Legends *
  CollectionSize : Value of the 'CollectionSize' parameter
  Mean           : Arithmetic mean of all measurements
  Error          : Half of 99.9% confidence interval
  StdDev         : Standard deviation of all measurements
  Ratio          : Mean of the ratio distribution ([Current]/[Baseline])
  RatioSD        : Standard deviation of the ratio distribution ([Current]/[Baseline])
  Gen 0          : GC Generation 0 collects per 1000 operations
  Gen 1          : GC Generation 1 collects per 1000 operations
  Gen 2          : GC Generation 2 collects per 1000 operations
  Allocated      : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 us           : 1 Microsecond (0.000001 sec)

// * Diagnostic Output - MemoryDiagnoser *
```

Outliers removal

Customer's requirements:

Given: Sequence of objects
Sorting criterion
Percentage value

Request: Remove the specified percentage of items from the ends of the sort order

.vscode
ConsoleDemo
ConsoleDemo.csproj
CurrencyCreationBen...
Operators.cs
Program.cs
Models
Collections
FullySortedList.cs
IOrderedList.cs
OrderedListProjectio...
Common
Analytics
OrderedOutliersRe...
OrderedPercentiles...
Percentile.cs
Caching
Formatting
Pagination
Shuffling
ArgumentExtension...
IOutliersRemover.cs
IPage.cs
IPaginated.cs
IPercentilesReader.cs
Operators.cs
SinglePassSequence...
TransparentCaching....
Currency.cs
CurrencyCodeEqualit...
Employee.cs

```
19
20      private IOrderedList<T> List { get; }
21
22      public IEnumerator<T> GetEnumerator()
23      {
24          for (int i = this.Offset; i < this.ExclusiveEndOffset; i++)
25              yield return this.List[i];
26      }
27
28      public IOrderedList<T> GetRange(int index, int count) =>
29
30
31
32
33      IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();
34  }
```

Summary

Demonstrated use of linear collections

- Applied a list far away from the domain
- Implemented percentiles calculator
- Implemented outliers remover

Building a hierarchy of abstractions

- Improves composability of solutions
- Enables performance optimizations

Summary

Practical guidelines

- General-purpose operations belong to infrastructure
- Infrastructure should be optimal
- Domain-related operations rely on composability
- Domain should not depend on collections

Up Next: Engineering Queuing Solutions