

My code is slow.
What can I do about that?

Primož Gabrijelčič

@thedelphigeek

**MY CODE IS SLOW.
WHAT CAN I DO ABOUT THAT?**

Primož Gabrijelčič

<http://primoz.gabrijelcic.org>

- programmer, MVP, writer, blogger, consultant, speaker
- Blog *<http://thedelphigeek.com>*
- Twitter *[@thedelphigeek](#)*
- Skype *[gabr42](#)*
- LinkedIn *[gabr42](#)*
- GitHub *[gabr42](#)*
- SO *[gabr](#)*

Monday, September 30, 2019

CompareValue for booleans

CompareValue function is incredibly practical when you are writing comparers (functions that determine how some data structure is ordered). *System.Math* and *System.StrUtils* define a bunch of functions that can be used to compare integers, doubles, strings ... There's, however, no *CompareValue* for booleans.

A *CompareValue* function compares two parameters, traditionally named *left* and *right*, and returns 0 if they are the same, -1 if *left* is smaller and 1 if *right* is smaller.

If we use the usual ordering of *false* < *true*, we can write the missing function as follows:

```
function CompareValue(left, right: boolean): integer; overload;
begin
  if left < right then
    Result := -1
  else if left > right then
    Result := 1
  else
    Result := 0;
end;
```

Your task for today – if you choose to accept it – is: Write this function without any *if* statements.

[Read more »](#)

Posted by [gabr42](#) at 20:22 9 comments: [Links to this post](#) 

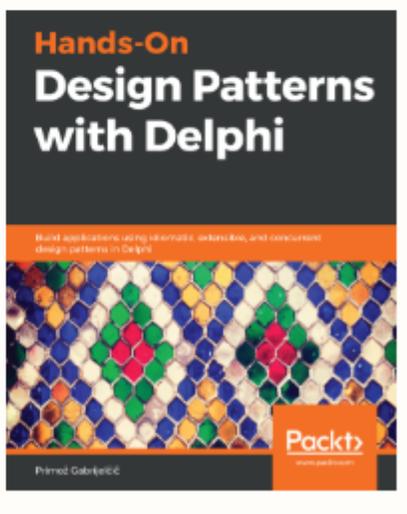
Labels: [Delphi](#), [programming](#), [tips](#)



Embarcadero
MVP

Pages

[Presentations](#)

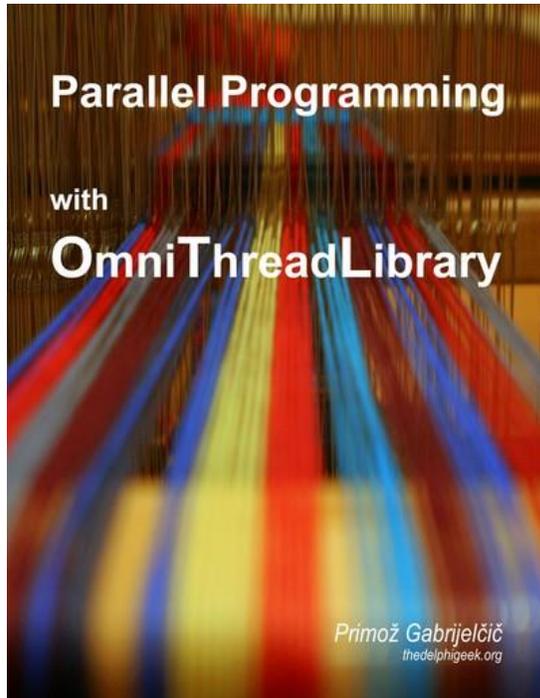


Hands-On
Design Patterns
with Delphi

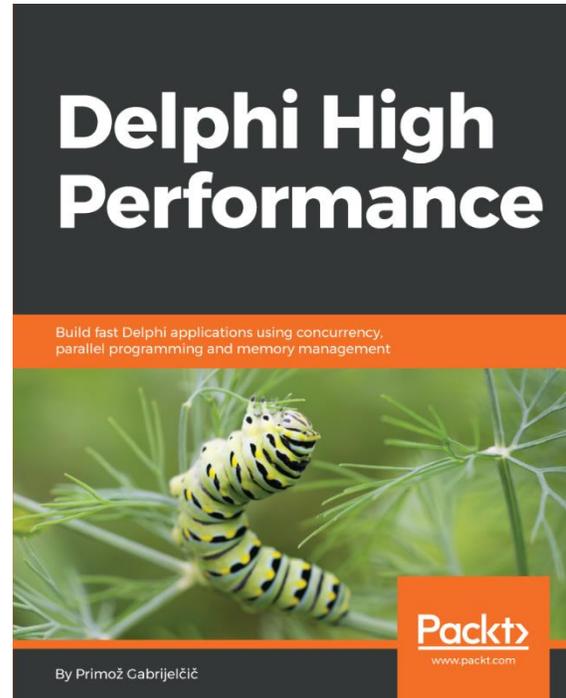
Build applications using observables, actors, and concurrent design patterns in Delphi

Packt

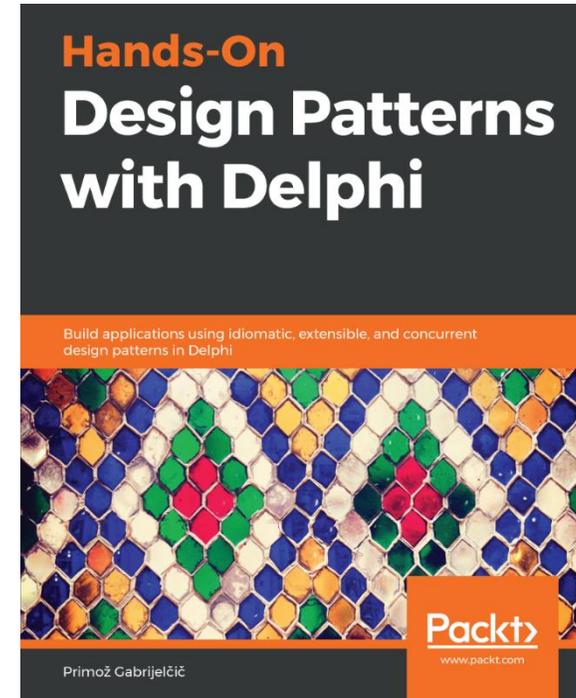
Primo Gabriel



[http://tiny.cc/
pg-ppotl](http://tiny.cc/pg-ppotl)



[http://tiny.cc/
pg-dhp](http://tiny.cc/pg-dhp)



[http://tiny.cc/
pg-dpd](http://tiny.cc/pg-dpd)

PERFORMANCE



Performance

- What is performance?
- How do we “add it to the program”?
 - There is no silver bullet!



What defines performance?

- Running “fast enough”
- Raw speed
- Responsiveness
 - Non-blocking



Improving performance

- Analyzing algorithms
- Measuring execution time
- Fixing algorithms
- Fine tuning the code
- Writing parallel code

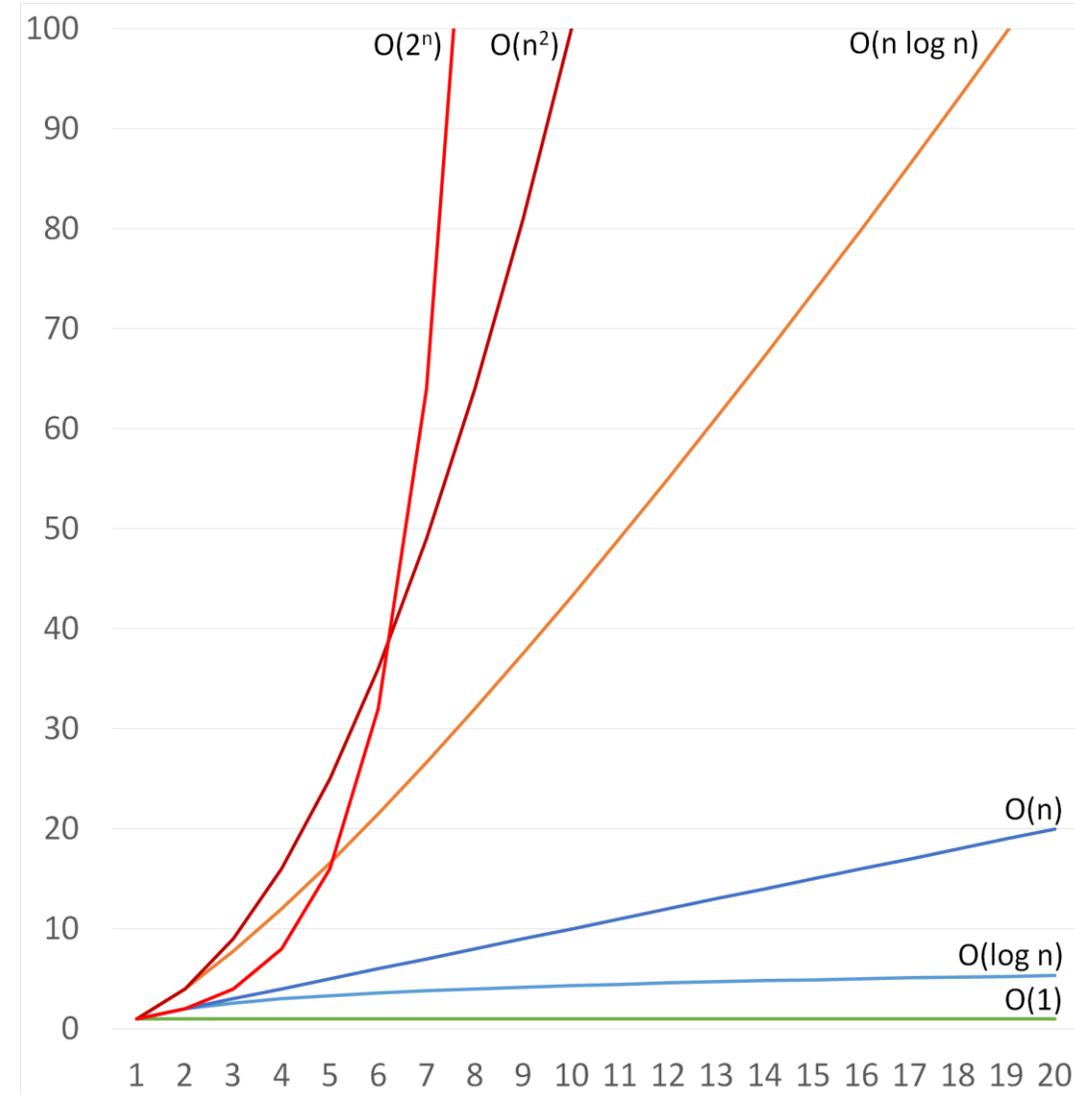
ALGORITHM COMPLEXITY

Algorithm complexity

- $O()$
 - Tells us how algorithm slows down if data size is increased by a **factor of n**
 - $O(n)$, $O(n^2)$, $O(n \log n)$...
- Time **and** space complexity

Frequently encountered $O()$

- $O(1)$ accessing array elements
- $O(\log n)$ searching in ordered list
- $O(n)$ linear search
- $O(n \log n)$ quick sort (average)
introsort
- $O(n^2)$ quick sort (worst),
naive sort (bubblesort,
insertion, selection)
- $O(c^n)$ recursive Fibonacci,
travelling salesman



Comparing complexities

Data size	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(c^n)$
1	1	1	1	1	1	1

Comparing complexities

Data size	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(c^n)$
1	1	1	1	1	1	1
10	1	4	10	43	100	512

Comparing complexities

Data size	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(c^n)$
1	1	1	1	1	1	1
10	1	4	10	43	100	512
100	1	8	100	764	10.000	10^{29}

Comparing complexities

Data size	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(c^n)$
1	1	1	1	1	1	1
10	1	4	10	43	100	512
100	1	8	100	764	10.000	10^{29}
300	1	9	300	2.769	90.000	10^{90}

DATA STRUCTURES

[HTTP://BIGOCHEATSHEET.COM/](http://bigocheatsheet.com/)

Array

Operation	Complexity
Access	$O(1)$
Search	$O(n)$
Insertion	$O(n)$
Deletion	$O(n)$

Sorted array

Operation	Complexity
Access	$O(1)$
Search	$O(\log n)$
Insertion	$O(n)$
Deletion	$O(n)$

- Bisection search
- `TArray.BinarySearch<T>`



TList, Tlist<T>, TObjectList, TStringList ...

- Array / Sorted array
 - Faster when inserting/deleting
- TStringList.IndexOf, Find
- TList<T>.BinarySearch

TQueue<T> / TStack<T>

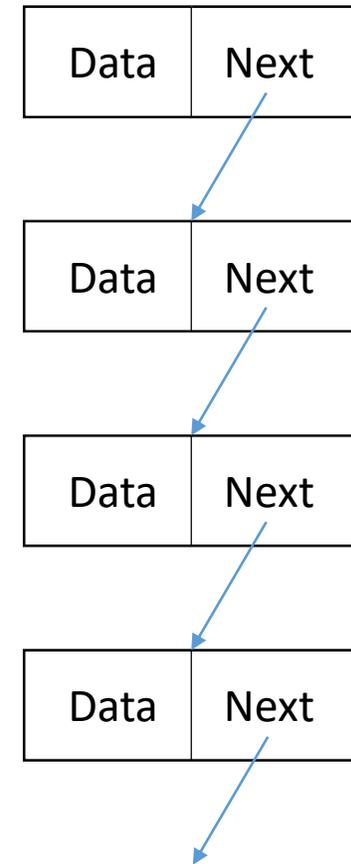
Operation	Complexity
Access	O(1)
Search	O(n)
Insertion	O(1) / O(n)
Deletion	O(1)

- Array-based
- Access via .List
- Clumsy Search
- Insertion may cause the storage to grow

Linked list

Operation	Complexity
Access	$O(n)$
Search	$O(n)$
Insertion	$O(1)$
Deletion	$O(1)$

- Not CPU cache-friendly
- Adding order to other structures
- TGpDoublyLinkedList / GpLists.pas
 - <https://github.com/gabr42/GpDelphiUnits>



TDictionary<K,V>

Operation	Complexity
Access	N/A
Search	O(1)
Insertion	O(1)
Deletion	O(1)

- Relatively slow O(1)
- Increased memory consumption
 - “index” (hash) + data
- Unordered

TRedBlackTree<K,V> [Spring4D]

Operation	Complexity
Access	$O(\log n)$
Search	$O(\log n)$
Insertion	$O(\log n)$
Deletion	$O(\log n)$

- Worst complexity = average complexity
- Ordered



Spring4D collections

- List, SortedList
 - Array
- Stack, Queue
 - Array
- Dictionary
 - TDictionary
- SortedDictionary
 - TRedBlackTree
- Set
 - Dictionary
- OrderedSet
 - Dictionary + List
- MultiMap
 - Dictionary<K, List<V>>
- BidiDictionary
 - Dictionary<K,V> + Dictionary<V,K>

TIPS & TRICKS



Constant factor can be important

- Accessing array element vs. accessing dictionary element
- → Fine tuning (optimizing)
- **Measure first!**



If something is slow, do it less often

- Don't update UI million times a second
- Caching!

TGpCache<K,V>

Operation	Complexity
Access	N/A / O(1)
Search	O(1)
Insertion	O(1)
Deletion	O(1)

- Ordered (by time)
- Accessing oldest/newest: O(1)
- Dictionary + Linked list
- TGpDoublyLinkedList / GpLists.pas
 - <https://github.com/gabr42/GpDelphiUnits>



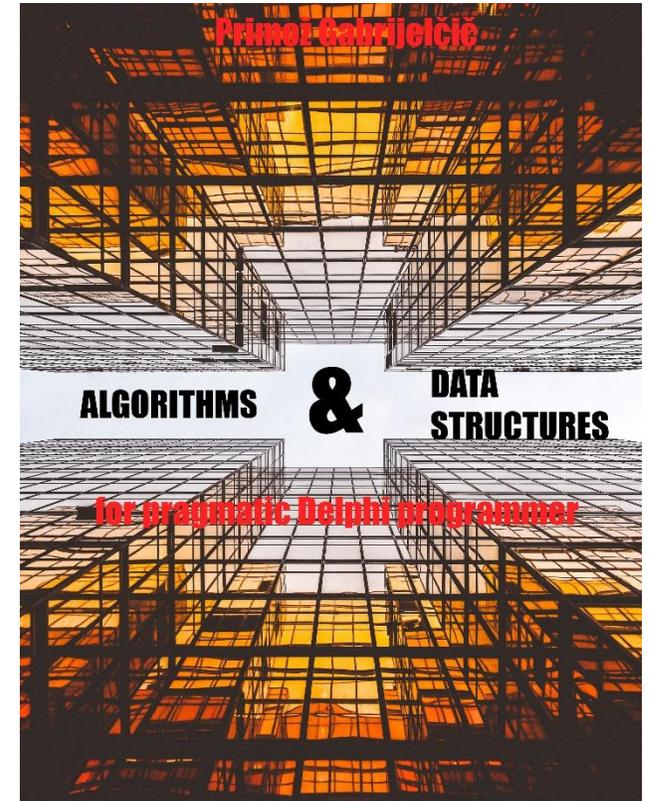
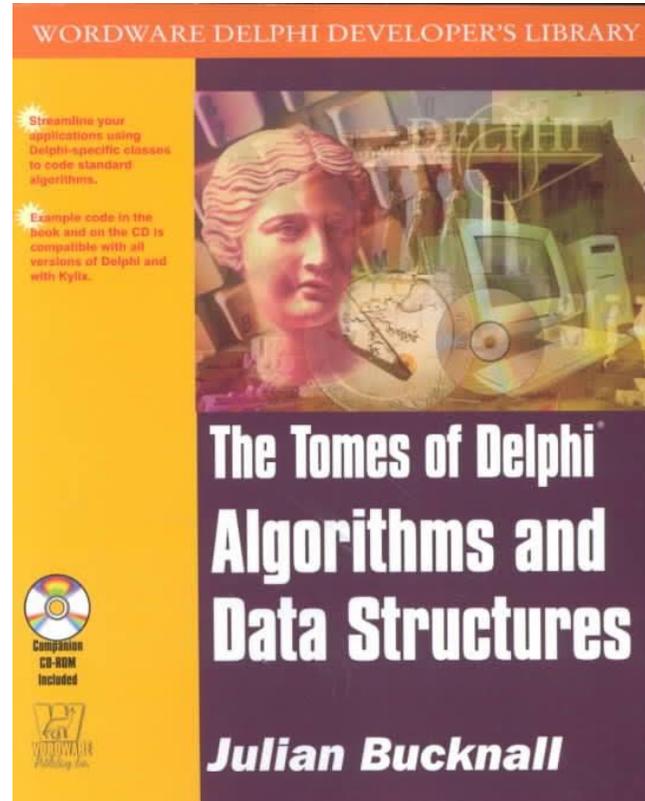
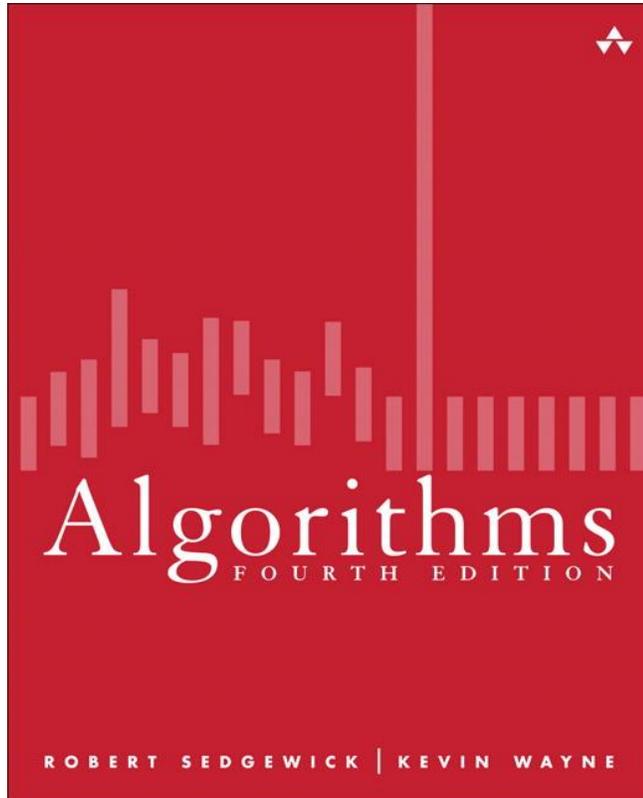
If something is slow, don't do it

- BeginUpdate/EndUpdate
- UI virtualization
 - Virtual listbox
 - Virtual TreeView

LEARN MORE



Learn more



Q&A