

Writing High Performance Delphi Applications

Primož Gabrijelčič

<http://tiny.cc/dk2018gp1>

<http://tiny.cc/dk2018hp>

About performance



Performance

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

What is performance?

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

Improving performance

- Analyzing algorithms
- *Measuring execution time*
- Fixing algorithms
- Fine tuning the code
- Mastering memory manager
- *Writing parallel code*
- *Importing libraries*

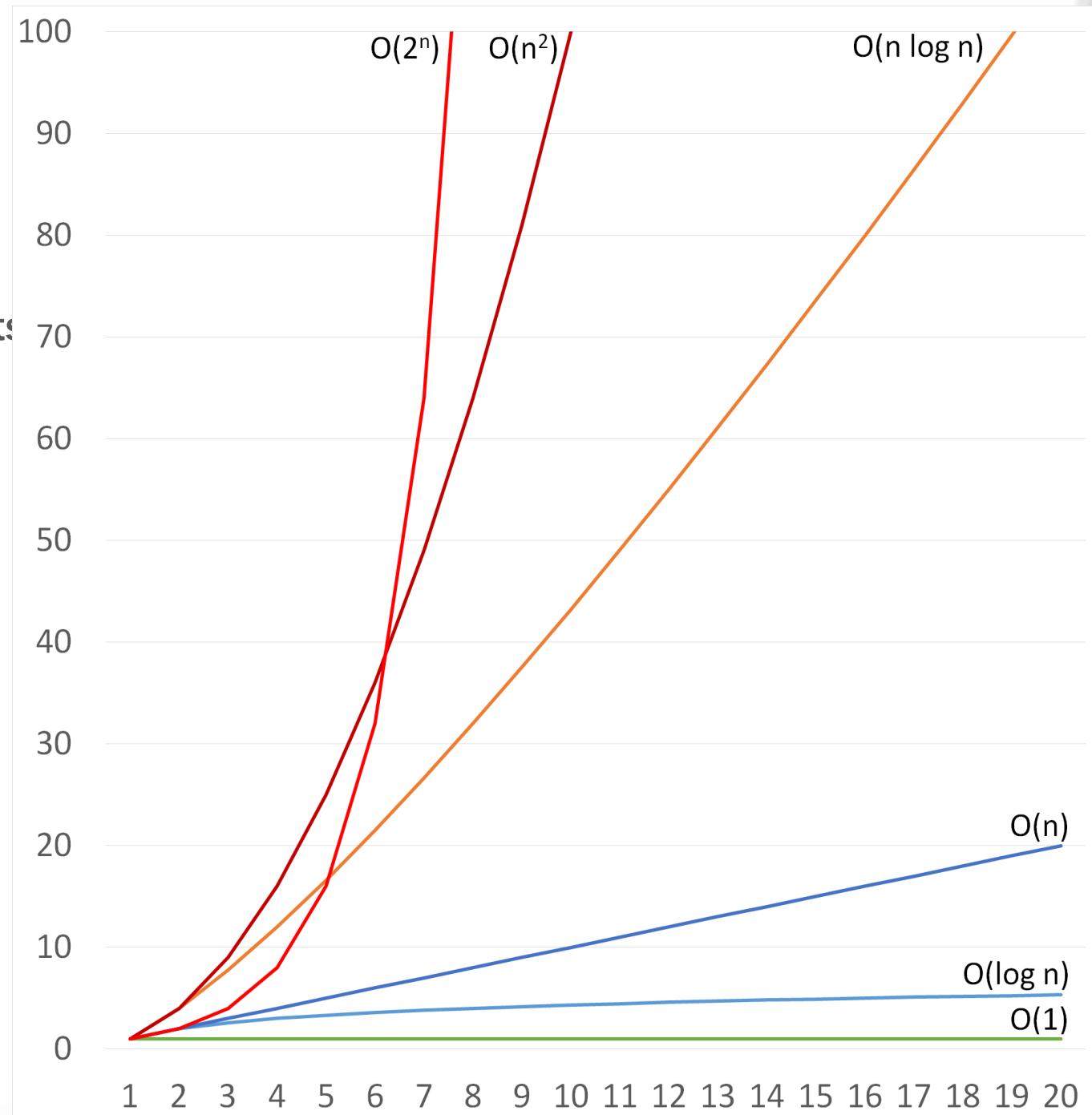
Algorithm complexity



Algorithm complexity

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

- $O(1)$ accessing array elements
- $O(\log n)$ searching in ordered list
- $O(n)$ linear search
- $O(n \log n)$ quick sort (average)
- $O(n^2)$ quick sort (worst),
naive sort (bubblesort,
insertion, selection)
- $O(cn)$ 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
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}

Complexities in RTL

- Lists
 - access $O(1)$
 - search $O(n) / O(n \log n)$
 - sort $O(n \log n)$
- Dictionary
 - * $O(1)$
 - search by value $O(n)$
 - Unordered!
 - Spring4D 1.2.1: CreateSortedDictionary
- Trees
 - * $O(\log n)$
 - Spring4D 1.2.1: TRedBlackTree

<http://bigocheatsheet.com/>

Measuring performance



Measuring

- Manual
 - GetTickCount
 - QueryPerformanceCounter
 - TStopwatch
- Automated - Profilers
 - Sampling
 - Instrumenting
 - Binary
 - Source

Free profilers

- **AsmProfiler**
 - André Mussche
 - Instrumenting and sampling
 - 32-bit
 - <https://github.com/andremussche/asmprofiler>
- **Sampling Profiler**
 - Eric Grange
 - Sampling
 - 32-bit
 - <https://www.delphitools.info/samplingprofiler>

Commercial profilers

- **AQTime**
 - Instrumenting and sampling
 - 32- and 64-bit
 - <https://smartbear.com/>
- **Nexus Quality Suite**
 - Instrumenting
 - 32- and 64-bit
 - <https://www.nexusdb.com>
- **ProDelphi**
 - Instrumenting (source)
 - 32- and 64-bit
 - <http://www.prodelphi.de/>

A task for you!

If you choose to accept it ...

Task 1_1 Primoz\Task11

Move one line in function Test to a different place (in the same function) to make the code run faster.

Fixing the algorithm



Fixing the algorithm

- Find a better algorithm
- If a part of program is slow, don't execute it so much
- If a part of program is slow, don't execute it at all

“Don’t execute it so much”

- Don’t update UI thousands of times per second
- Call BeginUpdate/EndUpdate
- Don’t send around millions of messages per second

“Don’t execute it at all”

- UI virtualization
 - Virtual listbox
 - Virtual TreeView
- Memoization
 - Caching
 - Dynamic programming
 - **TGpCache<K,V>**
 - O(1) all operations
 - GpLists.pas, <https://github.com/gabr42/GpDelphiUnits/>

A task for you!

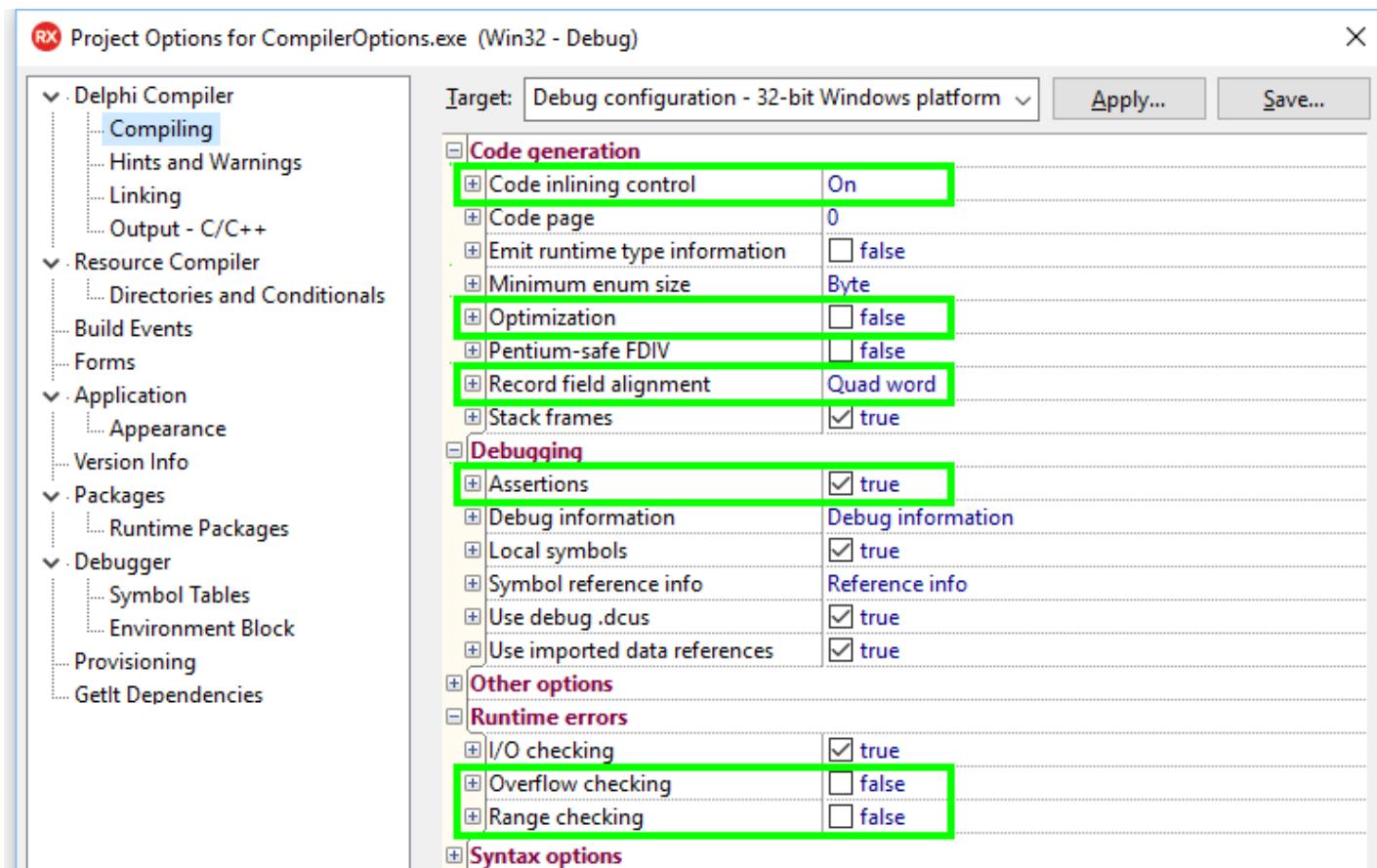
If you choose to accept it ...

Task 1_2 Primoz\Task12

Make function Test run faster!

Fine tuning the code

Compiler settings



Behind the scenes

- Strings
 - Reference counted, Copy on write
- Arrays
 - Static
 - Dynamic
 - Reference counted, Cloned
- Records
 - Initialized if managed
- Classes
 - Reference counted on ARC
- Interfaces
 - Reference counted

Calling methods

- Parameter passing
 - Dynamic arrays are strange
- Inlining
 - Single pass compiler!

A task for you!

If you choose to accept it ...

Task 1_3 Primoz\Task13

Without changing the program architecture, make it faster!

Memory management

Why memory manager?

- No fine-grained allocation on OS level
- Speed
- Additional functionality (debugging)

String and memory allocations

- NO[†]: `string := string + 'c'`
- NO[†]: `SetLength(array, Length(array) + 1)`
- KIND OF OK[‡]: `for ... do list.Add(something)`

[†] KIND OF OK with a good memory manager

[‡] May waste LOTS of memory [but improved in 10.3]

Memory management functions

- `GetMem`, `AllocMem`, `ReallocMem`, `FreeMem`
- `GetMemory`, `ReallocMemory`, `FreeMemory`
- `New`, `Dispose`
- `Initialize`, `Finalize`

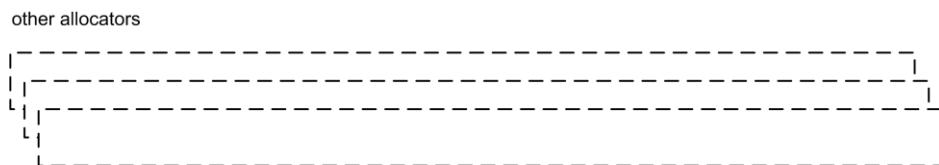
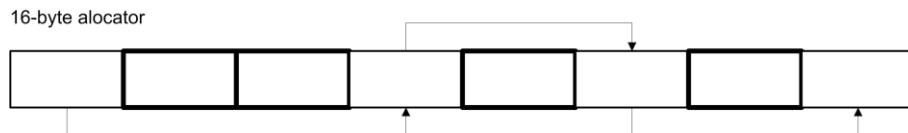
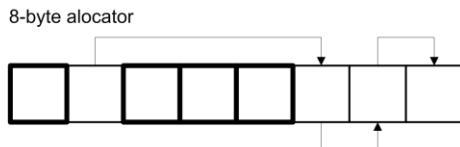
Records vs. objects

- Objects
 - `TObject.NewInstance`
 - `TObject.InstanceSize`
 - `GetMem`
 - `InitInstance`
 - `constructor (inherited ...)`
 - `AfterConstruction (inherited ...)`
- Records[†]
 - `GetMem`
 - `[_Initialize]`

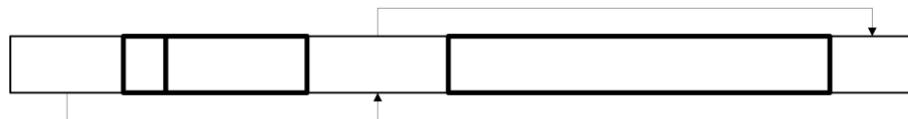
[†] Will change in 10.3

FastMM4: 58 memory managers in one!

Small block allocators



Medium block allocator



Large block allocator



Image source: 'Delphi High Performance'

© 2018 Packt Publishing

Optimizations

- Reallocation
 - Small blocks: New size = at least 2x old size
 - Medium blocks: New size = at least 1.25x old size
- Allocator locking
 - Small block only
 - Will try 2 'larger' allocators
 - Problem: Freeing memory

```
allocIdx := find best allocator for the memory block
repeat
    if can lock allocIdx then
        break;
    Inc(allocIdx);
    if can lock allocIdx then
        break;
    Inc(allocIdx);
    if can lock allocIdx then
        break;
    Dec(allocIdx, 2)
until false
```

Optimizing parallel allocations

- FastMM4 from GitHub
 - <https://github.com/pleriche/FastMM4>
- **DEFINE LogLockContention**

or
- **DEFINE UseReleaseStack**

Alternatives

- ScaleMM
 - <https://github.com/andremussche/scalemm>
- TBBMAlloc
 - <https://www.threadingbuildingblocks.org>

A task for you!

If you choose to accept it ...

Task 1_4 Primoz\Task14

Optimize memory allocations to make the program run faster!