Desempenho poliglota

Melhoria de performance independente de linguagem ou plataforma

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

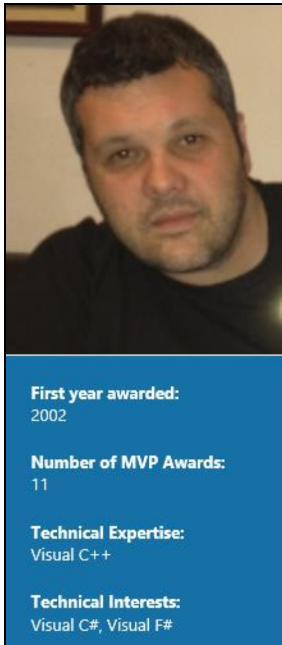
fabiogaluppo@acm.org

Engenheiro de Software BM&FBovespa

http://www.bmfbovespa.com.br

Microsoft MVP Visual C++

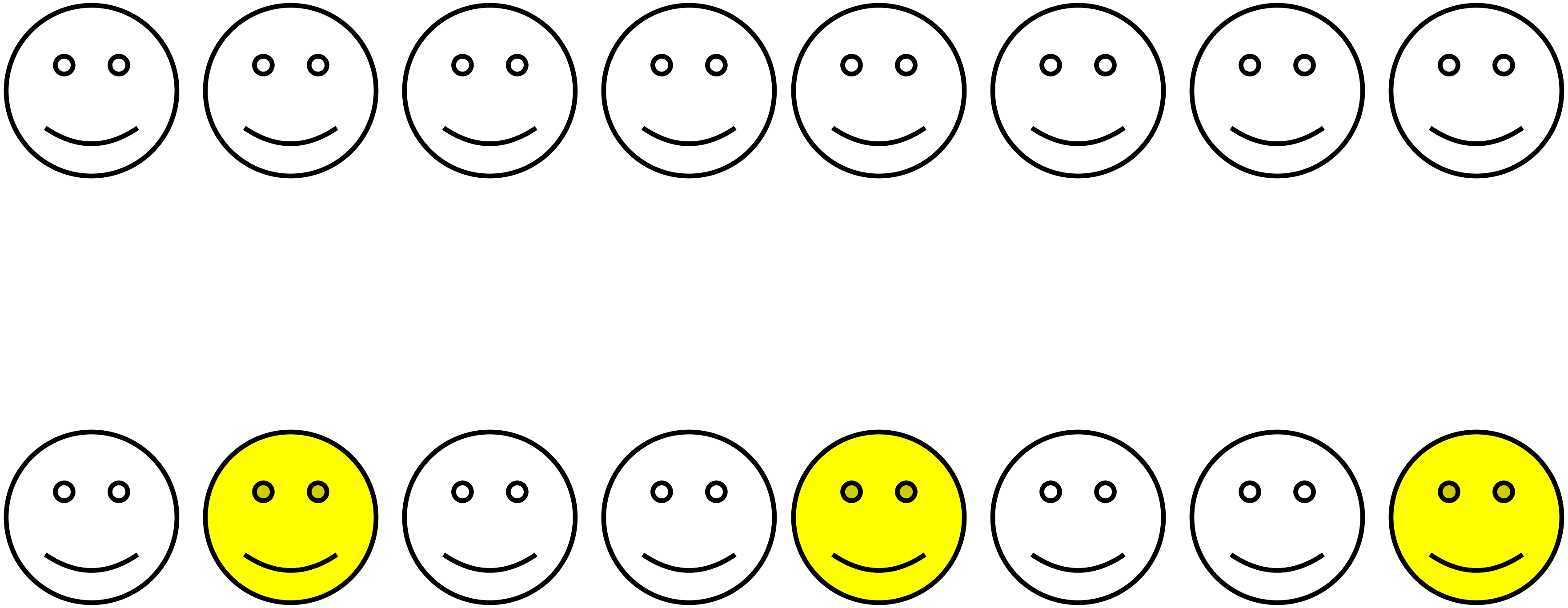
http://bit.ly/desempenho_poliglota

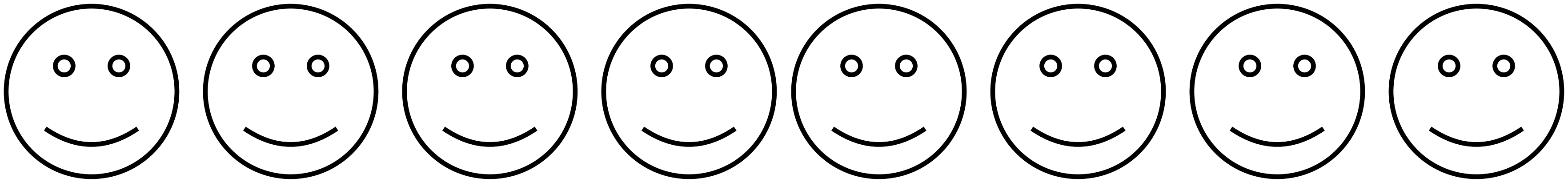


Fabio Razzo Galuppo, M.Sc.

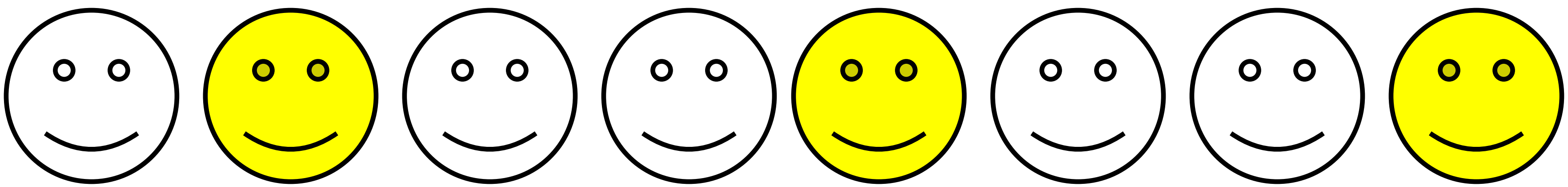
Novembro 1973

- Mestrado em Engenharia Elétrica (Universidade Presbiteriana Mackenzie)
  - Ciência da Computação - Inteligência Artificial
- Por mais de 10 anos premiado com Microsoft MVP em Visual C++
- Engenheiro de Software (Programador)
- Matemática Aplicada
- Linguagens de programação prediletas:
  - C++
  - F#
  - Haskell
- Rock’n’Roll
  - E boa música em geral
- http://fabiogaluppo.com
- https://github.com/fabiogaluppo
- http://simplycpp.com





$$N = 8 \rightarrow Contagem = 8$$



$$N = 8 \rightarrow Contagem = 3$$

$$3 = \log_2 8$$

Imperativo

```
long count = 0;
for (int i = 0; i < N; ++i)
    ++count;
```

$$N$$

```
long count = 0;
for (int i = 1; i < N; i = 2 * i)
    ++count;
```

$$\lg N$$

```
long count = 0;
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        ++count;
```

$$N^2$$

```
long count = 0;
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        for (int k = 0; k < N; ++k)
            ++count;
```

$$N^3$$

```java
long count = 0;
for (int i = 1; i * i <= N; ++i)
    for (int j = 1; j * j <= N; ++j)
        for (int k = 1; k * k <= N; ++k)
            for (int l = 1; l <= 4; ++l)
                ++count;
```

$$4\sqrt{N^3}$$

```java
long count = 0;

//N
Integer[] xs = new Integer[N];
for (int i = 0; i < N; ++i) {
    xs[i] = i + 1;
    ++count;
}

count += shuffle(xs); //N - 1

Comp comp = new Comp();
Arrays.binarySearch(xs, N, comp); //lg N
count += comp.count;
```

$$2\,N + lg\,N - 1$$

Funcional

```
let N = List.length xs
let count = ref 0
xs |> List.map (fun x -> count := !count + 1;  x * 2) |> ignore
```

$$N$$

```
let N = List.length xs
let count = ref 0
xs |> List.collect (fun i -> [for j = 1 to N do count := !count + 1;
                                                 yield (i, j)]) |> ignore
```

$$N^2$$

```
let N = List.length xs
let count = ref 0
xs |> List.toArray |> Array.sortWith (fun x y -> count := !count + 1; x - y) |> ignore
```

$$O(N \lg N) \qquad \sim 1.2\, N \lg N$$

Composição

```
let N = List.length xs
let count = ref 0
let f (x) = count := !count + 1; x * 2
let g (x) = count := !count + 1; x + 1
let h = f >> g //g . f
xs |> List.map (h) |> ignore
```

$O(N)$ $\sim 2\,N$

```
N =     16 count =       32 [2 N =       32]
N =     32 count =       64 [2 N =       64]
N =     64 count =      128 [2 N =      128]
N =    128 count =      256 [2 N =      256]
N =    256 count =      512 [2 N =      512]
N =    512 count =     1024 [2 N =     1024]
N =   1024 count =     2048 [2 N =     2048]
N =   2048 count =     4096 [2 N =     4096]
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

‘Big-Oh’ (O) Omega (Ω) Theta (Θ)

- *Upper bound* – ‘Big-Oh’ (O) – análogo a ≤

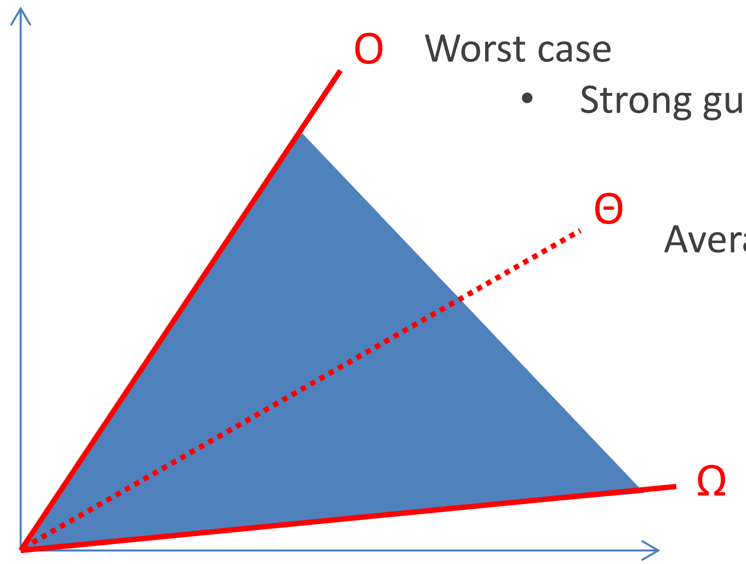
$$g(N) = O\big(f(n)\big)\ quando\ a\ proporção\ \left|\frac{g(n)}{f(n)}\right| \text{é}\ delimitada\ por\ cima\ quando\ N\ tende\ ao\ infinito$$

- *Lower bound* – Omega (Ω) – análogo a ≥

$$g(N) = \Omega\big(f(n)\big)\ quando\ a\ proporção\ \left|\frac{g(n)}{f(n)}\right| \text{é}\ delimitada\ por\ baixo\ quando\ N\ tende\ ao\ infinito$$

- Theta (Θ) – análogo a =

$$g(N) = \Theta\big(f(n)\big)\ quando\ for\ g(N) = O\big(f(n)\big)\ e\ g(N) = \Omega\big(f(n)\big)$$



Worst case

- Strong guarantees, maybe unrealistic data

Average case

- “Typical” input data

‘Big-Oh’ (O) Omega (Ω) Theta (Θ)

$$f_1(N) = N^2 \qquad f_2(N) = 3N + 10 \qquad f_3(N) = N + 2$$

N	f1	f2	f3	f2/f1	f1/f2	f2/f3	f3/f2
0	0.000000	10.000000	2.000000	#DIV/0!	0.000000	5.000000	0.200000
1	1.000000	13.000000	3.000000	13.000000	0.076923	4.333333	0.230769
2	4.000000	16.000000	4.000000	4.000000	0.250000	4.000000	0.250000
5	25.000000	25.000000	7.000000	1.000000	1.000000	3.571429	0.280000
10	100.000000	40.000000	12.000000	0.400000	2.500000	3.333333	0.300000
100	10000.000000	310.000000	102.000000	0.031000	32.258065	3.039216	0.329032
250	62500.000000	760.000000	252.000000	0.012160	82.236842	3.015873	0.331579
500	250000.000000	1510.000000	502.000000	0.006040	165.562914	3.007968	0.332450
1000	1000000.000000	3010.000000	1002.000000	0.003010	332.225914	3.003992	0.332890
10000	100000000.000000	30010.000000	10002.000000	0.000300	3332.222592	3.000400	0.333289
100000	10000000000.000000	300010.000000	100002.000000	0.000030	33332.222259	3.000040	0.333329
1000000	1000000000000.000000	3000010.000000	1000002.000000	0.000003	333332.222226	3.000004	0.333333
10000000	100000000000000.000000	30000010.000000	10000002.000000	0.000000	3333332.222223	3.000000	0.333333
100000000	10000000000000000.000000	300000010.000000	100000002.000000	0.000000	33333332.222222	3.000000	0.333333
1000000000	1000000000000000000.000000	3000000010.000000	1000000002.000000	0.000000	333333332.222222	3.000000	0.333333

$$f_2(N) = O(f_1(N))$$

$$f_2(N) = O(f_3(N)) \land f_3(N) = O(f_2(N)) \Rightarrow f_2(N) = \Theta(f_3(N))$$ relação antisimétrica

$$f_1(N) = \Omega(f_3(N))$$

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Dominância

- $Exponecial\ domina\ Polinomial$
- $N^a\ domina\ N^b\ quando\ a > b$
- $Polinomial\ domina\ Logaritmo$

regras de senso comun

$$lg\,N < \sqrt{N} < lg^2 N < N < N\,lg\,N < N^2 < 2^N < N! < 2^{N^2} < 2^{2^N} < N^{2^N}$$

lg N	N^1/2	N	N lg N	N^2	N^3
4	4	16	64	256	4096
5	6	32	160	1024	32768
6	8	64	384	4096	262144
7	11	128	896	16384	2097152
8	16	256	2048	65536	16777216
9	23	512	4608	262144	134217728
10	32	1024	10240	1048576	1073741824
11	45	2048	22528	4194304	8589934592

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Função e Função Inversa

f	f^{-1}
$\lg N$	2^N
$\sqrt{N}$	N^2
N	N
$N \lg N$	$(N \lg N)^{-1}$ deve ser uma aproximação, verifique código do link no rodapé
N^2	$\sqrt{N}$
N^3	$\sqrt[3]{N}$
2^N	$\lg N$
$N!$	$N!^{-1}$ "$!^{-1}$" deve ser uma aproximação

$$f \circ f^{-1} = id$$

$$(\lg \circ 2^x)(N) = N$$

https://github.com/fabiogaluppo/samples/blob/master/events/qconrio2015/code/fsharp/FunctionsAndInverses.fs

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

“$!^{-1}$” Inversa do Fatorial

$$\Gamma(N) = (N - 1)! \qquad \Gamma(N + 1) = N!$$

$$\Gamma^{-1}$$ Approximated Inverse Gamma

http://fssnip.net/rw

Load in: tsunami.io tryfsharp.org tryfs.net Tools: Copy Link Copy Source

```
//More info and native code at:
//https://github.com/fabiogaluppo/samples/tree/master/fragments/InverseGamma

//References:
//David W. Cantrell's Inverse gamma function (and Inverse factorial): http://ma
//DarkoVeberic's C++ implementation of the Lambert W(x) function: https://githu

open System
open System.Runtime.InteropServices

[<DllImport("bin\\LambertW.dll", CallingConvention=CallingConvention.Cdecl)>]
extern float LambertW_0(float x)

let c = 0.036534
let ln = Math.Log
let pi = Math.PI
let L x = ln((x + c) / Math.Sqrt(2. * pi))
let W x = LambertW_0 x
let e = Math.E

let AIG x =
  //Approximated Inverse Gamma
  L(x) / (W (L(x) / e)) + 1. / 2.

let InvFact x =
  //Inverse Factorial in terms of rounded AIG
  Math.Round(AIG x) - 1.
```

```
showAIG 1.
showAIG 24.
showAIG 362880.
showAIG 1.216451e+17
showInvFact 6.
showInvFact 24.
showInvFact 3628800.
showInvFact 2.432902e+18

(*

Inverse Gamma function and Inverse Factorial
AIG(                    1.0) =  2.021203
AIG(                   24.0) =  4.994871
AIG(               362880.0) =  9.998053
AIG(     121645100000000000.0) = 19.999281
InvFact(                 6.0) =  3.000000
InvFact(                24.0) =  4.000000
InvFact(           3628800.0) = 10.000000
InvFact(2432902000000000000.0) = 20.000000

*)
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

CLRS Problem 1.1

- http://answers-by-me.blogspot.com.br/2010/07/clrs-2e-problem-1-1.html

"I couldn't find how to achieve the value for n lg(n) for 1 second."

"Sorry, I'm not good at this type of math. I asked the computer. Wolfram Alpha is really good at this stuff."

- 1 segundo

$$\lg(N) = 10^6 \qquad N = 2^{10^6}$$

WolframAlpha computational.. knowledge engine

Input:

$2^{1000000}$

Decimal approximation: More digits

$9.900656229295898250697923616301903250733624241787567 33\ldots \times 10^{301029}$

- 1 minuto

$$\lg(N) = 10^6 * 60 \qquad N = 2^{6*10^7}$$

$5.493370256404490239091681579060385311908847922601753\ldots \times 10^{18061799}$

- 1 segundo

$$N! = 10^6 \qquad N = 10^6!^{-1}$$

```
> showInvFact 1000000.;;
InvFact(           1000000.0) =  9.000000
```

$$\frac{f(n)\ complexity}{x\,\frac{instructions}{second}} = t(n)\ second(s)$$

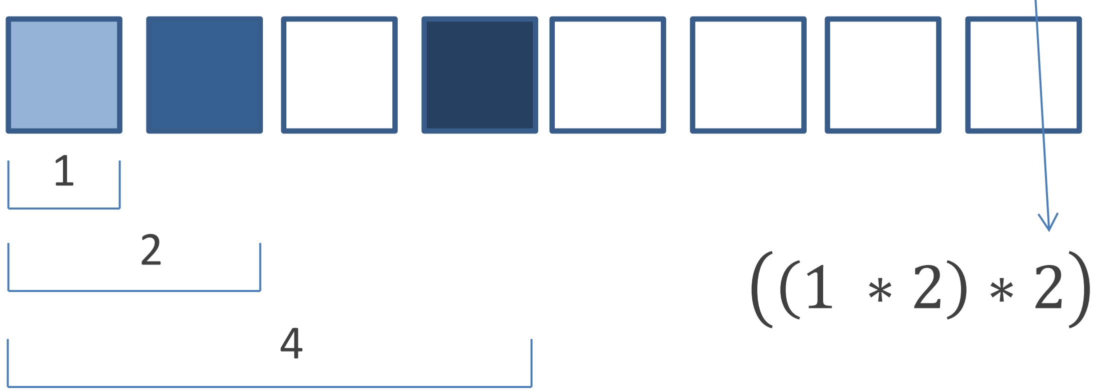
Equação do Problema

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Alguma observação?

```
long count = 0;
for (int i = 1; i < N; i = 2 * i)
    ++count;
```

$$\lg N$$

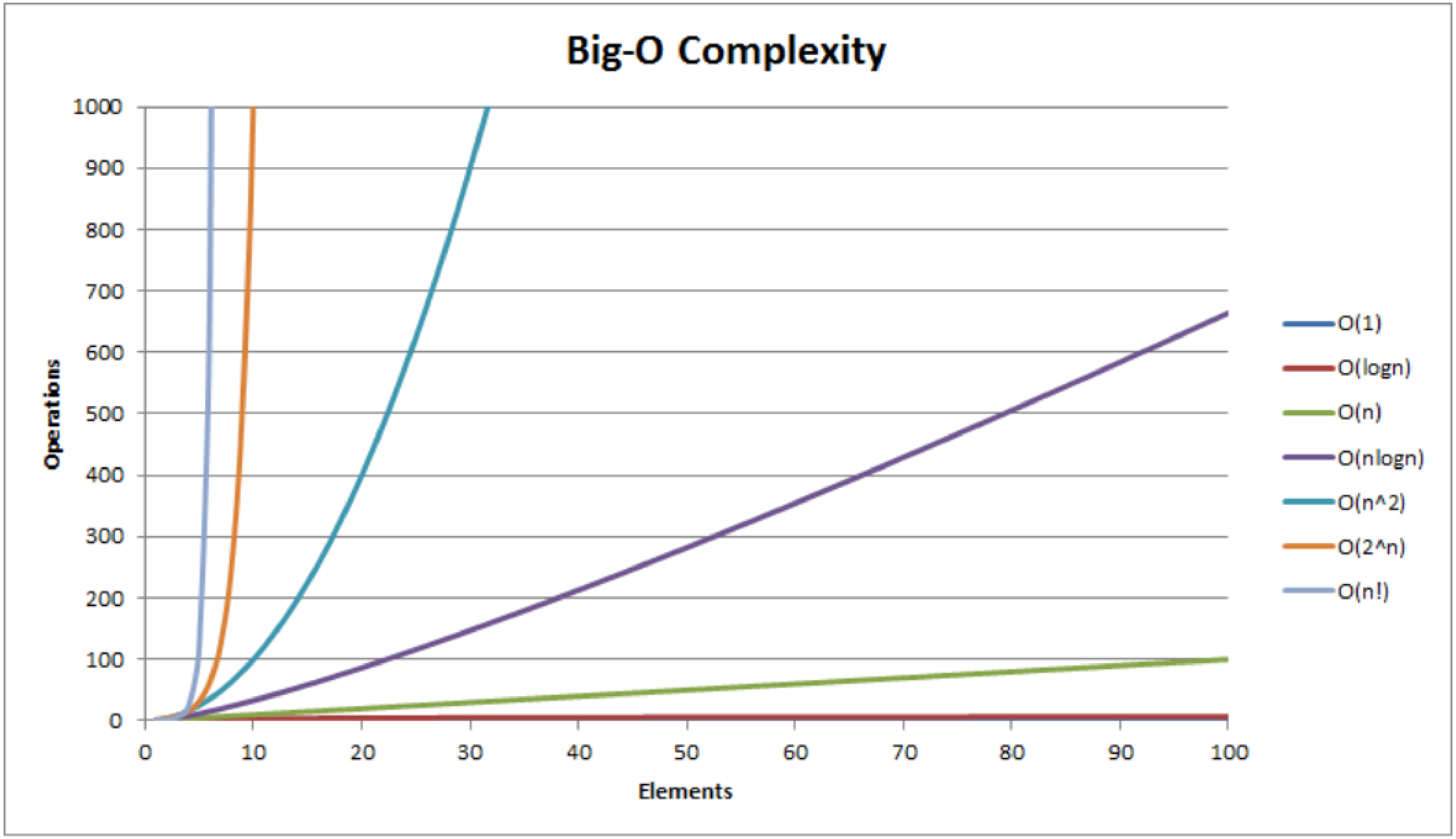


$$\big((1 \ast 2) \ast 2\big) \ast 2 = 2^3$$

f	f^{-1}
$\lg N$	2^N

Know Thy Complexities!

- http://bigocheatsheet.com/



Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Know Thy Machine!

Peter Norvig

Approximate timing for various operations on a typical PC:

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

http://norvig.com/21-days.html

Teach Yourself Programming in Ten Years

Timings for various operations on a typical PC on human scale

execute typical instruction	1 second
fetch from L1 cache memory	0.5 seconds
branch misprediction	5 seconds
fetch from L2 cache memory	7 seconds
Mutex lock/unlock	½ minute
fetch from main memory	1½ minutes
send 2K bytes over 1Gbps network	5½ hours
read 1MB sequentially from memory	3 days
fetch from new disk location (seek)	13 weeks
read 1MB sequentially from disk	6½ months
send packet US to Europe and back	5 years

Erik Meijer

https://www.coursera.org/course/reactive

Principles of Reactive Programming

Medindo o tempo de execução

```
#include <chrono>

struct stop_watch final
{
    stop_watch() : Start_(now()) {}

    std::chrono::seconds elapsed_s() const
    {
        using std::chrono::seconds;
        return std::chrono::duration_cast<seconds>(elapsed());
    }

    std::chrono::milliseconds elapsed_ms() const
    {
        using std::chrono::milliseconds;
        return std::chrono::duration_cast<milliseconds>(elapsed());
    }

    std::chrono::microseconds elapsed_us() const
    {
        using std::chrono::microseconds;
        return std::chrono::duration_cast<microseconds>(elapsed());
    }

    std::chrono::nanoseconds elapsed_ns() const
    {
        using std::chrono::nanoseconds;
        return std::chrono::duration_cast<nanoseconds>(elapsed());
    }

    void restart() { Start_ = now(); }

    stop_watch(const stop_watch&) = delete;
    stop_watch& operator=(const stop_watch&) = delete;

private:
    static std::chrono::high_resolution_clock::time_point now()
    {
        return std::chrono::high_resolution_clock::now();
    }
```

```
stop_watch sw;
…
auto us = sw.elapsed_us().count();
sw.restart();
…
auto ms = sw.elapsed_ms().count();
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Análise Empírica

- Medir o tempo de execução com variações no tamanho de entrada (N)

```
Elapsed time:       0 ms max_element algorithm N:  1048576 result:  1048576.
Elapsed time:       0 ms max_element algorithm N:  2097152 result:  2097152.
Elapsed time:      15 ms max_element algorithm N:  4194304 result:  4194304.
Elapsed time:      20 ms max_element algorithm N:  8388608 result:  8388608.
Elapsed time:      42 ms max_element algorithm N: 16777216 result: 16777216.
Elapsed time:      74 ms max_element algorithm N: 33554432 result: 33554432.
Elapsed time:     163 ms max_element algorithm N: 67108864 result: 67108864.
```

```
stats_table results = Test_max_element<ElapsedMilliseconds>(Ns);
double b = display_stats(results);
```

“Framework” para Análise Empírica

```
template<class ElapsedPolicy>
stats_table Test_max_element(std::initializer_list<size_t> Ns)
{
    stats_table results;

    for (size_t N : Ns)
    {
        std::vector<int> xs;
        xs.resize(N);
        std::iota(xs.begin(), xs.end(), 1);

        unsigned seed = 1234567890;
        std::shuffle(xs.begin(), xs.end(), std::default_random_engine(seed));

        double elapsed{};
        int result{};
        auto m = do_measurement<ElapsedPolicy>([&]() {

            const auto& max_iter = std::max_element(xs.begin(), xs.end());
            result = *max_iter;

        }, "max_element algorithm", elapsed);

        std::cout << "Elapsed time: " << m << " N: " << std::setw(8) << N
            << " result: " << std::setw(8) << result << ".\n";

        results.push_back(std::make_tuple(static_cast<double>(N), elapsed));
    }

    return results;
}
```

Preparar

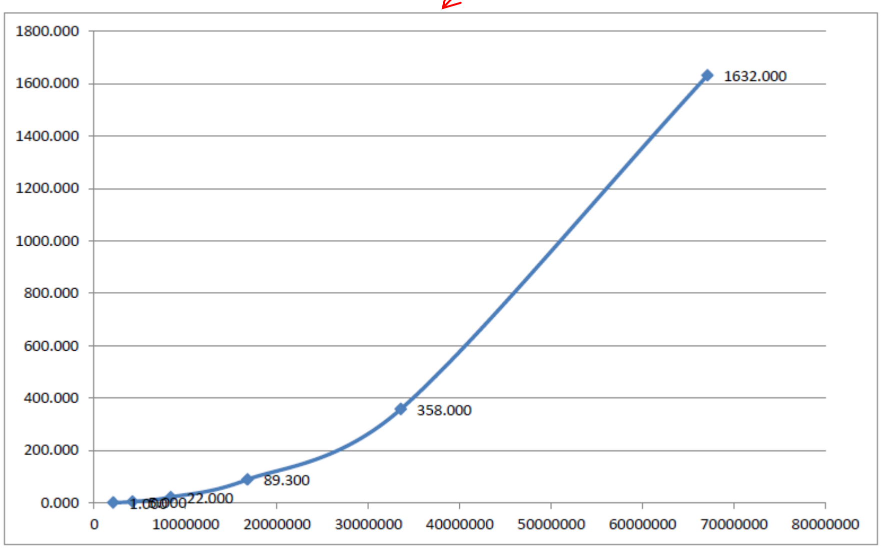
Executar

Coletar

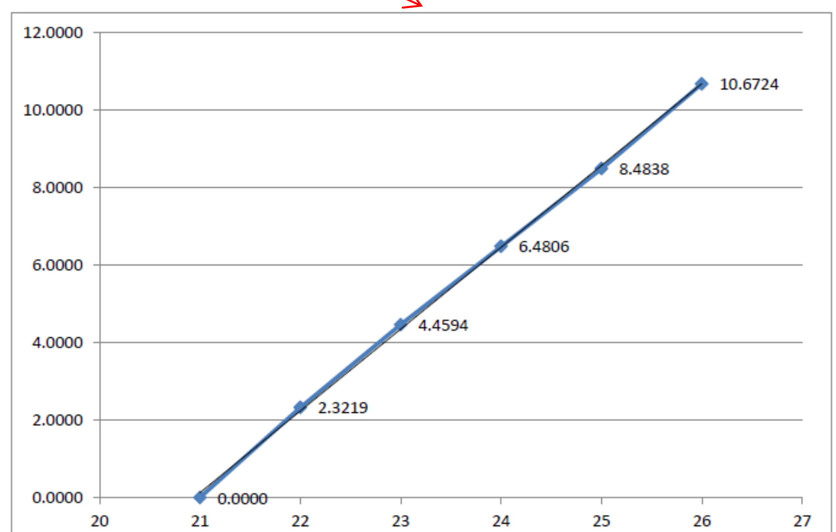
https://github.com/fabiogaluppo/samples/tree/master/fragments/Measure

Análise dos dados

N	T	LG N	LG T	RATIO	LG RATIO
1048576	0.220	20	-2.1844	#VALUE!	#VALUE!
2097152	1.000	21	0.0000	4.5455	2.1844
4194304	5.000	22	2.3219	5.0000	2.3219
8388608	22.000	23	4.4594	4.4000	2.1375
16777216	89.300	24	6.4806	4.0591	2.0212
33554432	358.000	25	8.4838	4.0090	2.0032
67108864	1632.000	26	10.6724	4.5587	2.1886



Standard plot: T(N) x N



Log-log plot: : log(T(N)) x log(N)

Tempo de execução estimado

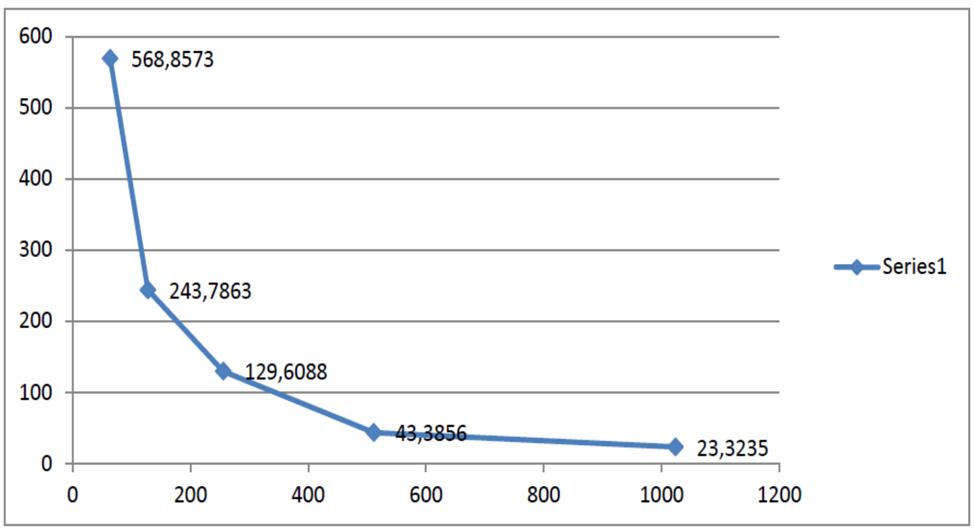
```
Elapsed time:      0 ms max_element algorithm N:  1048576 result:  1048576.
Elapsed time:      0 ms max_element algorithm N:  2097152 result:  2097152.
Elapsed time:     15 ms max_element algorithm N:  4194304 result:  4194304.
Elapsed time:     15 ms max_element algorithm N:  8388608 result:  8388608.
Elapsed time:     42 ms max_element algorithm N: 16777216 result: 16777216.
Elapsed time:     86 ms max_element algorithm N: 33554432 result: 33554432.
Elapsed time:    162 ms max_element algorithm N: 67108864 result: 67108864.

        N               T     log N     log T     ratio log ratio
  1048576          0.0000        20   -1.#INF   -1.#INF   -1.#INF
  2097152          0.0000        21   -1.#INF   -1.#IND   -1.#IND
  4194304         15.0000        22    3.9069    1.#INF    1.#INF
  8388608         15.0000        23    3.9069    1.0000    0.0000
 16777216         42.0000        24    5.3923    2.8000    1.4854
 33554432         86.0000        25    6.4263    2.0476    1.0339
 67108864        162.0000        26    7.3399    1.8837    0.9136

Elapsed time:    165 ms max_element algorithm N: 67108864 result: 67108864.
Elapsed time:    168 ms max_element algorithm N: 67108864 result: 67108864.
Elapsed time:    165 ms max_element algorithm N: 67108864 result: 67108864.
Elapsed time:    167 ms max_element algorithm N: 67108864 result: 67108864.
Elapsed time:    165 ms max_element algorithm N: 67108864 result: 67108864.

Estimated runnning time is 1.1740e-005 x N^0.9136 ms
```

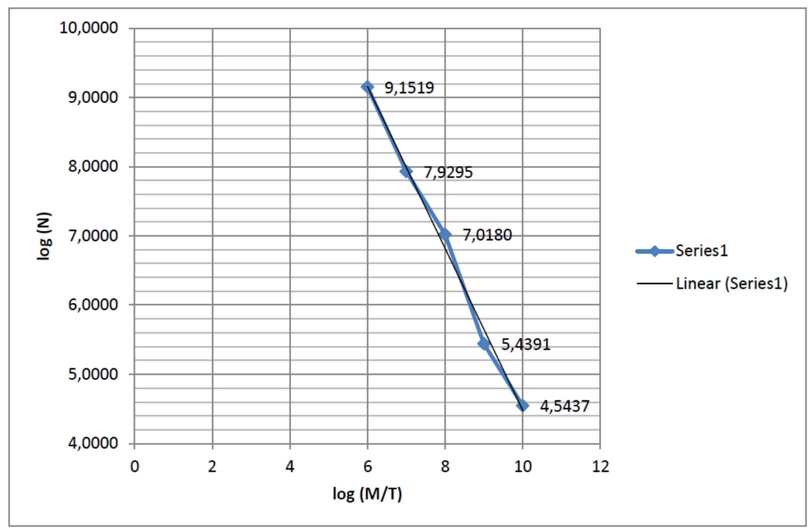
$$Lei\ da\ Potência:\ aN^b$$

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Wrapping up

```
  32   10240     9000 us 1137.7778 msgs/ms
  64   10240    18001 us  568.8573 msgs/ms
 128   10240    42004 us  243.7863 msgs/ms
 256   10240    79007 us  129.6088 msgs/ms
 512   10240   236023 us   43.3856 msgs/ms
1024   10240   439043 us   23.3235 msgs/ms
2048   10240   793079 us   12.9117 msgs/ms
```





Considerações sobre medição

- Medir com precisão é um grande desafio

- Efeitos independentes do sistema
  - Algoritmo e sua complexidade
  - Quantidade de dados (N)
  - Influência o expoente e a constante na lei da potência

- Efeitos dependentes do sistema
  - Hardware
    - CPU, Memória, Cache, …
  - Software
    - Compilador, Máquina virtual, GC, …
    - Sistema operacional, Rede, …
  - Influência a constante na lei da potência

$Lei\ da\ Potência: aN^b$

Ref.: http://algs4.cs.princeton.edu/14analysis/

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Medindo código gerenciado

```
N = 100
515582 4485 515582 112649 515582 112649 318373 112649 309426 309426
      3 ms  (GCs=  0) Naive Median Maintenance
515582 4485 515582 112649 515582 112649 318373 112649 309426 309426
      9 ms  (GCs=  0) Median Maintenance
------------------------
------------------------
N = 100
822444 598188 598188 598188 822444 598188 598188 586475 598188 586475
      0 ms  (GCs=  0) Naive Median Maintenance
822444 598188 598188 598188 822444 598188 598188 586475 598188 586475
      0 ms  (GCs=  0) Median Maintenance
------------------------
------------------------
N = 1000
302787 302787 302787 302787 302787 302787 302787 267396 267396 267396
      5 ms  (GCs=  0) Naive Median Maintenance
302787 302787 302787 302787 302787 302787 302787 267396 267396 267396
      2 ms  (GCs=  0) Median Maintenance
------------------------
------------------------
N = 10000
500315 500315 500315 447911 500315 500315 671360 671360 703316 703316
    642 ms  (GCs=  0) Naive Median Maintenance
500315 500315 500315 447911 500315 500315 671360 671360 703316 703316
     19 ms  (GCs=  2) Median Maintenance
------------------------
------------------------
N = 100000
408615 408615 408615 408615 664838 664838 696500 664838 664838 664838
 78,101 ms  (GCs=  0) Naive Median Maintenance
408615 408615 408615 408615 664838 664838 696500 664838 664838 664838
    291 ms  (GCs= 19) Median Maintenance
```

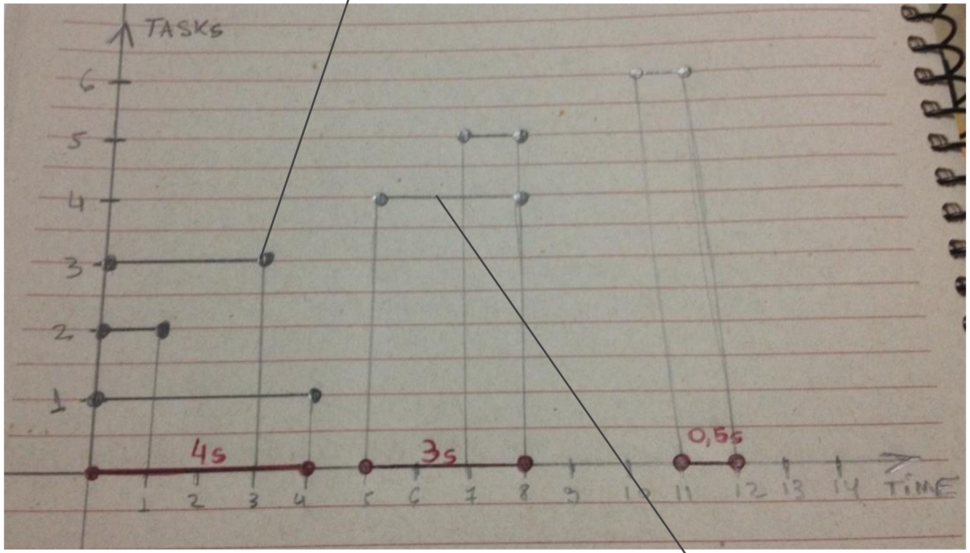
```csharp
static void Test(int N)
{
    Console.WriteLine("------------------------");
    Console.WriteLine("N = " + N);

    var xs = RandomRange(N).ToList();

    List<int> ys = new List<int>();
    var t1 = InstrumentedOperation.Test(() => {
        ys.AddRange(NaiveMedianMaintenance(xs));
    }, "Naive Median Maintenance");
    Console.Write(ys[0]);
    for (int i = 1; i < 10; ++i) Console.Write(" " + ys[i]);
    Console.WriteLine();
    Console.WriteLine(t1);

    ys = new List<int>();
    var t2 = InstrumentedOperation.Test(() => {
         ys.AddRange(MedianMaintenance(xs));
    }, "Median Maintenance");
    Console.Write(ys[0]);
    for (int i = 1; i < 10; ++i) Console.Write(" " + ys[i]);
    Console.WriteLine();
    Console.WriteLine(t2);

    Console.WriteLine("------------------------");
}
```

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Medindo tarefas em paralelo

```
time_line_segment_1D_collection xs;
time_ruler m1, m2, m3;
std::this_thread::sleep_for(std::chrono::seconds(1));
xs.collect(m2.get());
std::this_thread::sleep_for(std::chrono::seconds(2));
xs.collect(m3.get());
std::this_thread::sleep_for(std::chrono::seconds(1));
xs.collect(m1.get());
std::this_thread::sleep_for(std::chrono::seconds(1));
time_ruler m4;
std::this_thread::sleep_for(std::chrono::seconds(2));
time_ruler m5;
std::this_thread::sleep_for(std::chrono::seconds(1));
xs.collect(m4.get());
//std::this_thread::sleep_for(std::chrono::milliseconds(100));
xs.collect(m5.get());
std::this_thread::sleep_for(std::chrono::seconds(3));
time_ruler m6;
std::this_thread::sleep_for(std::chrono::milliseconds(500));
xs.collect(m6.get());

auto result = xs.compute();
```

time_point



time_line_segment_1D

```
Count    =      6
Total    =   7578 ms
Average  =   1263 ms
Std Dev  =   1899 ms
Min      =    512 ms
Max      =   4046 ms
```

Desempenho poliglota

Melhoria de performance independente de linguagem ou plataforma

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

fabiogaluppo@acm.org

Engenheiro de Software BM&FBovespa

http://www.bmfbovespa.com.br

Microsoft MVP Visual C++

http://bit.ly/desempenho_poliglota