Institut für Theoretische Physik
Universität zu Köln

Prof. Dr. Simon Trebst

M.Sc. Carsten Bauer

Computerphysik
Vorlesung — Programmiertechniken 3

Sommersemester 2019

Website: [http://www.thp.uni-koeln.de/trebst/Lectures/2019-CompPhys.shtml (http://www.thp.uni-koeln.de/trebst/Lectures/2019-CompPhys.shtml)](http://www.thp.uni-koeln.de/trebst/Lectures/2019-CompPhys.shtml)

0. Erinnerung

Schleifen

In [6]:
```
for i in 1:10
    println(i)
end
```

```
1
2
3
4
5
6
7
8
9
10
```

In [7]:
```
i = 1
while i <= 10
    println(i)
    i += 1
end
```

```
1
2
3
4
5
6
7
8
9
10
```

Verzweigungen

In [9]:
```
v = 11

if v > 10
    println("Die Variable ist größer als 10.")
elseif v < 5
    println("Die Variable ist kleiner als 5.")
else
    println("Die Variable ist irgendwo dazwischen.")
end
```

```
Die Variable ist größer als 10.
```

Arrays

In [10]:
```
a = [5, 1, 8, 9, 3, 7]
```

Out[10]:
```
6-element Array{Int64,1}:
 5
 1
 8
 9
 3
 7
```

In [11]:
```
a[3]
```

Out[11]:
```
8
```

In [12]:
```
a[3] = 0
```

Out[12]:
```
0
```

In [13]:
```
a
```

Out[13]:
```
6-element Array{Int64,1}:
 5
 1
 0
 9
 3
 7
```

In [14]:
```
a[100]
```

```
BoundsError: attempt to access 6-element Array{Int64,1} at index [100]

Stacktrace:
 [1] getindex(::Array{Int64,1}, ::Int64) at .\array.jl:729
 [2] top-level scope at In[14]:1
```

In [15]:
```
length(a)
```

Out[15]:
```
6
```

In [1]:
```
?length
```

search: **length**

Out[1]:
```
length(collection) -> Integer
```

Return the number of elements in the collection.

Use `lastindex` (@ref) to get the last valid index of an indexable collection.

Examples

```
julia> length(1:5)
5

julia> length([1, 2, 3, 4])
4

julia> length([1 2; 3 4])
4
```

```
length(A::AbstractArray)
```

Return the number of elements in the array, defaults to `prod(size(A))` .

Examples

```
julia> length([1, 2, 3, 4])
4

julia> length([1 2; 3 4])
4
```

```
length(s::AbstractString) -> Int
length(s::AbstractString, i::Integer, j::Integer) -> Int
```

The number of characters in string `s` from indices `i` through `j` . This is computed as the number of code unit indices from `i` to `j` which are valid character indices. With only a single string argument, this computes the number of characters in the entire string. With `i` and `j` arguments it computes the number of indices between `i` and `j` inclusive that are valid indices in the string `s` . In addition to in-bounds values, `i` may take the out-of-bounds value `ncodeunits(s) + 1` and `j` may take the out-of-bounds value `0` .

See also: `isvalid` (@ref), `ncodeunits` (@ref), `lastindex` (@ref), `thisind` (@ref), `nextind` (@ref), `prevind` (@ref)

Examples

```
julia> length("jμΛIα")
5
```

In [17]:
```
size(a)
```

Out[17]:
```
(6,)
```

In [20]:
```
Y = rand(2,4,8)
```

Out[20]:
```
2×4×8 Array{Float64,3}:
[:, :, 1] =
 0.152435  0.979689  0.0074239  0.420801
 0.354892  0.150715  0.39802    0.564168

[:, :, 2] =
 0.618326  0.546292  0.561061  0.122623
 0.476645  0.131552  0.920055  0.241142

[:, :, 3] =
 0.952622  0.994377  0.783856  0.856578
 0.683555  0.737067  0.144527  0.793784

[:, :, 4] =
 0.828183  0.57101    0.995672  0.589804
 0.961323  0.0506661  0.272348  0.563574

[:, :, 5] =
 0.261386  0.466898  0.899692  0.030649
 0.753715  0.185559  0.48811   0.743139

[:, :, 6] =
 0.736457  0.263446  0.51642   0.0545521
 0.773255  0.647353  0.152371  0.354608

[:, :, 7] =
 0.670055  0.797061  0.298981   0.391179
 0.301899  0.83288   0.0433871  0.172567

[:, :, 8] =
 0.761125  0.995194  0.676004  0.459685
 0.234348  0.507026  0.111185  0.740016
```

In [21]:
```
size(Y)
```

Out[21]:
```
(2, 4, 8)
```

In [18]:
```
X = rand(2,2)
```

Out[18]:
```
2×2 Array{Float64,2}:
 0.409139  0.590969
 0.776841  0.329583
```

In [19]:
```
size(X)
```
Out[19]:
```
(2, 2)
```

In [22]:
```
M = rand(2,2)
```
Out[22]:
```
2×2 Array{Float64,2}:
 0.739909  0.0670009
 0.997385  0.708088
```

In [23]:
```
M * X # Matrix Multiplikation
```
Out[23]:
```
2×2 Array{Float64,2}:
 0.354775  0.459345
 0.958141  0.822797
```

In [24]:
```
M .* X # Elementweise Multiplikation
```
Out[24]:
```
2×2 Array{Float64,2}:
 0.302726  0.0395955
 0.77481   0.233374
```

Funktionen

In [25]:
```
quadrat(x) = x^2
```
Out[25]:
```
quadrat (generic function with 1 method)
```

In [26]:
```
quadrat(3)
```
Out[26]:
```
9
```

In [27]:
```
x = range(-2, stop=2, length=100)
```
Out[27]:
```
-2.0:0.04040404040404041:2.0
```

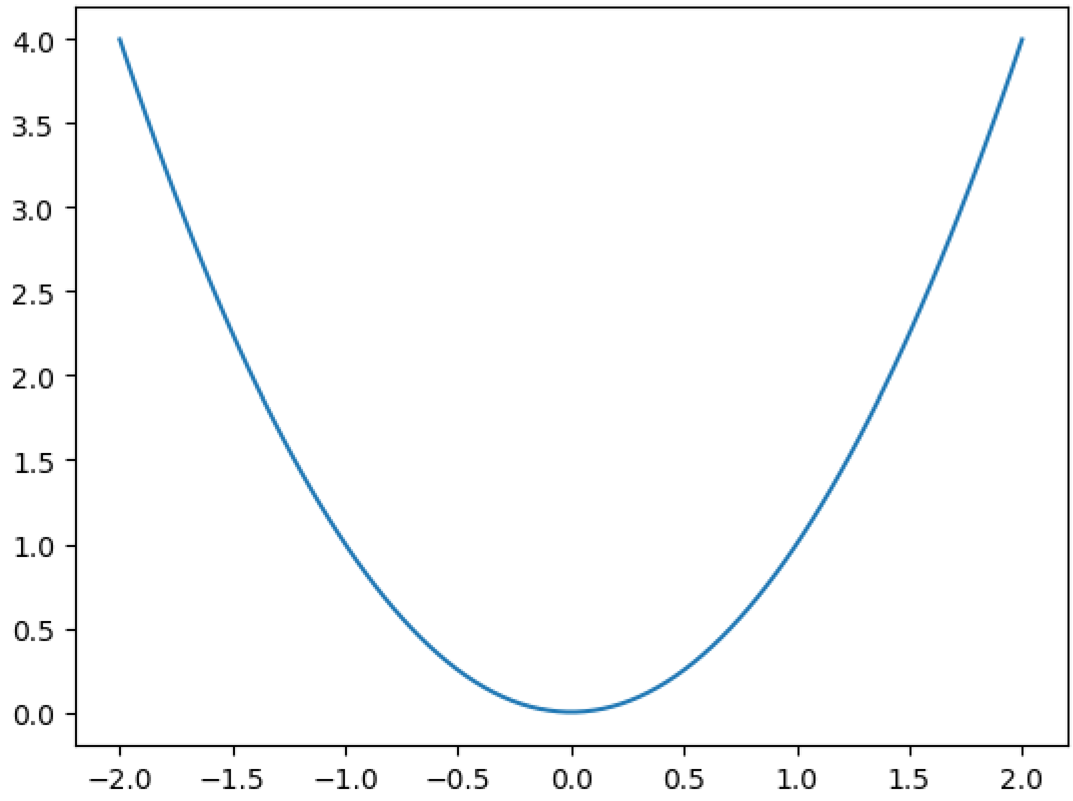
In [30]:
```
y = quadrat.(x)
```

Out[30]:
```
100-element Array{Float64,1}:
 4.0
 3.8400163248648096
 3.683297622691562
 3.5298438934802574
 3.379655137230895
 3.232731353943475
 3.089072543617998
 2.9486787062544635
 2.8115498418528717
 2.6776859504132235
 2.5470870319355168
 2.419753086419753
 2.295684113865932
 ⋮
 2.419753086419753
 2.5470870319355168
 2.6776859504132235
 2.8115498418528717
 2.9486787062544635
 3.089072543617998
 3.232731353943475
 3.379655137230895
 3.5298438934802574
 3.683297622691562
 3.8400163248648096
 4.0
```

Plots

In [29]:
```
using PyPlot
```

In [31]:
```
plot(x,y)
```



Out[31]:
```
1-element Array{PyCall.PyObject,1}:
 PyObject <matplotlib.lines.Line2D object at 0x0000000002284400>
```

In [ ]:

1. Funktionen (cont'd)

Mehrzeilige Funktionen

In [32]:
```
shortfunc(x) = 2x + 3
```

Out[32]:
```
shortfunc (generic function with 1 method)
```

In [33]:
```
shortfunc(2)
```

Out[33]:
```
7
```

In [34]:
```
function shortfunc2(x)
    2x + 3
end
```

Out[34]:
```
shortfunc2 (generic function with 1 method)
```

In [35]:
```
shortfunc2(2)
```

Out[35]: 7

In [36]:
```
function longfunc(x)
    a = 2x
    a + 3
end
```

Out[36]: longfunc (generic function with 1 method)

In [37]:
```
longfunc(2)
```

Out[37]: 7

Eine Funktion *returned* automatisch den Wert der letzten Zeile. Oft ist es besser explizit anzugeben, was zurückgegeben wird.

In [38]:
```
function longfunc2(x)
    a = 2x
    return a + 3
end
```

Out[38]: longfunc2 (generic function with 1 method)

In [39]:
```
longfunc2(2)
```

Out[39]: 7

In [40]:
```
function longfunc3(x)
    a = 2x
    return a + 3
    4 + 4
end
```

Out[40]: longfunc3 (generic function with 1 method)

In [41]:
```
longfunc3(2)
```

Out[41]: 7

In [ ]:
```
function machtwasanderes(x)
    if x > 10
        println("x > 10")
        return 1
    else
        println("x <= 10")
        return 0
    end
end
```

Mehrere Rückgabewerte

In [42]:
```
function longfunc_multiple(x)
    a = 2x
    b = a + 3
    return a, b
end
```
Out[42]: `longfunc_multiple (generic function with 1 method)`

In [43]:
```
longfunc_multiple(2)
```
Out[43]: `(4, 7)`

In [44]:
```
a, b = longfunc_multiple(2)
```
Out[44]: `(4, 7)`

In [45]:
```
a
```
Out[45]: `4`

In [46]:
```
b
```
Out[46]: `7`

In [47]:
```
c = longfunc_multiple(2)
```
Out[47]: `(4, 7)`

In [48]:
```
c[1]
```
Out[48]: `4`

In [49]:
```
c[2]
```
Out[49]: `7`

In [50]:
```
d, e = longfunc_multiple(3)
```
Out[50]: `(6, 9)`

Globale und lokale Variablen

Eine Funktion sollte *autonom* sein und nur mit den Eingabeparametern arbeiten.

In [51]:
```
function f(x)
    2x + 3
end
```
Out[51]: `f (generic function with 1 method)`

In [52]:
```
f(5)
```

Out[52]:
```
13
```

In [53]:
```
# Schlecht!!!!!!!!!!

x = 2

function f()
    2x + 3
end

f()
```

Out[53]:
```
7
```

Übung: Elemente in einem Array vertauschen

In [54]:
```
function swap(a, i, j)
    tmp = a[i]
    a[i] = a[j]
    a[j] = tmp
    return a
end
```

Out[54]:
```
swap (generic function with 1 method)
```

In [55]:
```
a = collect(1:10)
```

Out[55]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [56]:
```
swap(a, 6, 10)
```

Out[56]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
 10
  7
  8
  9
  6
```

In [57]:
```
a
```

Out[57]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
 10
  7
  8
  9
  6
```

In Julia gibt es die Konvention, dass Funktionen die mindestens eines ihrer Funktionsargumente modifizieren, mit einem Ausrufezeichen am Ende versehen werden.

In [58]:
```
function swap!(a, i, j)
    tmp = a[i]
    a[i] = a[j]
    a[j] = tmp
    return a
end
```

Out[58]:
```
swap! (generic function with 1 method)
```

In [59]:
```
swap!(a, 2,3)
```

Out[59]:
```
10-element Array{Int64,1}:
  1
  3
  2
  4
  5
 10
  7
  8
  9
  6
```

In [60]:
```
a
```

Out[60]:
```
10-element Array{Int64,1}:
  1
  3
  2
  4
  5
 10
  7
  8
  9
  6
```

In [61]:
```
issorted(a)
```

Out[61]:
```
false
```

In [62]:
```
sort(a)
```

Out[62]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [63]:
```
a
```

Out[63]:
```
10-element Array{Int64,1}:
  1
  3
  2
  4
  5
 10
  7
  8
  9
  6
```

In [64]:
```
sort!(a)
```

Out[64]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [65]:
```
a
```

Out[65]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [70]:
```
a = rand(2,2)
```

Out[70]:
```
2×2 Array{Float64,2}:
 0.080248  0.76882
 0.416823  0.0826709
```

In [71]:
```
b = a
```

Out[71]:
```
2×2 Array{Float64,2}:
 0.080248  0.76882
 0.416823  0.0826709
```

In [72]:
```
b[1] = 123
```

Out[72]:
```
123
```

In [73]:
```
b
```

Out[73]:
```
2×2 Array{Float64,2}:
 123.0       0.76882
   0.416823  0.0826709
```

In [74]:
```
a
```

Out[74]:
```
2×2 Array{Float64,2}:
 123.0       0.76882
   0.416823  0.0826709
```

In [66]:
```
function swap(b, i, j)
    a = copy(b)
    tmp = a[i]
    a[i] = a[j]
    a[j] = tmp
    return a
end
```

Out[66]:
```
swap (generic function with 1 method)
```

In [67]:
```
a
```

Out[67]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [68]:
```
swap(a, 3, 7)
```

Out[68]:
```
10-element Array{Int64,1}:
  1
  2
  7
  4
  5
  6
  3
  8
  9
 10
```

In [69]:
```
a
```

Out[69]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

2. Selbst sortieren: BubbleSort

"Größte Elemente steigen nacheinander ans Ende des Arrays auf"

In [78]:
```
collect(1:10) # start:ende
```

Out[78]:
```
10-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

In [79]:
```
collect(1:0.1:10) # start:schrittweite:ende
```

Out[79]:
```
91-element Array{Float64,1}:
  1.0
  1.1
  1.2
  1.3
  1.4
  1.5
  1.6
  1.7
  1.8
  1.9
  2.0
  2.1
  2.2
  ⋮
  8.9
  9.0
  9.1
  9.2
  9.3
  9.4
  9.5
  9.6
  9.7
  9.8
  9.9
 10.0
```

In [80]:
```
collect(10:-1:1)
```

Out[80]:
```
10-element Array{Int64,1}:
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

In [94]:
```julia
function bubblesort!(a)
    N = length(a)

    for rechts in N:-1:2
        for i in 1:(rechts-1)
            if a[i] > a[i+1] # wenn links größer als rechts
                swap!(a, i, i+1)
            end
        end
    end

    return a
end
```

Out[94]:
```
bubblesort! (generic function with 1 method)
```

In [82]:
```julia
a = rand(1:10, 10)
```

Out[82]:
```
10-element Array{Int64,1}:
  2
  3
  2
  6
  8
 10
  2
  3
  4
 10
```

In [84]:
```julia
rand(["hallo", "köln", "wasauchimmer"], 7)
```

Out[84]:
```
7-element Array{String,1}:
 "köln"
 "wasauchimmer"
 "hallo"
 "wasauchimmer"
 "hallo"
 "wasauchimmer"
 "köln"
```

In [95]:
```julia
a = rand(1:10, 10)
```

Out[95]:
```
10-element Array{Int64,1}:
  9
  8
  1
  4
 10
  4
  5
  3
 10
  7
```

In [96]:
```
bubblesort!(a)
```

Out[96]:
```
10-element Array{Int64,1}:
  1
  3
  4
  4
  5
  7
  8
  9
 10
 10
```

In [97]:
```
a
```

Out[97]:
```
10-element Array{Int64,1}:
  1
  3
  4
  4
  5
  7
  8
  9
 10
 10
```

Visualisierung

In [98]:
```
using PyPlot, Random

function show_bubble_schritt(n)
    a = shuffle(1:n)

    pygui(true)
    fig = figure()
    title("Bubble-Schritt")
    for rechts = length(a):-1:length(a)
        # bubble-Schritt
        for i in 1:rechts-1
            if a[i]>a[i+1]
                swap!(a, i, i+1)
            end
            fig.clear()
            bar(1:length(a), a)
            m, mind = findmax(a)
            bar(mind, m, color="red")
            sleep(0.001)
        end
    end
    pygui(false)
    nothing
end

function show_bubble_sort(n)
    a = shuffle(1:n)

    pygui(true)
    fig = figure()
    title("Bubble-Sort")
    for rechts = length(a):-1:2
        # bubble-Schritt
        for i in 1:rechts-1
            if a[i]>a[i+1]
                swap!(a, i, i+1)
            end
        end
        fig.clear()
        bar(1:length(a), a)
        m, mind = findmax(a[1:rechts])
        bar(mind, m, color="red")
        sleep(0.001)
    end
    pygui(false)
    nothing
end
```

Out[98]:
```
show_bubble_sort (generic function with 1 method)
```

In [100]:
```
show_bubble_schritt(40)
```

In [101]:
```
show_bubble_sort(40)
```

3. Timing und Komplexität

In [6]:
```
b = rand(50000);
```

In [7]:
```
@time bubblesort!(b);
```

```
  4.237893 seconds (26.33 k allocations: 1.333 MiB, 0.08% gc time)
```

In [8]:
```
function benchmark_bubblesort()
    number_count = [0.0]
    elapsed_time = [0.0]

    for i in 1:16
        b = rand(2^i)
        t = @elapsed bubblesort!(b)
        println(2^i, "\t", t)
        push!(number_count, 2^i)
        push!(elapsed_time, t)
    end

    return number_count, elapsed_time
end
```

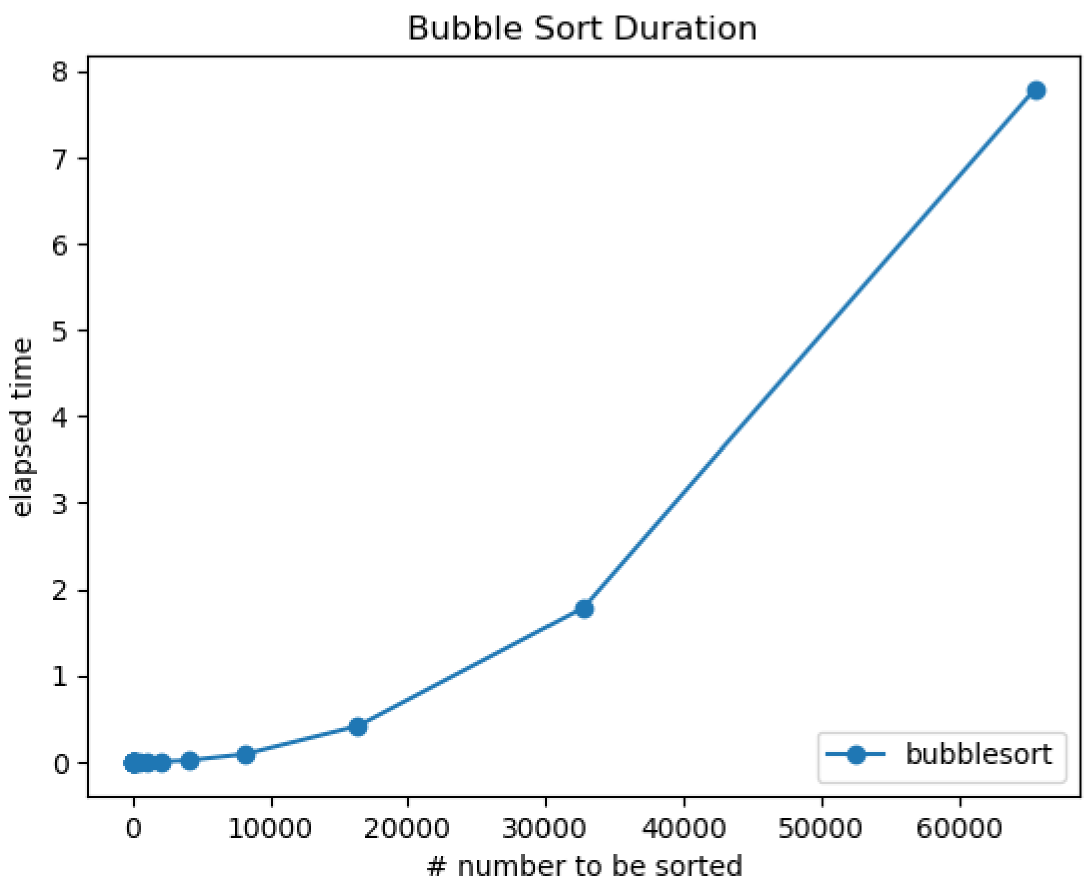
Out[8]:
```
benchmark_bubblesort (generic function with 1 method)
```

In [9]:
```
number_count, elapsed_time = benchmark_bubblesort();
```

```
2       4.99e-7
4       4.0e-7
8       2.99e-7
16      6.0e-7
32      1.599e-6
64      5.899e-6
128     1.67e-5
256     0.000196
512     0.0002221
1024    0.0008542
2048    0.003413601
4096    0.0172408
8192    0.090724101
16384   0.416059101
32768   1.7874153
65536   7.7880827
```

In [10]:
```
using PyPlot

plot(number_count, elapsed_time, marker="o", label="bubblesort");
legend(loc=4);
xlabel("# number to be sorted");
ylabel("elapsed time");
title("Bubble Sort Duration")
```

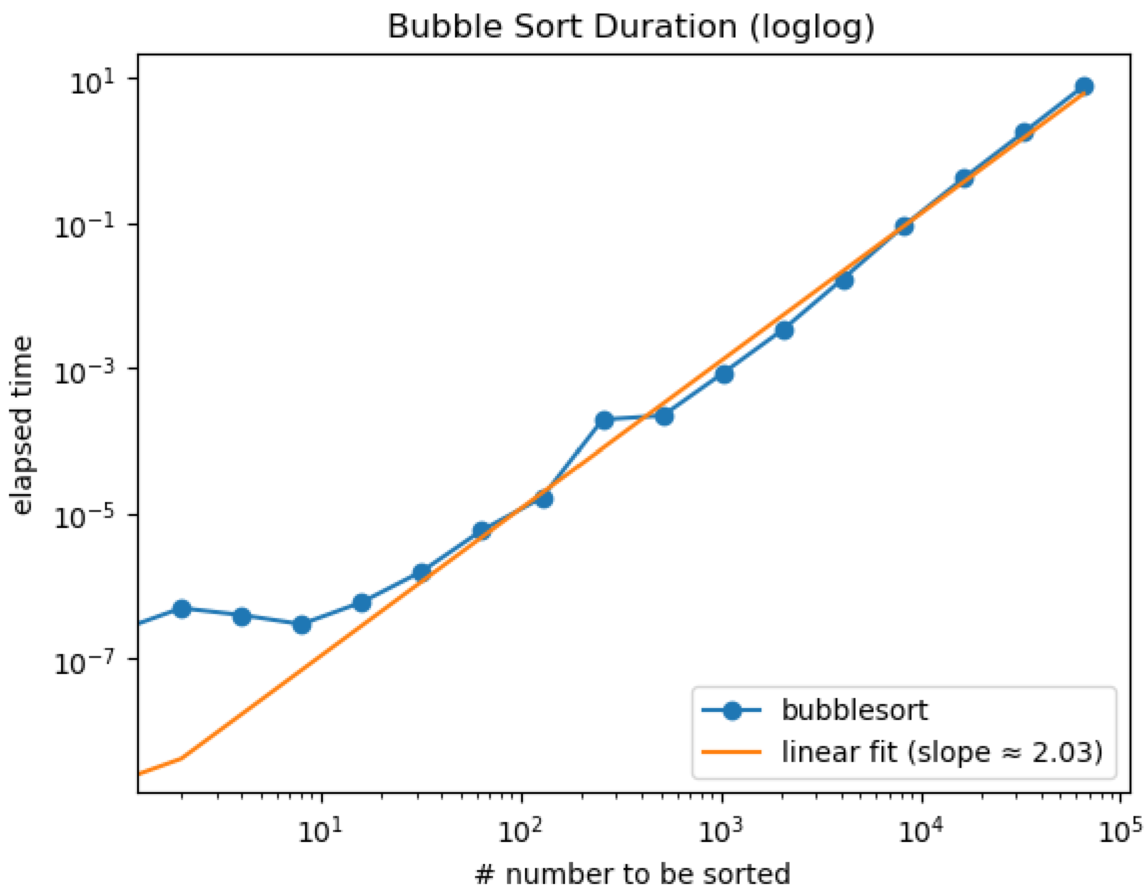


Out[10]:
```
PyObject Text(0.5, 1.0, 'Bubble Sort Duration')
```

In [11]:
```
using PyPlot, Polynomials

# fit straight line in loglog space (ignoring first couple of datapoints)
p = polyfit(log.(number_count[7:end]), log.(elapsed_time[7:end]), 1)
m = p.a[2]

plot(number_count, elapsed_time, marker="o", label="bubblesort");
plot(number_count, exp.(p.(log.(number_count))), label="linear fit (slope ≈ $(rou
legend(loc=4);
xscale("log")
yscale("log")
xlabel("# number to be sorted");
ylabel("elapsed time");
title("Bubble Sort Duration (loglog)");
```



Komplexität (asymptotisches Verhalten): BubbleSort $\in \mathcal{O}(n^2)$

O-Notation: [https://de.wikipedia.org/wiki/Landau-Symbole#Beispiele_und_Notation (https://de.wikipedia.org/wiki/Landau-Symbole#Beispiele_und_Notation)](https://de.wikipedia.org/wiki/Landau-Symbole#Beispiele_und_Notation)

Vergleich mit Julias `sort!`

In [12]:
```
function benchmark_juliasort()
    number_count = [0.0]
    elapsed_time = [0.0]

    for i in 1:16
        b = rand(2^i)
        t = @elapsed sort!(b)
        println(2^i, "\t", t)
        push!(number_count, 2^i)
        push!(elapsed_time, t)
    end

    return number_count, elapsed_time
end
```

Out[12]:
```
benchmark_juliasort (generic function with 1 method)
```

In [13]:
```
number_count, elapsed_time = benchmark_juliasort();
```

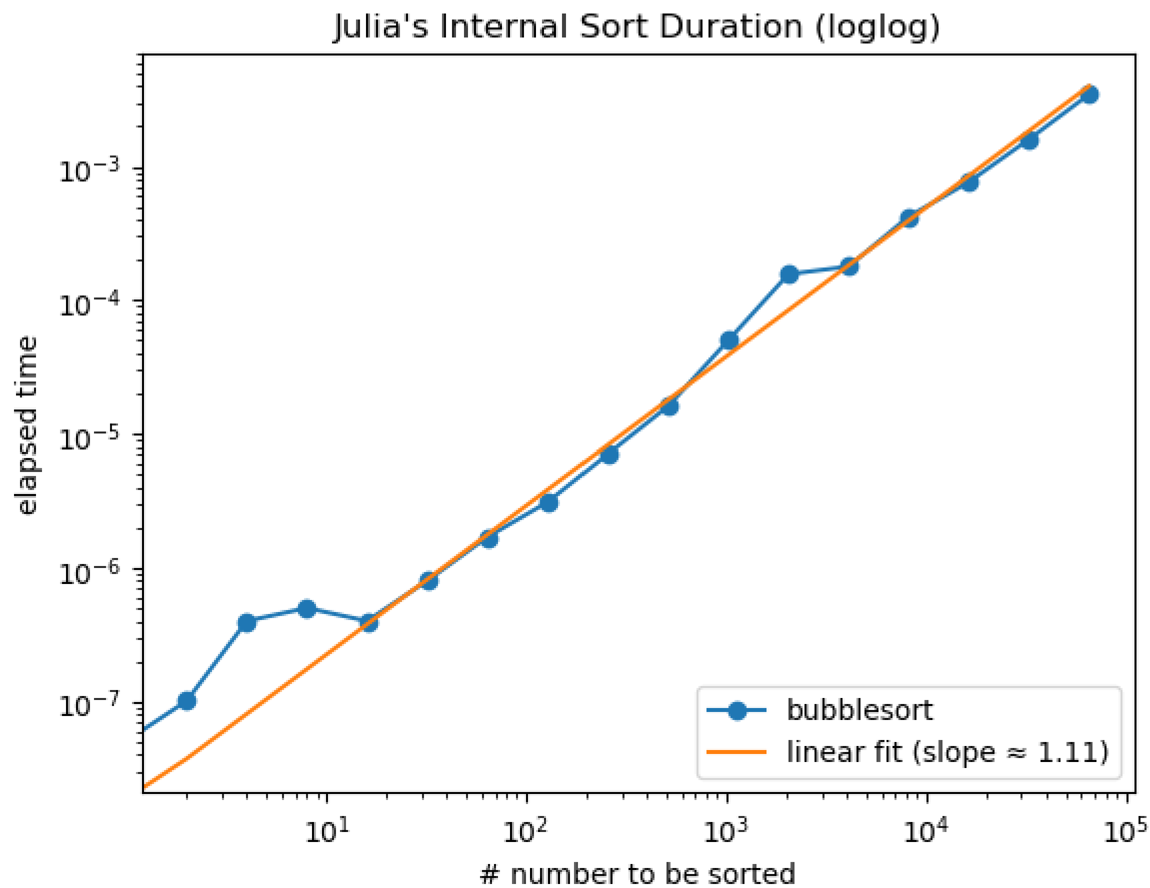
```
2       1.01e-7
4       4.0e-7
8       5.01e-7
16      4.0e-7
32      8.0e-7
64      1.699e-6
128     3.101e-6
256     7.099e-6
512     1.61e-5
1024    5.01e-5
2048    0.000156601
4096    0.000178901
8192    0.000422901
16384   0.000766699
32768   0.0016012
65536   0.003472799
```

In [14]:

```
using PyPlot, Polynomials

# fit straight line in loglog space (ignoring first couple of datapoints)
p = polyfit(log.(number_count[7:end]), log.(elapsed_time[7:end]), 1)
m = p.a[2]

plot(number_count, elapsed_time, marker="o", label="bubblesort");
plot(number_count, exp.(p.(log.(number_count))), label="linear fit (slope ≈ $(rou
legend(loc=4);
xscale("log")
yscale("log")
xlabel("# number to be sorted");
ylabel("elapsed time");
title("Julia's Internal Sort Duration (loglog)");
```



Übersicht der Komplexität verschiedener Sortierverfahren: [https://de.wikipedia.org/wiki/Sortierverfahren (https://de.wikipedia.org/wiki/Sortierverfahren)](https://de.wikipedia.org/wiki/Sortierverfahren)

Randnotiz: Die Macros `@time` und `@elapsed` sind hilfreich, sollten jedoch meistens vermieden werden, da Nebeneffekte das Messergebnis verzerren können. Führen Sie beispielsweise `@time sort(rand(1000));` zweimal aus und beobachten Sie, wie sich das Ergebnis ändert.

Es ist stattdessen empfehlenswert auf `@btime` und `@belapsed` aus dem Paket [BenchmarkTools.jl (https://github.com/JuliaCI/BenchmarkTools.jl)](https://github.com/JuliaCI/BenchmarkTools.jl) verwenden.

In [15]:
```
@time sort(rand(1000));
```

```
0.004349 seconds (6.02 k allocations: 336.401 KiB)
```

In [16]:
```
@time sort(rand(1000));
```

```
0.000041 seconds (6 allocations: 16.031 KiB)
```

In [ ]:
```
] add BenchmarkTools
```

In [17]:
```
using BenchmarkTools
```

In [18]:
```
@btime sort(rand(1000));
```

```
30.499 μs (2 allocations: 15.88 KiB)
```

In [19]:
```
@btime sort(rand(1000));
```

```
31.800 μs (2 allocations: 15.88 KiB)
```