

In Julia, there are many different number types. All are subtypes of the abstract supertype `Number`.

```
Any[Complex, Real]
```

```
· subtypes(Number)
```

Subtypes can also be abstract, concrete, or parametric.

```
Bool[false, true]
```

```
· isabstracttype.(subtypes(Number))
```

```
Bool[false, false]
```

```
· isconcretetype.(subtypes(Number))
```

We can conclude that `Complex` is a parametric type while `Real` is an abstract type. Usually, but not always, abstract types have subtypes.

```
Any[AbstractFloat, AbstractIrrational, Integer, Rational]
```

```
· subtypes(Real)
```

```
Any[BigFloat, Float16, Float32, Float64]
```

```
· subtypes(AbstractFloat)
```

```
Any[Bool, Signed, Unsigned]
```

```
· subtypes(Integer)
```

```
Any[BigInt, Int128, Int16, Int32, Int64, Int8]
```

```
· subtypes(Signed)
```

```
Any[UInt128, UInt16, UInt32, UInt64, UInt8]
```

```
· subtypes(Unsigned)
```

```
Complex
```

```
· Complex
```

```
AbstractIrrational
```

AbstractIrrational

Rational

Rational

Mathematically, it's not inconceivable that would wish to work with a rational number type. There are a few problems with this when it comes to arithmetic on a computer. The main issue is that of overflow and underflow. Take, for example, the Hilbert matrix:

$$H_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}.$$

This matrix is easy enough to create. Julia even allows us to find its inverse with rationals. The catch is that if n is too large, this seemingly innocent-looking matrix's inverse is no longer representable as a ratio of two 64-bit integers. It could be done with arbitrary precision, but this comes at a significant computational expense.

H = #1 (generic function with 1 method)

```
H = n -> inv.((1:n) .+ (1:n)' .- 1)
```

```
5×5 Array{Rational{Int64},2}:
 1//1  1//2  1//3  1//4  1//5
 1//2  1//3  1//4  1//5  1//6
 1//3  1//4  1//5  1//6  1//7
 1//4  1//5  1//6  1//7  1//8
 1//5  1//6  1//7  1//8  1//9
```

```
H(5//1)
```

```
5×5 Array{Rational{Int64},2}:
 25//1  -300//1  1050//1  -1400//1  630//1
 -300//1  4800//1  -18900//1  26880//1  -12600//1
 1050//1  -18900//1  79380//1  -117600//1  56700//1
 -1400//1  26880//1  -117600//1  179200//1  -88200//1
 630//1  -12600//1  56700//1  -88200//1  44100//1
```

```
inv(H(5//1))
```

```
5×5 Array{Rational{Int64},2}:
 1//1  0//1  0//1  0//1  0//1
 0//1  1//1  0//1  0//1  0//1
 0//1  0//1  1//1  0//1  0//1
 0//1  0//1  0//1  1//1  0//1
 0//1  0//1  0//1  0//1  1//1
```

```
inv(H(5//1))*H(5//1)
```

```
15×15 Array{Rational{Int64},2}:
```

```
1//1  1//2  1//3  1//4  1//5  1//6  ...  1//11  1//12  1//13  1//14  1//15
1//2  1//3  1//4  1//5  1//6  1//7  ...  1//12  1//13  1//14  1//15  1//16
1//3  1//4  1//5  1//6  1//7  1//8  ...  1//13  1//14  1//15  1//16  1//17
1//4  1//5  1//6  1//7  1//8  1//9  ...  1//14  1//15  1//16  1//17  1//18
1//5  1//6  1//7  1//8  1//9  1//10 ...  1//15  1//16  1//17  1//18  1//19
1//6  1//7  1//8  1//9  1//10 1//11 ...  1//16  1//17  1//18  1//19  1//20
1//7  1//8  1//9  1//10 1//11 1//12 ...  1//17  1//18  1//19  1//20  1//21
⋮
1//10 1//11 1//12 1//13 1//14 1//15 ...  1//20 1//21 1//22 1//23 1//24
1//11 1//12 1//13 1//14 1//15 1//16 ...  1//21 1//22 1//23 1//24 1//25
1//12 1//13 1//14 1//15 1//16 1//17 ...  1//22 1//23 1//24 1//25 1//26
1//13 1//14 1//15 1//16 1//17 1//18 ...  1//23 1//24 1//25 1//26 1//27
1//14 1//15 1//16 1//17 1//18 1//19 ...  1//24 1//25 1//26 1//27 1//28
1//15 1//16 1//17 1//18 1//19 1//20 ...  1//25 1//26 1//27 1//28 1//29
```

```
H(15//1)
```

OverflowError: 8855 * 1176346566046080 overflowed for type Int64

1. **throw_overflowerr_binaryop**(::Symbol, ::Int64, ::Int64) @ *checked.jl:154*
2. **checked_mul** @ *checked.jl:288* [inlined]
3. **//**(::Rational{Int64}, ::Rational{Int64}) @ *rational.jl:74*
4. **/** @ *rational.jl:320* [inlined]
5. **** @ *operators.jl:574* [inlined]
6. **naivesub!**
(::LinearAlgebra.UpperTriangular{Rational{Int64},Array{Rational{Int64},2}},
::Array{Rational{Int64},1}, ::Array{Rational{Int64},1}) @ *triangular.jl:1332*
7. **naivesub!** @ *triangular.jl:1325* [inlined]
8. **ldiv!** @ *bidiag.jl:761* [inlined]
9. **ldiv!**(::LinearAlgebra.UpperTriangular{Rational{Int64},Array{Rational{Int64},2}},
::Array{Rational{Int64},2}) @ *bidiag.jl:774*
10. **ldiv!**(::LinearAlgebra.LU{Rational{Int64},Array{Rational{Int64},2}},
::Array{Rational{Int64},2}) @ *lu.jl:396*
11. **ldiv!**(::Array{Rational{Int64},2},
::LinearAlgebra.LU{Rational{Int64},Array{Rational{Int64},2}},
::Array{Rational{Int64},2}) @ *factorization.jl:139*
12. **inv!**(::LinearAlgebra.LU{Rational{Int64},Array{Rational{Int64},2}}) @ *lu.jl:477*
13. **inv**(::Array{Rational{Int64},2}) @ *dense.jl:781*
14. **top-level scope** @ **Local: 1**

```
inv(H(15//1))
```

This is one reason we tend to use floating-point types and arithmetic in numerical analysis.

H(15)

$$\text{inv}(H(15))$$

```
bitstring(1.0)
```

$$2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

Page 4 of 6

Page 5 of 6

```
colorbitstring(1.0+ $\epsilon$ )
```