Package ‘Zseq’

September 7, 2022

Type Package

Title Integer Sequence Generator

Version 0.2.1

Description Generates well-known integer sequences. 'gmp' package is adopted for computing with arbitrarily large numbers. Every function has hyperlink to its corresponding item in OEIS (The On-Line Encyclopedia of Integer Sequences) in the function help page. For interested readers, see Sloane and Plouffe (1995, ISBN:978-0125586306).

License MIT + file LICENSE

Imports gmp, utils

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>)

Maintainer Kisung You <kisungyou@outlook.com>

Repository CRAN

Date/Publication 2022-09-07 07:50:18 UTC

R topics documented:

`Zseq-package` *Zseq : Integer Sequence Generator*

Description

The world of integer sequence has long history, which has been accumulated in The On-Line Encyclopedia of Integer Sequences. Even though `R` is not a first pick for many number theorists, we introduce our package to enrich the `R` ecosystem as well as provide pedagogical toolset. We adopted **gmp** for flexible large number computations in that users can easily experience large number sequences on a non-exclusive generic computing platform.

Abundant — *Abundant numbers*

Description

Under OEIS A005101, an *abundant* number is a number whose proper divisors sum up to the extent greater than the number itself. First 6 abundant numbers are 12, 18, 20, 24, 30, 36.

Usage

```
Abundant(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

See Also

Deficient, Perfect

Examples

```
## generate first 30 Abundant numbers and print it
print(Abundant(30))
```

Achilles — *Achilles numbers*

Description

Under OEIS A052486, an *Achilles* number is a number that is *powerful* but *not perfect*. First 6 Achilles numbers are 72, 108, 200, 288, 392, 432.

Usage

```
Achilles(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 3 Achilles numbers and print
print(Achilles(3))
```

Bell *Bell numbers*

Description

Under OEIS A000110, the *n*th *Bell* number is the number of ways to partition a set of n labeled elements, where the first 6 entries are 1, 1, 2, 5, 15, 52.

Usage

```
Bell(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Bell numbers and print
print(Bell(30))
```

Carmichael	*Carmichael numbers*

Description

Under OEIS A002997, a *Carmichael* number is a composite number n such that

$$b^{n-1} = 1 (mod n)$$

for all integers b which are relatively prime to n. First 6 Carmichael numbers are 561, 1105, 1729, 2465, 2821, 6601.

Usage

```
Carmichael(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 3 Carmichael numbers
print(Carmichael(3))
```

Catalan	*Catalan numbers*

Description

Under OEIS A000108, the *n*th *Catalan* number is given as

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

where the first 6 entries are 1, 1, 2, 5, 14, 42 with $n \geq 0$.

Usage

```
Catalan(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Catalan numbers
print(Catalan(30))
```

Composite — *Composite numbers*

Description

Under OEIS A002808, a *composite* number is a positive integer that can be represented as multiplication of two smaller positive integers. The first 6 composite numbers are 4, 6, 8, 9, 10, 12.

Usage

```
Composite(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Composite numbers
print(Composite(30))
```

`Deficient` *Deficient numbers*

Description

Under OEIS A005100, a *deficient* number is a number whose proper divisors sum up to the extent smaller than the number itself. First 6 deficient numbers are 1, 2, 3, 4, 5, 7

Usage

```
Deficient(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Abundant`, `Perfect`

Examples

```
## generate first 30 Deficient numbers
print(Deficient(30))
```

`Equidigital` *Equidigital numbers*

Description

Under OEIS A046758, an *Equidigital* number has equal digits as the number of digits in its prime factorization including exponents. First 6 Equidigital numbers are 1, 2, 3, 5, 7, 10. Though it doesn't matter which base we use, here we adopt only a base of 10.

Usage

```
Equidigital(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Frugal`, `Extravagant`

Examples

```
## generate first 20 Equidigital numbers
print(Equidigital(20))
```

`Evil` *Evil numbers*

Description

Under OEIS A001969, an *Evil* number has an even number of 1's in its binary expansion. First 6 Evil numbers are 0, 3, 5, 6, 9, 10.

Usage

```
Evil(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Odious`

Examples

```
## generate first 20 Evil numbers
print(Evil(20))
```

`Extravagant` *Extravagant numbers*

Description

Under OEIS A046760, an *Extravagant* number has less digits than the number of digits in its prime factorization including exponents. First 6 Extravagant numbers are 4, 6, 8, 9, 12, 18. Though it doesn't matter which base we use, here we adopt only a base of 10.

Usage

```
Extravagant(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Frugal`, `Equidigital`

Examples

```
## generate first 20 Extravagant numbers
print(Extravagant(20))
```

`Factorial` *Factorial numbers*

Description

Under OEIS A000142, a *Factorial* is the product of all positive integers smaller than or equal to the number. First 6 such numbers are 1, 1, 2, 6, 24, 120

Usage

```
Factorial(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 10 Factorials
print(Factorial(10))
```

`Factorial.Alternating` *Alternating Factorial numbers*

Description

Under OEIS A005165, an *Alternating Factorial* is the absolute value of the alternating sum of the first `n` factorials of positive integers. First 6 such numbers are 0, 1, 1, 5, 19, 101.

Usage

```
Factorial.Alternating(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Factorial`

Examples

```
## generate first 5 Alternating Factorial numbers
print(Factorial.Alternating(5))
```

Factorial.Double *Double Factorial numbers*

Description

Under OEIS A000165 and A001147, a *Double Factorial* is the factorial of numbers with same parity. For example, if $n = 5$, then $n!! = 5 * 3 * 1$.

Usage

```
Factorial.Double(n, gmp = TRUE, odd = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.
odd	a logical; TRUE for double factorial of odd numbers, FALSE for even numbers.

Value

a vector of length n containing first entries from the sequence.

See Also

Factorial

Examples

```
## generate first 10 double factorials
print(Factorial.Double(10))
```

Fibonacci *Fibonacci numbers*

Description

Under OEIS A000045, the *n*th *Fibonnaci* number is given as

$$F_n = F_{n-1} + F_{n-2}$$

where the first 6 entries are 0, 1, 1, 2, 3, 5 with $n \geq 0$.

Usage

```
Fibonacci(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise (default: TRUE).

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Fibonacci numbers
print(Fibonacci(30))
```

Frugal — *Frugal numbers*

Description

Under OEIS A046759, a *Frugal* number has more digits than the number of digits in its prime factorization including exponents. First 6 Frugal numbers are 125, 128, 243, 256, 343, 512. Though it doesn't matter which base we use, here we adopt only a base of 10.

Usage

```
Frugal(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

See Also

Extravagant, Equidigital

Examples

```
## generate first 5 Frugal numbers
print(Frugal(5))
```

Happy	*Happy numbers*

Description

Under OEIS A007770, a *Happy* number is defined by the process that starts from arbitrary positive integer and replaces the number by the sum of the squares of each digit until the number is 1. First 6 Happy numbers are 1, 7, 10, 13, 19, 23.

Usage

```
Happy(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 happy numbers
print(Happy(30))
```

Juggler	*Juggler sequence*

Description

Under OEIS A094683, a *Juggler* sequence is an integer-valued sequence that starts with a nonnegative number iteratively follows that $J_{k+1} = floor(J_k^{1/2})$ if J_k is even, or $J_{k+1} = floor(J_k^{3/2})$ if odd. No first 6 terms are given since it all depends on the starting value.

Usage

```
Juggler(start, gmp = TRUE)
```

Arguments

`start`	the starting nonnegative integer.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector recording the sequence of unknown length a priori.

Examples

```
## let's start from 9 and show the sequence
print(Juggler(9))
```

`Juggler.Largest` *Largest value for Juggler sequence*

Description

Under OEIS A094716, the *Largest value for Juggler sequence* is the largest value in trajectory of a sequence that starts from `n`. First 6 terms are 0, 1, 2, 36, 4, 36 that `n` starting from 0 is conventional choice.

Usage

```
Juggler.Largest(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Juggler`

Examples

```
## generate first 10 numbers of largest values for Juggler sequences
print(Juggler.Largest(10))
```

`Juggler.Nsteps` *Number of steps for Juggler sequence*

Description

Under OEIS A007320, a *Number of steps for Juggler sequence* literally counts the number of steps required for a sequence that starts from `n`. First 6 terms are 0, 1, 6, 2, 5, 2 that `n` starting from 0 is conventional choice. Note that when it counts *number of steps*, not the length of the sequence including the last 1.

Usage

```
Juggler.Nsteps(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Juggler`

Examples

```
## generate first 10 numbers of steps for Juggler sequences
print(Juggler.Nsteps(10))
```

`Lucas` *Lucas numbers*

Description

Under OEIS A000032, the *n*th *Lucas* number is given as

$$F_n = F_{n-1} + F_{n-2}$$

where the first 6 entries are 2, 1, 3, 4, 7, 11.

Usage

```
Lucas(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

See Also

`Fibonacci`

Examples

```
## generate first 30 Lucas numbers
print(Lucas(30))
```

`Motzkin` *Motzkin numbers*

Description

Under OEIS A001006, a *Motzkin* number for a given n is the number of ways for drawing non-intersecting chords among n points on a circle, where the first 7 entries are 1, 1, 2, 4, 9, 21, 51.

Usage

```
Motzkin(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 30 Motzkin numbers
print(Motzkin(30))
```

Odious *Odious numbers*

Description

Under OEIS A000069, an *Odious* number has an odd number of 1's in its binary expansion. First 6 Odious numbers are 1, 2, 4, 7, 8, 11.

Usage

```
Odious(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

See Also

Evil

Examples

```
## generate first 20 Odious numbers
print(Odious(20))
```

Padovan *Padovan numbers*

Description

Under OEIS A000931, the *n*th *Padovan* number is given as

$$F_n = F_{n-2} + F_{n-3}$$

where the first 6 entries are 1, 0, 0, 1, 0, 1.

Usage

```
Padovan(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 30 Padovan numbers
print(Padovan(30))
```

`Palindromic` *Palindromic numbers*

Description

Under OEIS A002113, a *Palindromic* number is a number that remains the same when its digits are reversed. First 6 Palindromic numbers in decimal are 0, 1, 2, 3, 4, 5. This function supports various base by specifying the parameter `base` but returns are still represented in decimal.

Usage

```
Palindromic(n, base = 10, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`base`	choice of base.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 30 palindromic number in decimal
print(Palindromic(30))
```

Palindromic.Squares *Palindromic squares*

Description

Under OEIS A002779, a *Palindromic square* is a number that is both Palindromic and Square. First 6 such numbers are 0, 1, 4, 9, 121, 484. It uses only the base 10 decimals.

Usage

```
Palindromic.Squares(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 10 palindromic squares
print(Palindromic.Squares(10))
```

Perfect *Perfect numbers*

Description

Under OEIS A000396, a *Perfect* number is a number whose proper divisors sum up to the extent equal to the number itself. First 6 abundant numbers are 6, 28, 496, 8128, 33550336, 8589869056.

Usage

```
Perfect(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length n containing first entries from the sequence.

See Also

Deficient, Abundant

Examples

```
## Not run:
## generate first 7 Perfect numbers
print(Perfect(10))

## End(Not run)
```

Perrin *Perrin numbers*

Description

Under OEIS A001608, the *n*th *Perrin* number is given as

$$F_n = F_{n-2} + F_{n-3}$$

where the first 6 entries are 3, 0, 2, 3, 2, 5.

Usage

```
Perrin(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Perrin numbers
print(Perrin(30))
```

Powerful *Powerful numbers*

Description

Under OEIS A001694, a *Powerful* number is a positive integer such that for every prime p dividing the number, p^2 also divides the number. First 6 powerful numbers are 1, 4, 8, 9, 16, 25.

Usage

```
Powerful(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 20 Powerful numbers
print(Powerful(20))
```

Prime *Prime numbers*

Description

Under OEIS A000040, a *Prime* number is a natural number with no positive divisors other than 1 and itself. First 6 prime numbers are 2, 3, 5, 7, 11, 13.

Usage

```
Prime(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Regular numbers
print(Prime(30))
```

`Regular` *Regular numbers*

Description

Under OEIS A051037, a *Regular* number - also known as 5-smooth - is a positive integer that even divide powers of 60, or equivalently, whose prime divisors are only 2,3, and 5. First 6 Regular numbers are 1, 2, 3, 4, 5, 6.

Usage

```
Regular(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 20 Regular numbers
print(Regular(20))
```

`Square` *Square numbers*

Description

Under OEIS A000290, a *Square* number is

$$A_n = n^2$$

for $n \geq 0$. First 6 Square numbers are 0, 1, 4, 9, 16, 25.

Usage

```
Square(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 20 Square numbers
print(Square(20))
```

Squarefree *Squarefree numbers*

Description

Under OEIS A005117, a *Squarefree* number is a number that are not divisible by a square of a smaller integer greater than 1. First 6 Squarefree numbers are 1, 2, 3, 5, 6, 7.

Usage

```
Squarefree(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Squarefree numbers
print(Squarefree(30))
```

`Telephone` *Telephone numbers*

Description

Under OEIS A000085, a *Telephone* number - also known as *Involution* number - is counting the number of connection patterns in a telephone system with `n` subscribers, or in a more mathematical term, the number of self-inverse permutations on `n` letters. First 6 Telephone numbers are 1, 1, 2, 4, 10, 26,

Usage

```
Telephone(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 20 Regular numbers
print(Telephone(20))
```

`Thabit` *Thabit numbers*

Description

Under OEIS A055010, the *n*th *Thabit* number is given as

$$A_n = 3 * 2^{n-1} - 1$$

where the first 6 entries are 0, 2, 5, 11, 23, 47 with $A_0 = 0$.

Usage

```
Thabit(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 30 Thabit numbers
print(Thabit(30))
```

Triangular   *Triangular numbers*

Description

Under OEIS A000217, a *Triangular* number counts objects arranged in an equilateral triangle. First 6 Triangular numbers are 0, 1, 3, 6, 10, 15.

Usage

```
Triangular(n, gmp = TRUE)
```

Arguments

n	the number of first n entries from the sequence.
gmp	a logical; TRUE to use large number representation, FALSE otherwise.

Value

a vector of length n containing first entries from the sequence.

Examples

```
## generate first 20 Triangular numbers
print(Triangular(20))
```

`Unusual` *Unusual numbers*

Description

Under OEIS A064052, an *Unusual* number is a natural number whose largest prime factor is strictly greater than square root of the number. First 6 Unusual numbers are 2, 3, 5, 6, 7, 10.

Usage

```
Unusual(n, gmp = TRUE)
```

Arguments

`n`	the number of first `n` entries from the sequence.
`gmp`	a logical; `TRUE` to use large number representation, `FALSE` otherwise.

Value

a vector of length `n` containing first entries from the sequence.

Examples

```
## generate first 20 Unusual numbers
print(Unusual(20))
```

Index