

[Discover Packages](#) > [Standard library](#) > [sort](#) 





sort

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [2](#) |Imported by: [276,297](#)

Details



- ✓ Valid [go.mod](#) file 
- ✓ Redistributable license 
- ✓ Tagged version 
- ✓ Stable version 

[Learn more](#)

Repository

[cs.opensource.google/go/go](#)

Links

 [Report a Vulnerability](#) Documentation 

<> Documentation

Overview

Package sort provides primitives for sorting slices and user-defined collections.

[▶ Example](#)[▶ Example \(SortKeys\)](#)[▶ Example \(SortMultiKeys\)](#)[▶ Example \(SortWrapper\)](#)

Index

```
func Find(n int, cmp func(int) int) (i int, found bool)
```

```
func Float64s(x []float64)
```

```
func Float64sAreSorted(x []float64) bool
```

```
func Ints(x []int)
```

```
func IntsAreSorted(x []int) bool
```

```
func IsSorted(data Interface) bool
```

```
func Search(n int, f func(int) bool) int
```

```
func SearchFloat64s(a []float64, x float64) int
```

```
func SearchInts(a []int, x int) int
func SearchStrings(a []string, x string) int
func Slice(x any, less func(i, j int) bool)
func SliceIsSorted(x any, less func(i, j int) bool) bool
func SliceStable(x any, less func(i, j int) bool)
func Sort(data Interface)
func Stable(data Interface)
func Strings(x []string)
func StringsAreSorted(x []string) bool
type Float64Slice
    func (x Float64Slice) Len() int
    func (x Float64Slice) Less(i, j int) bool
    func (p Float64Slice) Search(x float64) int
    func (x Float64Slice) Sort()
    func (x Float64Slice) Swap(i, j int)
type IntSlice
    func (x IntSlice) Len() int
    func (x IntSlice) Less(i, j int) bool
    func (p IntSlice) Search(x int) int
    func (x IntSlice) Sort()
    func (x IntSlice) Swap(i, j int)
type Interface
    func Reverse(data Interface) Interface
type StringSlice
    func (x StringSlice) Len() int
    func (x StringSlice) Less(i, j int) bool
    func (p StringSlice) Search(x string) int
    func (x StringSlice) Sort()
    func (x StringSlice) Swap(i, j int)
```

Examples

Package

Package (SortKeys)

Package (SortMultiKeys)

Package (SortWrapper)

Float64s

Float64sAreSorted

Ints

IntsAreSorted

Reverse

Search

Search (DescendingOrder)

SearchFloat64s

SearchInts

Slice

SliceStable

Constants

This section is empty.

Variables

This section is empty.

Functions

func Find

added in go1.19

```
func Find(n int, cmp func(int) int) (i int, found bool)
```

Find uses binary search to find and return the smallest index i in $[0, n)$ at which $\text{cmp}(i) \leq 0$. If there is no such index i , Find returns $i = n$. The found result is true if $i < n$ and $\text{cmp}(i) == 0$. Find calls $\text{cmp}(i)$ only for i in the range $[0, n)$.

To permit binary search, Find requires that $\text{cmp}(i) > 0$ for a leading prefix of the range, $\text{cmp}(i) == 0$ in the middle, and $\text{cmp}(i) < 0$ for the final suffix of the range. (Each subrange could be empty.) The usual way to establish this condition is to interpret $\text{cmp}(i)$ as a comparison of a desired target value t against entry i in an underlying indexed data structure x , returning <0 , 0 , and >0 when $t < x[i]$, $t == x[i]$, and $t > x[i]$, respectively.

For example, to look for a particular string in a sorted, random-access list of strings:

```
i, found := sort.Find(x.Len(), func(i int) int {
    return strings.Compare(target, x.At(i))
})
if found {
    fmt.Printf("found %s at entry %d\n", target, i)
} else {
    fmt.Printf("%s not found, would insert at %d", target, i)
}
```

func Float64s

```
func Float64s(x []float64)
```

Float64s sorts a slice of float64s in increasing order. Not-a-number (NaN) values are ordered before other values.

► [Example](#)

func Float64sAreSorted

```
func Float64sAreSorted(x []float64) bool
```

Float64sAreSorted reports whether the slice x is sorted in increasing order, with not-a-number (NaN) values before any other values.

► [Example](#)

func Ints

```
func Ints(x []int)
```

Ints sorts a slice of ints in increasing order.

► [Example](#)

func IntsAreSorted

```
func IntsAreSorted(x []int) bool
```

IntsAreSorted reports whether the slice x is sorted in increasing order.

► [Example](#)

func IsSorted

```
func IsSorted(data Interface) bool
```

IsSorted reports whether data is sorted.

func Search

```
func Search(n int, f func(int) bool) int
```

Search uses binary search to find and return the smallest index i in [0, n) at which f(i) is true, assuming that on the range [0, n), f(i) == true implies f(i+1) == true. That is, Search requires that f is false for some (possibly empty) prefix of the input range [0, n) and then true for the (possibly empty) remainder; Search returns the first true index. If there is no such index, Search returns n. (Note that the "not found" return value is not -1 as in, for instance, strings.Index.) Search calls f(i) only for i in the range [0, n).

A common use of Search is to find the index i for a value x in a sorted, indexable data structure such as an array or slice. In this case, the argument f, typically a closure, captures the value to be searched for, and how the data structure is indexed and ordered.

For instance, given a slice data sorted in ascending order, the call Search(len(data), func(i int) bool { return data[i] >= 23 }) returns the smallest index i such that data[i] >= 23. If the caller wants to find

whether 23 is in the slice, it must test `data[i] == 23` separately.

Searching data sorted in descending order would use the `<=` operator instead of the `>=` operator.

To complete the example above, the following code tries to find the value `x` in an integer slice data sorted in ascending order:

```
x := 23
i := sort.Search(len(data), func(i int) bool { return data[i] >= x })
if i < len(data) && data[i] == x {
    // x is present at data[i]
} else {
    // x is not present in data,
    // but i is the index where it would be inserted.
}
```

As a more whimsical example, this program guesses your number:

```
func GuessingGame() {
    var s string
    fmt.Printf("Pick an integer from 0 to 100.\n")
    answer := sort.Search(100, func(i int) bool {
        fmt.Printf("Is your number <= %d? ", i)
        fmt.Scanf("%s", &s)
        return s != "" && s[0] == 'y'
    })
    fmt.Printf("Your number is %d.\n", answer)
}
```

► [Example](#)

► [Example \(DescendingOrder\)](#)

func [SearchFloat64s](#)

```
func SearchFloat64s(a []float64, x float64) int
```

`SearchFloat64s` searches for `x` in a sorted slice of `float64s` and returns the index as specified by `Search`. The return value is the index to insert `x` if `x` is not present (it could be `len(a)`). The slice must be sorted in ascending order.

► [Example](#)

func [SearchInts](#)

```
func SearchInts(a []int, x int) int
```

SearchInts searches for x in a sorted slice of ints and returns the index as specified by Search. The return value is the index to insert x if x is not present (it could be len(a)). The slice must be sorted in ascending order.

► [Example](#)

func SearchStrings

```
func SearchStrings(a []string, x string) int
```

SearchStrings searches for x in a sorted slice of strings and returns the index as specified by Search. The return value is the index to insert x if x is not present (it could be len(a)). The slice must be sorted in ascending order.

func Slice

added in go1.8

```
func Slice(x any, less func(i, j int) bool)
```

Slice sorts the slice x given the provided less function. It panics if x is not a slice.

The sort is not guaranteed to be stable: equal elements may be reversed from their original order. For a stable sort, use SliceStable.

The less function must satisfy the same requirements as the Interface type's Less method.

► [Example](#)

func SliceIsSorted

added in go1.8

```
func SliceIsSorted(x any, less func(i, j int) bool) bool
```

SliceIsSorted reports whether the slice x is sorted according to the provided less function. It panics if x is not a slice.

func SliceStable

added in go1.8

```
func SliceStable(x any, less func(i, j int) bool)
```

SliceStable sorts the slice x using the provided less function, keeping equal elements in their original order. It panics if x is not a slice.

The less function must satisfy the same requirements as the Interface type's Less method.

► [Example](#)

func Sort

```
func Sort(data Interface)
```

Sort sorts data in ascending order as determined by the `Less` method. It makes one call to `data.Len` to determine `n` and $O(n \cdot \log(n))$ calls to `data.Less` and `data.Swap`. The sort is not guaranteed to be stable.

func [Stable](#)

added in go1.2

```
func Stable(data Interface)
```

Stable sorts data in ascending order as determined by the `Less` method, while keeping the original order of equal elements.

It makes one call to `data.Len` to determine `n`, $O(n \cdot \log(n))$ calls to `data.Less` and $O(n \cdot \log(n) \cdot \log(n))$ calls to `data.Swap`.

func [Strings](#)

```
func Strings(x []string)
```

Strings sorts a slice of strings in increasing order.

► [Example](#)

func [StringsAreSorted](#)

```
func StringsAreSorted(x []string) bool
```

StringsAreSorted reports whether the slice `x` is sorted in increasing order.

Types

type [Float64Slice](#)

```
type Float64Slice []float64
```

Float64Slice implements `Interface` for a `[]float64`, sorting in increasing order, with not-a-number (NaN) values ordered before other values.

func (Float64Slice) [Len](#)

```
func (x Float64Slice) Len() int
```

func (Float64Slice) [Less](#)

```
func (x Float64Slice) Less(i, j int) bool
```

Less reports whether `x[i]` should be ordered before `x[j]`, as required by the `sort` Interface. Note that floating-point comparison by itself is not a transitive relation: it does not report a consistent ordering for not-a-number (NaN) values. This implementation of `Less` places NaN values before any others, by using:

```
x[i] < x[j] || (math.IsNaN(x[i]) && !math.IsNaN(x[j]))
```

func (Float64Slice) Search

```
func (p Float64Slice) Search(x float64) int
```

`Search` returns the result of applying `SearchFloat64s` to the receiver and `x`.

func (Float64Slice) Sort

```
func (x Float64Slice) Sort()
```

`Sort` is a convenience method: `x.Sort()` calls `Sort(x)`.

func (Float64Slice) Swap

```
func (x Float64Slice) Swap(i, j int)
```

type IntSlice

```
type IntSlice []int
```

`IntSlice` attaches the methods of `Interface` to `[]int`, sorting in increasing order.

func (IntSlice) Len

```
func (x IntSlice) Len() int
```

func (IntSlice) Less

```
func (x IntSlice) Less(i, j int) bool
```

func (IntSlice) Search

```
func (p IntSlice) Search(x int) int
```

`Search` returns the result of applying `SearchInts` to the receiver and `x`.

func (IntSlice) Sort

```
func (x IntSlice) Sort()
```

`Sort` is a convenience method: `x.Sort()` calls `Sort(x)`.

func (IntSlice) Swap

```
func (x IntSlice) Swap(i, j int)
```

type Interface

```
type Interface interface {  
    // Len is the number of elements in the collection.  
    Len() int  
  
    // Less reports whether the element with index i  
    // must sort before the element with index j.  
    //  
    // If both Less(i, j) and Less(j, i) are false,  
    // then the elements at index i and j are considered equal.  
    // Sort may place equal elements in any order in the final result,  
    // while Stable preserves the original input order of equal elements.  
    //  
    // Less must describe a transitive ordering:  
    // - if both Less(i, j) and Less(j, k) are true, then Less(i, k) must be true as well  
    // - if both Less(i, j) and Less(j, k) are false, then Less(i, k) must be false as well  
    //  
    // Note that floating-point comparison (the < operator on float32 or float64 values  
    // is not a transitive ordering when not-a-number (NaN) values are involved.  
    // See Float64Slice.Less for a correct implementation for floating-point values.  
    Less(i, j int) bool  
  
    // Swap swaps the elements with indexes i and j.  
    Swap(i, j int)  
}
```

An implementation of Interface can be sorted by the routines in this package. The methods refer to elements of the underlying collection by integer index.

func Reverse

added in go1.1

```
func Reverse(data Interface) Interface
```

Reverse returns the reverse order for data.

► [Example](#)

type StringSlice

```
type StringSlice []string
```

StringSlice attaches the methods of Interface to []string, sorting in increasing order.

func (StringSlice) Len

```
func (x StringSlice) Len() int
```

func (StringSlice) Less

```
func (x StringSlice) Less(i, j int) bool
```

func (StringSlice) Search

```
func (p StringSlice) Search(x string) int
```

Search returns the result of applying SearchStrings to the receiver and x.

func (StringSlice) Sort

```
func (x StringSlice) Sort()
```

Sort is a convenience method: x.Sort() calls Sort(x).

func (StringSlice) Swap

```
func (x StringSlice) Swap(i, j int)
```



Source Files

[View all](#) 

[search.go](#)
[slice.go](#)

[sort.go](#)
[zsortfunc.go](#)

[zsortinterface.go](#)

Why Go

[Use Cases](#)

[Case Studies](#)

Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

Packages

[Standard Library](#)

[About Go Packages](#)

About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

Connect

[Twitter](#)

[GitHub](#)

Slack

r/golang

Meetup

Golang Weekly

Copyright

Terms of Service

Privacy Policy

Report an Issue



Google