# Binary Searching

A few Ruby methods support binary searching in a collection:

| | |
|---|---|
| **`Array#bsearch`** | Returns an element selected via a binary search as determined by a given block. |

**Array#bsearch_index** Returns the index of an element selected via a binary search as determined by a given block.

**Range#bsearch** Returns an element selected via a binary search as determined by a given block.

Each of these methods returns an enumerator if no block is given.

Given a block, each of these methods returns an element (or element index) from `self` as determined by a binary search. The search finds an element of `self` which meets the given condition in `O(log n)` operations, where `n` is the count of elements. `self` should be sorted, but this is not checked.

There are two search modes:

**Find-minimum mode** method `bsearch` returns the first element for which the block returns `true`; the block must return `true` or `false`.

**Find-any mode** method `bsearch` some element, if any, for which the block returns zero. the block must return a numeric value.

The block should not mix the modes by sometimes returning `true` or `false` and other times returning a numeric value, but this is not checked.

### Find-Minimum Mode

In find-minimum mode, the block must return `true` or `false`. The further requirement (though not checked) is that there are no indexes `i` and `j` such that:

- `0 <= i < j <= self.size`.
- The block returns `true` for `self[i]` and `false` for `self[j]`.

Less formally: the block is such that all `false`-evaluating elements precede all `true`-evaluating elements.

In find-minimum mode, method `bsearch` returns the first element for which the block returns `true`.

Examples:

```
a = [0, 4, 7, 10, 12]
a.bsearch {|x| x >= 4 } # => 4
a.bsearch {|x| x >= 6 } # => 7
a.bsearch {|x| x >= -1 } # => 0
a.bsearch {|x| x >= 100 } # => nil

r = (0...a.size)
r.bsearch {|i| a[i] >= 4 } #=> 1
r.bsearch {|i| a[i] >= 6 } #=> 2
r.bsearch {|i| a[i] >= 8 } #=> 3
r.bsearch {|i| a[i] >= 100 } #=> nil
r = (0.0...Float::INFINITY)
r.bsearch {|x| Math.log(x) >= 0 } #=> 1.0
```

These blocks make sense in find-minimum mode:

```
a = [0, 4, 7, 10, 12]
a.map {|x| x >= 4 } # => [false, true, true, true, true]
a.map {|x| x >= 6 } # => [false, false, true, true, true]
a.map {|x| x >= -1 } # => [true, true, true, true, true]
a.map {|x| x >= 100 } # => [false, false, false, false, false]
```

This would not make sense:

```
a.map {|x| x == 7 } # => [false, false, true, false, false]
```

### Find-Any Mode

In find-any mode, the block must return a numeric value. The further requirement (though not checked) is that there are no indexes `i` and `j` such that:

- `0 <= i < j <= self.size`.
- The block returns a negative value for `self[i]` and a positive value for `self[j]`.
- The block returns a negative value for `self[i]` and zero `self[j]`.
- The block returns zero for `self[i]` and a positive value for `self[j]`.

Less formally: the block is such that:

- All positive-evaluating elements precede all zero-evaluating elements.
- All positive-evaluating elements precede all negative-evaluating elements.
- All zero-evaluating elements precede all negative-evaluating elements.

In find-any mode, method `bsearch` returns some element for which the block returns zero, or `nil` if no such element is found.

Examples:

```
a = [0, 4, 7, 10, 12]
a.bsearch {|element| 7 <=> element } # => 7
a.bsearch {|element| -1 <=> element } # => nil
a.bsearch {|element| 5 <=> element } # => nil
a.bsearch {|element| 15 <=> element } # => nil

a = [0, 100, 100, 100, 200]
r = (0..4)
r.bsearch {|i| 100 - a[i] } #=> 1, 2 or 3
r.bsearch {|i| 300 - a[i] } #=> nil
r.bsearch {|i|  50 - a[i] } #=> nil
```

These blocks make sense in find-any mode:

```
a = [0, 4, 7, 10, 12]
a.map {|element| 7 <=> element } # => [1, 1, 0, -1, -1]
```

```
a.map {|element| -1 <=> element } # => [-1, -1, -1, -1, -1]
a.map {|element| 5 <=> element } # => [1, 1, -1, -1, -1]
a.map {|element| 15 <=> element } # => [1, 1, 1, 1, 1]
```

This would not make sense:

```
a.map {|element| element <=> 7 } # => [-1, -1, 0, 1, 1]
```