function template
# std::binary_search

<algorithm>

default (1)
```
template <class ForwardIterator, class T>
  bool binary_search (ForwardIterator first, ForwardIterator last,
                      const T& val);
```
custom (2)
```
template <class ForwardIterator, class T, class Compare>
  bool binary_search (ForwardIterator first, ForwardIterator last,
                      const T& val, Compare comp);
```

Test if value exists in sorted sequence

Returns `true` if any element in the range `[first,last)` is equivalent to *val*, and `false` otherwise.

The elements are compared using `operator<` for the first version, and *comp* for the second. Two elements, a and b are considered equivalent if `(!(a<b) && !(b<a))` or if `(!comp(a,b) && !comp(b,a))`.

The elements in the range shall already be sorted according to this same criterion (`operator<` or *comp*), or at least partitioned with respect to *val*.

The function optimizes the number of comparisons performed by comparing non-consecutive elements of the sorted range, which is specially efficient for random-access iterators.

The behavior of this function template is equivalent to:

```
template <class ForwardIterator, class T>
  bool binary_search (ForwardIterator first, ForwardIterator last, const T& val)
{
  first = std::lower_bound(first,last,val);
  return (first!=last && !(val<*first));
}
```

Parameters

`first, last`
Forward iterators to the initial and final positions of a sorted (or properly partitioned) sequence. The range used is `[first,last)`, which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.

`val`
Value to search for in the range.
For *(1)*, T shall be a type supporting being compared with elements of the range `[first,last)` as either operand of `operator<`.

`comp`
Binary function that accepts two arguments of the type pointed by `ForwardIterator` (and of type T), and returns a value convertible to `bool`. The value returned indicates whether the first argument is considered to go before the second.
The function shall not modify any of its arguments.
This can either be a function pointer or a function object.

Return value

`true` if an element equivalent to *val* is found, and `false` otherwise.

Example

```
// binary_search example
#include <iostream>     // std::cout
#include <algorithm>    // std::binary_search, std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  std::vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3 2 1

  // using default comparison:
  std::sort (v.begin(), v.end());

  std::cout << "looking for a 3... ";
  if (std::binary_search (v.begin(), v.end(), 3))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  // using myfunction as comp:
  std::sort (v.begin(), v.end(), myfunction);

  std::cout << "looking for a 6... ";
  if (std::binary_search (v.begin(), v.end(), 6, myfunction))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  return 0;
}
```

Output:
```
looking for a 3... found!
looking for a 6... not found.
```

Complexity

On average, logarithmic in the distance between *first* and *last*: Performs approximately `log`$_2$`(N)+2` element comparisons (where *N* is this distance).
On *non-random-access iterators*, the iterator advances produce themselves an additional linear complexity in *N* on average.

Data races

The objects in the range `[first,last)` are accessed.

Exceptions

Throws if either an element comparison or an operation on an iterator throws.
Note that invalid arguments cause *undefined behavior*.

See also

find	Find value in range (function template)
lower_bound	Return iterator to lower bound (function template)
upper_bound	Return iterator to upper bound (function template)
equal_range	Get subrange of equal elements (function template)