

# Binary Search

A fast search with a search time of  $O(\log n)$  has been predefined in C++.

## WE'LL COVER THE FOLLOWING ^

- Further information

The binary search algorithms use the fact that the ranges are already sorted. To search for an element, use `std::binary_search`. With `std::lower_bound`, we get an iterator for the first element, not smaller than the given value. With `std::upper_bound`, we get an iterator back for the first element which is bigger than the given value. `std::equal_range` combines both algorithms.

If the container has  $n$  elements, we need on average  $\log_2(n)$  comparisons for the search. The binary search requires that we use the same comparison criterion that you used for sorting the container. By default, the comparison criterion is `std::less`, but we can adjust it. Our sorting criterion has to obey [strict weak ordering](#). If not, the result is undefined.

If we have an unordered associative container, the methods of the [unordered associative container](#) are generally faster.

`std::binary_search`: Searches for the element `val` in the range.

```
bool binary_search(FwdIt first, FwdIt last, const T& val)
bool binary_search(FwdIt first, FwdIt last, const T& val, BiPre pre)
```



`lower_bound`: Returns the position of the first element of the range which is not smaller than `val`.

```
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val)
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val, BiPre pre)
```



`upper_bound`: Returns the position of the first element of the range which is bigger than `val`.

```
FwdIt upper_bound(FwdIt first, FwdIt last, const T& val)
FwdIt upper_bound(FwdIt first, FwdIt last, const T& val, BiPre pre)
```

`equal_range`: Returns the pair `std::lower_bound` and `std::upper_bound` for the element `val`.

```
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const T& val)
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const T& val, BiPre pre)
```

Finally, the code snippet:

```
#include <algorithm>
#include <cmath>
#include <iostream>
#include <vector>

bool isLessAbs(int a, int b){
    return std::abs(a) < std::abs(b);
}

int main(){

    std::cout << std::boolalpha << std::endl;

    std::vector<int> vec{-3, 0, -3, 2, -3, 5, -3, 7, -0, 6, -3, 5, -6, 8, 9, 0, 8, 7, -7, 8, 9, 0, -8, 7, 6, -5, 4, -3, 2, -1, 0};

    for ( auto v: vec ) std::cout << v << " ";

    std::sort(vec.begin(), vec.end(), isLessAbs);
    std::cout << std::endl;
    for ( auto v: vec ) std::cout << v << " ";

    std::cout << std::endl;

    std::cout << std::endl;
    std::cout << "std::binary_search(vec.begin(), vec.end(), -5, isLessAbs): " << std::binary_search(vec.begin(), vec.end(), -5, isLessAbs) << std::endl;
    std::cout << "std::binary_search(vec.begin(), vec.end(), 5, isLessAbs): " << std::binary_search(vec.begin(), vec.end(), 5, isLessAbs) << std::endl;

    auto pair= std::equal_range(vec.begin(), vec.end(), 3, isLessAbs);

    std::cout << std::endl;

    std::cout << "Position of first 3: " << std::distance(vec.begin(), pair.first) << std::endl;
    std::cout << "Position of last 3: " << std::distance(vec.begin(), pair.second)-1 << std::endl;
    for ( auto threeIt= pair.first; threeIt != pair.second ; ++threeIt ) std::cout << *threeIt << " ";

    std::cout << "\n\n";

}
```

## Further information #

- [strict weak ordering](#)
- [unordered associative container](#)

---

In the next lesson, we'll learn different ways of combining ranges.