Binary Search

The fast search which has a search time of O(log n) has been predefined in C++.

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` you get an iterator for the first element, being no smaller than the given value. With `std::upper_bound` you 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, you need on average $\mathbf{log_2(n)}$ comparisons for the search. The binary search requires that you use the same comparison criterion that you used for sorting the container. Per default the comparison criterion is `std::less`, but you can adjust it. Your sorting criterion has to obey the strict weak ordering. If not, the program is undefined.

If you have an unordered associative container, the methods of the unordered associative container are in general faster.

Searches 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)
```

Returns the position of the first element of the range, being 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)
```

Returns the position of the first element of the range, being 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)
```

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.

```cpp
#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,

  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_s
  std::cout << "std::binary_search(vec.begin(), vec.end(), 5, isLessAbs): " << std::binary_se

  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::en
  for ( auto threeIt= pair.first; threeIt != pair.second ; ++threeIt ) std::cout << *threeIt

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

}
```

Binary search algorithms