Longest Increasing Subsequence

Dada una lista, encuentra la subsecuencia creciente más larga. Puede configurarse qué significa "creciente". Ver ejemplos.

- Tiempo de procesamiento: $O(n \log(n))$

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

template <class T, class Compare = std::less<T>>
auto longest_increasing_subsequence(const std::vector<T>& X,
                                    Compare comp = std::less<T>())
{
    long n = X.size();

    using PII = std::pair<int, T>;

    // M[k] = index i of smallest X[i] for which
    // there is a subsequence of length k ending
    // at X[i]. Note that M will be increasing.
    std::vector<PII> M(2);
    M.reserve((n + 2) / 2);

    // P[i] = parent of i.
    std::vector<int> P(n);

    int L = 1;
    M[1].first = 0;
    M[1].second = X[0];
    for (long i = 1; i < n; ++i)
    {
        auto first = M.begin() + 1;
        auto last = M.begin() + L + 1;
        const auto& xi = X[i];
        auto newLIter =
          std::partition_point(first, last, [xi, &comp](const PII& p) {
              return comp(p.second, xi);
          });
        auto newL = newLIter - first + 1;
```

```
        P[i] = M[newL - 1].first;

        if (newL < M.size())
        {
            M[newL].first = i;
            M[newL].second = xi;
        }
        else
        {
            M.push_back({i, xi});
        }

        if (newL > L)
            L = newL;
    }
    std::vector<T> S(L);
    long k = M[L].first;
    for (auto it = S.rbegin(); it != S.rend(); ++it, k = P[k])
    {
        *it = X[k];
    }

    return S;
}

using namespace std;

template <class T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& A)
{
    for (const auto& x : A)
        os << x << ' ';
    return os;
}

int main()
{
    std::vector<int> A = {0, 4, 2, 3, 5, 2, 1, 7, 3, 5, 4, 3,
                          4, 5, 6, 4, 5, 3, 1, 5, 2, 6, 9};
    cout << "A = " << A << endl;
    cout << "Longest increasing subsequence: "
         << longest_increasing_subsequence(A) << endl;
    cout << "Longest non-decreasing subsequence: "
         << longest_increasing_subsequence(A, std::less_equal<>()) << endl;
    cout << "Longest decreasing subsequence: "
```

```
        << longest_increasing_subsequence(A, std::greater<>()) << endl;
    return 0;
}
```

Output:

```
A = 0 4 2 3 5 2 1 7 3 5 4 3 4 5 6 4 5 3 1 5 2 6 9
Longest increasing subsequence: 0 1 3 4 5 6 9
Longest non-decreasing subsequence: 0 2 2 3 3 4 4 5 5 6 9
Longest decreasing subsequence: 7 6 5 3 2
```