Итераторы

Итераторы

```cpp
#include <iostream>
#include <list>

int main() {
    std::list<int> l = { 10, 15, 20 };

    // Используем auto, чтобы не писать громоздкий тип
std::list<int>::iterator
    auto iter = l.begin();
    std::cout << *iter << "\n";  // печатаем начальный элемент
    ++iter;  // сдвигаемся к следующему элементу
    --iter;  // возвращаемся назад
}
```

Итераторы — это специальные объекты, предназначенные для навигации по контейнеру. Итераторы позволяют обращаться к текущему элементу контейнера и сдвигаться к соседним элементам. Итератор, указывающий на начальный элемент контейнера, возвращает функция begin.

Алгоритмы стандартной библиотеки

Алгоритмы C++

`ranges::all_of` (C++20) `ranges::any_of` (C++20) `ranges::none_of` (C++20)	checks if a predicate is `true` for all, any or none of the elements in a range (niebloid)
`for_each`	applies a function to a range of elements (function template)
`ranges::for_each` (C++20)	applies a function to a range of elements (niebloid)
`for_each_n` (C++17)	applies a function object to the first n elements of a sequence (function template)
`ranges::for_each_n` (C++20)	applies a function object to the first n elements of a sequence (niebloid)
`count` `count_if`	returns the number of elements satisfying specific criteria (function template)
`ranges::count` (C++20) `ranges::count_if` (C++20)	returns the number of elements satisfying specific criteria (niebloid)
`mismatch`	finds the first position where two ranges differ (function template)
`ranges::mismatch` (C++20)	finds the first position where two ranges differ (niebloid)

`find`
`find_if`
`find_if_not` (C++11)
	finds the first element satisfying specific criteria
	(function template)

`ranges::find` (C++20) `ranges::find_if` (C++20) `ranges::find_if_not` (C++20)	finds the first element satisfying specific criteria (niebloid)
`find_end`	finds the last sequence of elements in a certain range (function template)
`ranges::find_end` (C++20)	finds the last sequence of elements in a certain range (niebloid)

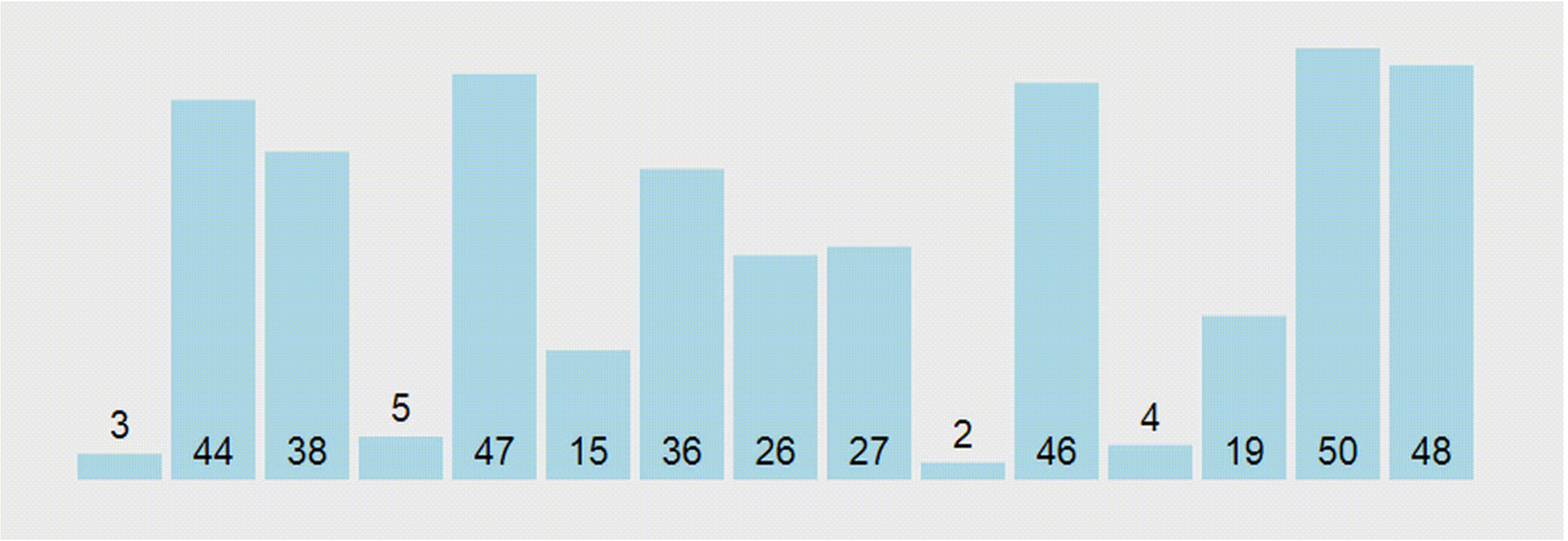
https://en.cppreference.com/w/cpp/algorithm

Сортировка

Сортировка пузырьком

```
vector <int> sorting_vec(vector <int> arr) {
    int temp;
    for (int i = 0; i < arr.size() - 1; i++) {
        for (int j = 0; j < arr.size() - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```

Сортировка пузырьком



Std::sort

```
sort(vec.begin(), vec.end());
```

Сравним время выполнения работы сортировок

```
vec = sorting_vec(vec);
```

VS

```
sort(vec.begin(), vec.end());
```

Сравним время выполнения работы сортировок

```
vec = sorting_vec(vec);
```

```
D1=24.094
D2=0.003
```

```
sort(vec.begin(), vec.end());
```

Операции над всеми элементами массива

```
int main()
{
    srand(time(NULL));
    vector <int> vec1, vec2;
    int N = 10;
    for (size_t i = 0; i < N; i++)
        vec1.push_back(rand() % 100);
    for (size_t i = 0; i < N; i++)
        cout << vec1[i] << " ";
    cout << endl;
}
```

Операции над всеми элементами массива

```
void print_vec(int x)
{
    cout << x << " ";
}
```

```
for (size_t i = 0; i < N; i++)
        print_vec(vec1[i]);
    cout << endl;
```

Операции над всеми элементами массива

```
void print_vec(int x)
{
    cout << x << " ";
}
```

```
for (size_t i = 0; i < N; i++)
        print_vec(vec1[i]);
    cout << endl;
```

```
for_each(vec1.begin(), vec1.end(), print_vec);
```

Операции над всеми элементами массива

Что будет, если изменить тип вектора на double?

```
void print_vec(int x)
{
    cout << x << " ";
}
```

```
for (size_t i = 0; i < N; i++)
        print_vec(vec1[i]);
    cout << endl;
```

```
for_each(vec1.begin(), vec1.end(), print_vec);
```

Шаблоны

Шаблоны

```
template <typename T>
void print_vec(const T& x)
{
    cout << x << " ";
}
```

Шаблон — это конструкция, которая создает обычный тип или функцию во время компиляции на основе аргументов, предоставленных пользователем для параметров шаблона.

Применим шаблон к for_each

```
for_each(vec1.begin(), vec1.end(), print_vec);
```

```
for_each(vec1.begin(), vec1.end(), print_vec<double>);
```

Что делать?

Можно ли как-то сэкономить и не создавать целую функцию?