# Алгоритмы стандартной библиотеки

# Алгоритмы C++

| | |
|---|---|
| `ranges::all_of` (C++20)<br>`ranges::any_of` (C++20)<br>`ranges::none_of` (C++20) | checks if a predicate is `true` for all, any or none of the elements in a range<br>(niebloid) |
| `for_each` | applies a function to a range of elements<br>(function template) |
| `ranges::for_each` (C++20) | applies a function to a range of elements<br>(niebloid) |
| `for_each_n` (C++17) | applies a function object to the first n elements of a sequence<br>(function template) |
| `ranges::for_each_n` (C++20) | applies a function object to the first n elements of a sequence<br>(niebloid) |
| `count`<br>`count_if` | returns the number of elements satisfying specific criteria<br>(function template) |
| `ranges::count` (C++20)<br>`ranges::count_if` (C++20) | returns the number of elements satisfying specific criteria<br>(niebloid) |
| `mismatch` | finds the first position where two ranges differ<br>(function template) |
| `ranges::mismatch` (C++20) | finds the first position where two ranges differ<br>(niebloid) |
| `find`<br>`find_if`<br>`find_if_not` (C++11) | finds the first element satisfying specific criteria<br>(function template) |
| `ranges::find` (C++20)<br>`ranges::find_if` (C++20)<br>`ranges::find_if_not` (C++20) | finds the first element satisfying specific criteria<br>(niebloid) |
| `find_end` | finds the last sequence of elements in a certain range<br>(function template) |
| `ranges::find_end` (C++20) | finds the last sequence of elements in a certain range<br>(niebloid) |

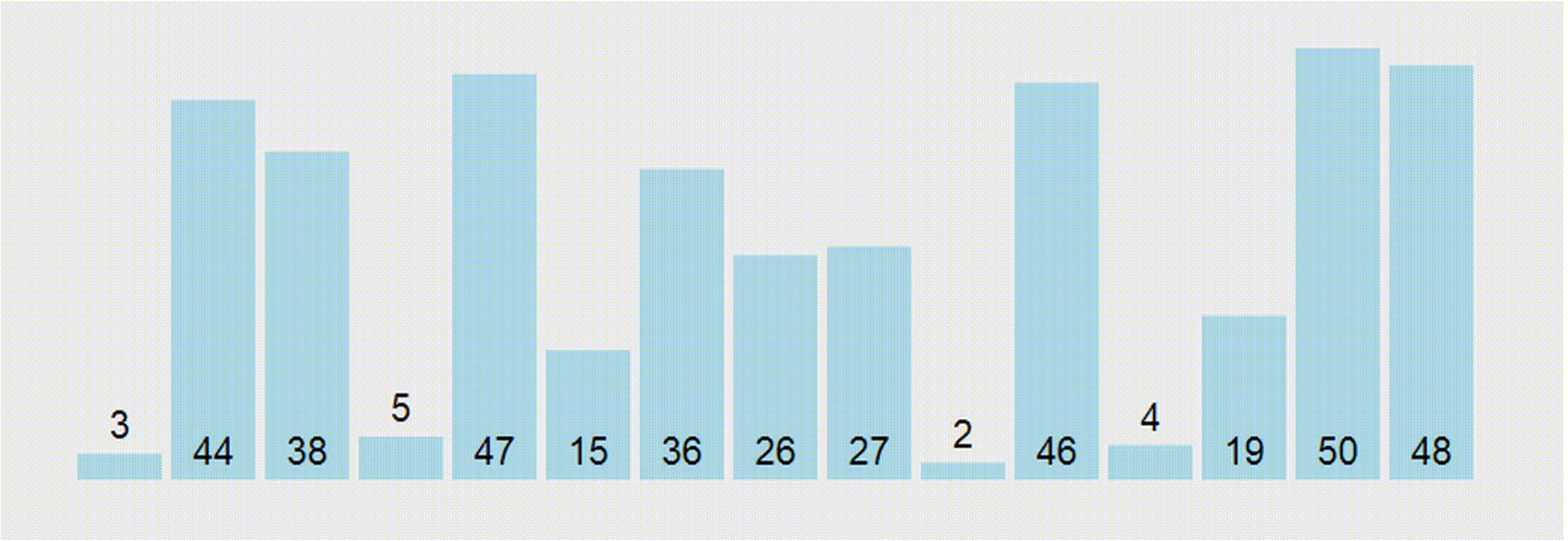
https://en.cppreference.com/w/cpp/algorithm

# 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

```
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?

# Лямбда функции

# Лямбда функции

```
[ capture ] (параметры) -> возвращаемое значение
{
        описание метода
}
```

**Лямбда-выражение** (также называемое лямбда или замыкание ) позволяет нам определить анонимную функцию внутри другой функции.

# Лямбда функции

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

```
47 94 68 64 77 34 46 4 69 44
```

```
auto f = [](auto value)
{
       print_vec(value);
};
for_each(vec1.begin(), vec1.end(), f);
```

# Лямбда функции

Упростим….

```
for_each(vec1.begin(), vec1.end(), [](auto value){cout << value << " "; });
```