

Яндекс

Яндекс

# Очень быстрый сортированный массив

Ярошевский Денис (разработчик)

Яндекс

быстрый

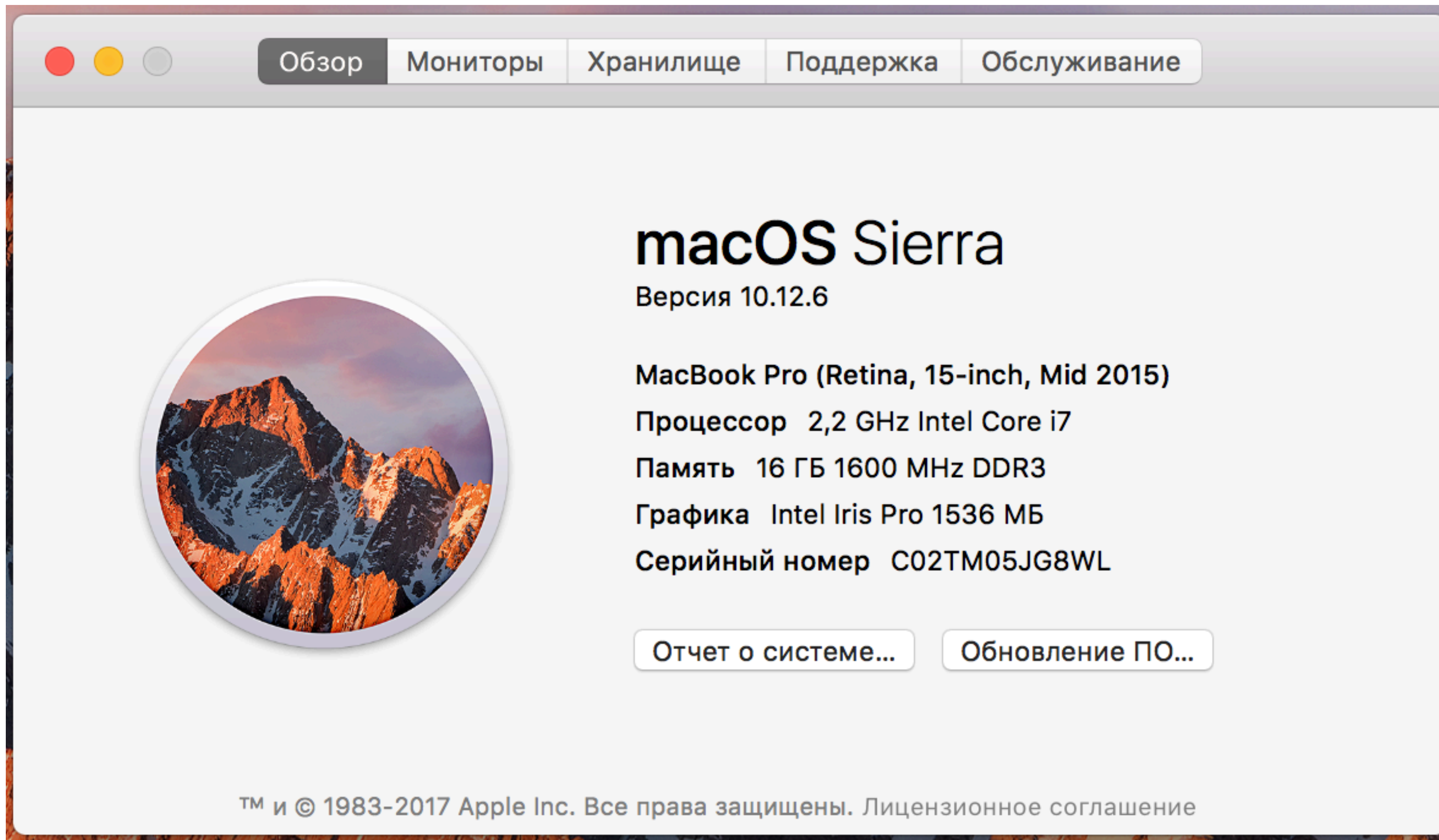
Яндекс

сортированный

# Предупреждения

- › Я люблю перформанс, но зарабатываю не этим.
- › Ноль ревью и использования в продакшн.
- › Почти все тестировалось на одной машине и с одним компилятором.
- › Оптимизация - процесс бесконечный. Я в какой-то момент остановился.
- › Материала сильно больше, чем я успею рассказать.

# О машине



# Компилятор

- › clang version: Apple LLVM version 9.0.0 (clang-900.0.38)  
Xcode: Version 9.1 (9B55)
- › Опции: --std=c++14 **-O3** -Werror -Wall  
(перепроверял на **-O2**)

# Set из сортированного массива



# Set из сортированного массива

- › Высокая локальность данных

# Set из сортированного массива

- › Высокая локальность данных
- › Быстрое итерирование

# Set из сортированного массива

- › Высокая локальность данных
- › Быстрое итерирование
- › Бинарный поиск

# Set из сортированного массива

- › Высокая локальность данных
- › Быстрое итерирование
- › Бинарный поиск
- › Малый перерасход памяти

# Set из сортированного массива

- › Высокая локальность данных
- › Быстрое итерирование
- › Бинарный поиск
- › Малый перерасход памяти
- › Медленная вставка/удаление по одному элементу

# Типичный use case

- › Маленькие размеры
- › POD

# Сравнение с std

Конструктор из 1000 int (64 bit)

srt::flat_set	16 мкс
std::unordered_set	171 мкс
std::set	196 мкс

# Сравнение с std

Копирование 1000 int (64 bit)

srt::flat_set	115 нс
std::unordered_set	124'626 нс
std::set	159'394 нс



# Сравнение с std

Поиск из 1000 элементов

srt::flat_set	23 нс
std::unordered_set	18 нс
std::set	56 нс

# Сравнение с std

Создание добавлением по одному (30'000 int 64bit)

srt::flat_set	25 мс
std::unordered_set	6 мс
std::set	8 мс

# Предыстория

- › Яндекс Браузер.
- › История.
- › Операции над множествами, кэширование.
- › Upstream в Chromium.

# Проблема

# Проблема

У вас есть flat\_set с 1000 элементов.

# Проблема

У вас есть `flat_set` с 1000 элементов.

› Как вставить один элемент?

# Проблема

У вас есть `flat_set` с 1000 элементов.

- › Как вставить один элемент?
- › 1000?

# Проблема

У вас есть `flat_set` с 1000 элементов.

- › Как вставить один элемент?
- › 1000?
- › Что на счет 100 элементов? А 10?



# `flat_set::insert(f, l)`

- › Не быть медленнее чем вставка по одному.
- › По максимуму использовать информацию о том, что новых элементов много.

# Существующие решения

- › `boost::flat_set`
- › `eastl::vector_set` (EA)
- › `folly::sorted_vector_set` (Facebook)
- › `base::flat_set` (Chromium)

# Существующие решения

Boost & eastl

```
std::copy(first, last, std::inserter(*this, end( )));
```

# Существующие решения

Folly & ~Chromium

(\*очень большое ~)

```
size_type old_size = size();  
body_.insert(end(), first, last);  
Iterator new_elements = begin() + old_size;  
std::sort(new_elements, end());  
std::inplace_merge(begin(), new_elements, end());  
auto new_end = std::unique(begin(), end());  
erase(new_end, end);
```

# Существующие решения

Folly & ~Chromium

(\*очень большое ~)

```
...  
std::sort(new_elements, end());
```

```
...
```

# Существующие решения

Folly & ~Chromium

(\*очень большое ~)

...

```
std::sort(new_elements, end());
```

```
std::inplace_merge(begin(), new_elements, end());
```

...

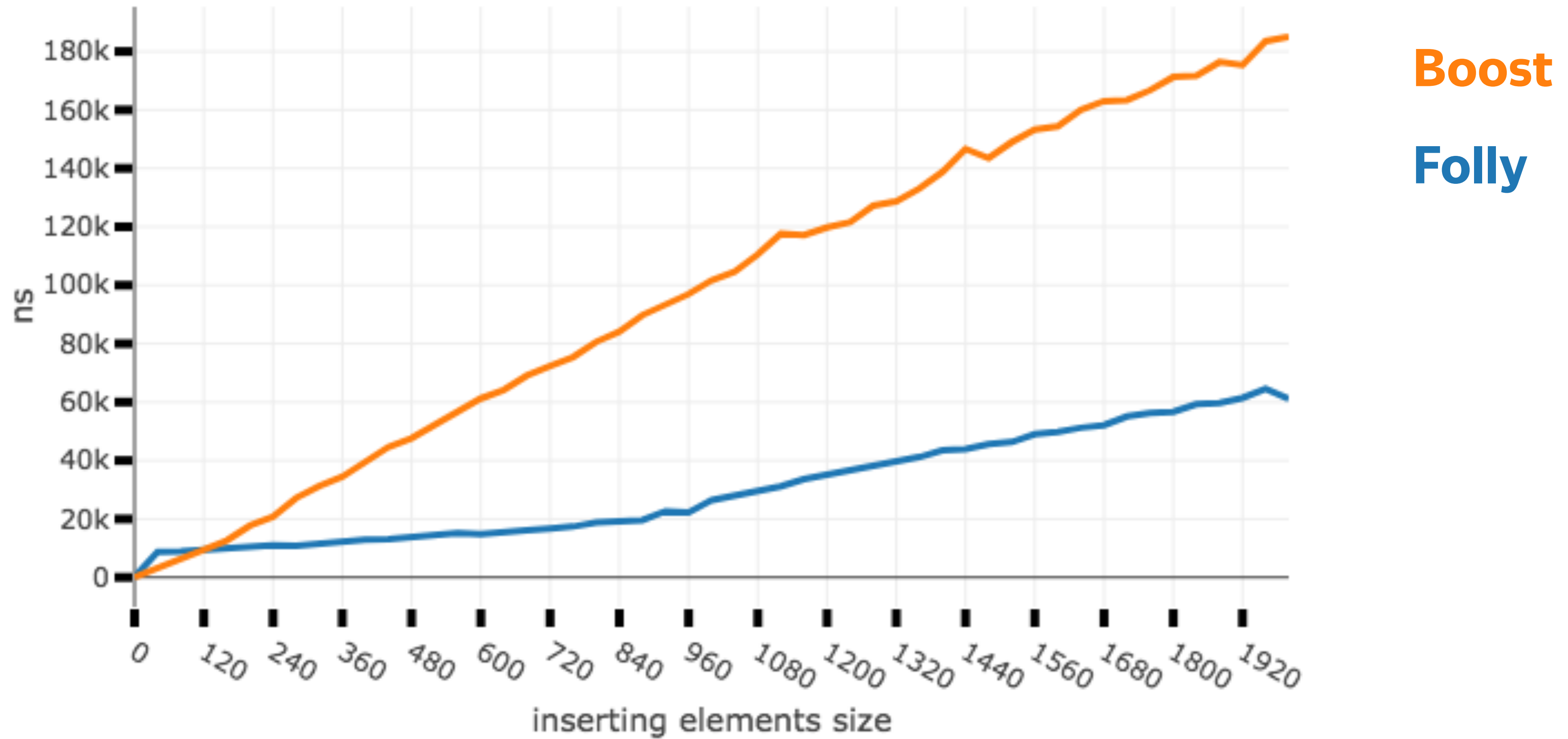
# Существующие решения

Folly & ~Chromium

(\*очень большое ~)

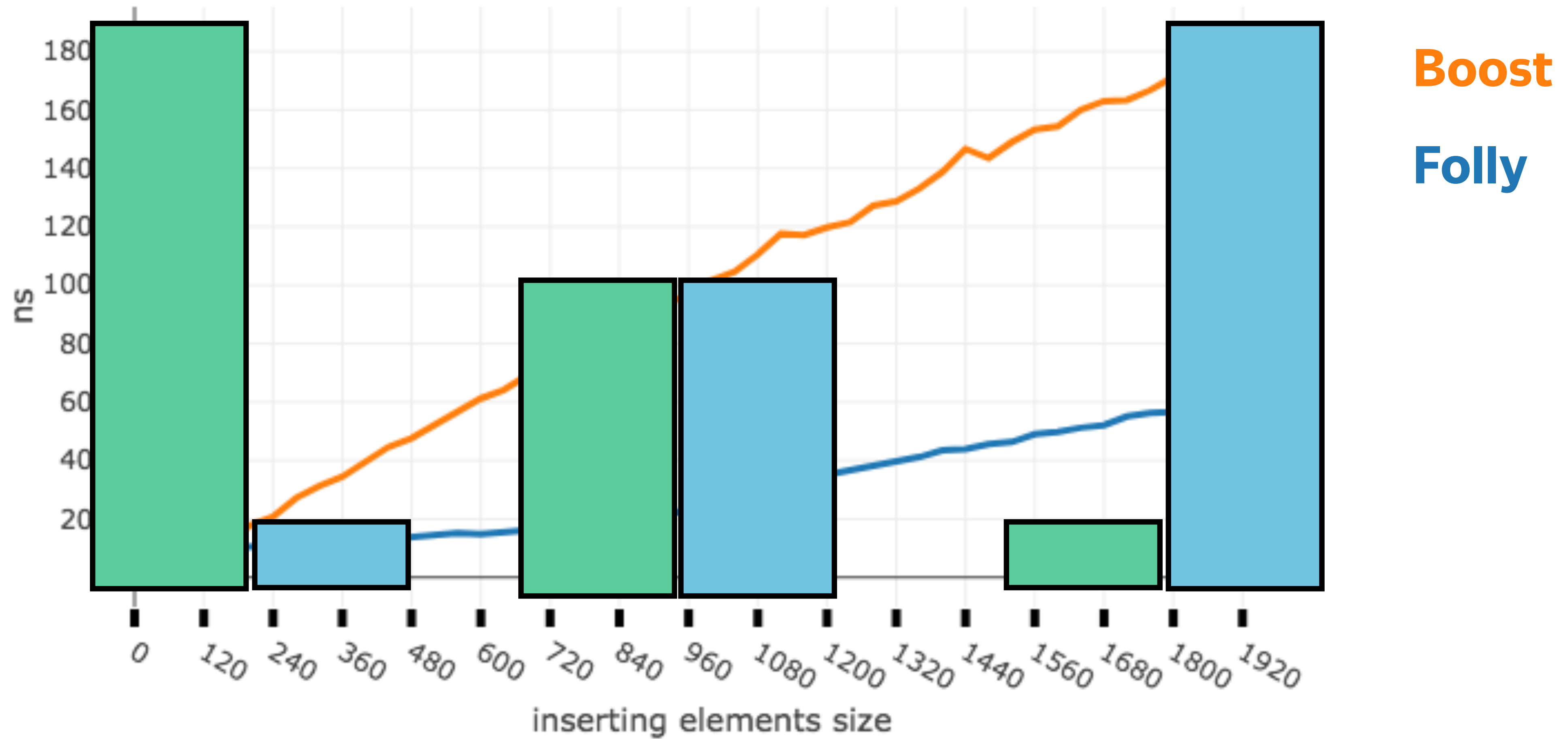
```
...  
std::sort(new_elements, end());  
std::inplace_merge(begin(), new_elements, end());  
auto new_end = std::unique(begin(), end());  
...
```

# Boost vs Folly

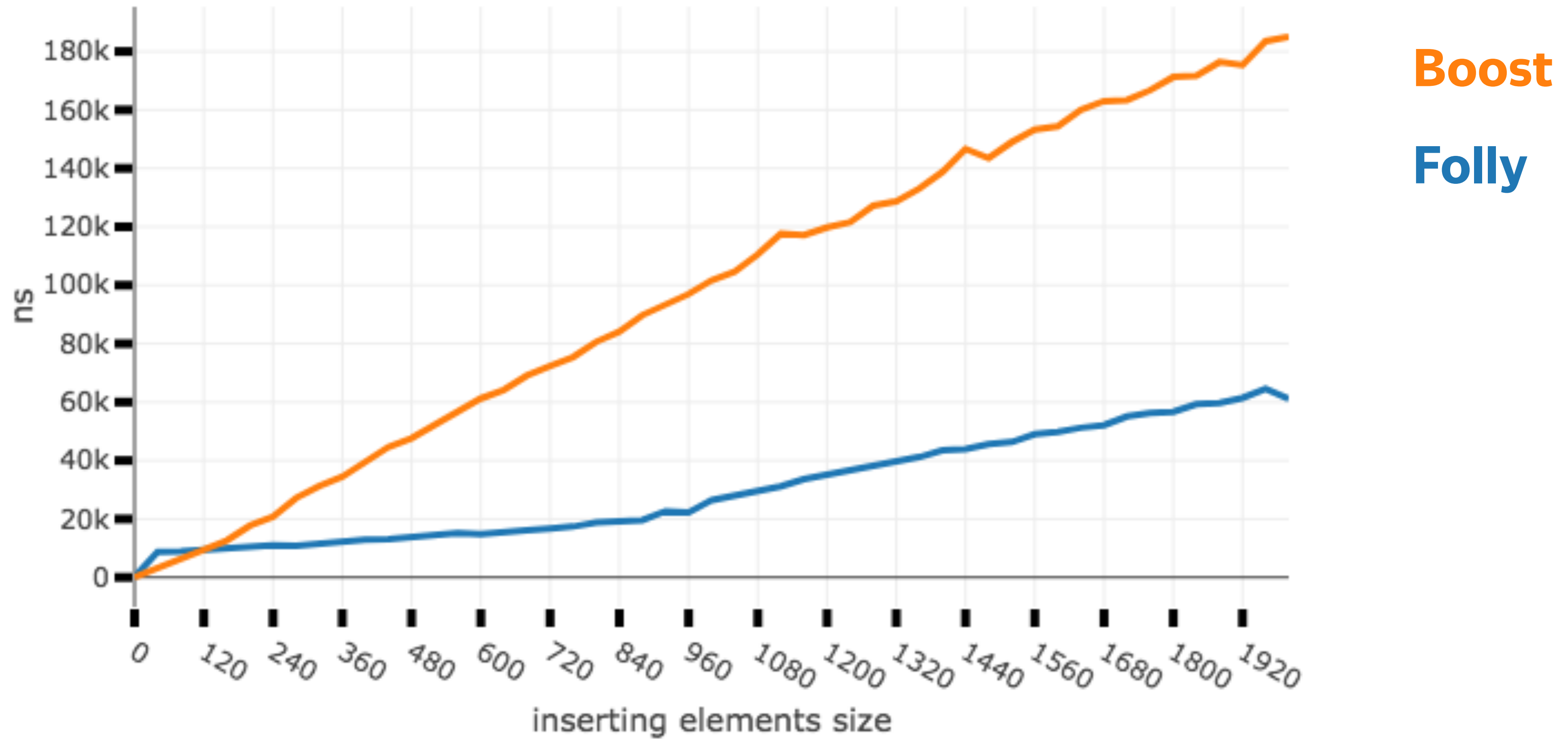




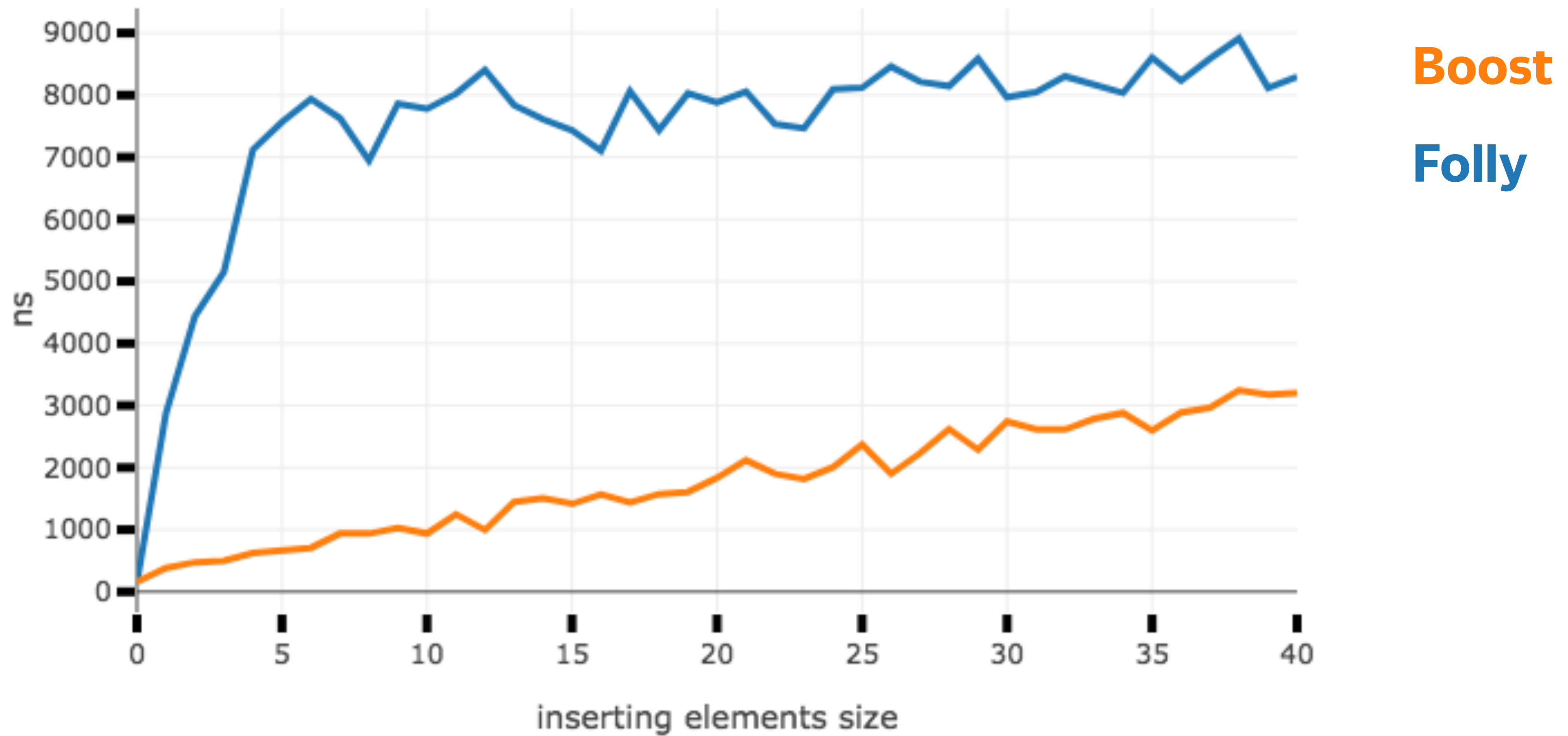
# Boost vs Folly



# Boost vs Folly



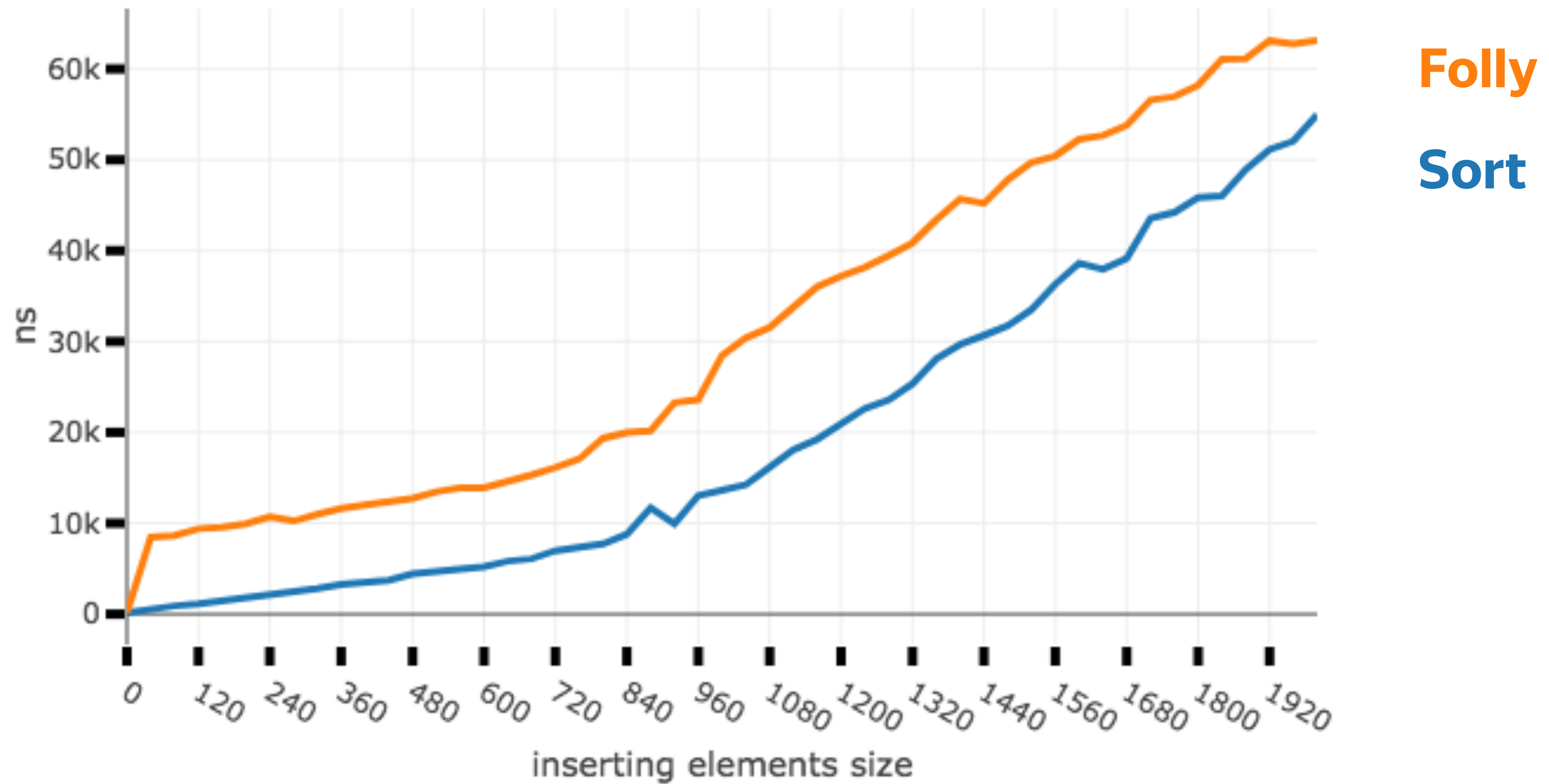
# Boost vs Folly (40)



# Анализ производительности

```
...  
std::sort(new_elements, end());  
  
std::inplace_merge(begin(), new_elements, end());  
  
auto new_end = std::unique(begin(), end());  
...
```

# Вклад сортировки



# Меньше усилий на дубликаты

```
...  
std::sort(new_elements, end());  
  
std::inplace_merge(begin(), new_elements, end());  
  
auto new_end = std::unique(begin(), end());  
...
```

# Меньше усилий на дубликаты

...

```
std::sort(new_elements, end());
```

```
auto new_end = std::unique(new_elements, end());
```

```
inplace_set_union(begin(), new_elements, end());
```

...

# Что нужно сделать?

- › Превратить `inplace_merge` в `set_union`
- › Написать `set_union` под нашу задачу.
- › Адаптировать бинарный поиск под нашу задачу.



# Что нужно сделать?

- › Превратить `inplace_merge` в `set_union`
- › **Написать `set_union` под нашу задачу.**
- › Адаптировать бинарный поиск под нашу задачу.

# Что нужно сделать?

- › Превратить `inplace_merge` в `set_union`  
*использовать capacity*
- › Написать `set_union` под нашу задачу.
- › Адаптировать бинарный поиск под нашу задачу.  
*Прыгать степенями 2ки.*

# std::set\_union

```
template <class I1, class I2, class O, class Comp>  
O set_union(I1 f1, I1 l1, // Left Input  
            I2 f2, I2 l2, // Right Input  
            O o,         // Output  
            Comp comp);  // StrictWeakOrdering
```

# std::set\_union (libc++)

```
template <class I1, class I2, class O, class Comp>
O set_union(I1 f1, I1 l1, I2 f2, I2 l2, O o, Comp comp) {
    for (; f1 != l1; ++o) {
        if (f2 == l2)
            return std::copy(f1, l1, o);
        if (comp(*f2, *f1)) {
            *o = *f2;
            ++f2;
        } else {
            *o = *f1;
            if (!comp(*f1, *f2))
                ++f2;
            ++f1;
        }
    }
    return std::copy(f2, l2, o);
}
```

# std::set\_union (libc++)

```
...  
*o = *f1;  
if ( !comp(*f1, *f2) ) ++f2;  
...
```

# std::set\_union (libc++)

```
...  
if ( !comp(*f1, *f2) ) ++f2;  
*o = *f1;  
...
```

# std::set\_union (libc++)

```
template <class I1, class I2, class O, class Comp>
O set_union(I1 f1, I1 l1, I2 f2, I2 l2, O o, Comp comp) {
    for (; f1 != l1; ++o) {
        if (f2 == l2)
            return std::copy(f1, l1, o);
        if (comp(*f2, *f1)) {
            *o = *f2;
            ++f2;
        } else {
            if (!comp(*f1, *f2))
                ++f2;
            *o = *f1;
            ++f1;
        }
    }
    return std::copy(f2, l2, o);
}
```

# std::set\_union (libc++)

```
for (; f1 != l1; ++o) {  
    if (f2 == l2)  
        return std::copy(f1, l1, o);  
    ...  
}  
return std::copy(f2, l2, o);
```



# std::set\_union (libc++)

```
...  
if ( comp(*f2, *f1) ) {  
    *o = *f2++;  
} else {  
    ...  
}
```

# std::set\_union (libc++)

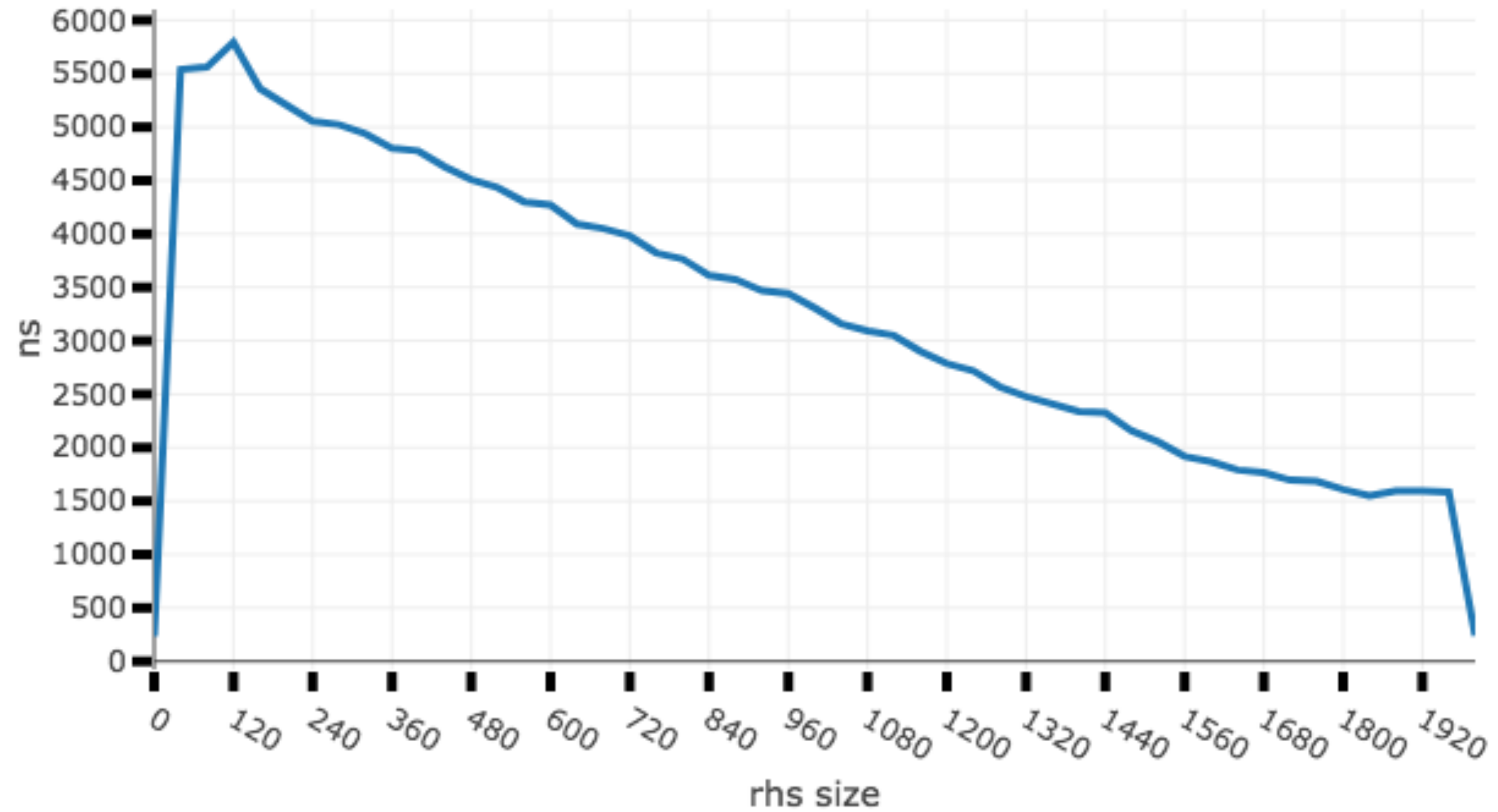
```
...  
} else {  
    if (!comp(*f1, *f2)) ++f2;  
    *o = *f1++;  
}  
...
```

# std::set\_union (libc++)

```
...  
} else {  
    if (!comp(*f1, *f2)) ++f2;  
    *o = *f1++;  
}  
...  

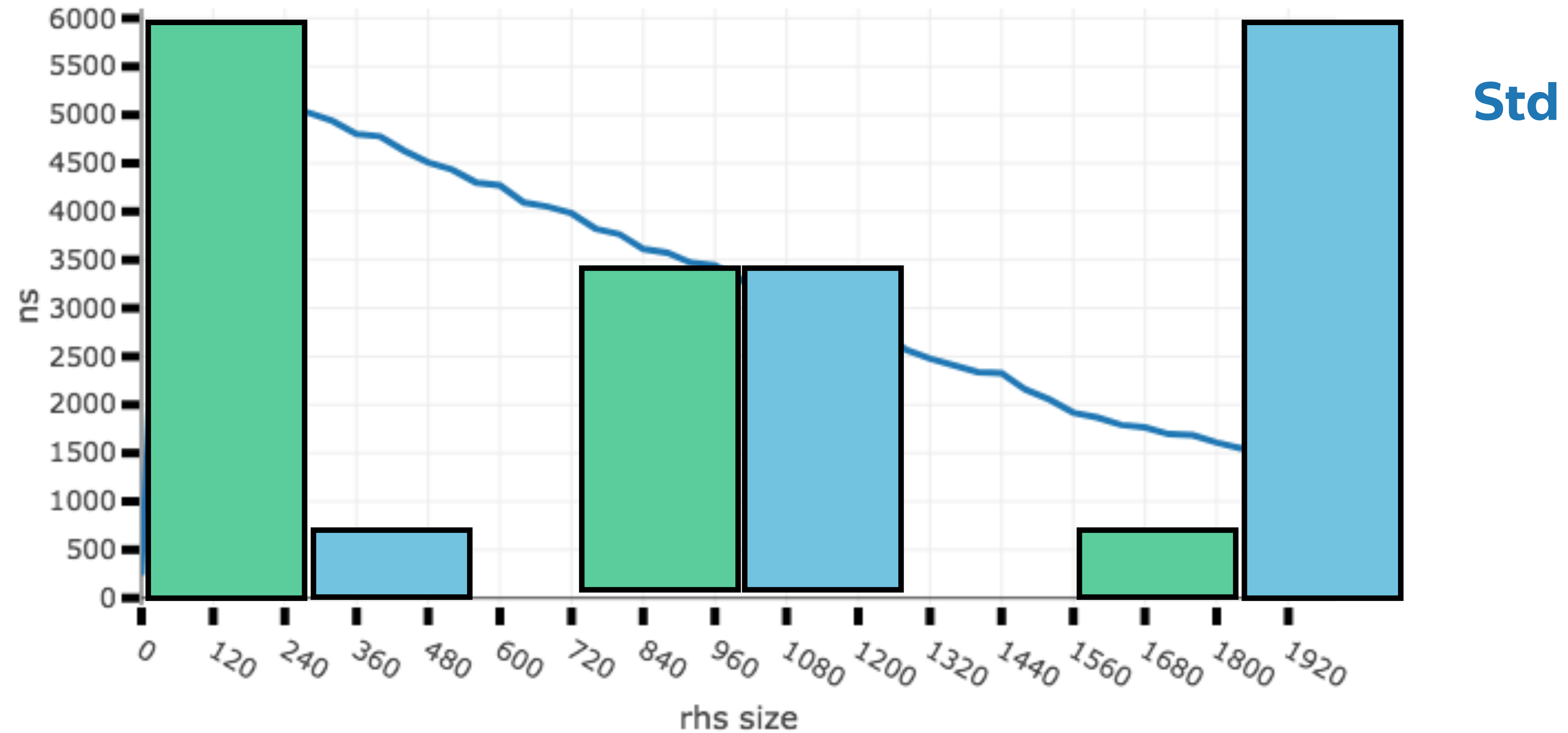
```

# std::set\_union

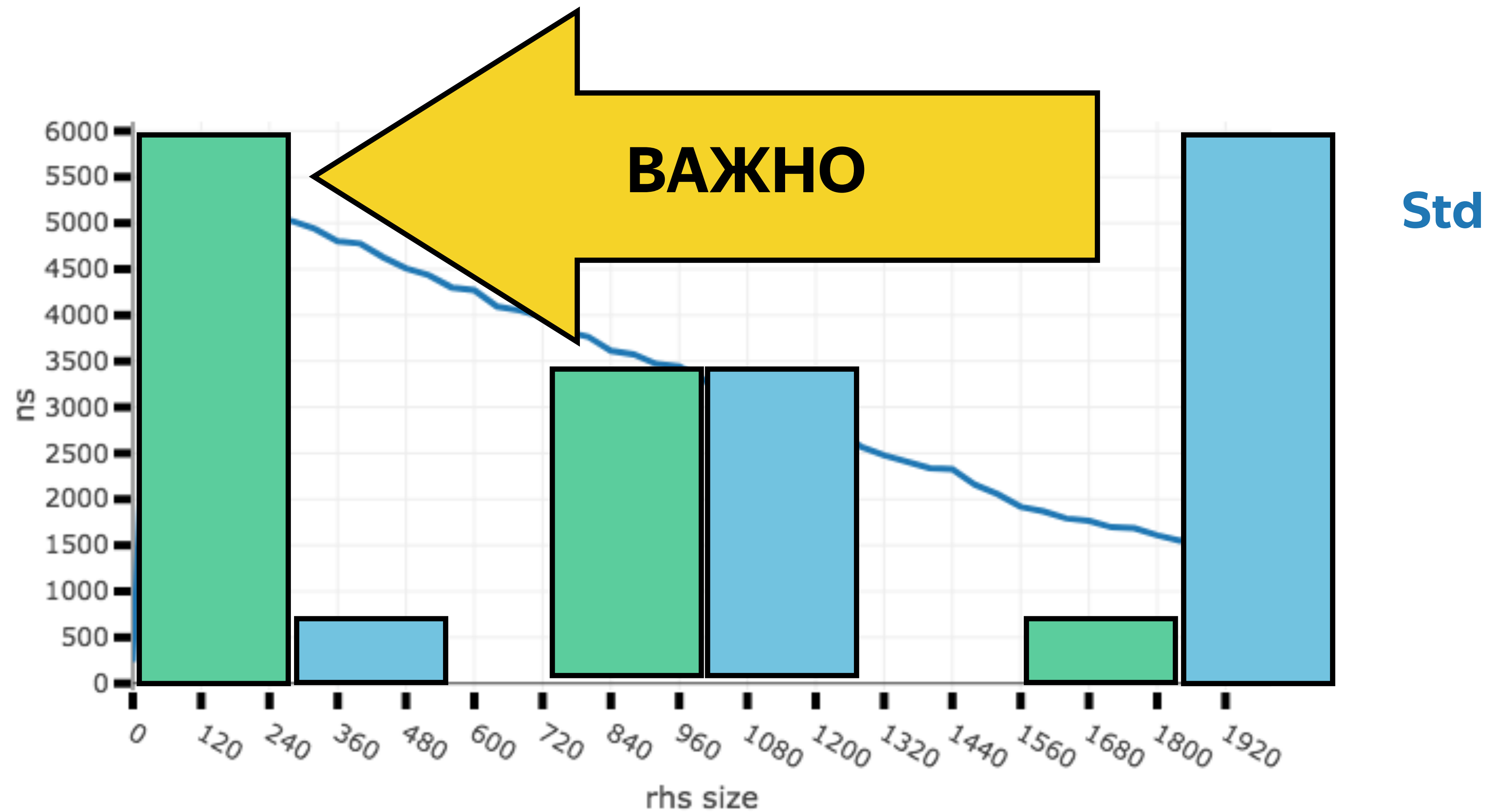


Std

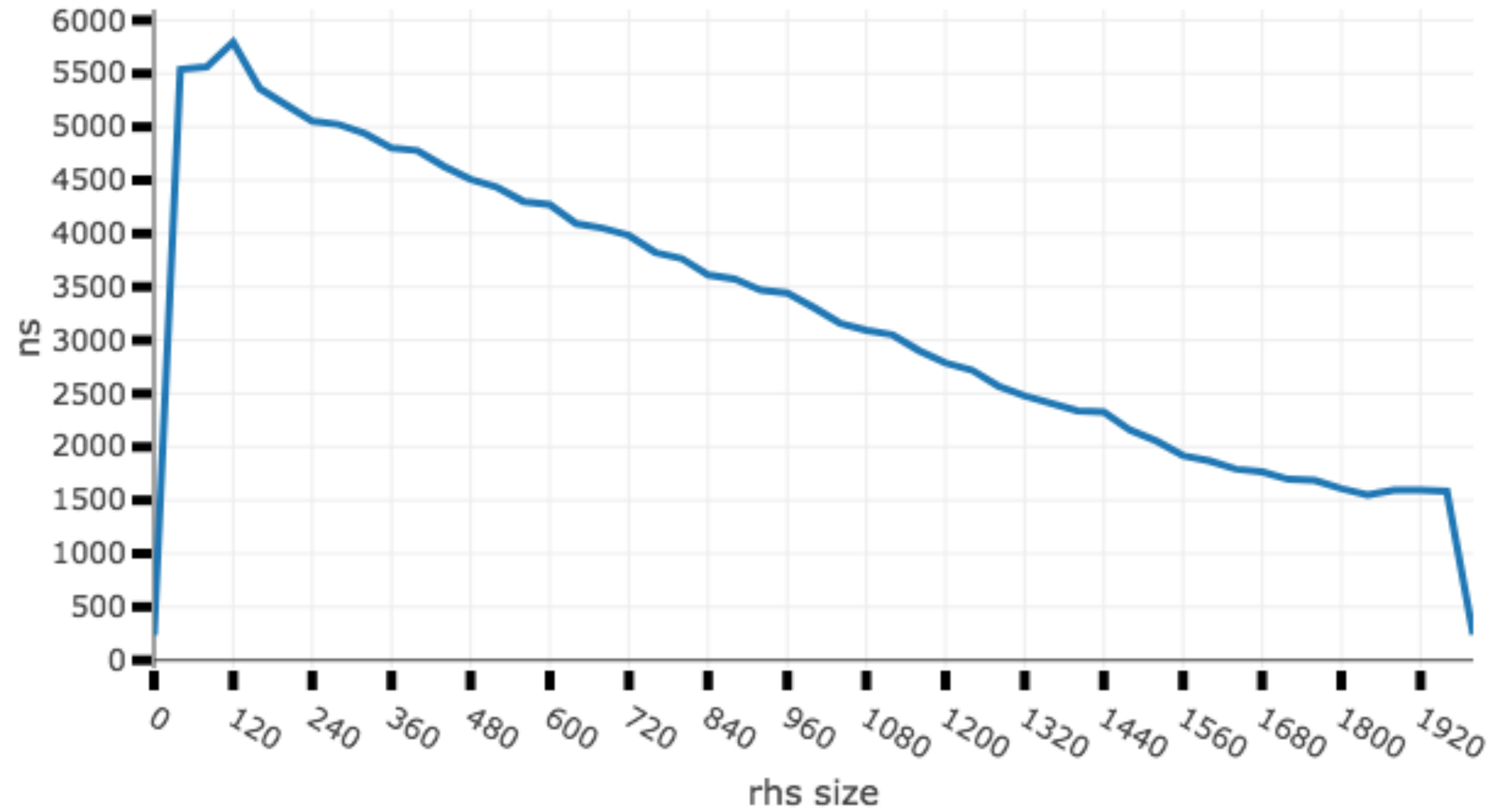
# std::set\_union



# std::set\_union



# std::set\_union



Std

# Проверка на границу

```
for ( ; f1 != l1; ++o ) {  
    if ( f2 == l2 )  
        ...  
}
```



# Проверка на границу

```
...  
if (comp(*f2, *f1)) {  
    *o++ = *f2++; if (f2 == l2) goto copyFirst;  
} else {  
    if (!comp(*f1, *f2)) {  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
}  
...
```

# Проверка на границу

```
...  
if (comp(*f2, *f1)) {  
    *o++ = *f2++; if (f2 == l2) goto copyFirst;  
} else {  
    if (!comp(*f1, *f2)) {  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
}  
...
```

# Проверка на границу

```
...  
if (comp(*f2, *f1)) {  
    *o++ = *f2++; if (f2 == l2) goto copyFirst;  
} else {  
    if (!comp(*f1, *f2)) {  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
}  
...
```

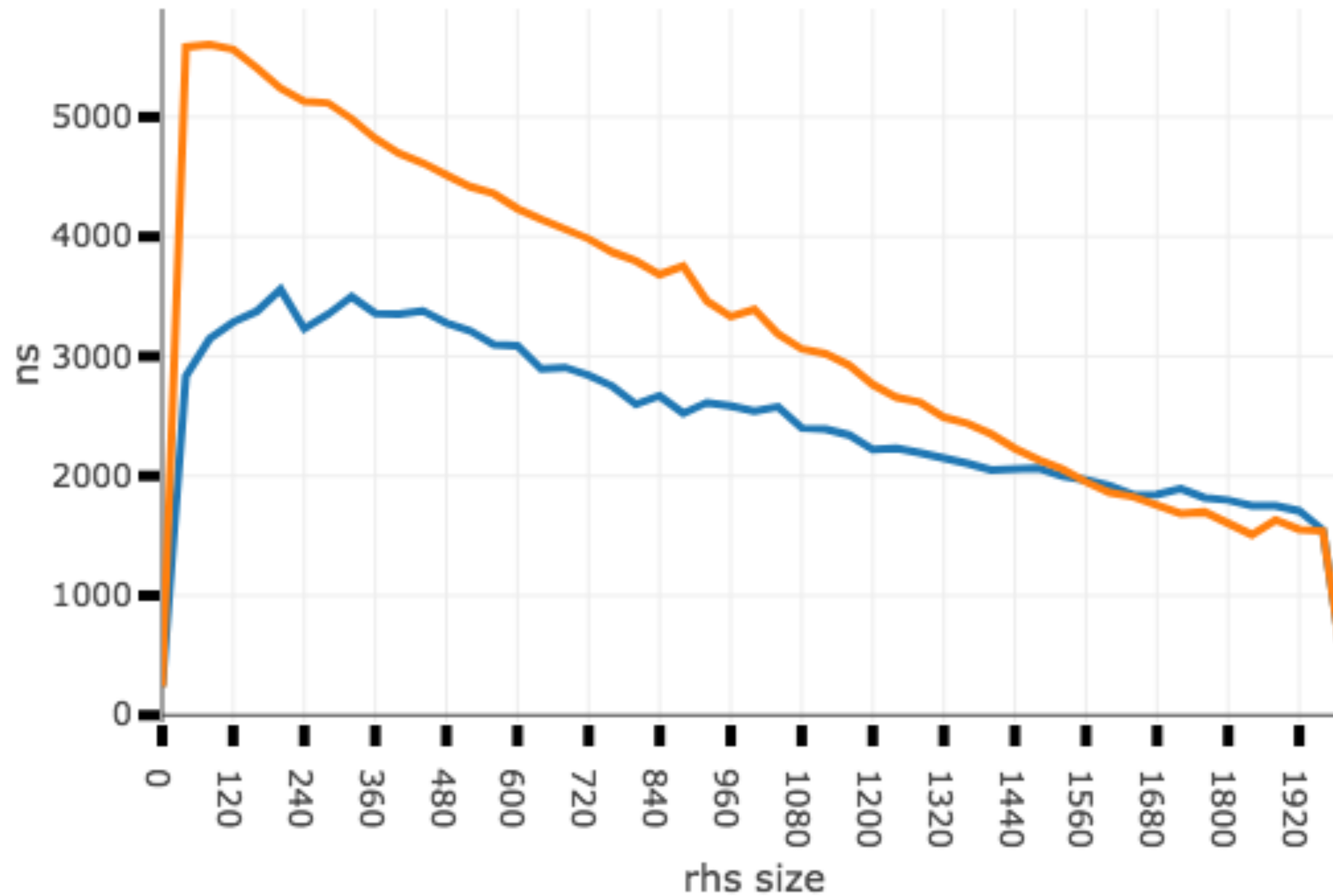
# Проверка на границу

```
template <class I1, class I2, class O, class Comp>
O set_union(I1 f1, I1 l1, I2 f2, I2 l2, O o, Comp comp) {
    if (f1 == l1) goto copySecond;
    if (f2 == l2) goto copyFirst;

    while (true) {
        if (comp(*f2, *f1)) {
            *o++ = *f2++; if (f2 == l2) goto copyFirst;
        } else {
            if (!comp(*f1, *f2)) {
                ++f2; if (f2 == l2) goto copyFirst;
            }
            *o++ = *f1++; if (f1 == l1) goto copySecond;
        }
    }
}

copySecond:
    return std::copy(f2, l2, o);
copyFirst:
    return std::copy(f1, l1, o);
}
```

# Проверка на границу



Было

Стало

# Правильный баланс

```
...  
if ( comp(*f2, *f1) ) {  
    *o++ = *f2++; if (f2 == l2) goto copyFirst;  
} else {  
    if ( !comp(*f1, *f2) ) {  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
}  
...
```

# Правильный баланс

```
while (true) {  
    if ( comp(*f1, *f2) ) {  
        *o++ = *f1++; if (f1 == l1) goto copySecond;  
    } else {  
        if ( comp(*f2, *f1) ) *o++ = *f2;  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
}
```

# Правильный баланс

```
while (true) {  
    if (__builtin_expect(comp(*f1, *f2), true)) {  
        *o++ = *f1++; if (f1 == l1) goto copySecond;  
    } else {  
        if (comp(*f2, *f1)) *o++ = *f2;  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
}
```



# Правильный баланс

```
while (true) {  
    if (__builtin_expect(comp(*f1, *f2), true)) {  
        *o++ = *f1++; if (f1 == l1) goto copySecond;  
    } else {  
        if (comp(*f2, *f1)) *o++ = *f2;  
        ++f2; if (f2 == l2) goto copyFirst;  
    }  
}
```

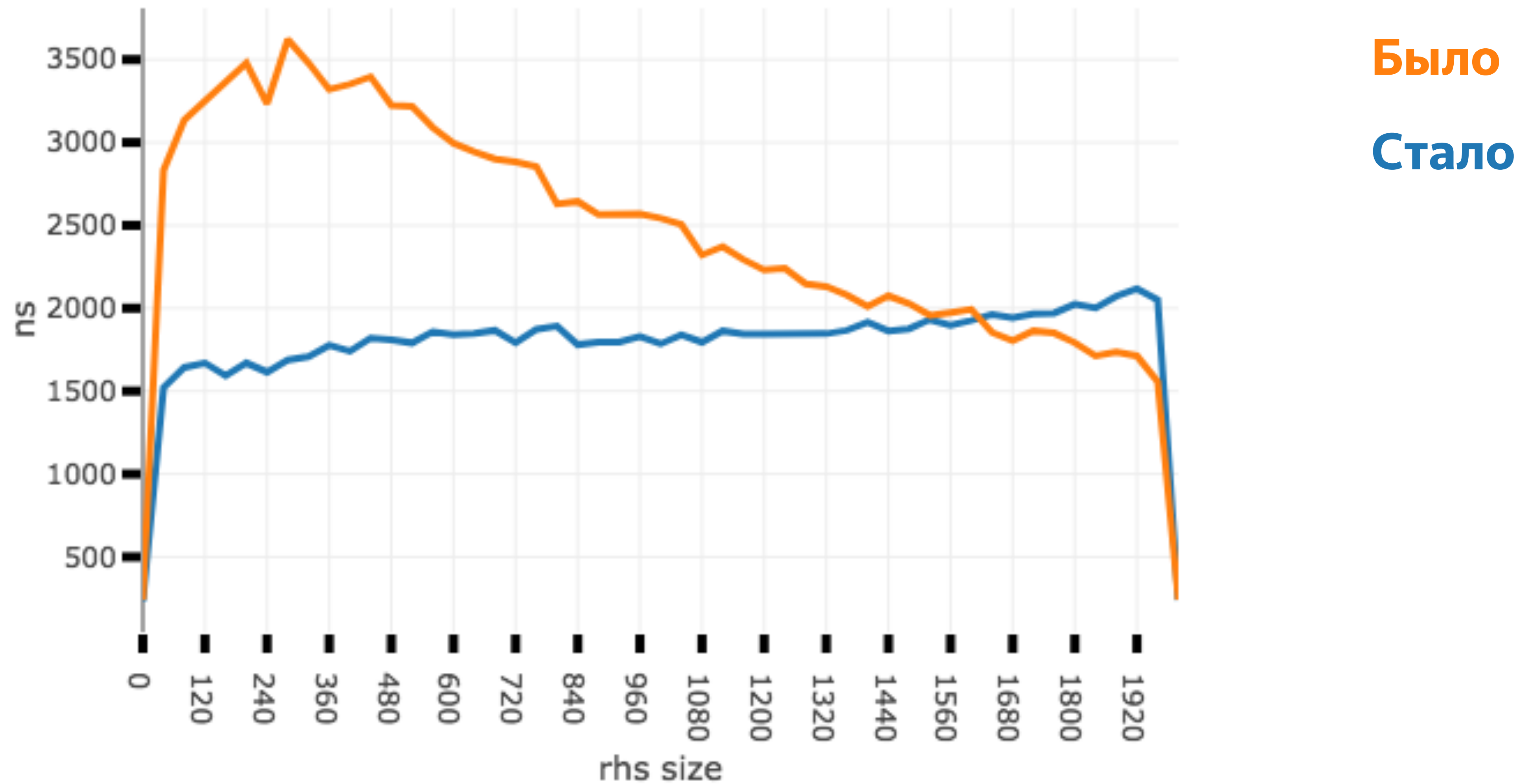
# Правильный баланс

```
template <class I1, class I2, class O, class Comp>
O set_union(I1 f1, I1 l1, I2 f2, I2 l2, O o, Comp comp) {
    if (f1 == l1) goto copySecond;
    if (f2 == l2) goto copyFirst;

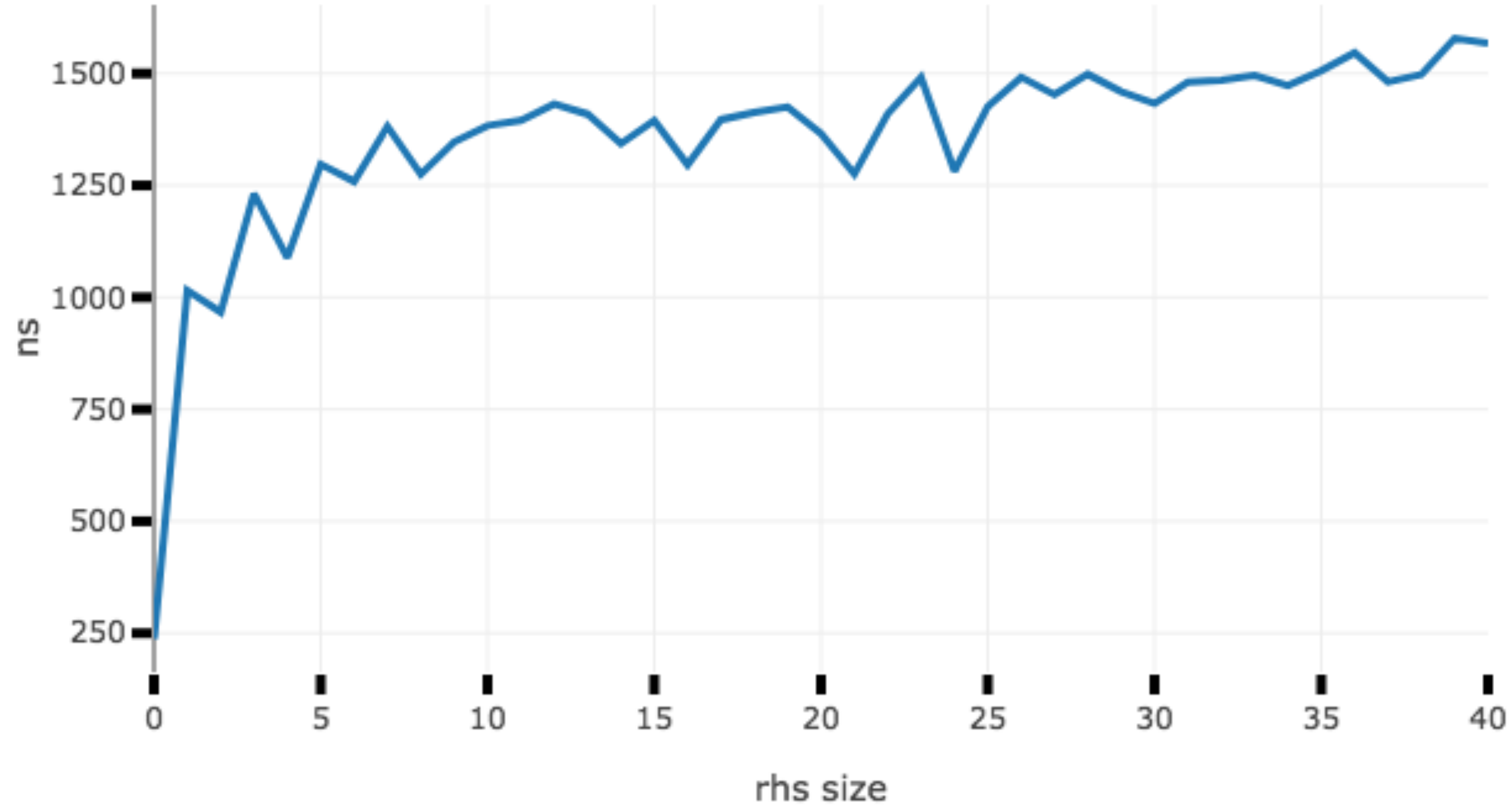
    while (true) {
        if (__builtin_expect(comp(*f1, *f2), true)) {
            *o++ = *f1++; if (f1 == l1) goto copySecond;
        } else {
            if (comp(*f2, *f1)) *o++ = *f2;
            ++f2; if (f2 == l2) goto copyFirst;
        }
    }
}

copySecond:
    return std::copy(f2, l2, o);
copyFirst:
    return std::copy(f1, l1, o);
}
```

# Правильный баланс



# Правильный баланс



Стало

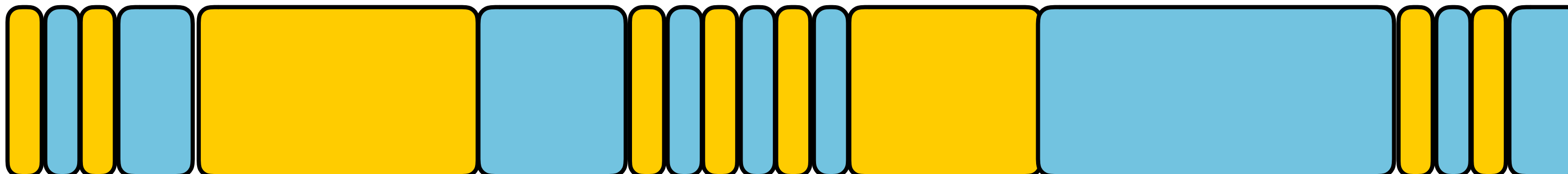
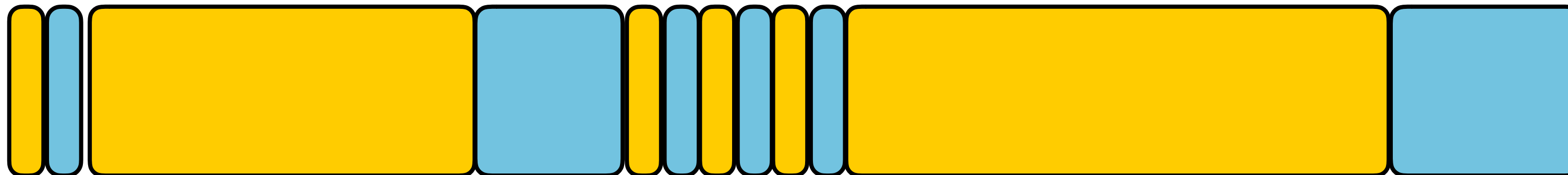
# Как пропускать сравнения?

```
while (true) {  
    if (__builtin_expect(comp(*f1, *f2), true)) {  
        *o++ = *f1++; if (f1 == l1) goto copySecond;  
    }  
    ...  
}
```

# Как пропускать сравнения?

```
...  
if next_f1 = skip_comparisons(f1, l1, *f2);  
std::copy(f1, next_f1, o);  
f1 = next_f1;  
...
```

# Эвристики на границы



# Как пропускать сравнения?

```
...  
if ( !comp(*f1, *f2) ) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
  
I1 next_f1 = skip_comparisons(f1, l1, *f2);  
...
```



# Как пропускать сравнения?

```
if ( !comp(*f1, *f2) ) goto checkSecond;  
*o++ = *f1++; if ( f1 == l1 ) goto copySecond;  
if ( !comp(*f1, *f2) ) goto checkSecond;  
*o++ = *f1++; if ( f1 == l1 ) goto copySecond;
```

```
I1 next_f1 = skip_comparisons(f1, l1, *f2);
```

# Как пропускать сравнения?

```
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
// two more...
```

```
I1 next_f1 = skip_comparisons(f1, l1, *f2);
```

**start:**

```
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
// two more...
```

**goto start;**

# Разворачивание цикла

**goto start;**

**checkSecond:**

```
if (comp(*f2, *f1)) *o++ = *f2;  
++f2; if (f2 == l2) goto copyFirst;
```

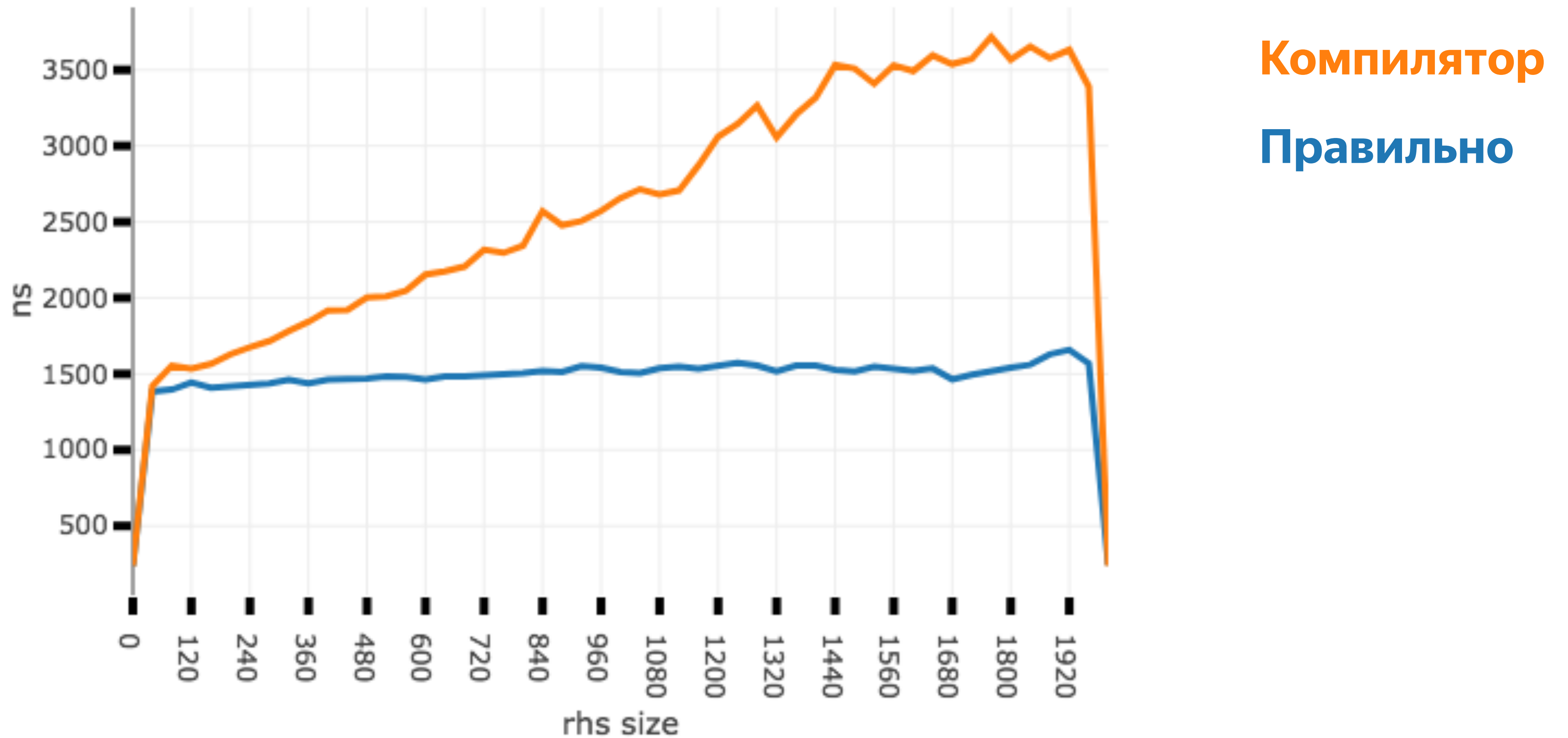
**start:**

```
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
if (!comp(*f1, *f2)) goto checkSecond;  
*o++ = *f1++; if (f1 == l1) goto copySecond;  
goto start;
```

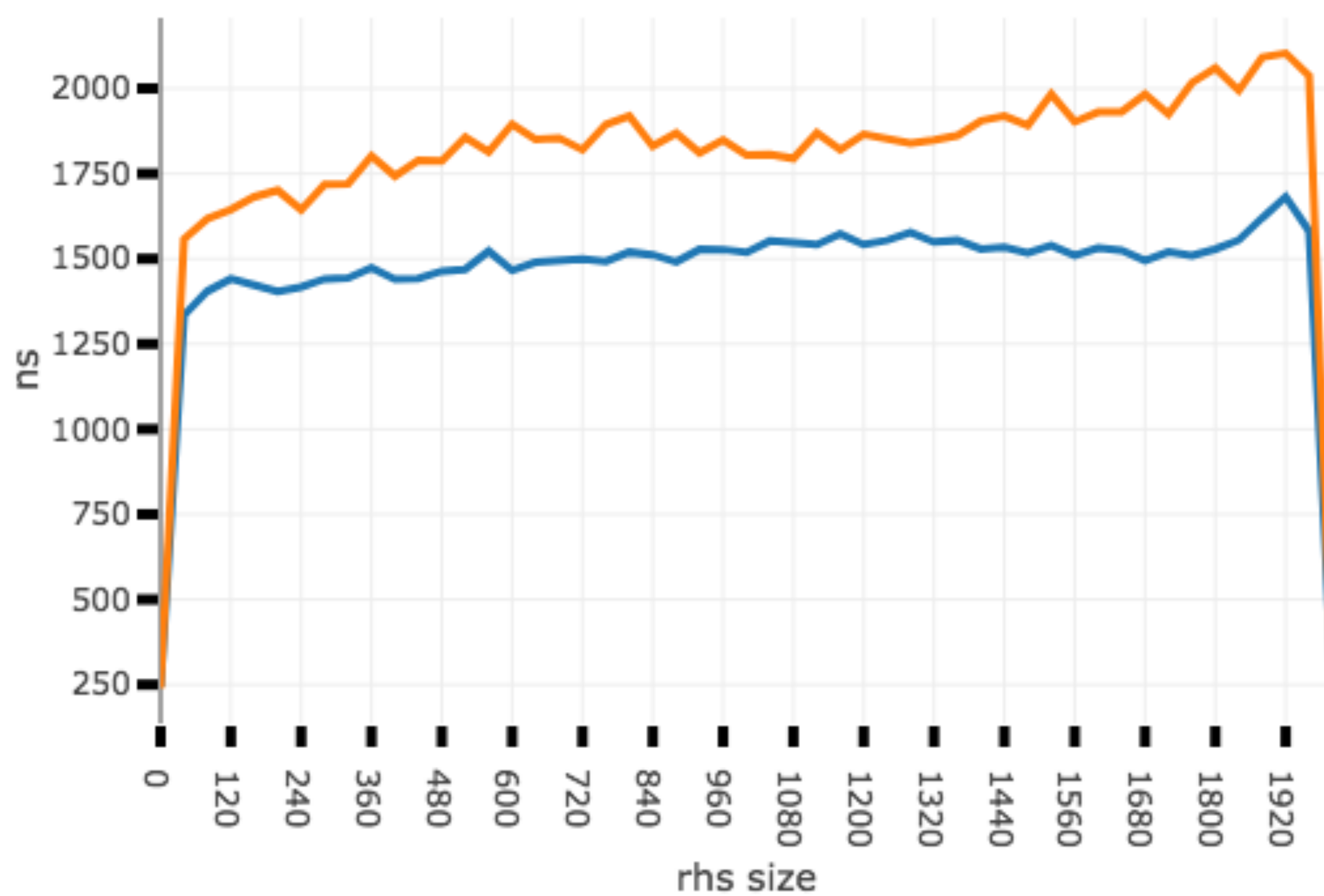
# Разворачивание цикла

```
while (true) {  
    if (!comp(*f1, *f2)) goto checkSecond;  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
    goto biased;  
  
checkSecond:  
    if (comp(*f2, *f1)) *o++ = *f2;  
    ++f2; if (f2 == l2) goto copyFirst;  
  
biased:  
    if (!comp(*f1, *f2)) goto checkSecond;  
    *o++ = *f1++; if (f1 == l1) goto copySecond;  
    // two more  
}
```

# Победа над компилятором



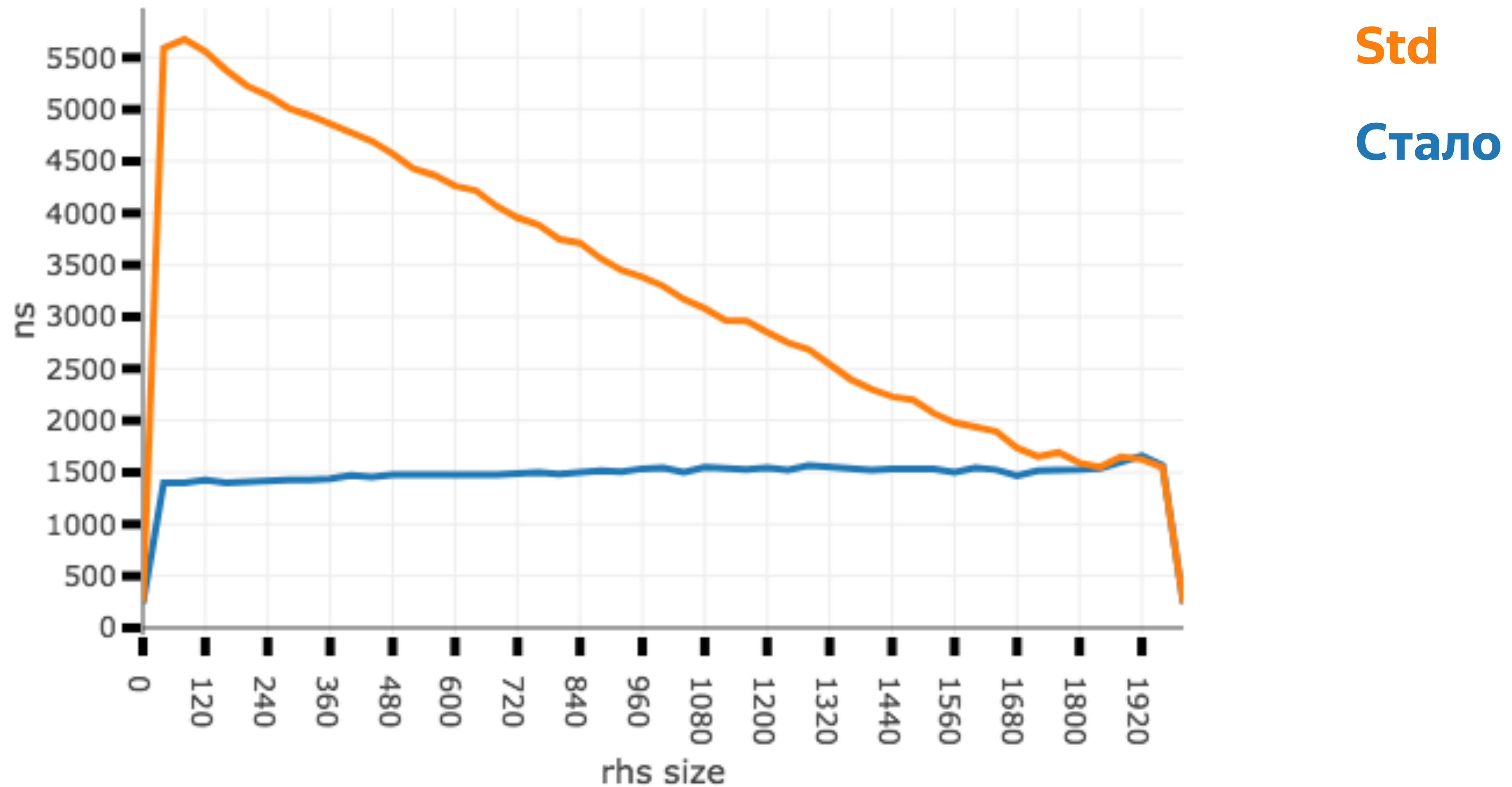
# Разворачивание цикла



Было

Стало

# Сравнение с std



[https://bugs.llvm.org/show\\_bug.cgi?id=35499](https://bugs.llvm.org/show_bug.cgi?id=35499)



# Пропуск сравнений

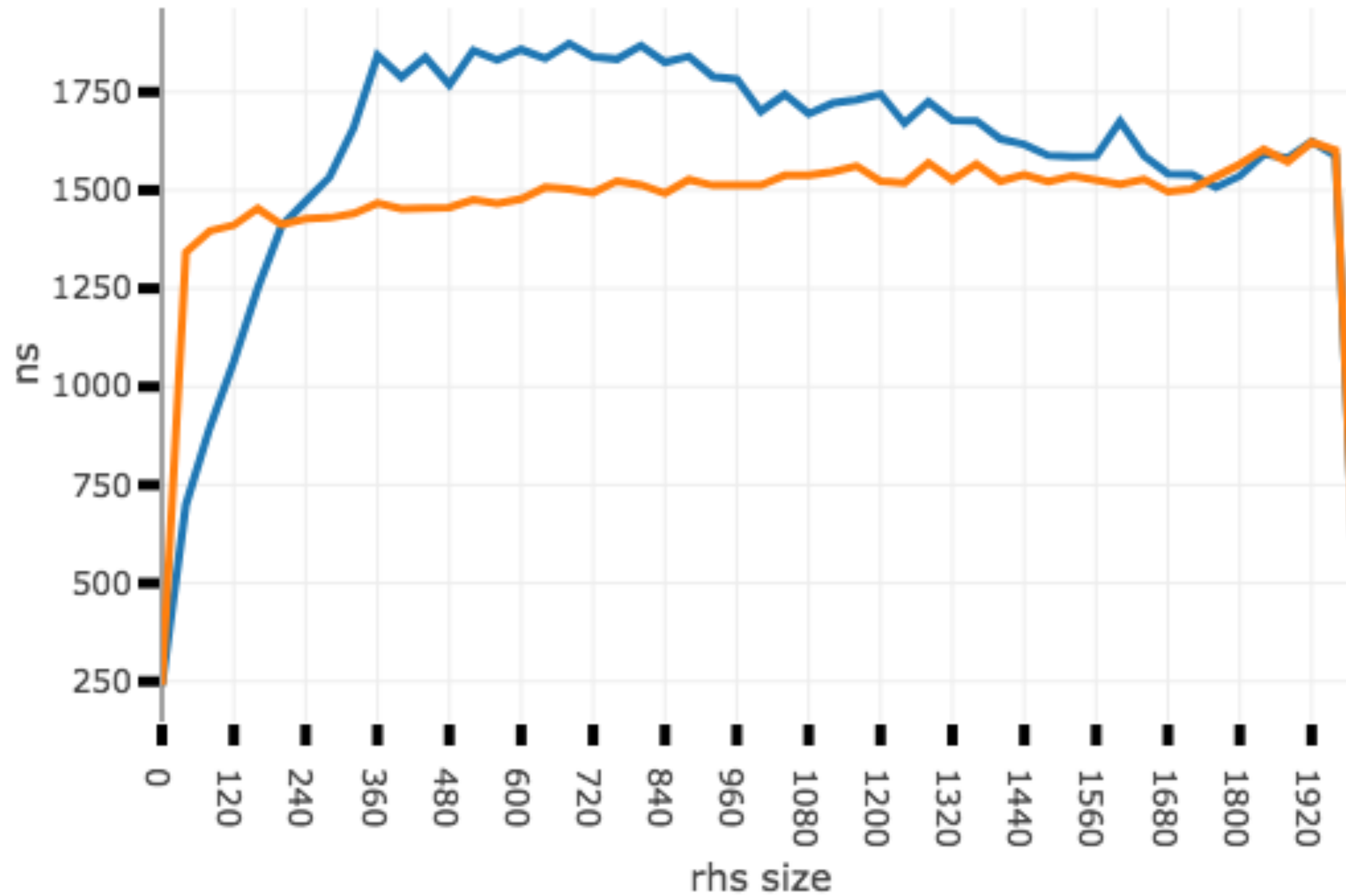
```
while (true) {
    if (!comp(*f1, *f2)) goto checkSecond;
    *o++ = *f1++; if (f1 == l1) goto copySecond;
    goto biased;

checkSecond:
    if (comp(*f2, *f1)) *o++ = *f2;
    ++f2; if (f2 == l2) goto copyFirst;

biased:
    if (!comp(*f1, *f2)) goto checkSecond;
    *o++ = *f1++; if (f1 == l1) goto copySecond;
    // 3 more

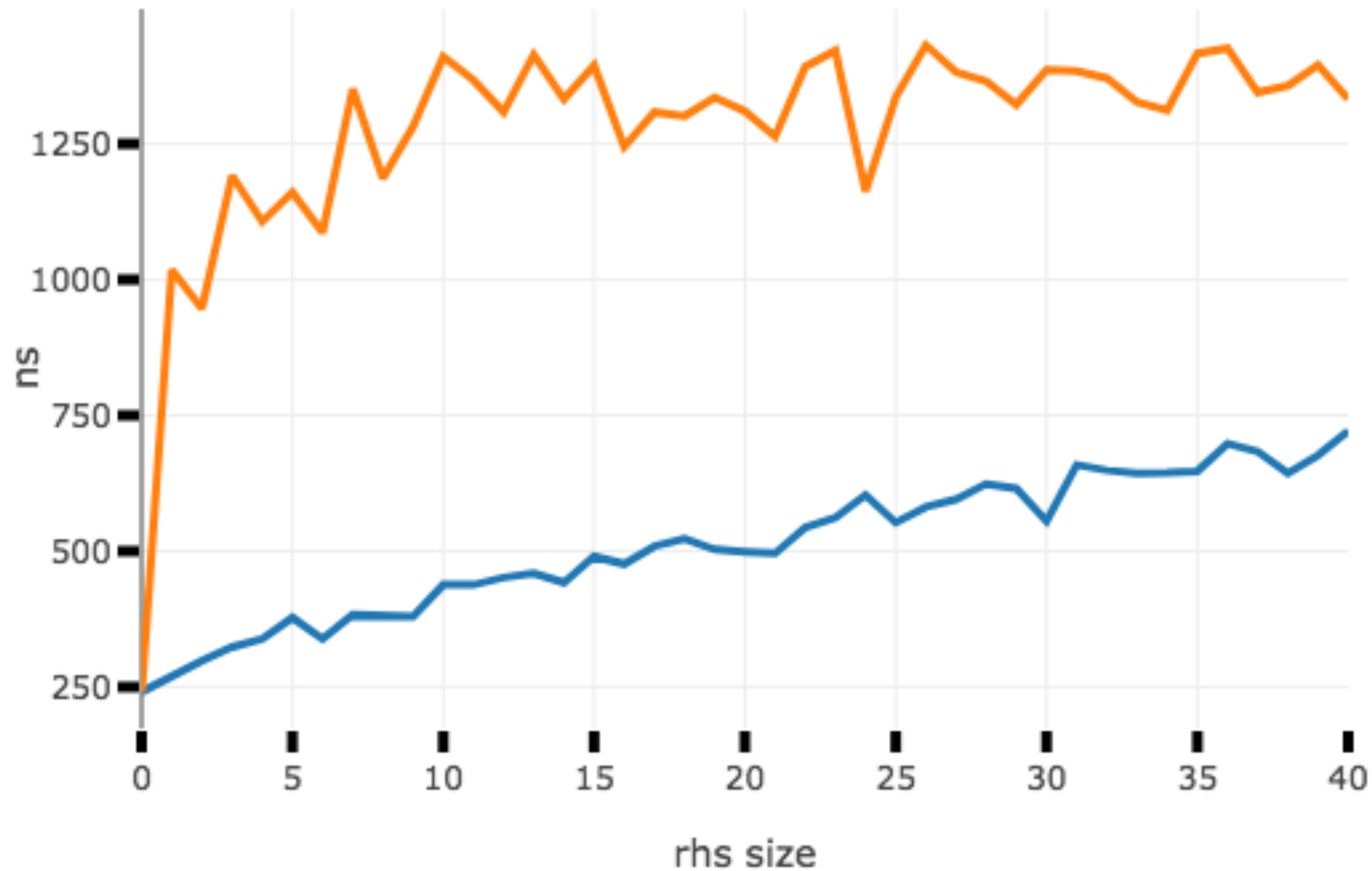
    I1 segment_end = find_boundary(f1, l1, [&](const auto& x) { return comp(x, *f2); });
    o = std::copy(f1, segment_end, o);
    f1 = segment_end;
}
```

# Пропуск сравнений



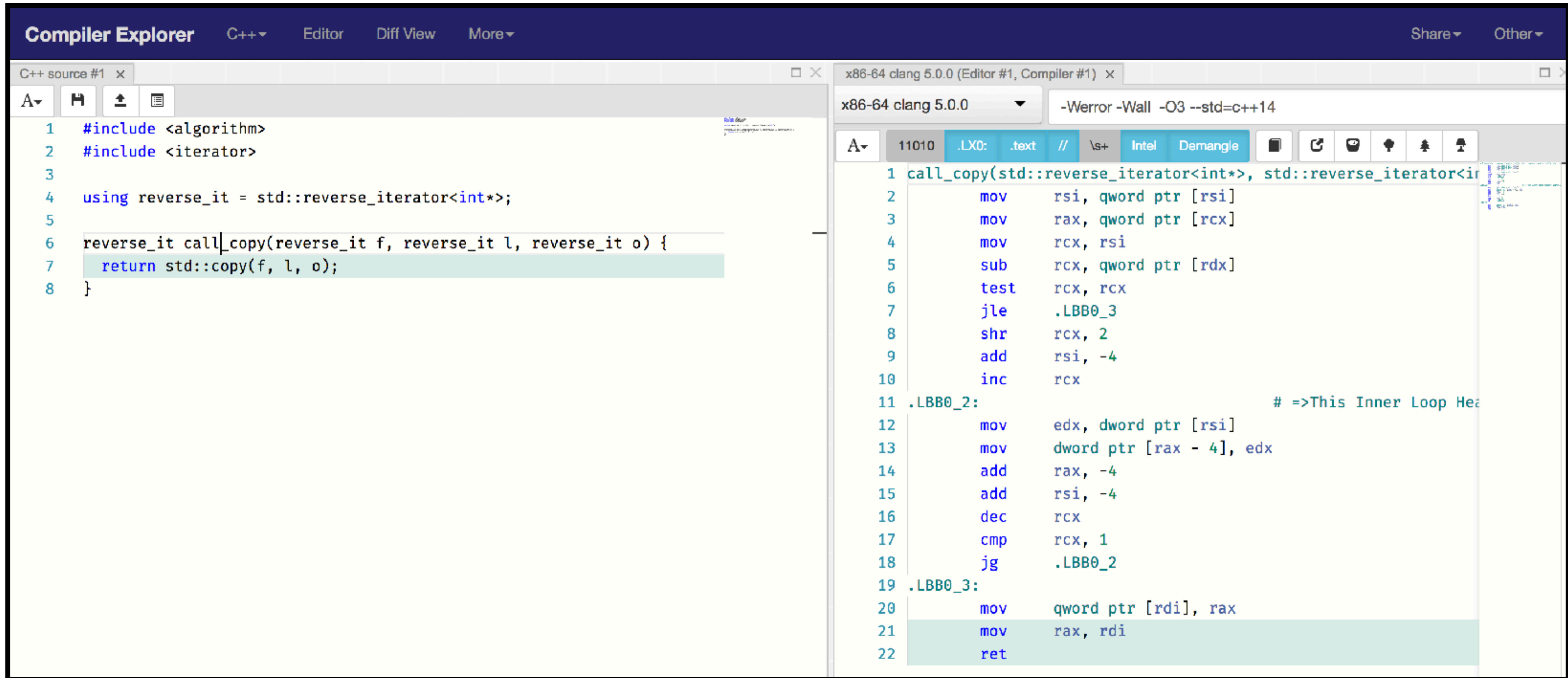
Линейный  
Пропускающая

# Пропуск сравнений (40)



**Линейный**  
**Пропуская**

# reverse\_iterator

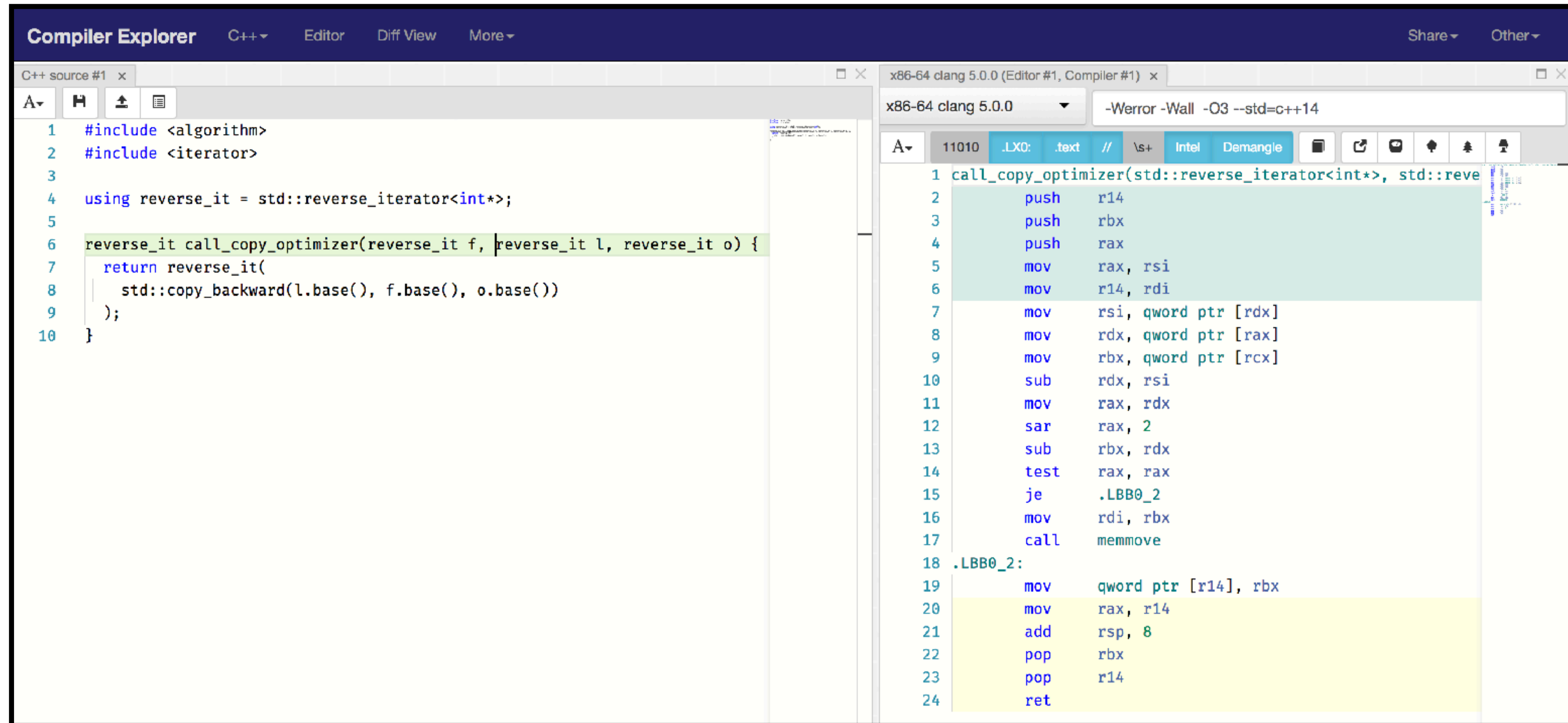


The image shows a screenshot of the Compiler Explorer interface. The left pane displays the C++ source code for a reverse\_iterator wrapper. The right pane shows the x86-64 assembly code generated by clang 5.0.0. The assembly code includes a call to std::copy, a loop to decrement pointers, and a final store and return instruction.

```
1  #include <algorithm>
2  #include <iterator>
3
4  using reverse_it = std::reverse_iterator<int*>;
5
6  reverse_it call_copy(reverse_it f, reverse_it l, reverse_it o) {
7      return std::copy(f, l, o);
8  }
```

```
1  call_copy(std::reverse_iterator<int*>, std::reverse_iterator<int*>, std::reverse_iterator<int*>)
2      mov     rsi, qword ptr [rsi]
3      mov     rax, qword ptr [rcx]
4      mov     rcx, rsi
5      sub     rcx, qword ptr [rdx]
6      test    rcx, rcx
7      jle     .LBB0_3
8      shr     rcx, 2
9      add     rsi, -4
10     inc     rcx
11 .LBB0_2:                                     # =>This Inner Loop Header
12     mov     edx, dword ptr [rsi]
13     mov     dword ptr [rax - 4], edx
14     add     rax, -4
15     add     rsi, -4
16     dec     rcx
17     cmp     rcx, 1
18     jg      .LBB0_2
19 .LBB0_3:
20     mov     qword ptr [rdi], rax
21     mov     rax, rdi
22     ret
```

# reverse\_iterator

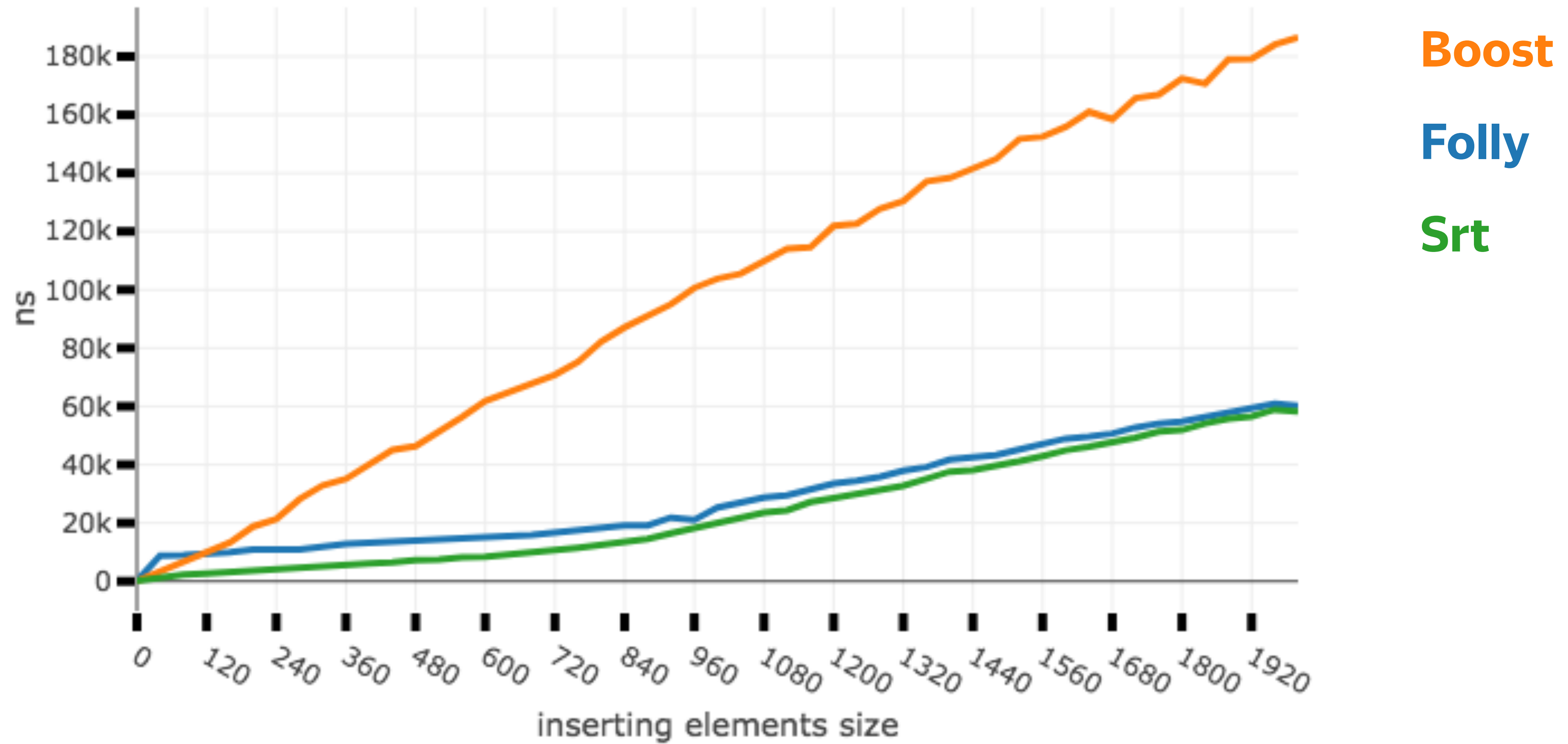


The image shows a screenshot of the Compiler Explorer interface. The left pane displays the C++ source code for a reverse iterator wrapper. The right pane shows the assembly code generated by x86-64 clang 5.0.0 with optimization level -O3 and standard --std=c++14. The assembly code includes a call to a memmove function and a return statement.

```
C++ source #1 x
1 #include <algorithm>
2 #include <iterator>
3
4 using reverse_it = std::reverse_iterator<int*>;
5
6 reverse_it call_copy_optimizer(reverse_it f, reverse_it l, reverse_it o) {
7     return reverse_it(
8         std::copy_backward(l.base(), f.base(), o.base())
9     );
10 }
```

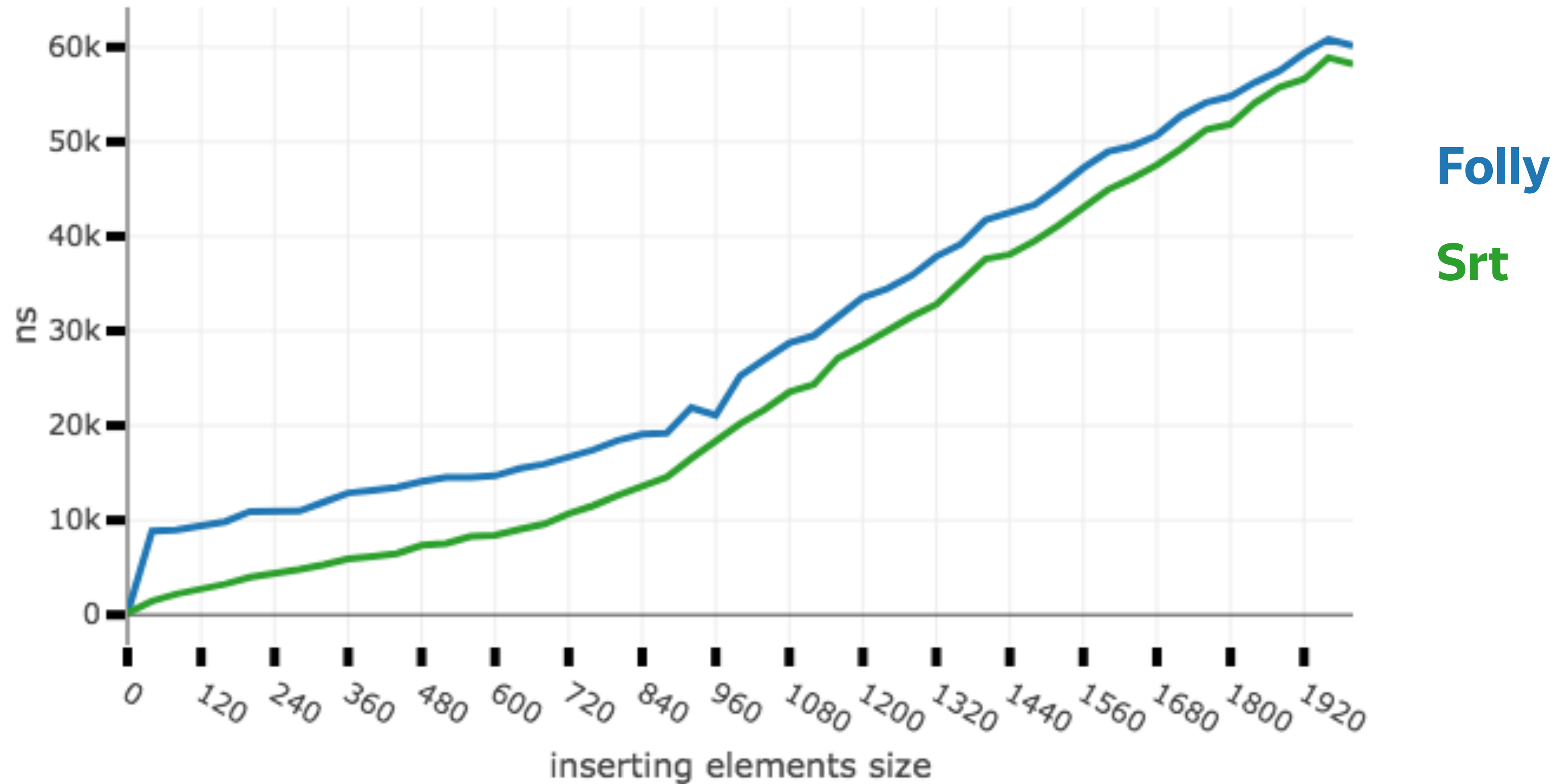
```
x86-64 clang 5.0.0 (Editor #1, Compiler #1) x
x86-64 clang 5.0.0 -Werror -Wall -O3 --std=c++14
11010 .LX0: .text // \s+ Intel Demangle
1 call_copy_optimizer(std::reverse_iterator<int*>, std::reve
2     push    r14
3     push    rbx
4     push    rax
5     mov     rax, rsi
6     mov     r14, rdi
7     mov     rsi, qword ptr [rdx]
8     mov     rdx, qword ptr [rax]
9     mov     rbx, qword ptr [rcx]
10    sub     rdx, rsi
11    mov     rax, rdx
12    sar     rax, 2
13    sub     rbx, rdx
14    test    rax, rax
15    je      .LBB0_2
16    mov     rdi, rbx
17    call    memmove
18 .LBB0_2:
19    mov     qword ptr [r14], rbx
20    mov     rax, r14
21    add     rsp, 8
22    pop     rbx
23    pop     r14
24    ret
```

# Srt vs Boost vs Folly

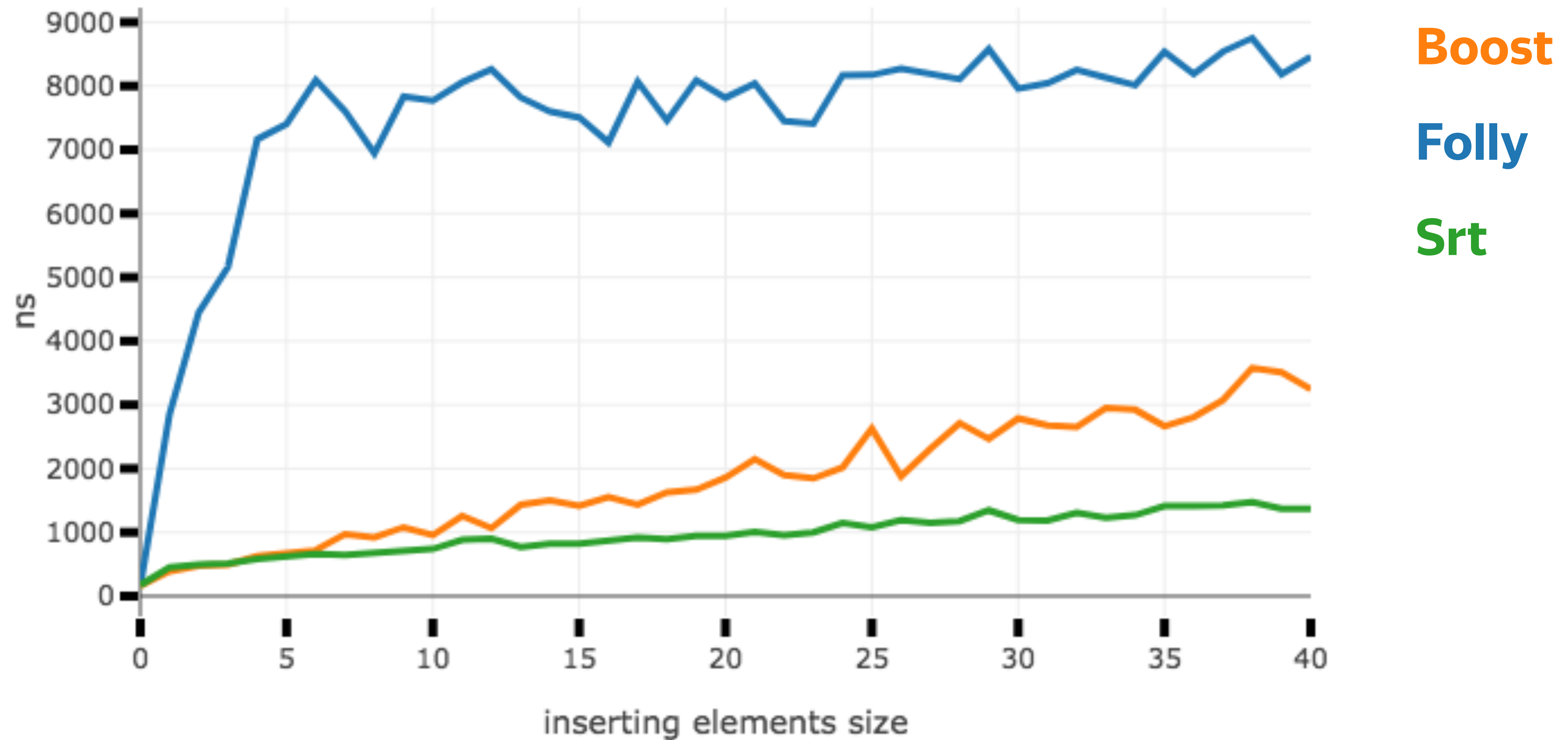




# Srt vs Folly

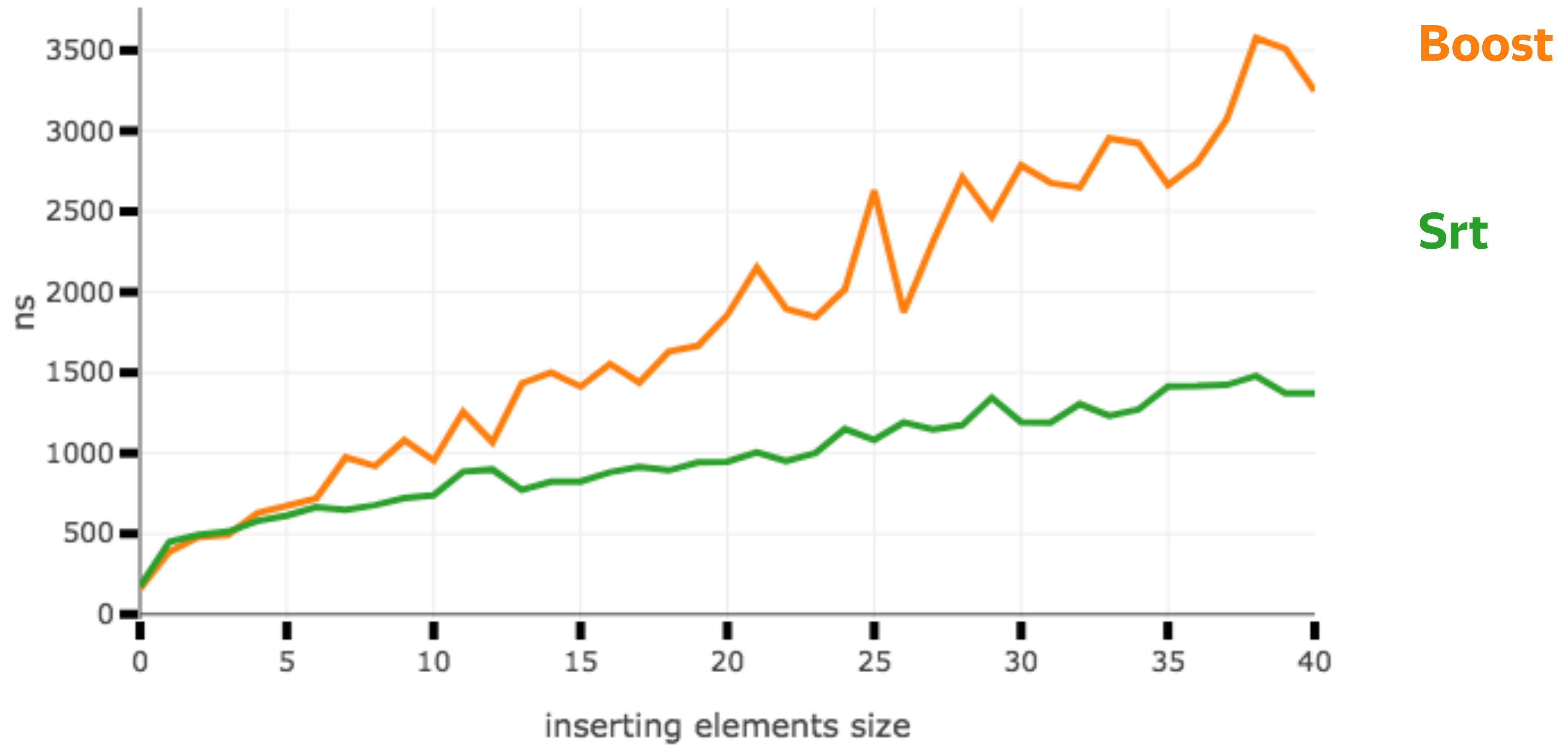


# Srt vs Boost vs Folly (40)





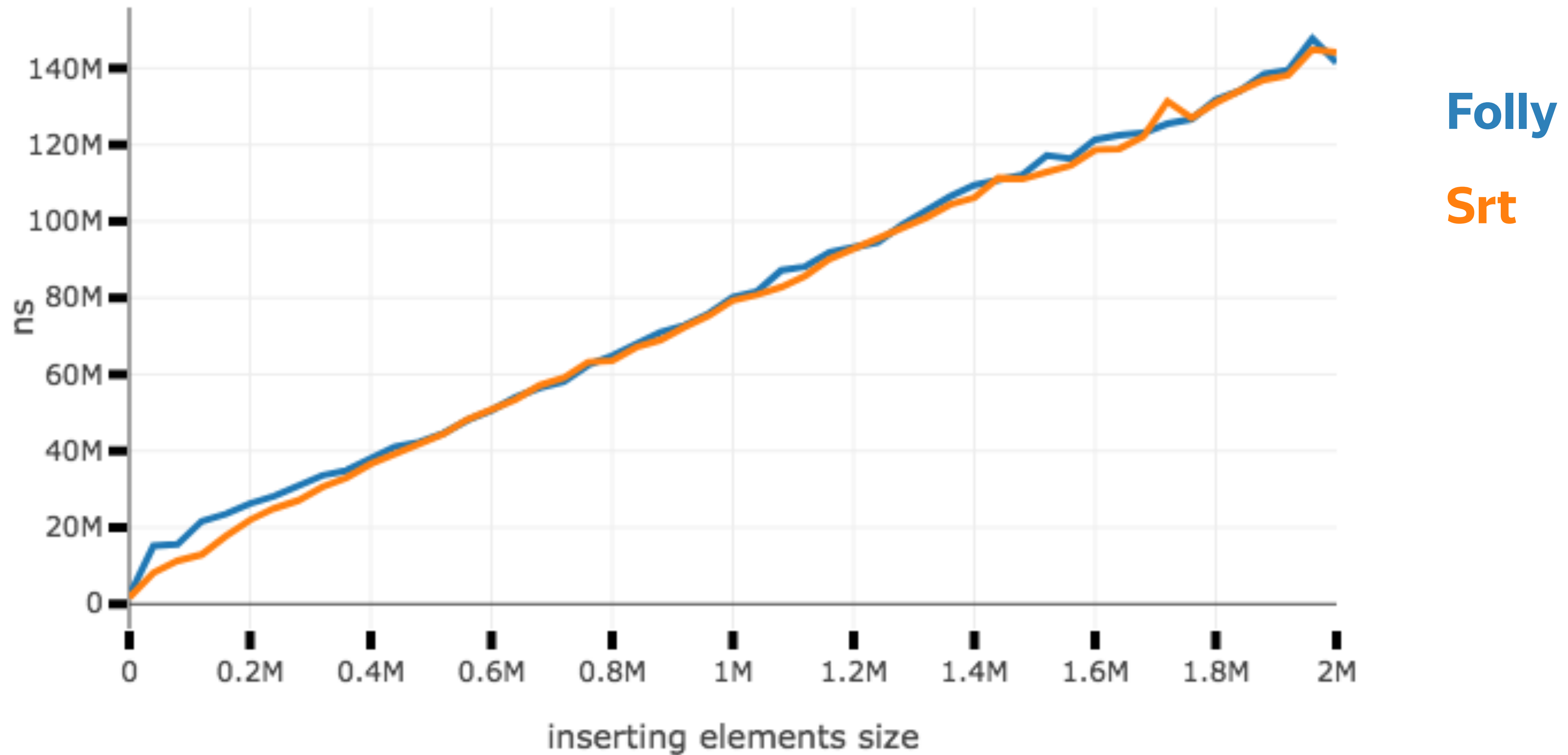
# Srt vs Boost (40)



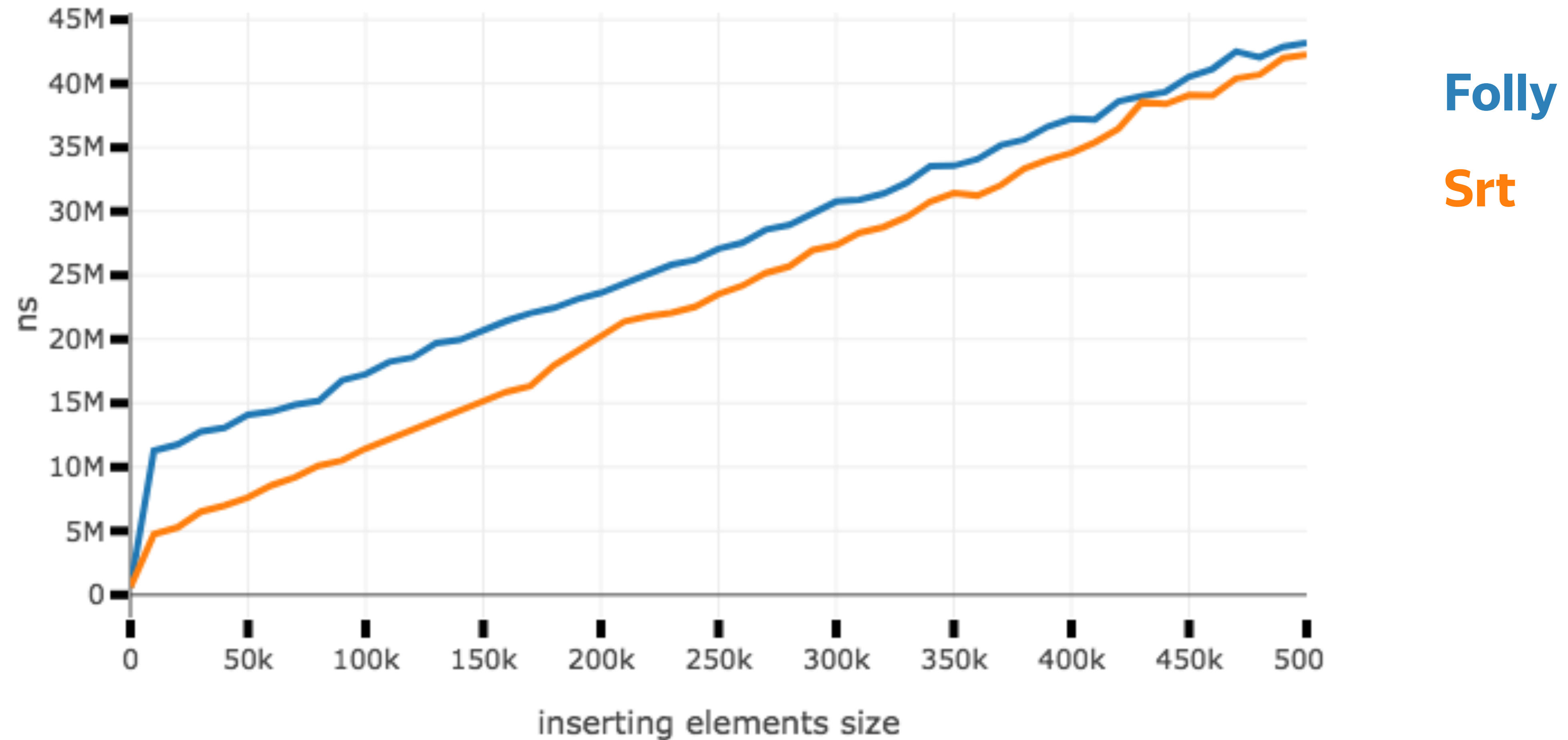
# Размер в инструкциях

srt::flat_set	973
boost::flat_set	290
folly::sorted_vector_set	> 1500

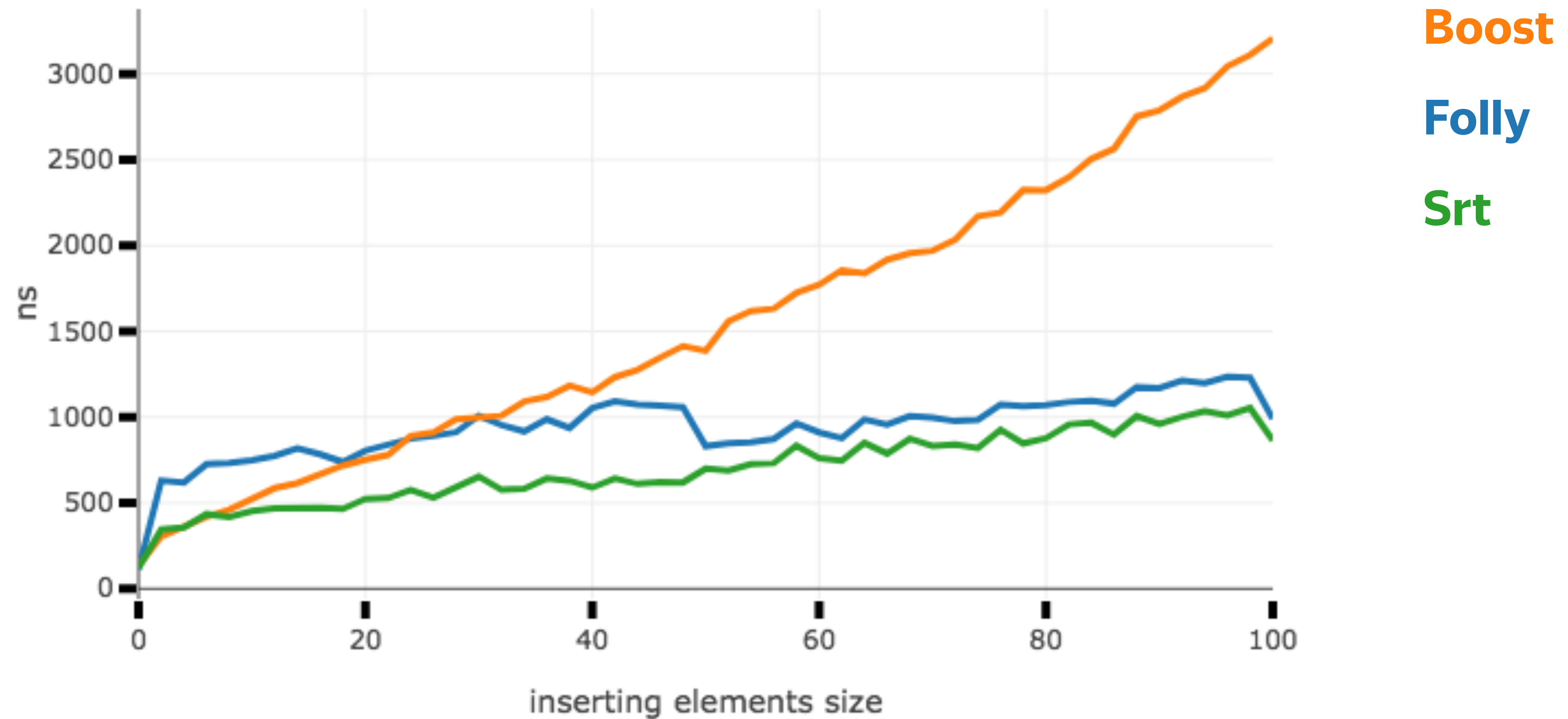
# Srt vs Folly - 2 миллиона



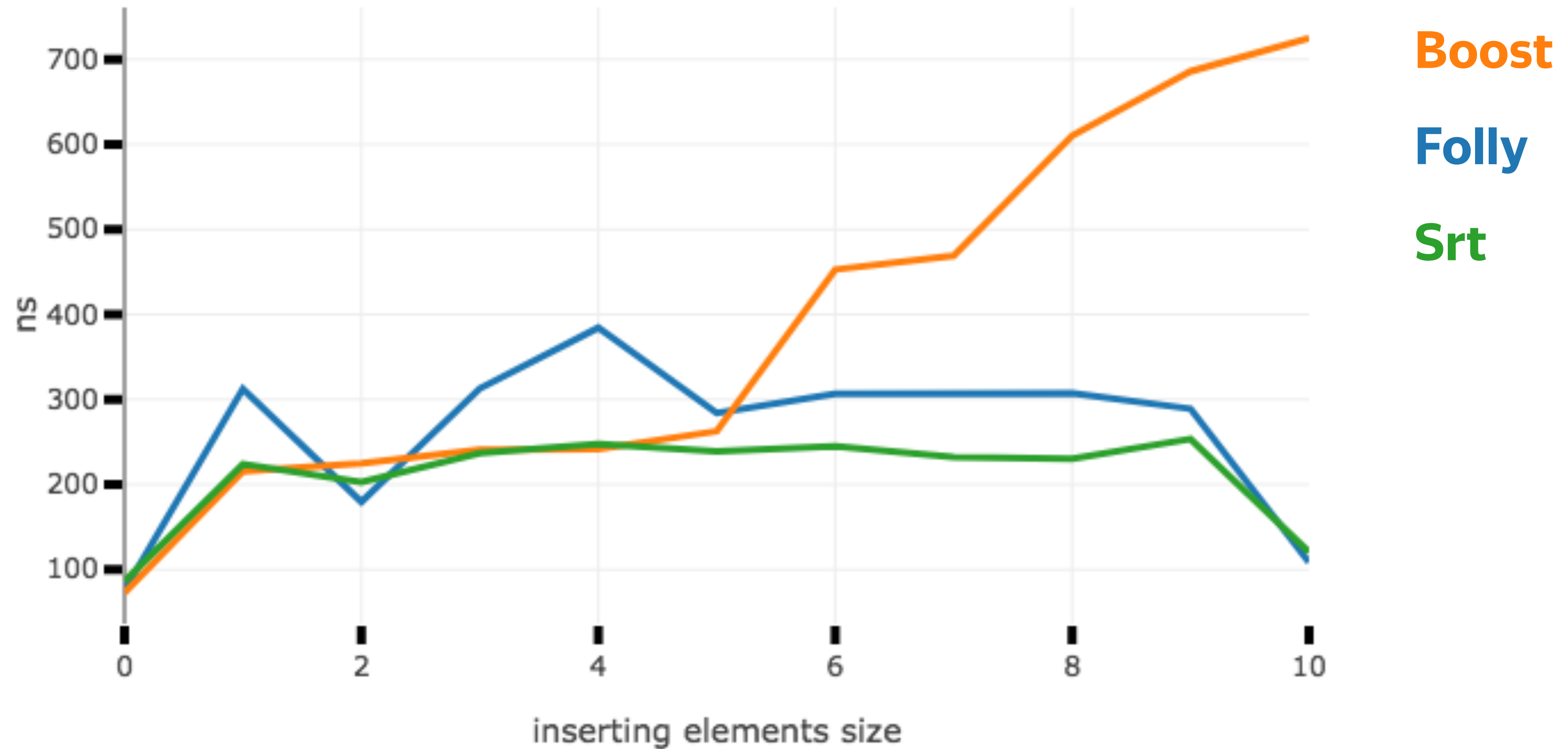
# Srt vs Folly - 2 миллиона (первые 500 тысяч)



# Srt vs Boost vs Folly (100)



# Srt vs Boost vs Folly (10)



# Пропуск элементов - худший случай

1000 int (64 bit), каждый бой элемент во втором массиве

Линейный	771 нс
Пропускающий	1545 нс

# Благодарности

- › Алексей Лошкарев
- › Зиновий Нис



<https://github.com/DenisYaroshevskiy/srt-library>

Ярошевский Денис

Разработчик



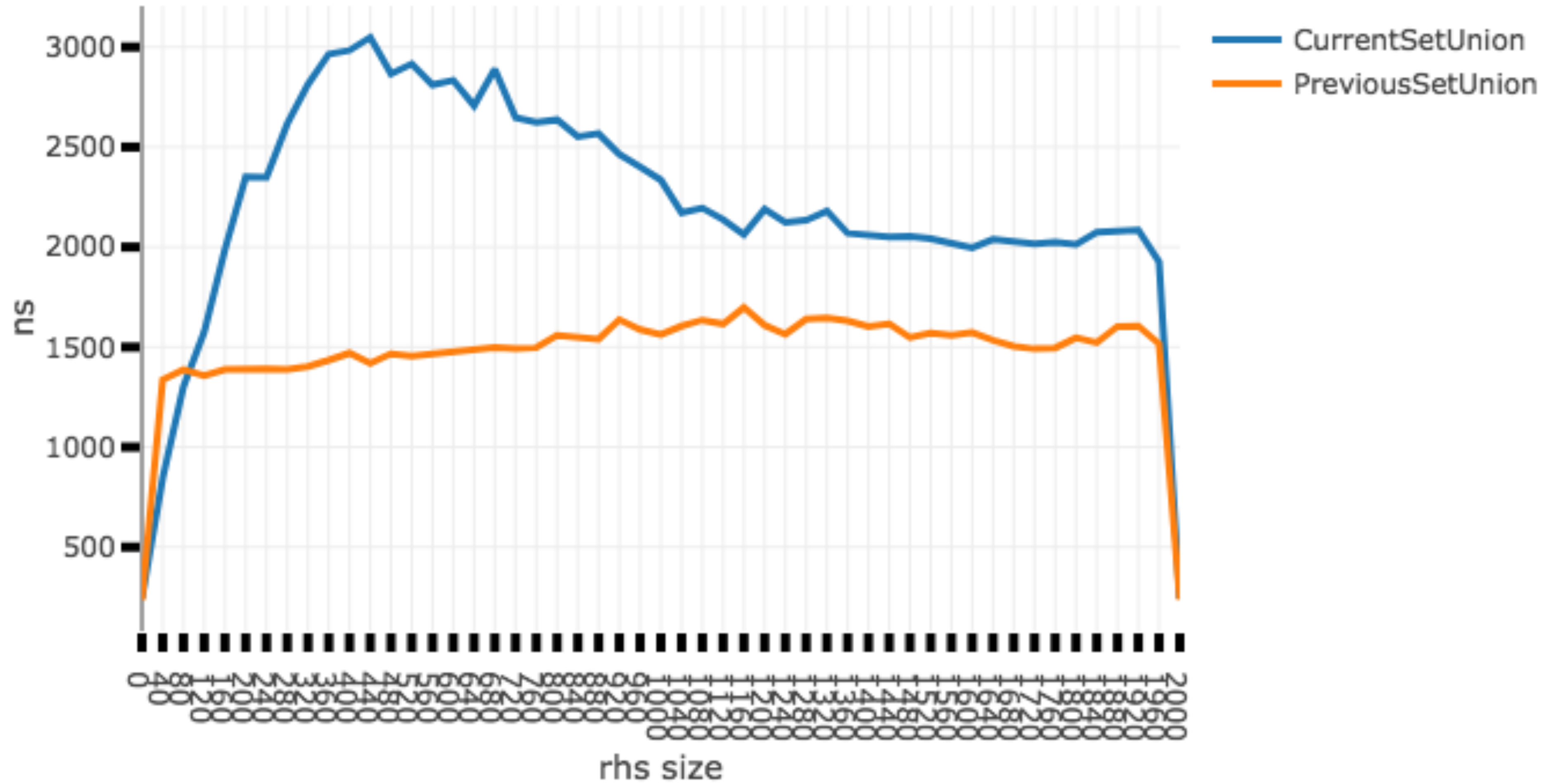
dyaroshev@yandex-team.ru

# std::lower\_bound

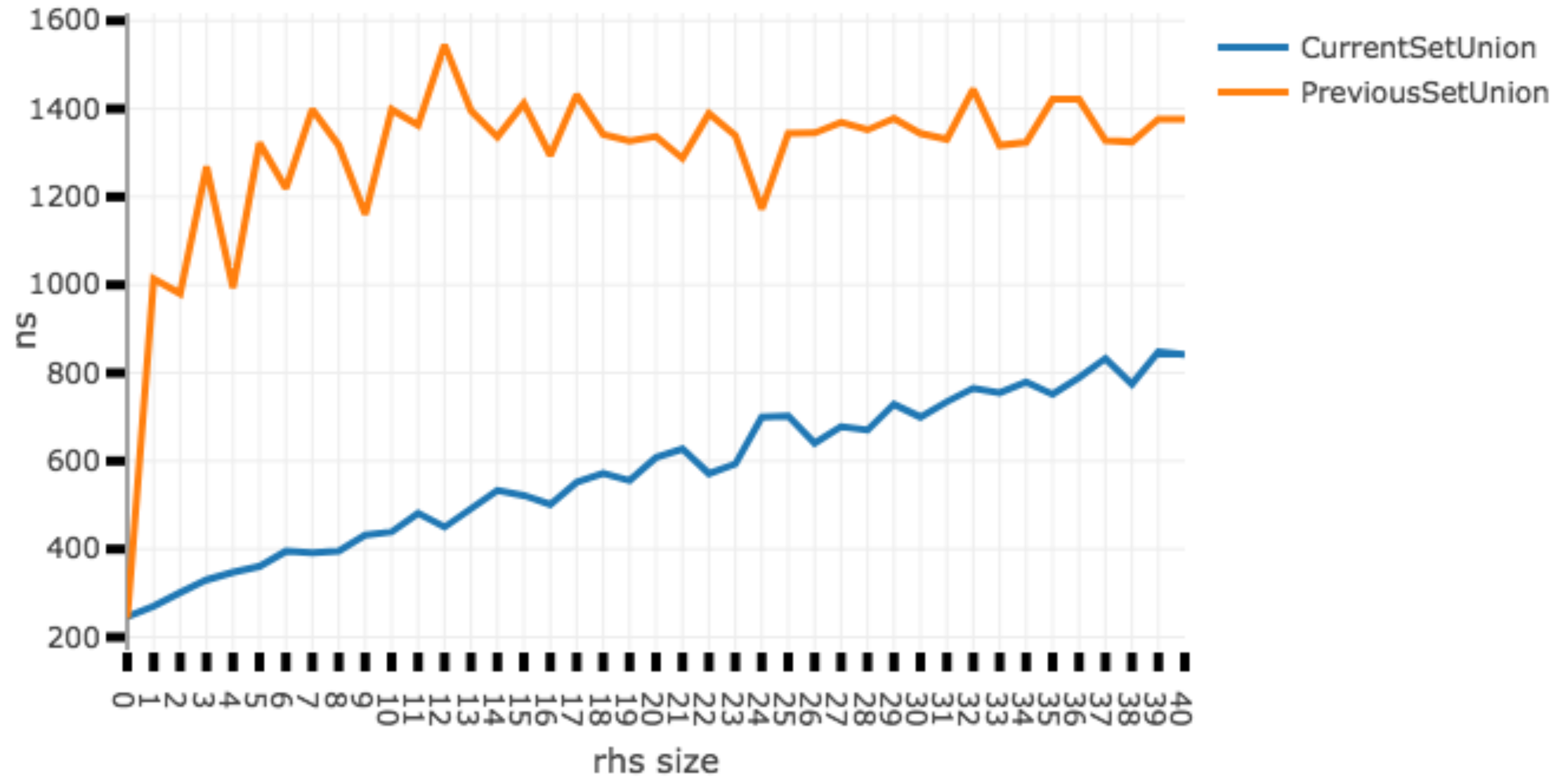
```
// unrolled loop here
if (!comp(*f1, *f2)) goto checkSecond;
*o++ = *f1++; if (f1 == l1) goto copySecond;

next_f1 = std::lower_bound(f1, l1, *f2, comp);
o = std::copy(f1, next_f1, o);
f1 = next_f1; if (f1 == l1) goto copySecond;
goto checkSecond;
```

# std::lower\_bound



# std::lower\_bound( 40 )



partition\_point\_biased

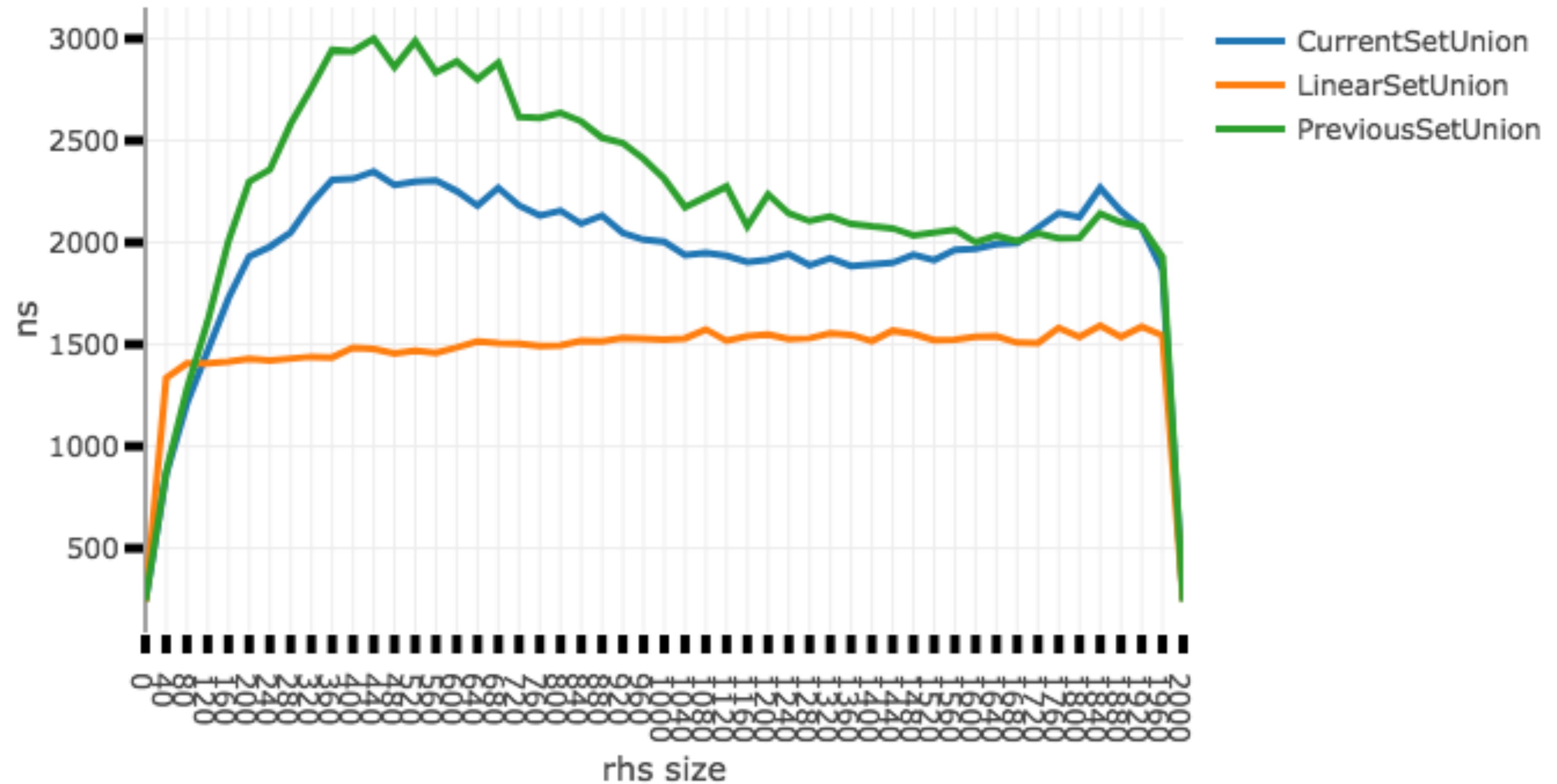
# partition\_point\_biased

```
template <typename I, typename P>
I partition_point_biased(I f, I l, P p) {
    auto len = std::distance(f, l);
    int step = 1;
    while (len > step) {
        I test = std::next(f, step);
        if (!p(*test)) {
            l = test;
            break;
        }
        f = ++test;
        len -= step + 1;
        step += step;
    }
    return std::partition_point(f, l, p);
}
```

# partition\_point\_biased

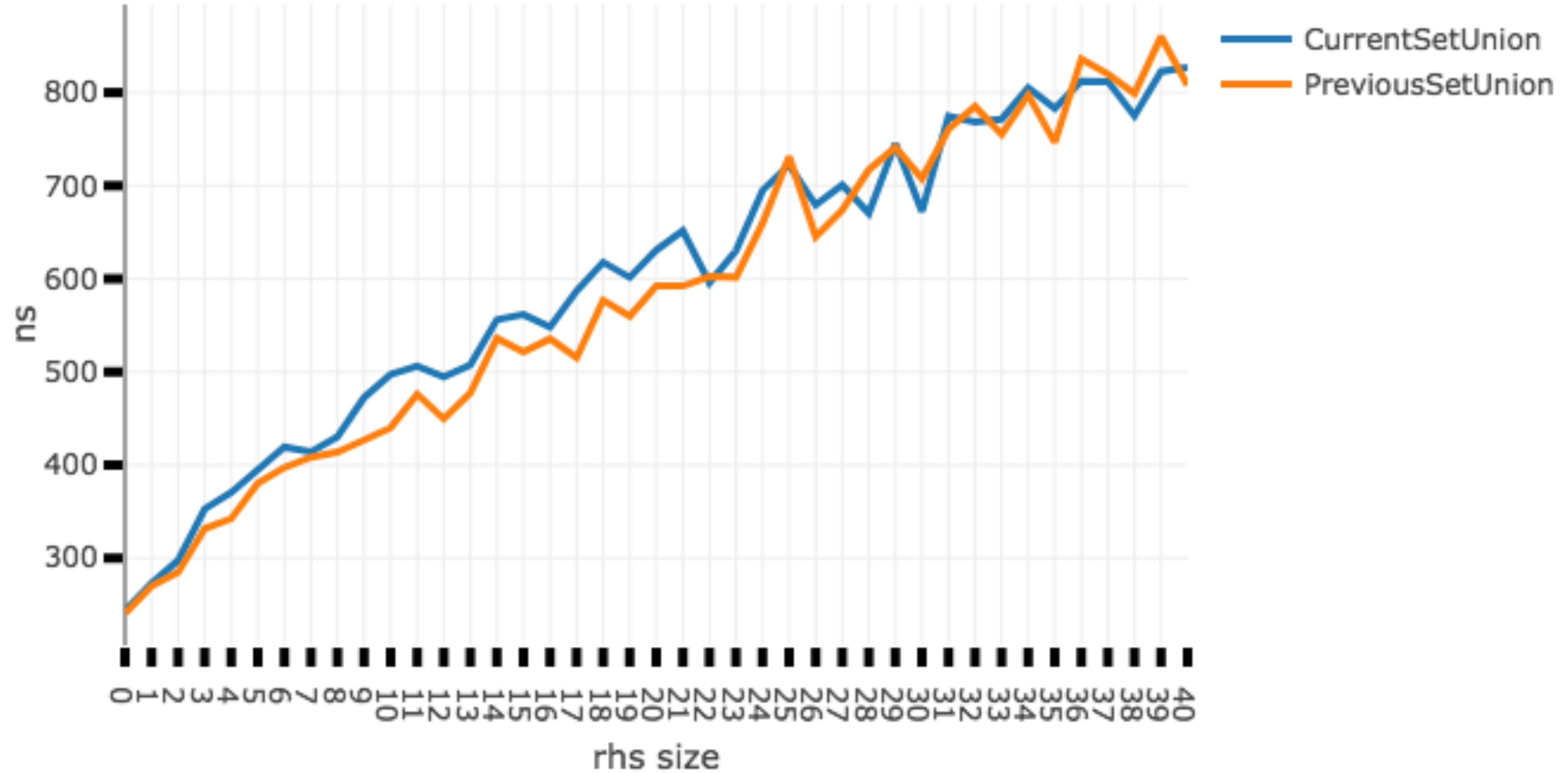
```
while (len > step) {  
    I test = std::next(f, step);  
    if (!p(*test)) { l = test; break; }  
    ...  
    step += step;  
}  
return std::partition_point(f, l, p);
```

# partition\_point\_biased





# partition\_point\_biased( 40 )



# partition\_point\_biased: sentinel

```
while (len > step) {  
    I test = std::next(f, step);  
    if (!p(*test)) { l = test; break; }  
    f = ++test;  
    len -= step + 1;  
    step += step;  
}  
return std::partition_point(f, l, p);
```

```
partition_point_linear: sentinel
```

```
partition_point_linear: sentinel
```

```
while (f != l) {  
    if (!p(*f)) break;  
    ++f;  
}  
return f;
```

partition\_point\_linear: sentinel

```
while (f != l) {  
    if (!p(*f)) break;  
    ++f;  
}  
return f;
```

```
partition_point_linear: sentinel
```

```
f (f == l || p(*prev(l)) return l;
```

```
while (true) {  
    if (!p(*f)) break;  
    ++f;  
}  
return f;
```

# partition\_point\_biased: sentinel

```
while (len > step) {  
    I test = std::next(f, step);  
    if (!p(*test)) {  
        l = test;  
        break;  
    }  
    f = ++test;  
    len -= step + 1;  
    step += step;  
}
```

# partition\_point\_biased: sentinel

$$\rangle (2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$$



# partition\_point\_biased: sentinel

$$\rangle (2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$$

$$\rangle \sum_{i=1..k} 2^i + k =$$

# partition\_point\_biased: sentinel

$$\rangle (2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$$

$$\rangle \sum_{i=1..k} 2^i + k =$$

$$\rangle 2^{k+1} + k =$$

# partition\_point\_biased: sentinel

$$\rangle (2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$$

$$\rangle \sum_{i=1..k} 2^i + k =$$

$$\rangle 2^{k+1} + k =$$

$$\rangle (\text{следующий шаг}) + k$$

# partition\_point\_biased: sentinel

- ›  $(2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$
- ›  $\sum_{i=1..k} 2^i + k =$
- ›  $2^{k+1} + k =$
- › (следующий шаг) + k
- › Если мы в левой половине - шаг не больше половины

# partition\_point\_biased: sentinel

- ›  $(2^0 + 1) + (2^1 + 1) + (2^2 + 1) + \dots + (2^k + 1) =$
- ›  $\sum_{i=1..k} 2^i + k =$
- ›  $2^{k+1} + k =$
- › (следующий шаг) + k
- › Если мы в левой половине - шаг не больше половины
- › Середина - sentinel

# partition\_point\_biased: sentinel

```
assert(len > 4);
```

```
I sent = std::next(f, len / 2);
```

```
if (!p(*sent)) ...
```

```
for (step = 1;; step += step) {
```

```
    ...
```

```
}
```

```
return std::partition_point(f, l, p);
```

# partition\_point\_biased: sentinel

```
auto len = std::distance(f, l);  
I sent = std::next(f, len / 2);  
if (!p(*sent)) ...  
if (!p(*f)) return f; ++f;  
if (!p(*f)) return f; ++f;  
if (!p(*f)) return f; ++f;  
for (step = 2; step += step) {  
    ...  
}  
return std::partition_point(f, l, p);
```

# partition\_point\_biased: sentinel

```
auto len = std::distance(f, l);  
I sent = std::next(f, len / 2);  
if (!p(*sent)) ...  
if (!p(*f)) return f; ++f;  
if (!p(*f)) return f; ++f;  
if (!p(*f)) return f; ++f;  
for (step = 2;; step += step) {  
    ...  
}  
return std::partition_point(f, l, p);
```



# partition\_point\_biased: sentinel

```
auto len = std::distance(f, l);  
I sent = std::next(f, len / 2);  
if (!p(*sent)) ...  
while(true) {  
    if (!p(*f)) return f; ++f;  
    if (!p(*f)) return f; ++f;  
    if (!p(*f)) return f; ++f;  
    for (step = 2;; step += step) {  
        ...  
    }  
}
```

```
partition_point_biased: sentinel
```

```
I find_boundary(I f, I l, P p) {
```

```
}
```

partition\_point\_biased: sentinel

```
I find_boundary(I f, I l, P p) {  
    I sent = middle(f, l);  
  
}
```

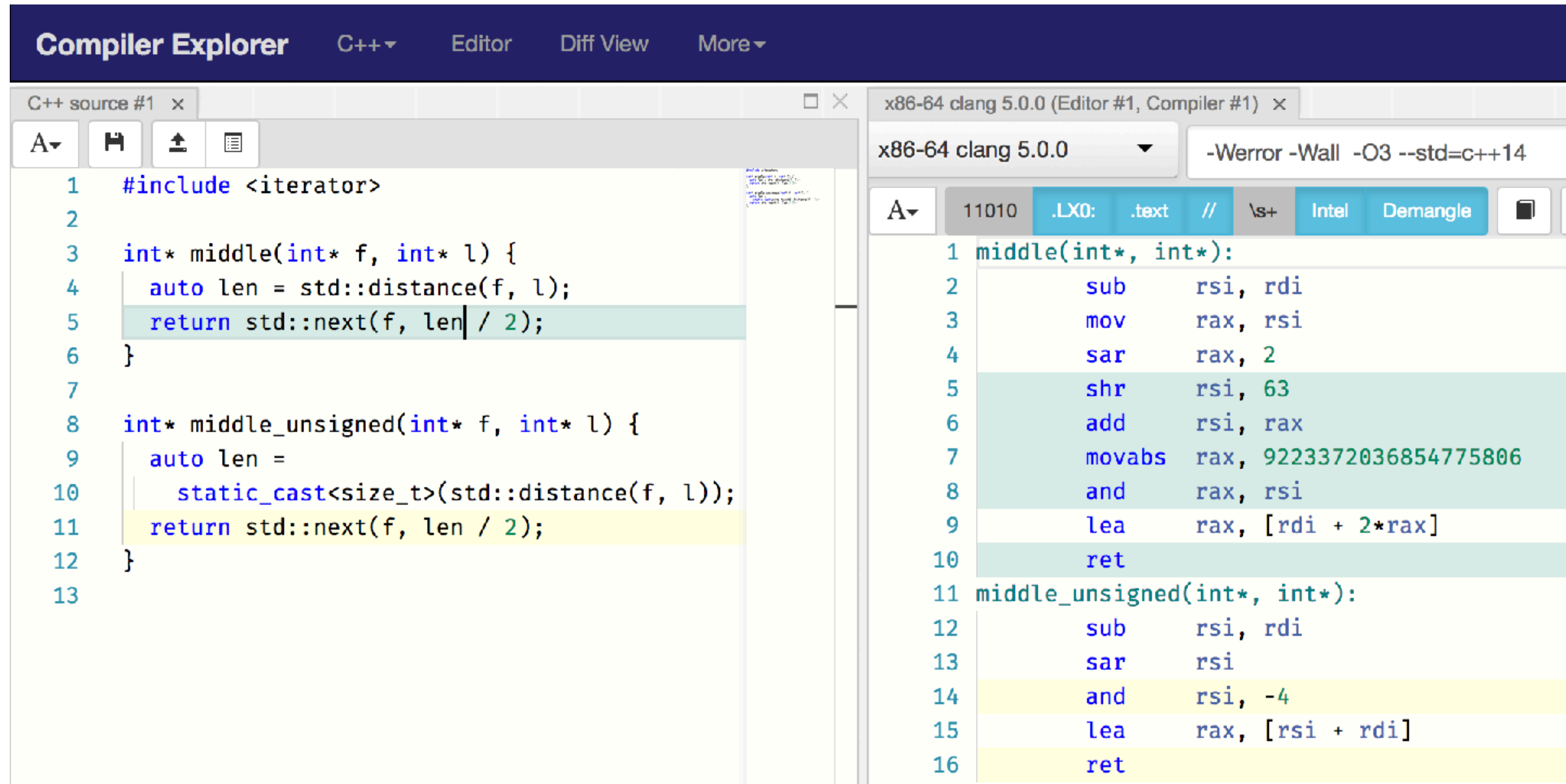
partition\_point\_biased: sentinel

```
I find_boundary(I f, I l, P p) {  
    I sent = middle(f, l);  
    if (p(*sent)) return sent;  
  
}
```

partition\_point\_biased: sentinel

```
I find_boundary(I f, I l, P p) {  
    I sent = middle(f, l);  
    if (p(*sent)) return sent;  
    return partition_point_biased_no_checks(f, p);  
}
```

# partition\_point\_biased: sentinel



The image shows a screenshot of the Compiler Explorer interface. The left pane displays C++ source code for a partitioning function. The right pane shows the corresponding x86-64 assembly code generated by clang 5.0.0 with optimization level -O3 and standard --std=c++14.

**C++ source code (C++ source #1):**

```
1 #include <iterator>
2
3 int* middle(int* f, int* l) {
4     auto len = std::distance(f, l);
5     return std::next(f, len / 2);
6 }
7
8 int* middle_unsigned(int* f, int* l) {
9     auto len =
10     static_cast<size_t>(std::distance(f, l));
11     return std::next(f, len / 2);
12 }
13
```

**x86-64 clang 5.0.0 (Editor #1, Compiler #1):**

Compiler options: -Werror -Wall -O3 --std=c++14

Assembly output (11010 instructions):

```
1 middle(int*, int*):
2     sub     rsi, rdi
3     mov     rax, rsi
4     sar     rax, 2
5     shr     rsi, 63
6     add     rsi, rax
7     movabs  rax, 9223372036854775806
8     and     rax, rsi
9     lea     rax, [rdi + 2*rax]
10    ret
11 middle_unsigned(int*, int*):
12    sub     rsi, rdi
13    sar     rsi
14    and     rsi, -4
15    lea     rax, [rsi + rdi]
16    ret
```

# partition\_point\_biased: sentinel

```
template <typename I, typename P>
I partition_point_biased_no_checks(I f, P p) {
    while(true) {
        if (!p(*f)) return f; ++f;
        if (!p(*f)) return f; ++f;
        if (!p(*f)) return f; ++f;
        for (DifferenceType<I> step = 2;; step += step) {
            I test = std::next(f, step);
            if (!p(*test)) break;
            f = ++test;
        }
    }
}
```

# partition\_point\_biased: sentinel

```
template <typename I>
I middle(I f, I l) {
    static_assert(
        std::numeric_limits<DifferenceType<I>>::max( ) <=
        std::numeric_limits<size_t>::max( ), «" );

    auto len = static_cast<size_t>(std::distance(f, l)) / 2;

    return std::next(f, len / 2);
}
```



# partition\_point\_biased: sentinel

```
template <typename I, typename P>
I find_boundary(I f, I l, P p) {
    I sent = middle(f, l);
    if (p(*sent)) return sent;
    return partition_point_biased_no_checks(f, p);
}
```

# partition\_point\_biased: sentinel

```
template <class I1, class I2, class O, class Comp>
O set_union_biased(I1 f1, I1 l1,
                   I2 f2, I2 l2,
                   O o, Comp comp) {
    if (f1 == l1) goto copySecond;
    if (f2 == l2) goto copyFirst;

    // main cycle

copySecond:
    return std::copy(f2, l2, o);
copyFirst:
    return std::copy(f1, l1, o);
}
```

# partition\_point\_biased: sentinel

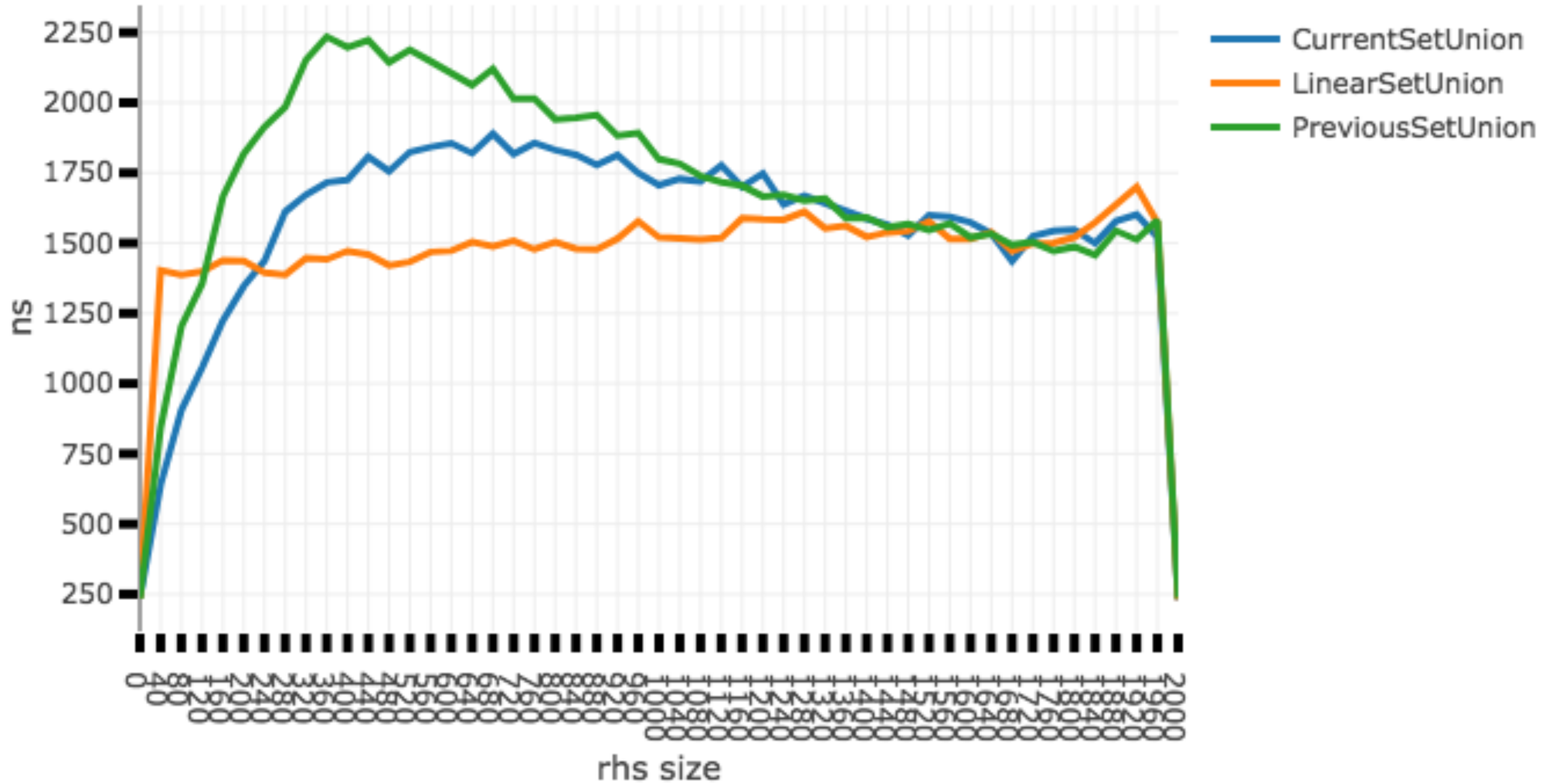
```
while (true) {
    if (!comp(*f1, *f2)) goto checkSecond;
    *o++ = *f1++; if (f1 == l1) goto copySecond;
    goto biased;

checkSecond:
    if (comp(*f2, *f1)) *o++ = *f2;
    ++f2; if (f2 == l2) goto copyFirst;

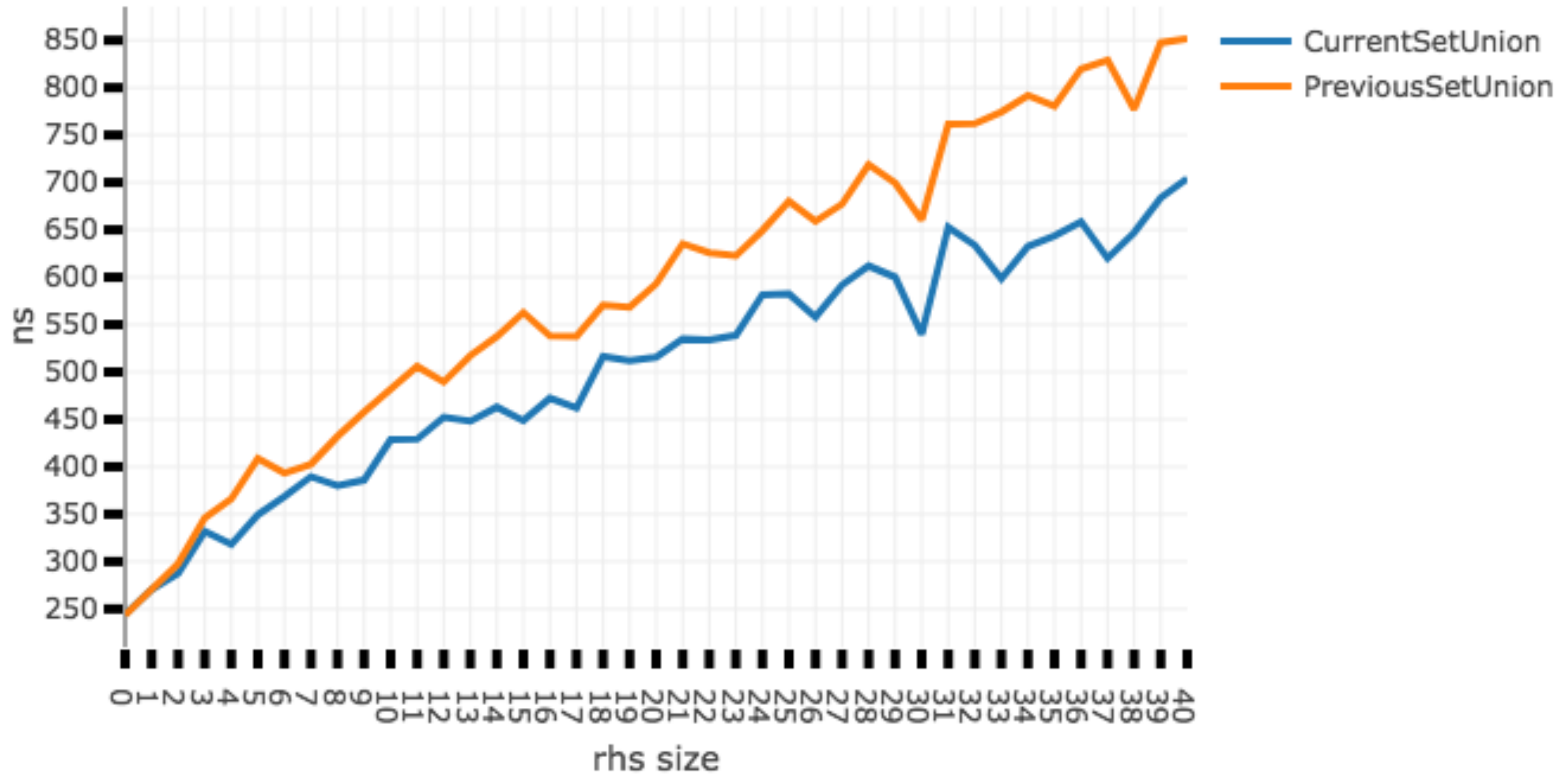
biased:
    if (!comp(*f1, *f2)) goto checkSecond;
    *o++ = *f1++; if (f1 == l1) goto copySecond;
    // 3 more

    l1 segment_end = find_boundary(f1, l1, [&](const auto& x) { return comp(x, *f2); });
    o = std::copy(f1, segment_end, o);
    f1 = segment_end;
}
```

# partition\_point\_biased: sentinel



`partition_point_biased:sentinel(40)`



# Как применить `set_union`

# std::inplace\_merge

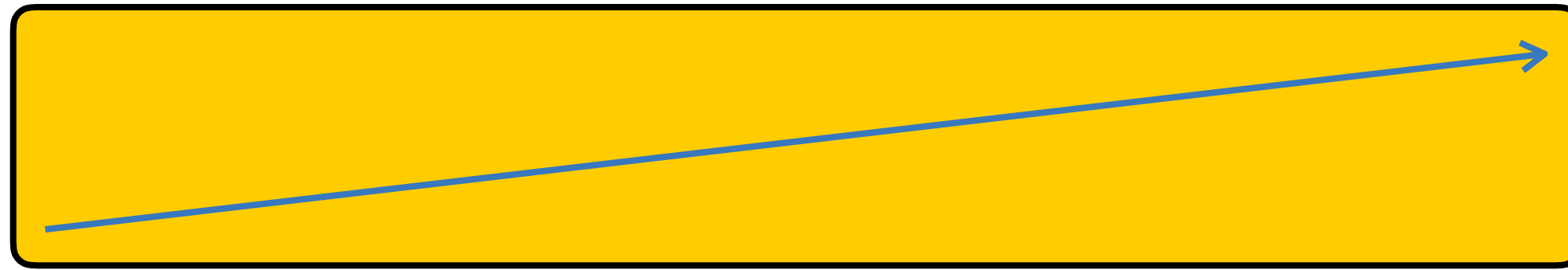
```
...  
if (distance(f, m) < distance(m, l)) {  
    buffer buf{f, m};  
  
} else {  
    buffer buf{m, l};  
  
}  
...
```

# std::inplace\_merge

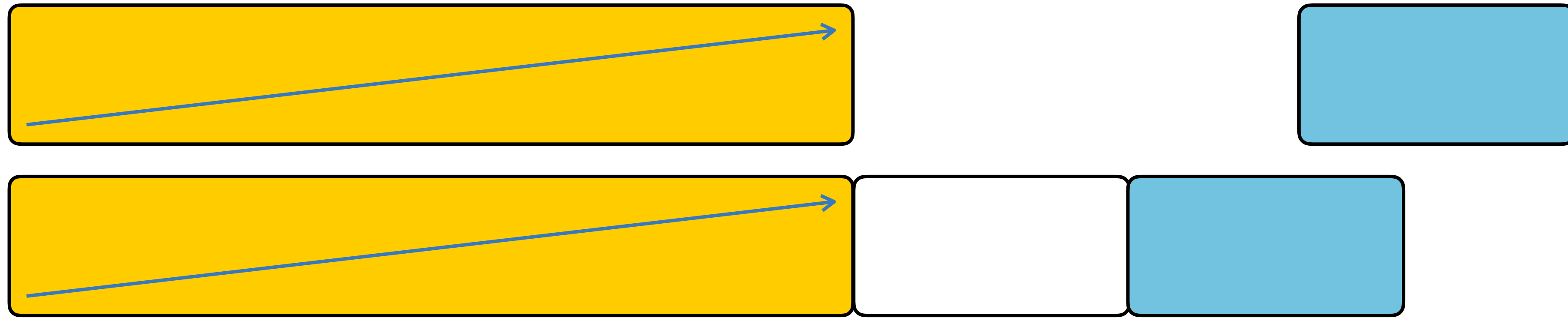
```
...  
if (distance(f, m) < distance(m, l)) {  
    buffer buf{f, m};  
  
} else {  
    buffer buf{m, l};  
    merge_backward(buf.begin(), buf.end(), f, m, l);  
}  
...
```



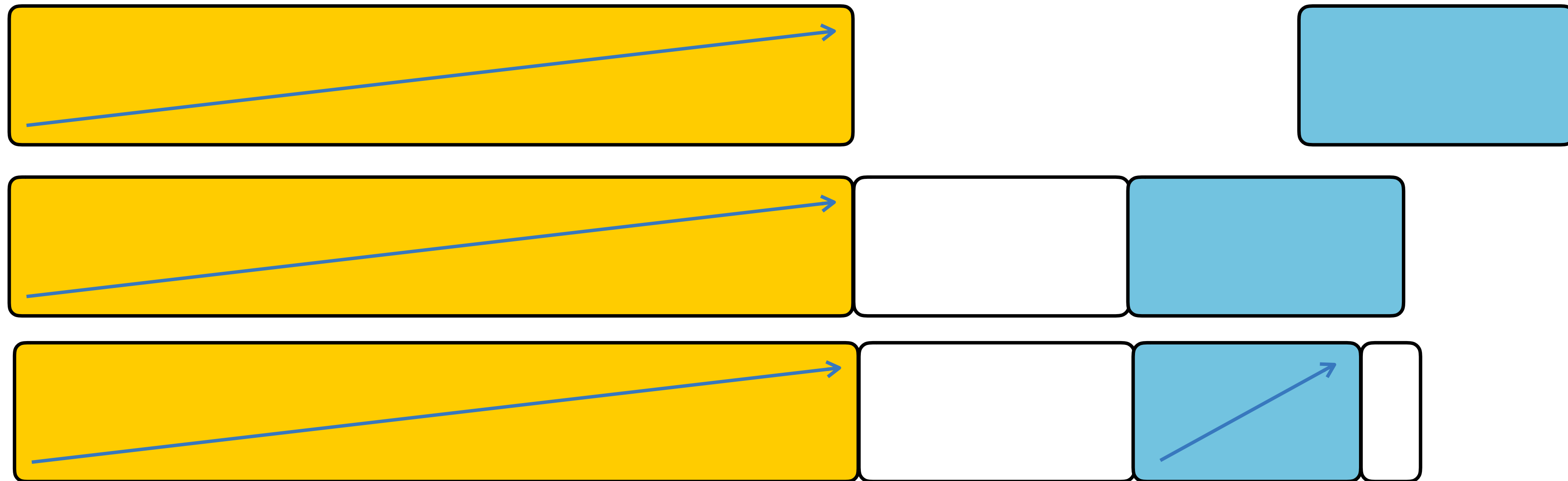
# Схема алгоритма



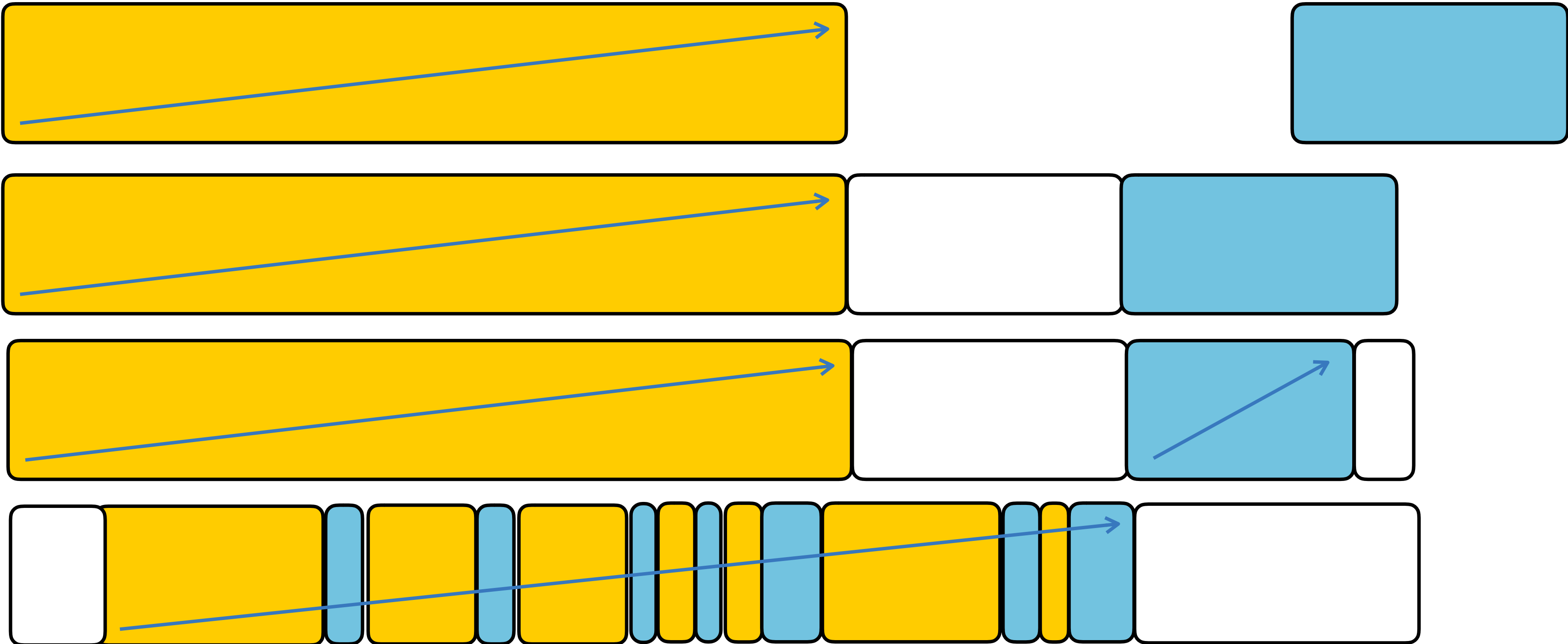
# Схема алгоритма



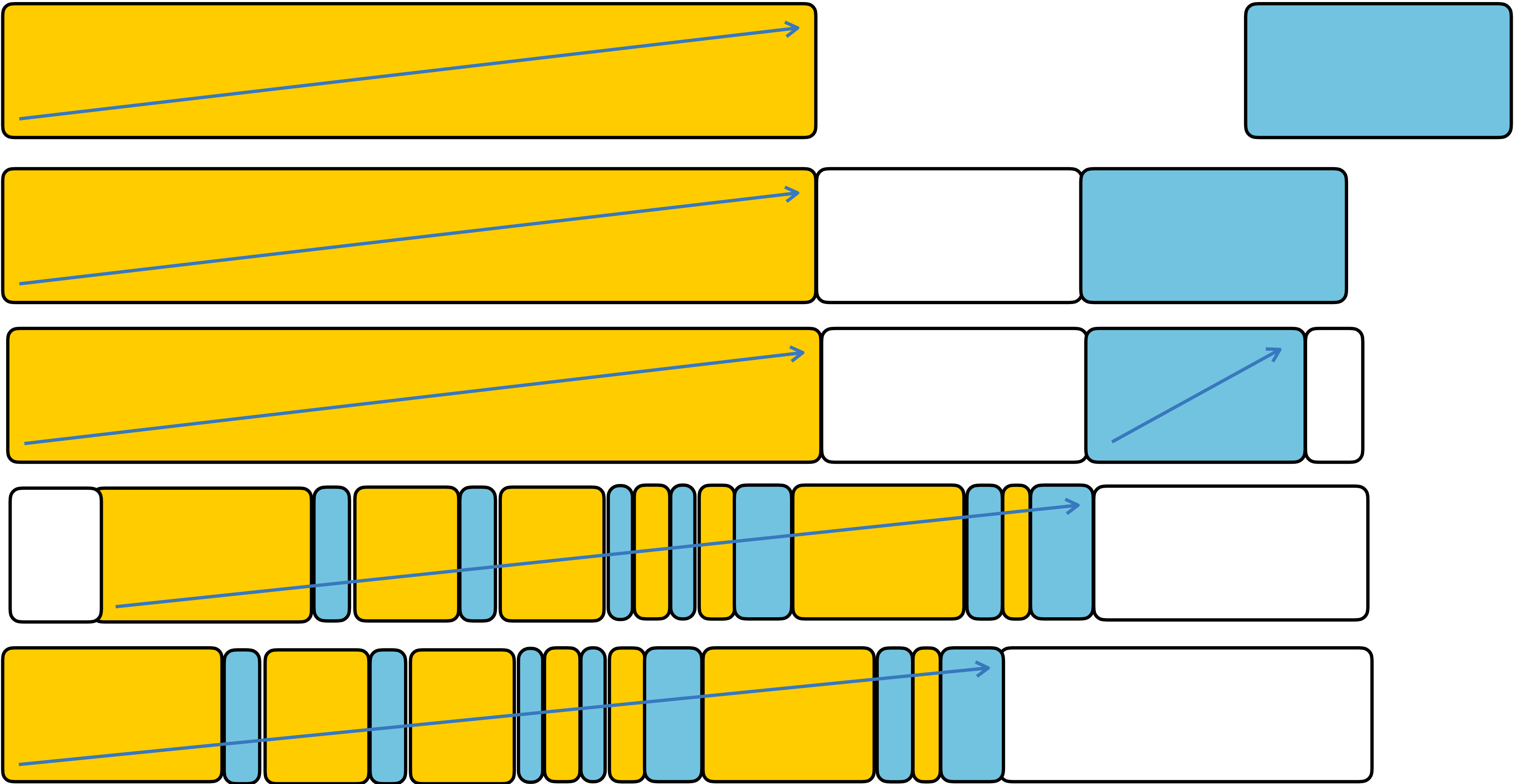
# Схема алгоритма



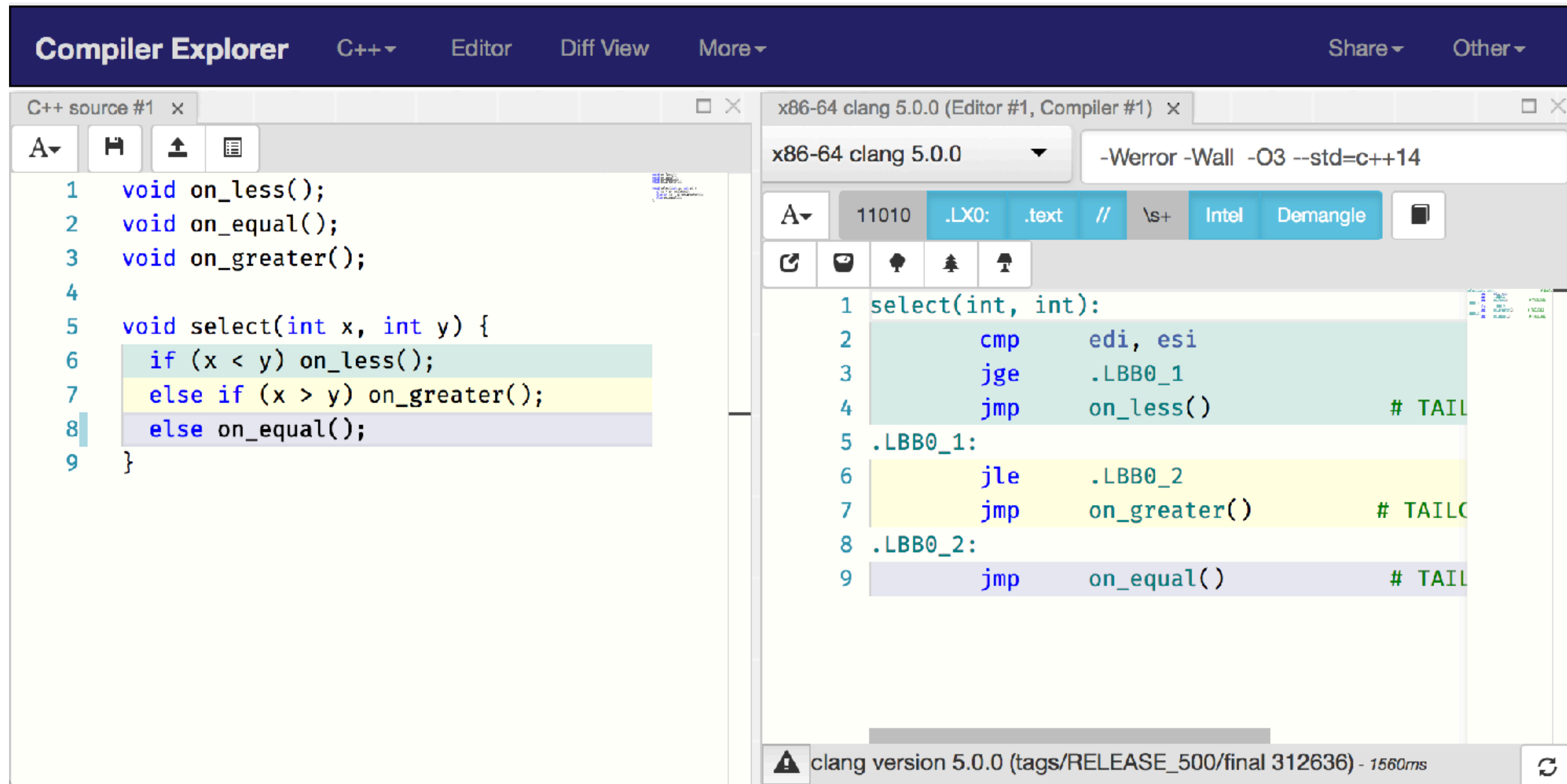
# Схема алгоритма



# Схема алгоритма



# Инструменты. Compiler explorer.



The screenshot displays the Compiler Explorer web application. The left pane, titled 'C++ source #1', contains the following C++ code:

```
1 void on_less();
2 void on_equal();
3 void on_greater();
4
5 void select(int x, int y) {
6     if (x < y) on_less();
7     else if (x > y) on_greater();
8     else on_equal();
9 }
```

The right pane, titled 'x86-64 clang 5.0.0 (Editor #1, Compiler #1)', shows the assembly output for the selected code. The compiler flags are set to '-Werror -Wall -O3 --std=c++14'. The assembly code is as follows:

```
1 select(int, int):
2     cmp     edi, esi
3     jge     .LBB0_1
4     jmp     on_less()          # TAIL
5 .LBB0_1:
6     jle     .LBB0_2
7     jmp     on_greater()      # TAIL
8 .LBB0_2:
9     jmp     on_equal()        # TAIL
```

The status bar at the bottom indicates 'clang version 5.0.0 (tags/RELEASE\_500/final 312636) - 1560ms'.

# Инструменты. Google benchmark

Пример

Benchmark	Time	CPU	Iterations
baseline/560/1440	154 ns	154 ns	4251752
LinearSetUnion/560/1440	2034 ns	2034 ns	341708
CurrentSetUnion/560/1440	2801 ns	2801 ns	257153

# Инструменты. Google benchmark

Пример

Benchmark	Time	CPU	Iterations
baseline/560/1440	154 ns	154 ns	4251752
LinearSetUnion/560/1440	2034 ns	2034 ns	341708
CurrentSetUnion/560/1440	2801 ns	2801 ns	257153

<https://github.com/google/benchmark/issues/461>

Benchmark	Time	CPU	Iterations
baseline/560/1440	159 ns	159 ns	4258633
LinearSetUnion/560/1440	4606 ns	4600 ns	151848
CurrentSetUnion/560/1440	3030 ns	3030 ns	213704



# Конструктор

```
template <typename I>
// requires InputIterator<I>
flat_set(I f, I l) :          {

}
}
```

# Конструктор

```
template <typename I>  
// requires InputIterator<I>  
flat_set(I f, I l) : body_(f, l) {  
  
  
}
```

# Конструктор

```
template <typename I>
// requires InputIterator<I>
flat_set(I f, I l) : body_(f, l) {
    std::sort(begin(), end(), value_compare());
}
```

# Конструктор

```
template <typename I>
// requires InputIterator<I>
flat_set(I f, I l) : body_(f, l) {
    std::sort(begin(), end(), value_compare());
    erase(std::unique(begin(), end(), not_fn(value_compare()))), end());
}
```

# Сравнение с библиотеками

Конструирование из 100 элементов

Benchmark	Time	CPU	Iterations
range_construction<OurSoulation>	750 ns	749 ns	908595
range_construction<Folly>	813 ns	813 ns	819260
range_construction<Chromium>	2095 ns	2094 ns	318914
range_construction<Boost>	3558 ns	3558 ns	202508

Конструирование из 1000 элементов

Benchmark	Time	CPU	Iterations
range_construction<OurSoulation>	16364 ns	16361 ns	42351
range_construction<Folly>	16898 ns	16898 ns	39620
range_construction<Chromium>	32566 ns	32564 ns	21503
range_construction<Boost>	72319 ns	72317 ns	9497

# Boost

```
void priv_range_insertion_construct(...) {  
...  
if (unique_insertion) {  
    for (; first != last; ++first) {  
        this->insert_unique(this->cend( ), *first);  
    }  
}  
}  
...
```

# Chromium

```
void sort_and_unique(iterator first,
                    iterator last,
                    FlatContainerDuples dupes) {
    ...
    std::stable_sort(first, last, impl_.get_value_comp());
    ...
}
```

# Folly

```
void bulk_insert(...) {  
    ...  
    std::copy(first, last, std::back_inserter(cont));  
    ...  
}
```

Benchmark	Time		CPU Iterations	
InsertFirstLast	156	ns	156	ns 4148960
BackInserterWithReserve	2274	ns	2273	ns 304247
BackInserterNoReserve	3668	ns	3666	ns 196330