

PARO 2019

Optymalizacje w C++

Prowadzący

Adam Badura

Software Architect

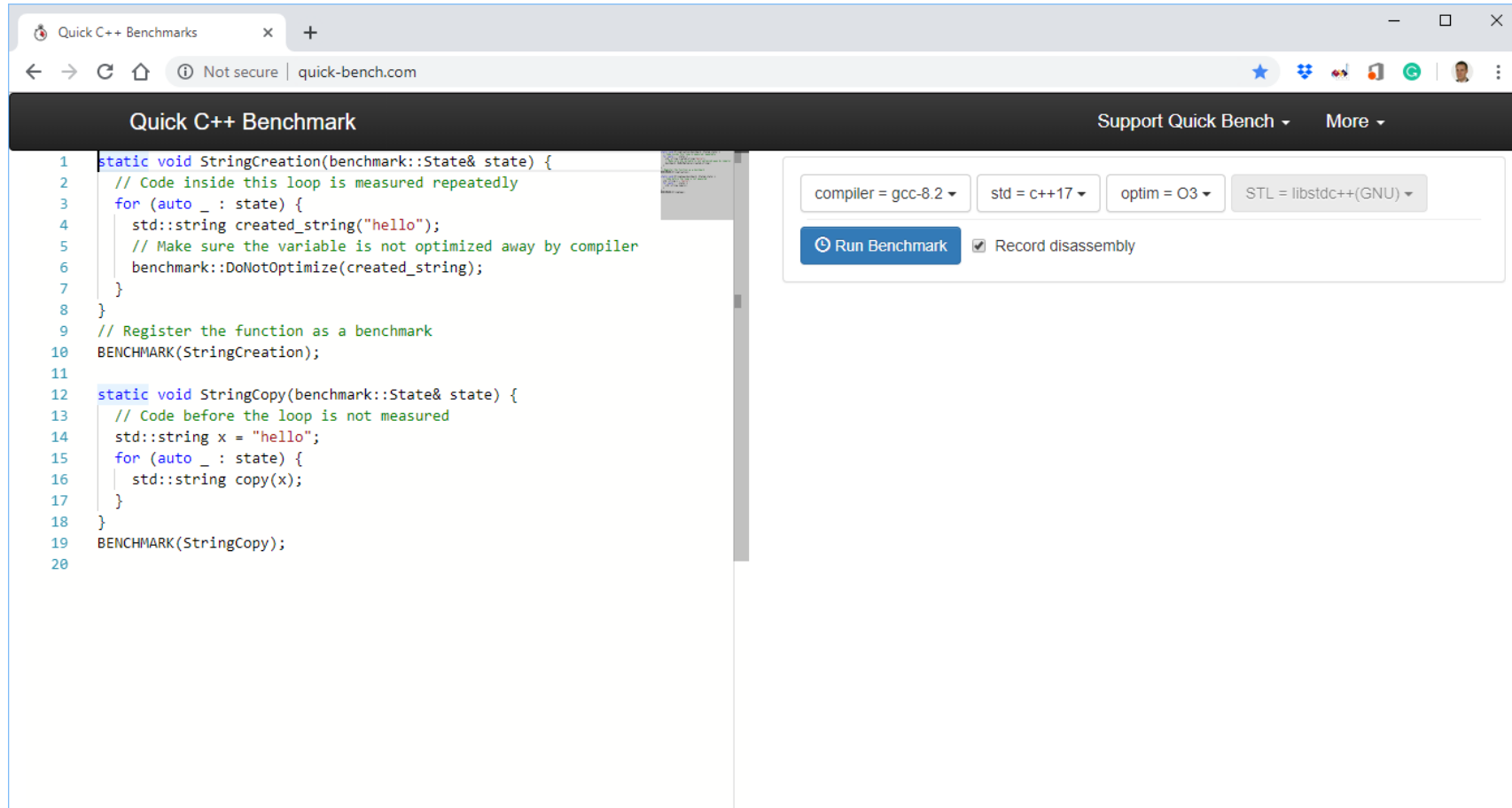
adam.badura@nokia.com

Krzysztof Pawluch

Senior Engineer

krzysztof.pawluch@nokia.com

Quick C++ Benchmarks



The screenshot shows the Quick C++ Benchmarks website interface. The browser address bar displays "Not secure | quick-bench.com". The page title is "Quick C++ Benchmark". The main content area is divided into two sections. The left section contains a C++ code editor with the following code:

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

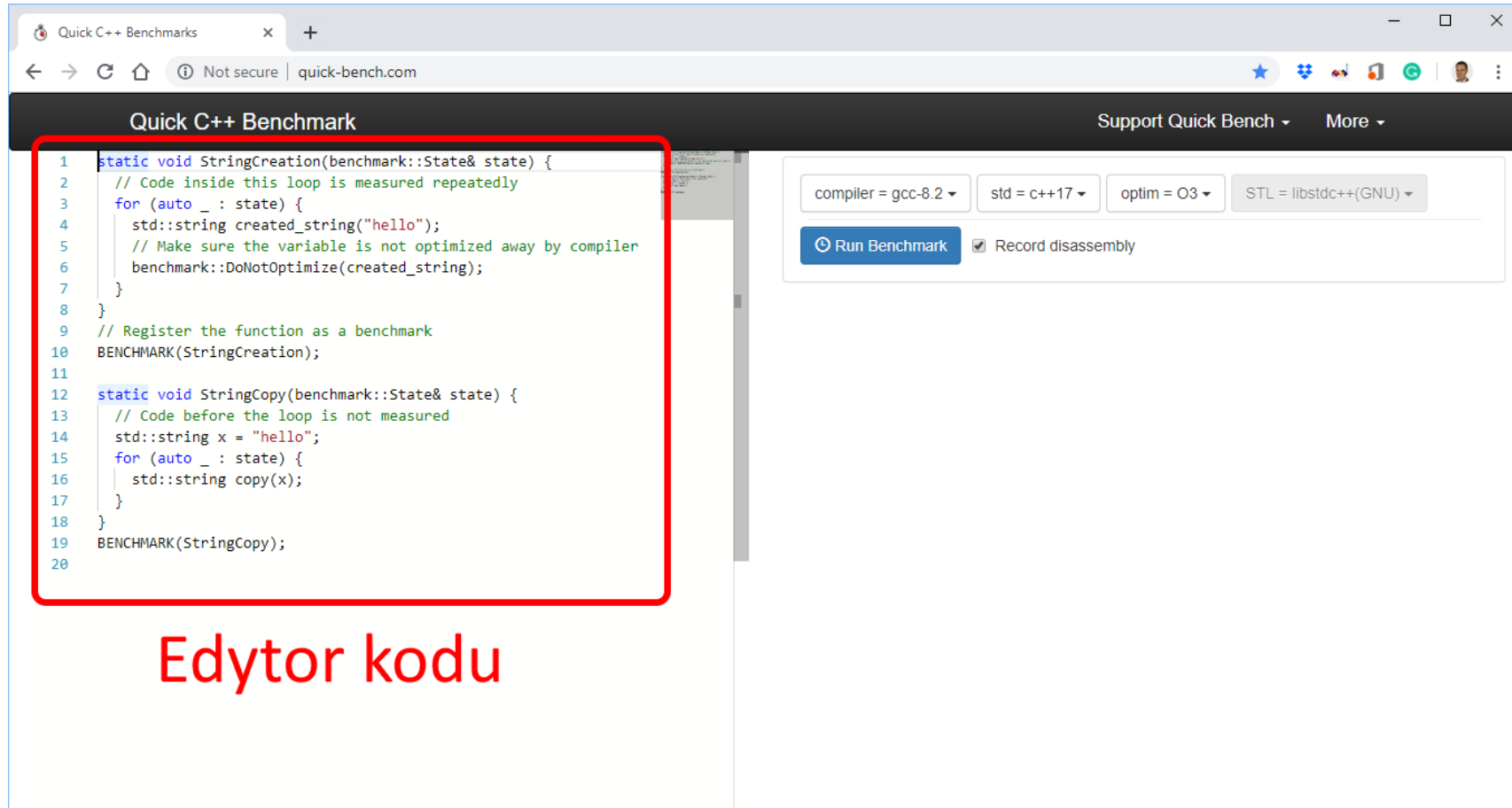
The right section contains benchmark configuration options:

- compiler = gcc-8.2
- std = c++17
- optim = O3
- STL = libstdc++(GNU)

Below these options are two buttons: "Run Benchmark" and "Record disassembly" (which is checked).

<http://quick-bench.com/>

Quick C++ Benchmarks



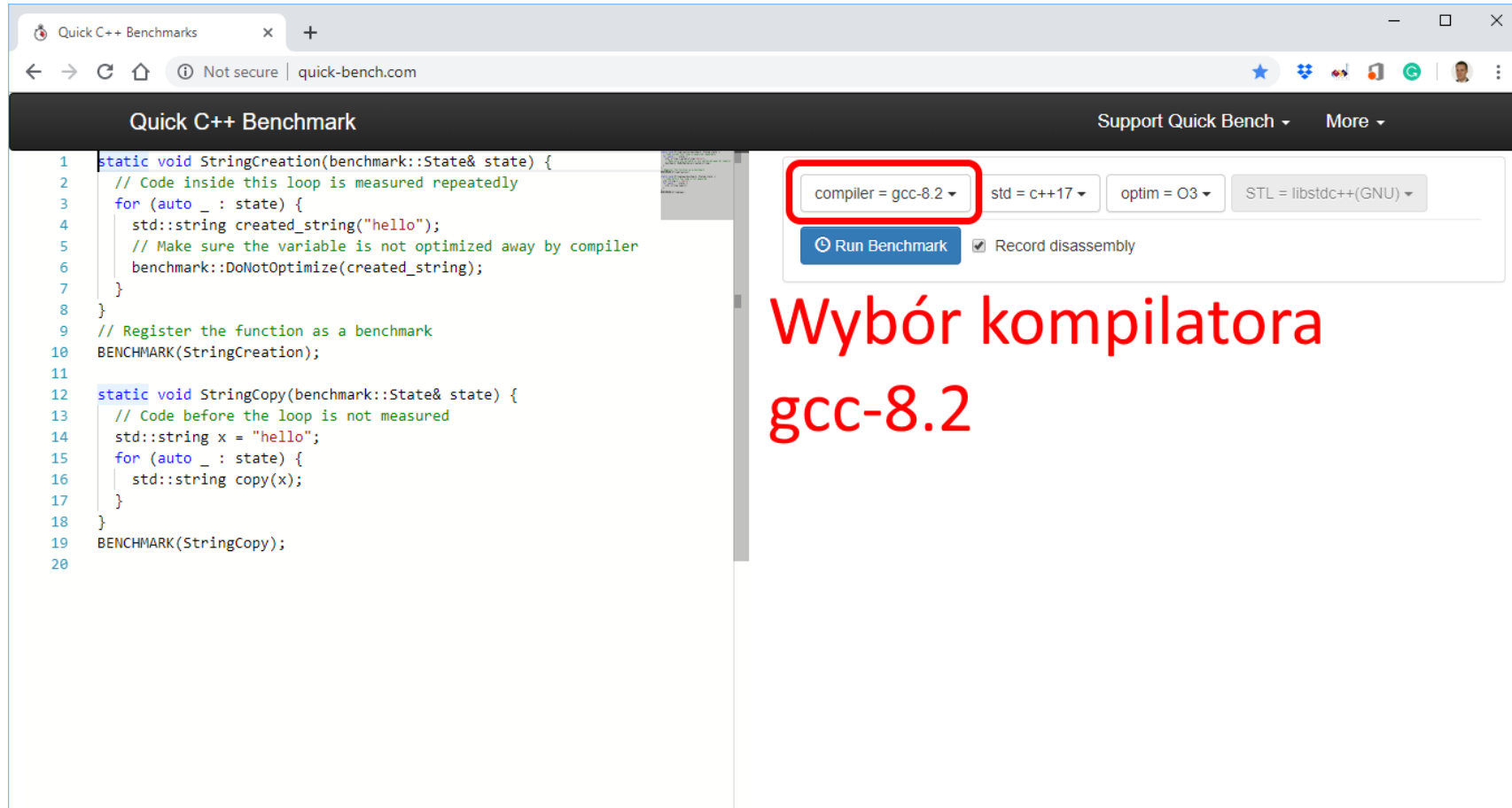
The screenshot shows the Quick C++ Benchmarks website interface. On the left, a code editor displays C++ code for two benchmarks: `StringCreation` and `StringCopy`. The `StringCreation` function is highlighted with a red box. On the right, there are configuration options for the compiler, standard, optimization level, and STL, along with a `Run Benchmark` button and a checkbox for `Record disassembly`.

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

Edytor kodu

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

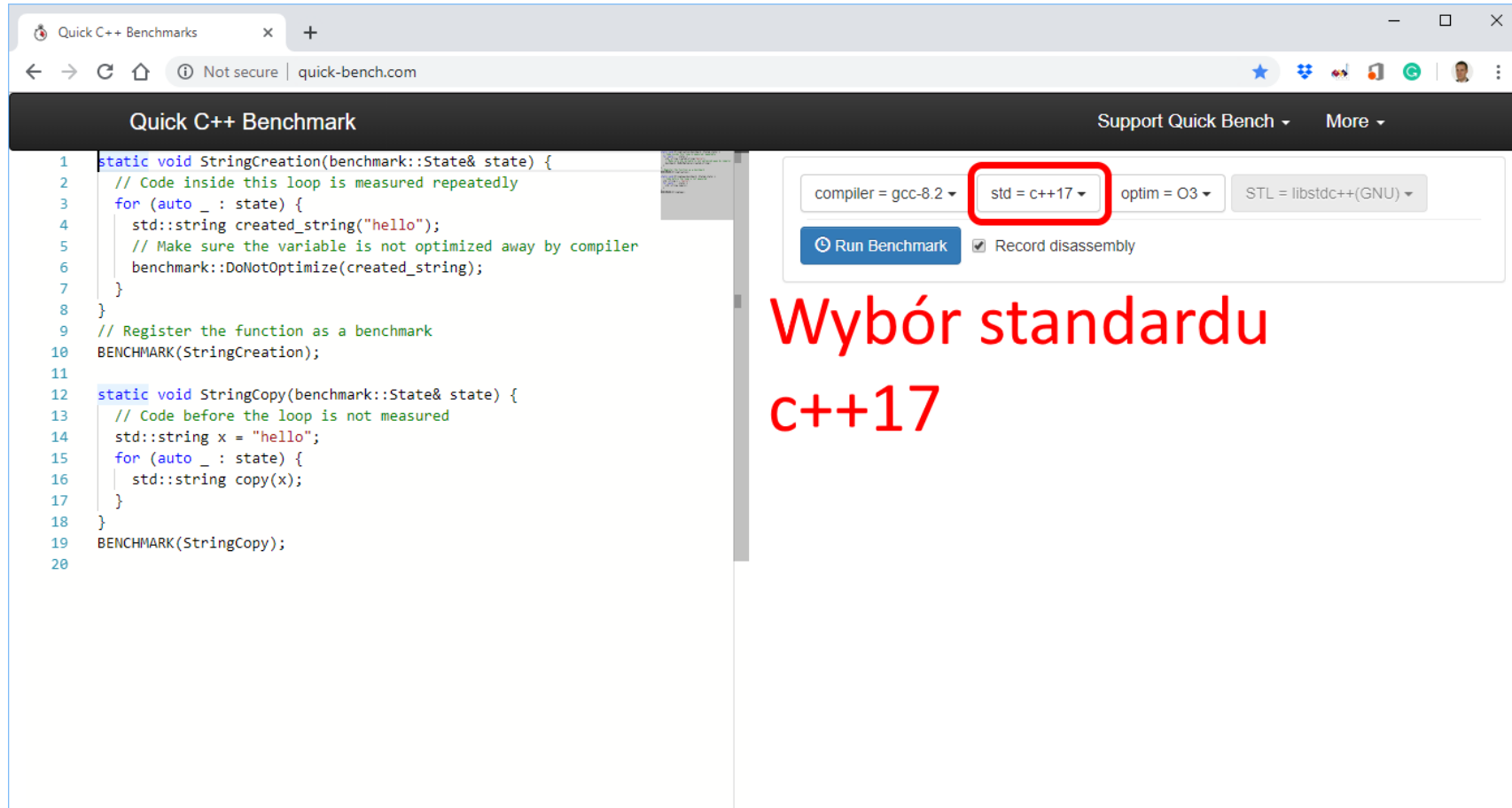
compiler = gcc-8.2 ▾ std = c++17 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Wybór kompilatora
gcc-8.2

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

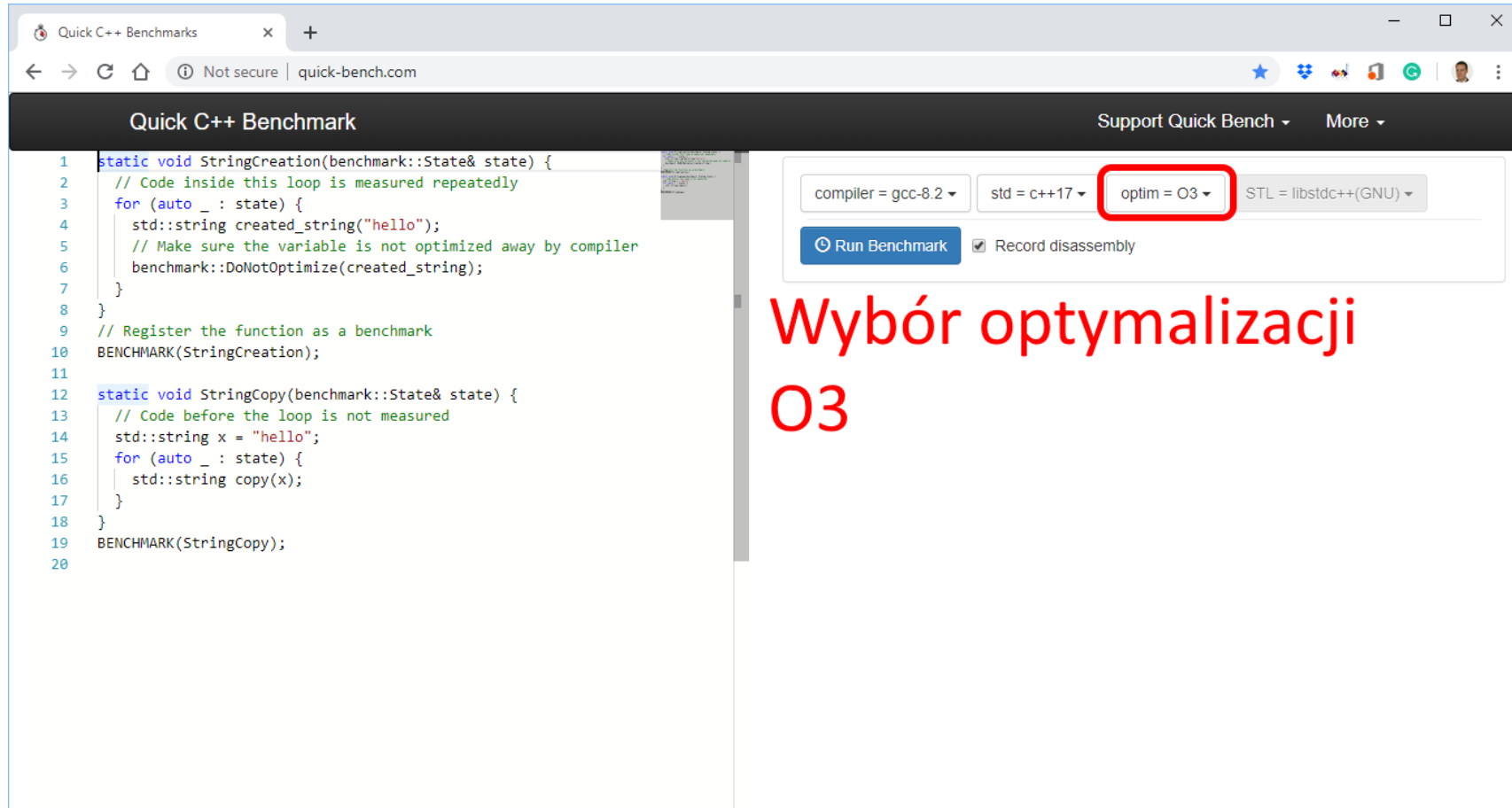
compiler = gcc-8.2 ▾ **std = c++17 ▾** optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Wybór standardu
c++17

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

compiler = gcc-8.2 ▾ std = c++17 ▾ **optim = O3 ▾** STL = libstdc++(GNU) ▾

☒ Record disassembly

Wybór optymalizacji
O3

<http://quick-bench.com/>

Quick C++ Benchmarks

Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

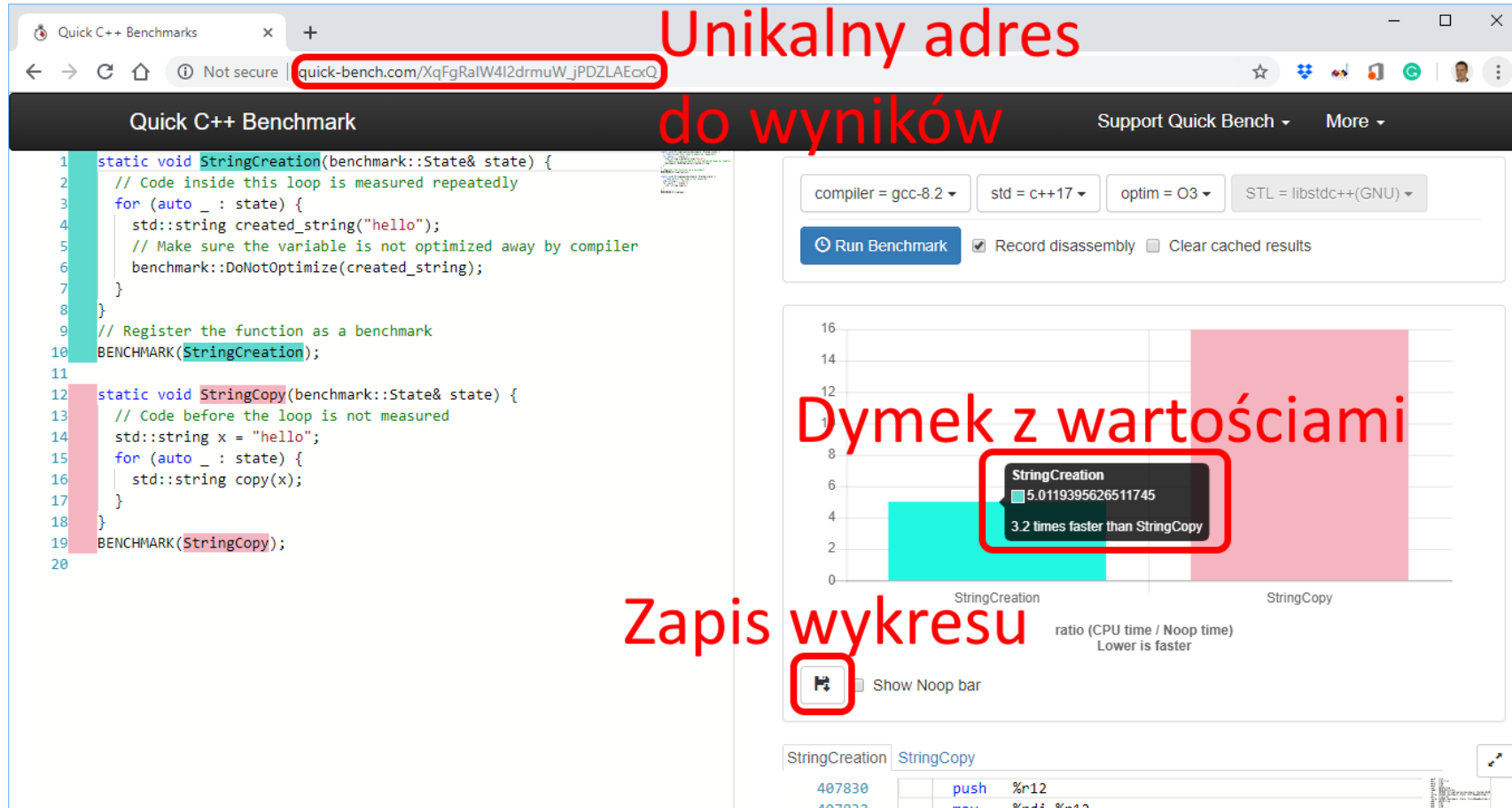
compiler = gcc-8.2 ▾ std = c++17 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Uruchomienie testów

<http://quick-bench.com/>

Quick C++ Benchmarks



<http://quick-bench.com/>

Złożoność algorytmiczna

Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.

Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.

Ćwiczenie 1 – Porównanie sortowań

Sortowanie bąbelkowe

średnio $O(n^2)$

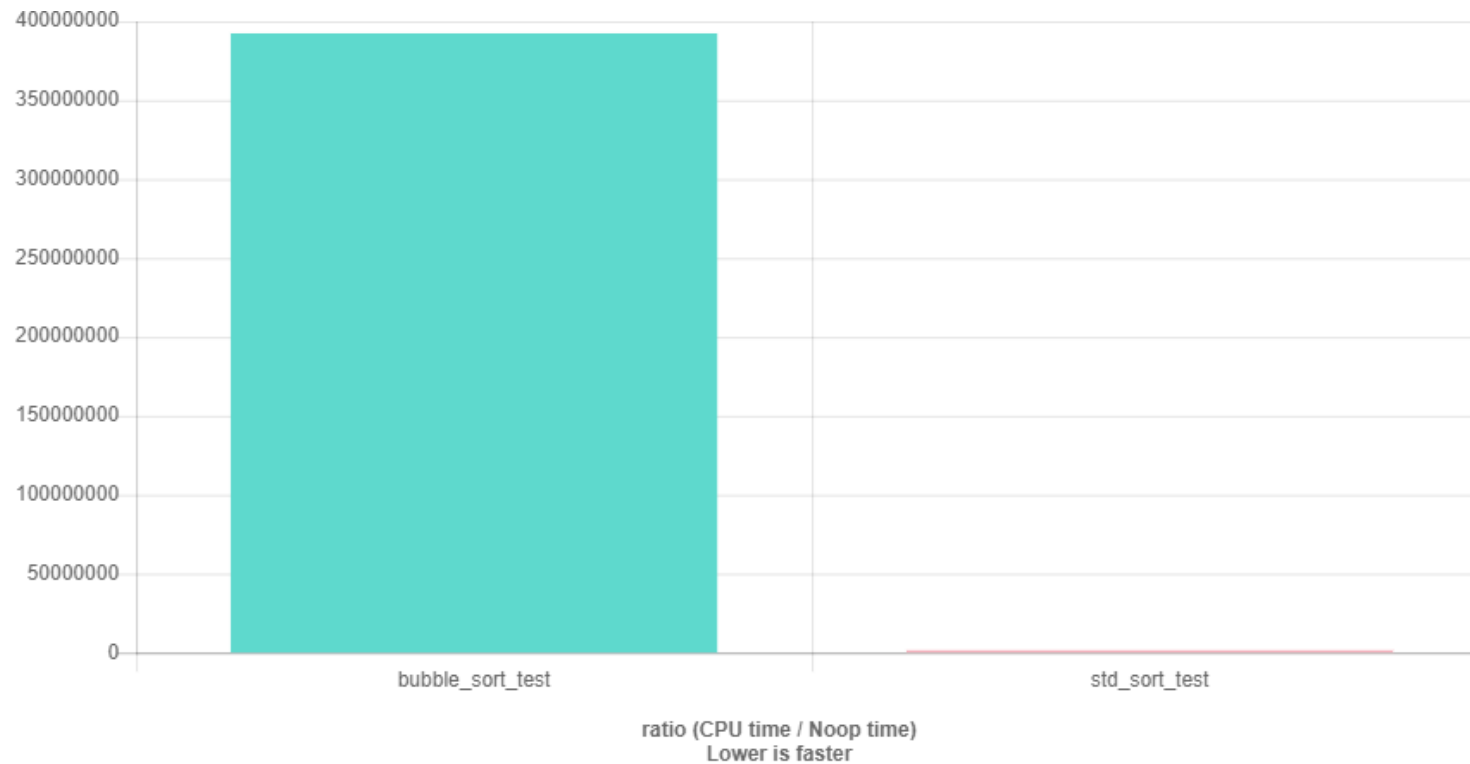
`std::sort`

średnio $O(n \log n)$

Ćwiczenie 1 – Porównanie sortowań

Sortowanie bąbelkowe
średnio $O(n^2)$

`std::sort`
średnio $O(n \log n)$



Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.
- Sama złożoność może być jednak myląca.

Ćwiczenie 2 – Porównanie sortowań

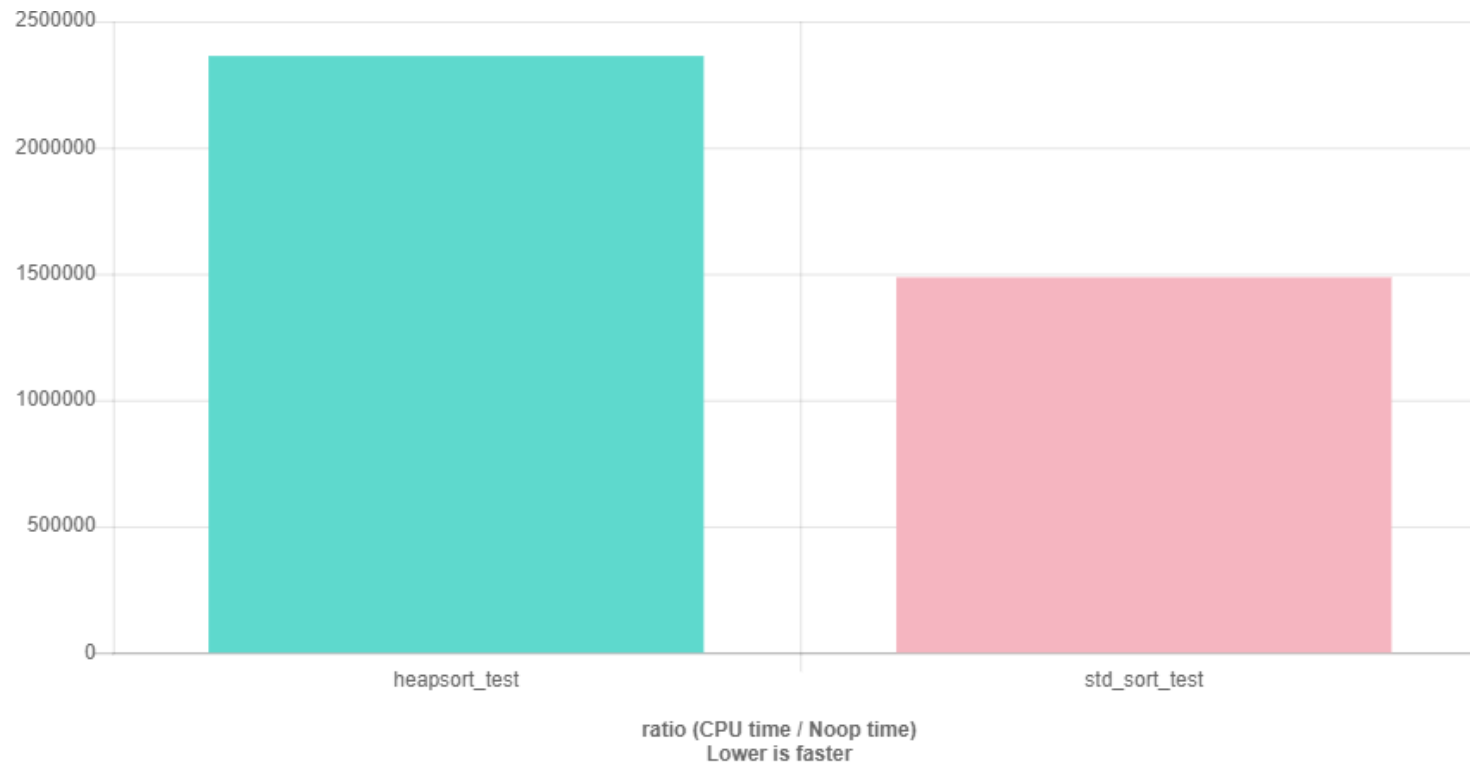
Sortowanie przez kopcowanie
pesymistycznie $O(n \log n)$

`std::sort`
średnio $O(n \log n)$

Ćwiczenie 2 – Porównanie sortowań

Sortowanie przez kopcowanie
pesymistycznie $O(n \log n)$

`std::sort`
średnio $O(n \log n)$



Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.
- Sama złożoność może być jednak myląca.
- Charakterystyka problemu pozwala dobrać dedykowany algorytm, często „out of the box”.

Ćwiczenie 3 – Porównanie sortowań

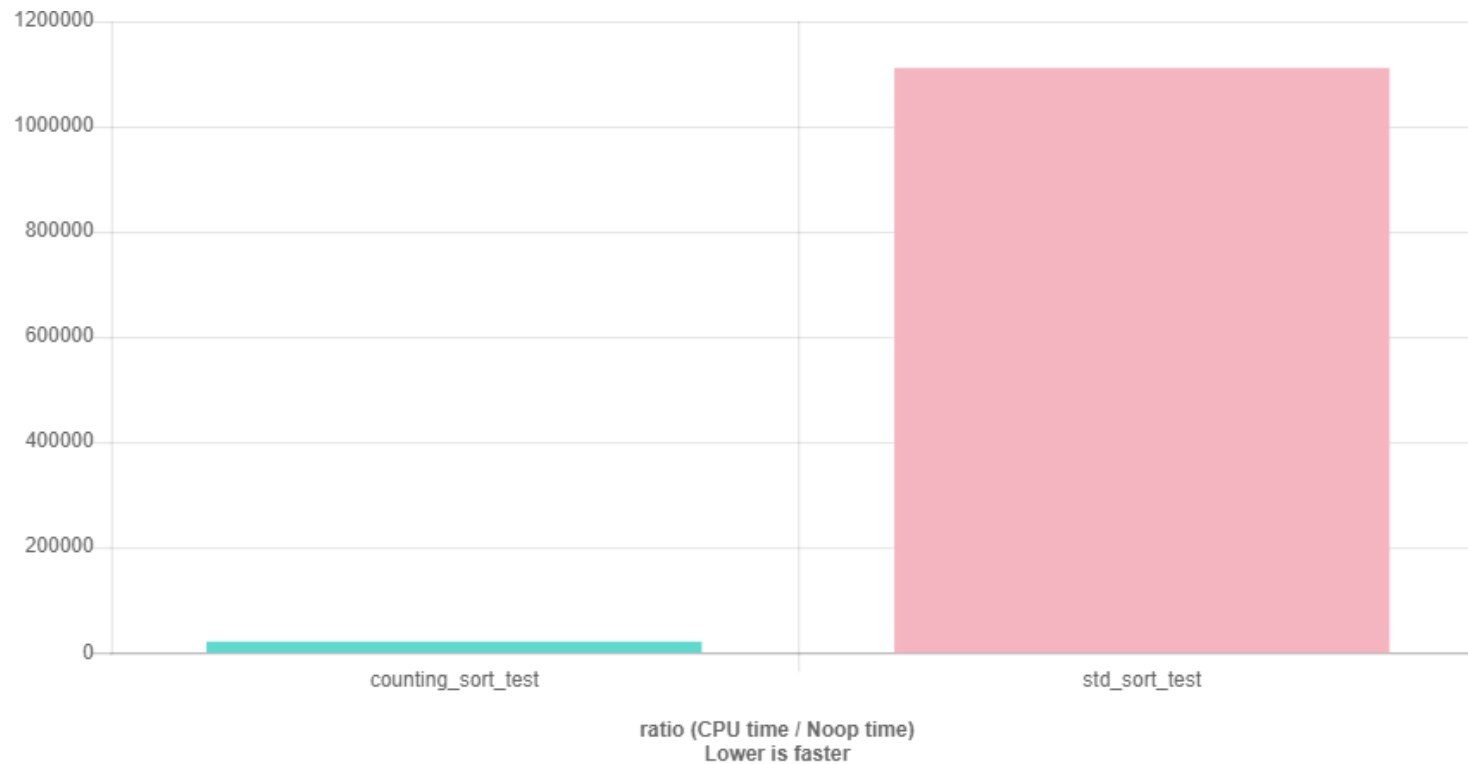
Sortowanie przez zliczanie
pesymistycznie $O(n)$

`std::sort`
średnio $O(n \log n)$

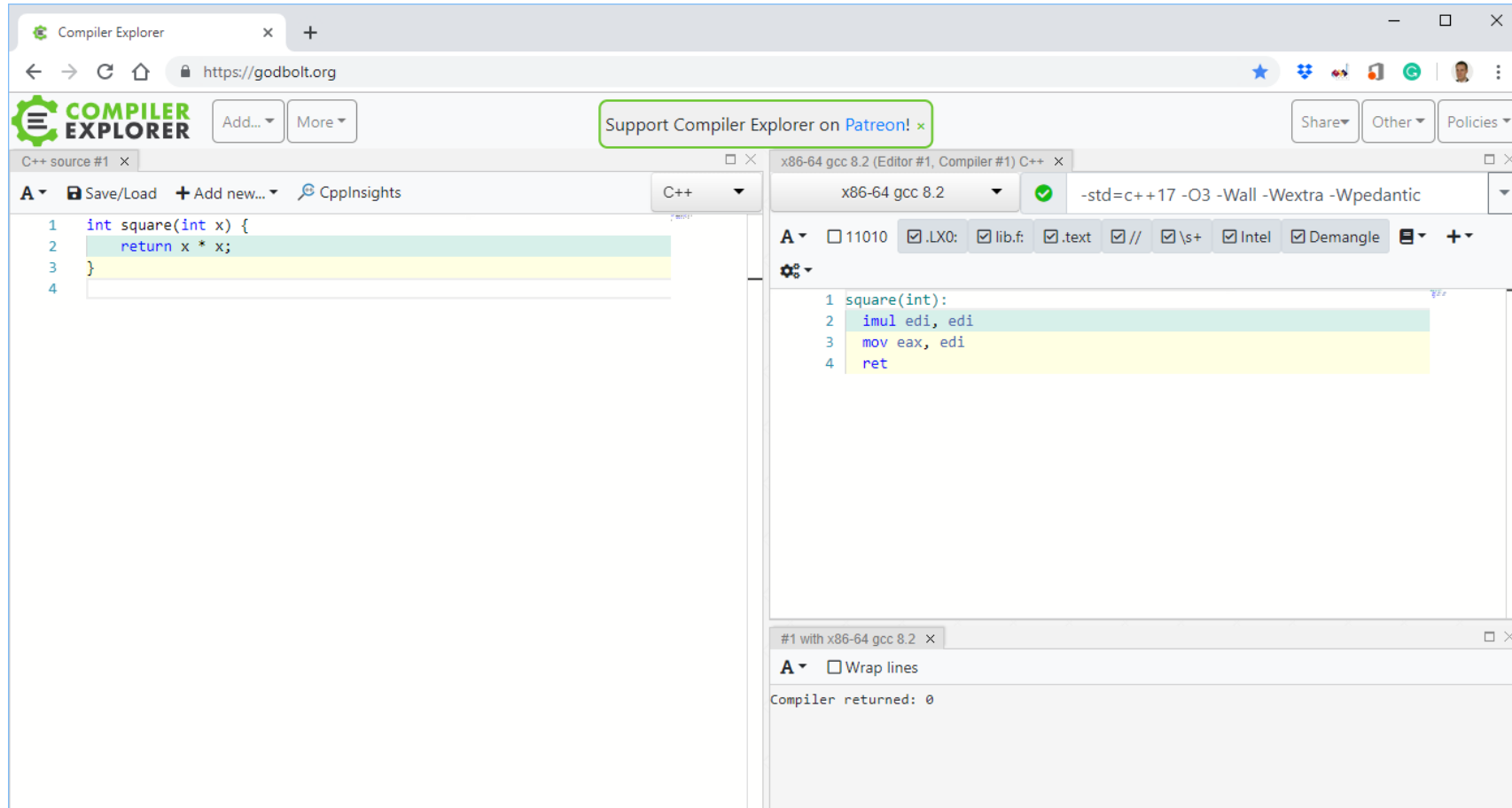
Ćwiczenie 3 – Porównanie sortowań

Sortowanie przez zliczanie
pesymistycznie $O(n)$

`std::sort`
średnio $O(n \log n)$

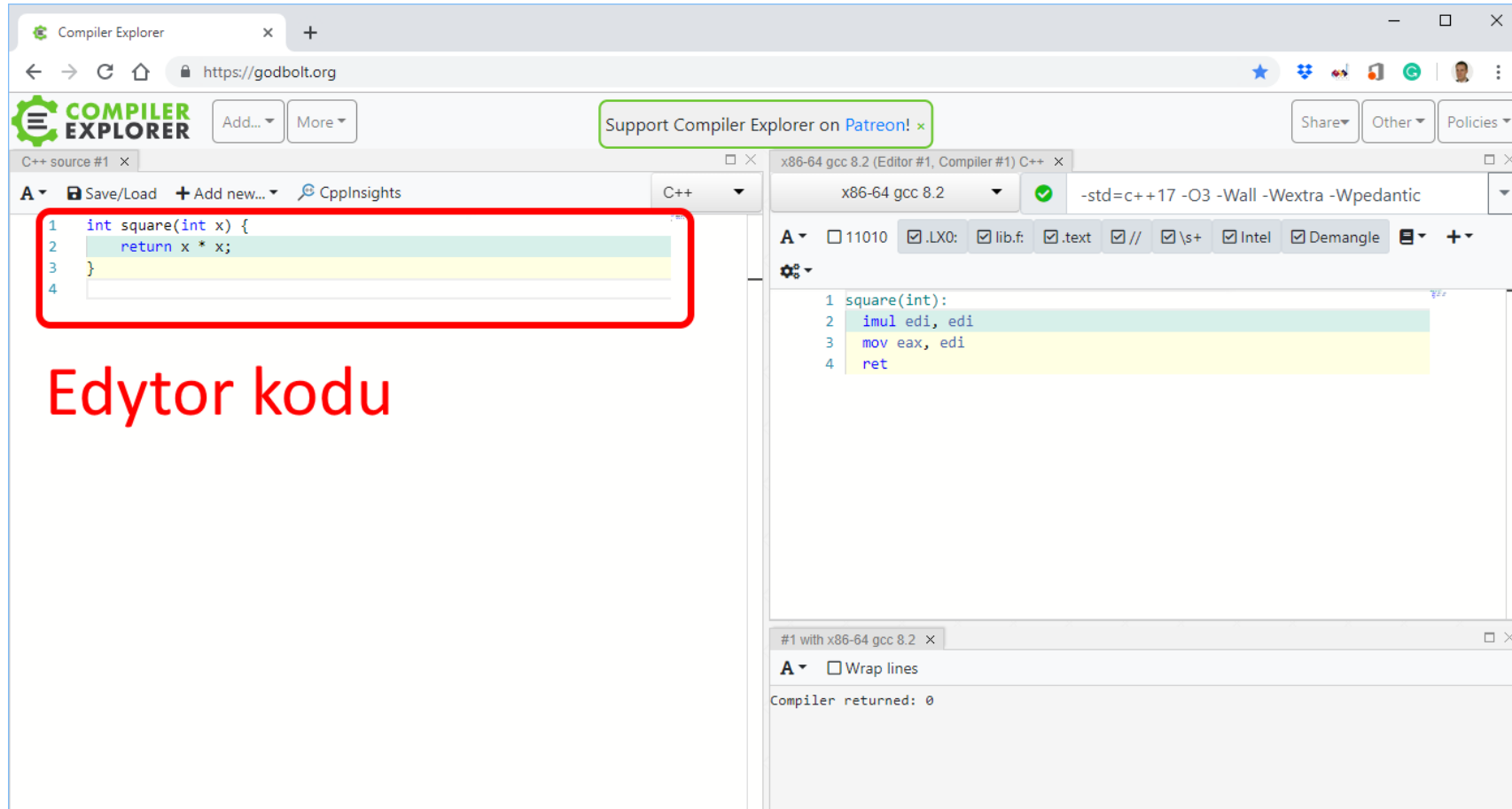


Compiler Explorer



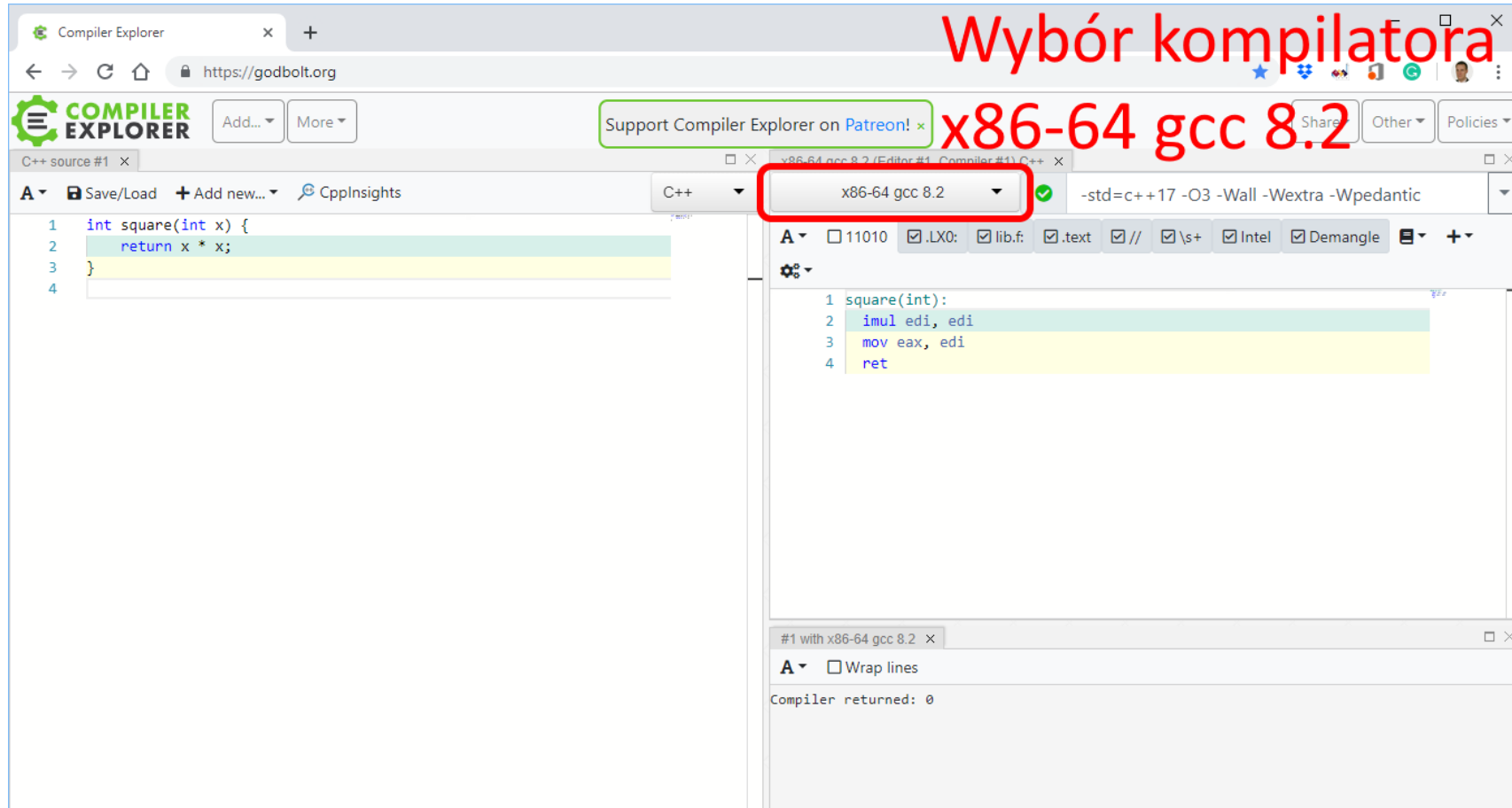
<https://godbolt.org/>

Compiler Explorer



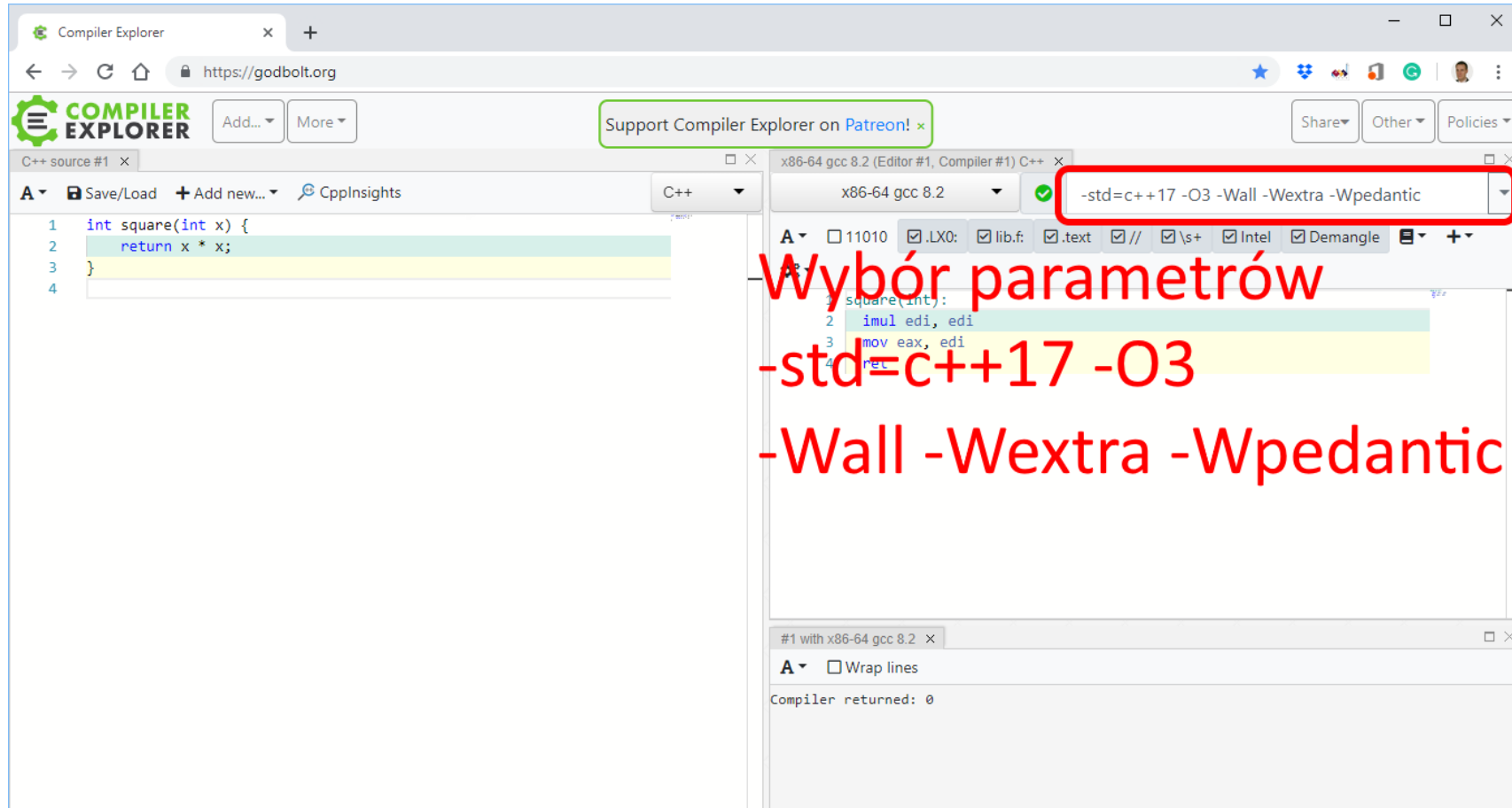
<https://godbolt.org/>

Compiler Explorer



<https://godbolt.org/>

Compiler Explorer



<https://godbolt.org/>

Compiler Explorer

The screenshot displays the Compiler Explorer interface at <https://godbolt.org>. The left pane shows the C++ source code for a `square` function. The right pane shows the assembly output for the same function, generated by `x86-64 gcc 8.2`. The assembly code is highlighted with a red box and labeled "Kod wynikowy". The compiler options are set to `-std=c++17 -O3 -Wall -Wextra -Wpedantic`. The output pane shows the compiler returned 0, which is also highlighted with a red box and labeled "Komunikaty". The "Share" button in the top right corner is highlighted with a red box and labeled "Unikalny adres".

Unikalny adres

```
1 int square(int x) {  
2     return x * x;  
3 }  
4
```

1 square(int):
2 imul edi, edi
3 mov eax, edi
4 ret

Compiler returned: 0

Kod wynikowy

Komunikaty

<https://godbolt.org/>

Optymalizacje kompilatora

Ćwiczenie 4 – Mnożenie a przesunięcie

mnożenie

```
int foo(int x) {  
    return x * 2;  
}
```

przesunięcie bitowe

```
int bar(int x) {  
    return x << 1;  
}
```

Ćwiczenie 4 – Mnożenie a przesunięcie

mnożenie

```
int foo(int x) {  
    return x * 2;  
}
```

```
foo(int):  
    lea eax, [rdi+rdi]  
    ret
```

przesunięcie bitowe

```
int bar(int x) {  
    return x << 1;  
}
```

```
bar(int):  
    lea eax, [rdi+rdi]  
    ret
```

Ćwiczenie 5 – Dodawanie

dodawanie

```
int foo(int x, int y) {  
    x = x + y;  
    return x;  
}
```

dodawanie z przypisaniem

```
int bar(int x, int y) {  
    x += y;  
    return x;  
}
```

Ćwiczenie 5 – Dodawanie

dodawanie

```
int foo(int x, int y) {  
    x = x + y;  
    return x;  
}
```

```
foo(int, int):  
    lea eax, [rdi+rsi]  
    ret
```

dodawanie z przypisaniem

```
int bar(int x, int y) {  
    x += y;  
    return x;  
}
```

```
bar(int, int):  
    lea eax, [rdi+rsi]  
    ret
```

Ćwiczenie 6 – Dodawanie a inkrementacja

dodawanie

```
int foo(int x) {  
    x = x + 1;  
    return x;  
}
```

inkrementowanie

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

Ćwiczenie 6 – Dodawanie a inkrementacja

dodawanie

```
int foo(int x) {  
    x = x + 1;  
    return x;  
}
```

```
foo(int):  
    lea eax, [rdi+1]  
    ret
```

inkrementowanie

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

```
bar(int):  
    lea eax, [rdi+1]  
    ret
```


Ćwiczenie 7 – Inkrementacja post i pre

Post-inkrementacja

```
int foo(int x) {  
    x++;  
    return x;  
}
```

Pre-inkrementacja

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

Ćwiczenie 7 – Inkrementacja post i pre

Post-inkrementacja

```
int foo(int x) {  
    x++;  
    return x;  
}
```

```
foo(int):  
    lea eax, [rdi+1]  
    ret
```

Pre-inkrementacja

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

```
bar(int):  
    lea eax, [rdi+1]  
    ret
```

Ćwiczenie 8.1 – Dzielenie przez zmienną

```
int foo(int x, int y) {  
    return x / y;  
}
```

Ćwiczenie 8.1 – Dzielenie przez zmienną

```
int foo(int x, int y) {  
    return x / y;  
}
```

```
foo(int, int):  
    mov eax, edi  
    cdq  
    idiv esi  
    ret
```

Ćwiczenie 8.2 – Dzielenie przez stałą cz. 1

```
int foo(int x) {  
    return x / 2;  
}
```

Ćwiczenie 8.2 – Dzielenie przez stałą cz. 1

```
int foo(int x) {  
    return x / 2;  
}
```

```
foo(int):  
    mov eax, edi  
    shr eax, 31  
    add eax, edi  
    sar eax  
    ret
```

Ćwiczenie 8.3 – Dzielenie przez stałą cz. 2

```
int foo(int x) {  
    return x / 10;  
}
```

Ćwiczenie 8.3 – Dzielenie przez stałą cz. 2

```
int foo(int x) {  
    return x / 10;  
}
```

```
foo(int):  
    mov eax, edi  
    mov edx, 1717986919  
    sar edi, 31  
    imul edx  
    sar edx, 2  
    mov eax, edx  
    sub eax, edi  
    ret
```


Ćwiczenie 9.1 – Nazwy enumeracji

```
enum class color {  
    black,  
    maroon,  
    green,  
    olive,  
    navy,  
    purple,  
    teal,  
    silver,  
    gray,  
    red,  
    lime,  
    yellow,  
    blue,  
    fuchsia,  
    aqua,  
    white  
};
```

Ćwiczenie 9.2 – Nazwy enumeracji, **switch**

```
char const* enum_to_c_str(color v) {  
    switch (v) {  
#define CASE(x) case x: return #x  
        CASE(color::black);  
        CASE(color::maroon);  
        CASE(color::green);  
        CASE(color::olive);  
        CASE(color::navy);  
        CASE(color::purple);  
        CASE(color::teal);  
        CASE(color::silver);  
        CASE(color::gray);  
        CASE(color::red);  
        CASE(color::lime);  
        CASE(color::yellow);  
        CASE(color::blue);  
        CASE(color::fuchsia);  
        CASE(color::aqua);  
        CASE(color::white);  
#undef CASE  
    }  
}
```

Ćwiczenie 9.2 – Nazwy enumeracji, **switch**

```
char const* enum_to_c_str(color v) {  
    switch (v) {  
#define CASE(x) case x: return #x  
        CASE(color::black);  
        CASE(color::maroon);  
        CASE(color::green);  
        CASE(color::olive);  
        CASE(color::navy);  
        CASE(color::purple);  
        CASE(color::teal);  
        CASE(color::silver);  
        CASE(color::gray);  
        CASE(color::red);  
        CASE(color::lime);  
        CASE(color::yellow);  
        CASE(color::blue);  
        CASE(color::fuchsia);  
        CASE(color::aqua);  
        CASE(color::white);  
#undef CASE  
    }  
}
```

```
enum_to_c_str(color):  
    mov edi, edi  
    mov rax, QWORD PTR CSWTCH.0[0+rdi*8]  
    ret  
// (...) stałe
```

Ćwiczenie 9.3 – Nazwy enumeracji, `std::map`

```
char const* enum_to_c_str(color v) {  
    static std::map<color, char const*> const  
    mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```

Ćwiczenie 9.3 – Nazwy enumeracji, std::map

```
char const* enum_to_c_str(color v) {  
    static std::map<color, char const*> const  
    mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```



Ćwiczenie 9.4 – Nazwy enumeracji, `std::unordered_map`

```
char const* enum_to_c_str(color v) {  
    static std::unordered_map<color, char const*>  
    const mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```

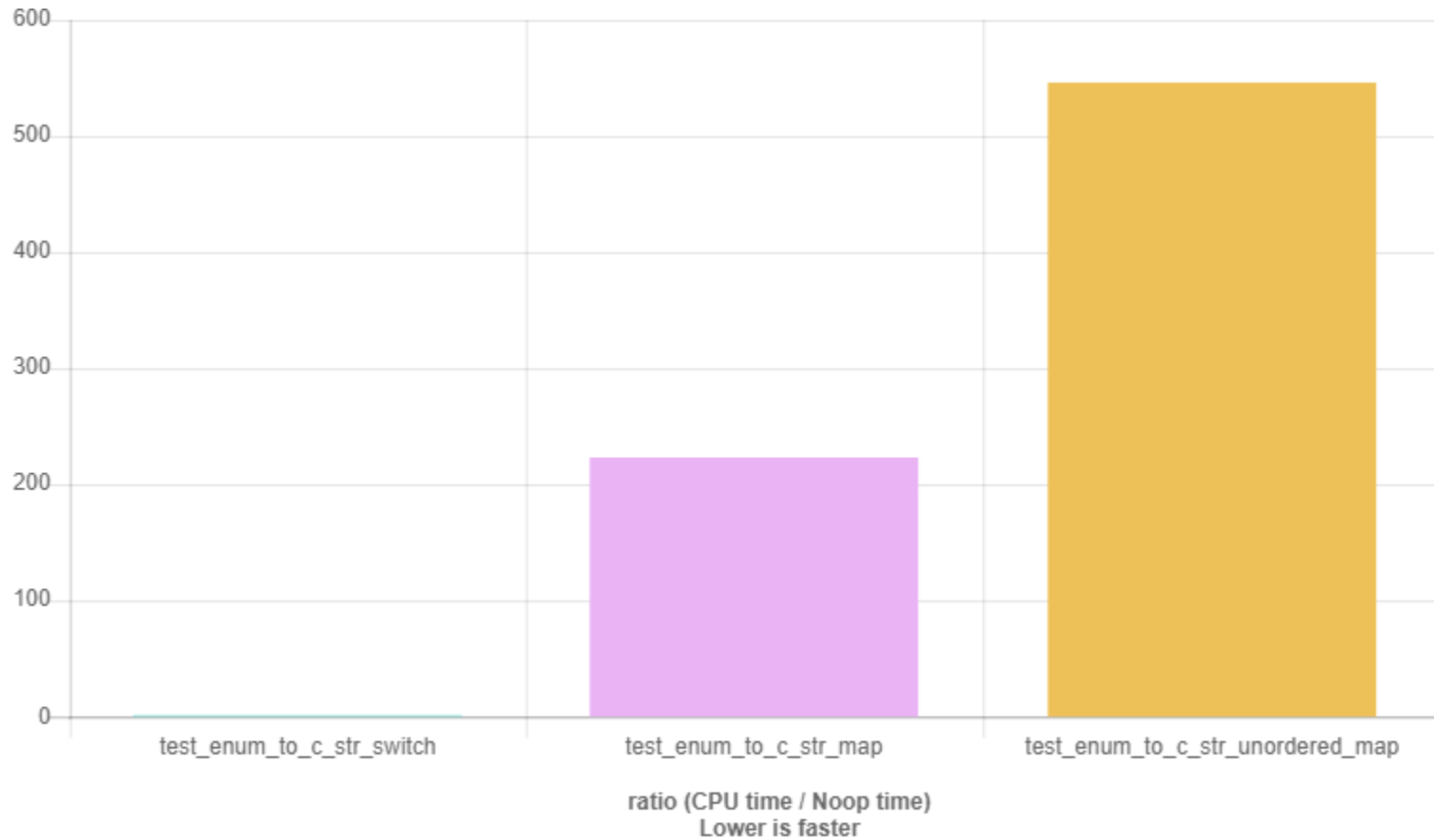
Ćwiczenie 9.4 – Nazwy enumeracji, `std::unordered_map`

```
char const* enum_to_c_str(color v) {  
    static std::unordered_map<color, char const*>  
    const mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```



Ćwiczenie 9.5 – Nazwy enumeracji, porównanie

Ćwiczenie 9.5 – Nazwy enumeracji, porównanie



Ćwiczenie 10 – Konkatenacja napisów

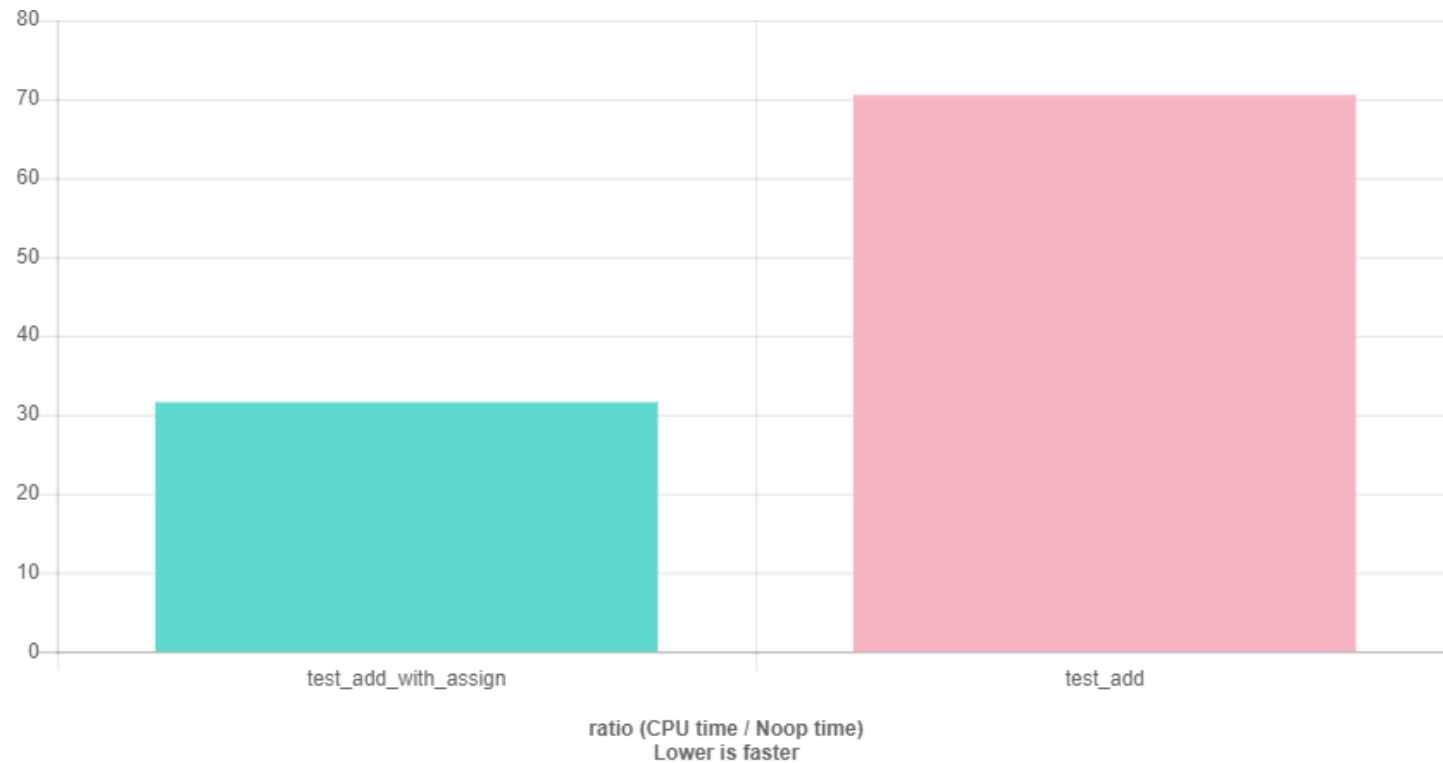
`text += right`

`text = text + right`

Ćwiczenie 10 – Konkatenacja napisów

text += right

text = text + right



Ćwiczenie 11 – Sumowanie, wiele wątków

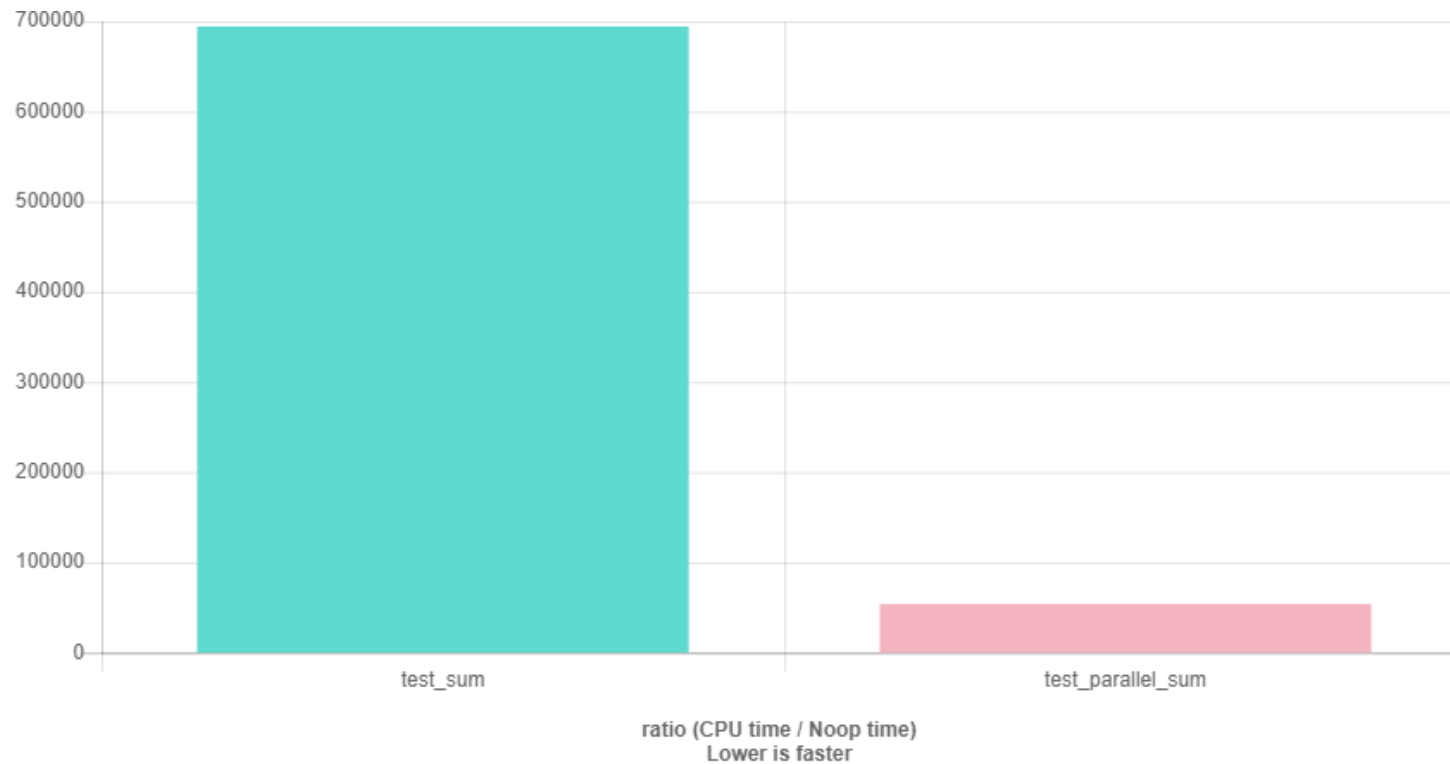
Jeden wątek

Sprzętowa liczba wątków

Ćwiczenie 11 – Sumowanie, wiele wątków

Jeden wątek

Sprzętowa liczba wątków



Ćwiczenie 12 – Sumowanie, bardzo wiele wątków

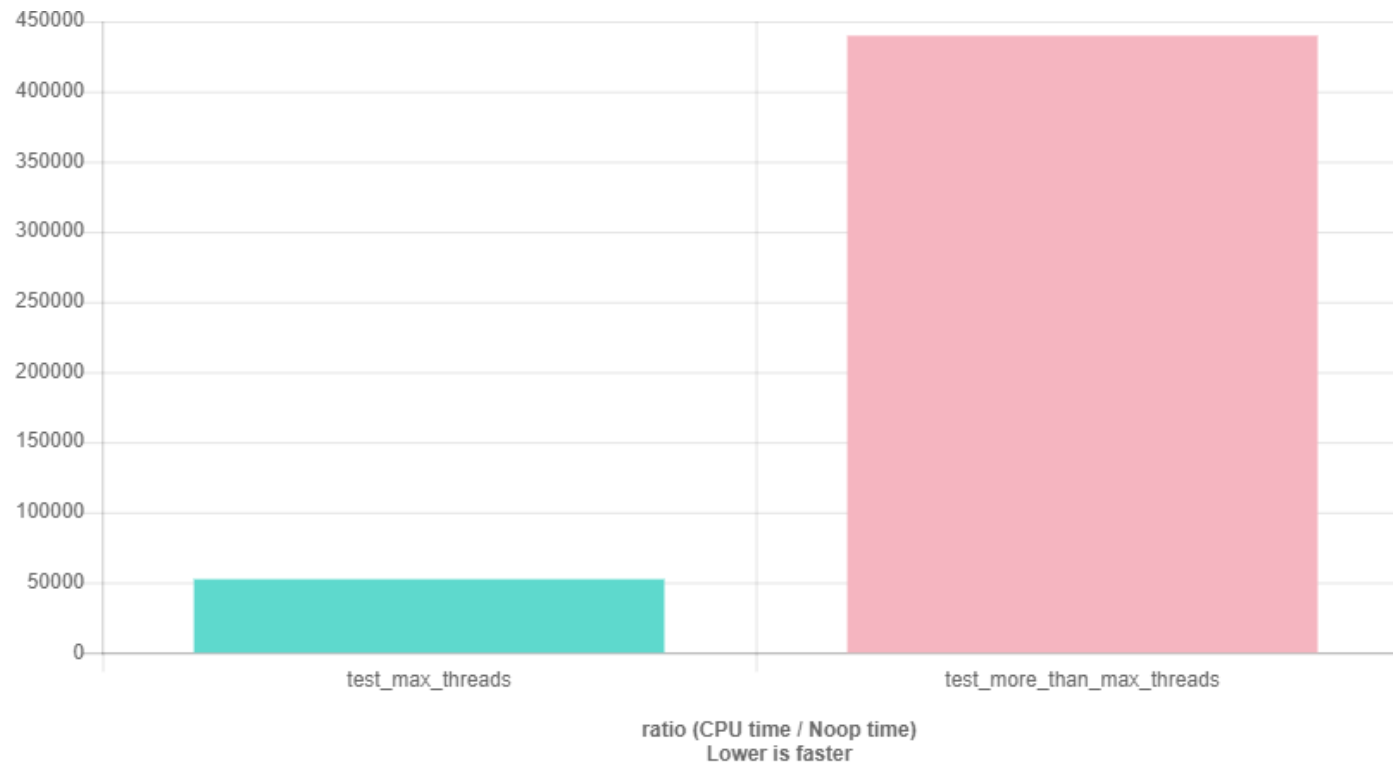
Sprzętowa liczba wątków

Jeszcze więcej wątków

Ćwiczenie 12 – Sumowanie, bardzo wiele wątków

Sprzętowa liczba wątków

Jeszcze więcej wątków



Ćwiczenie 13 – Sumowanie, małe dane

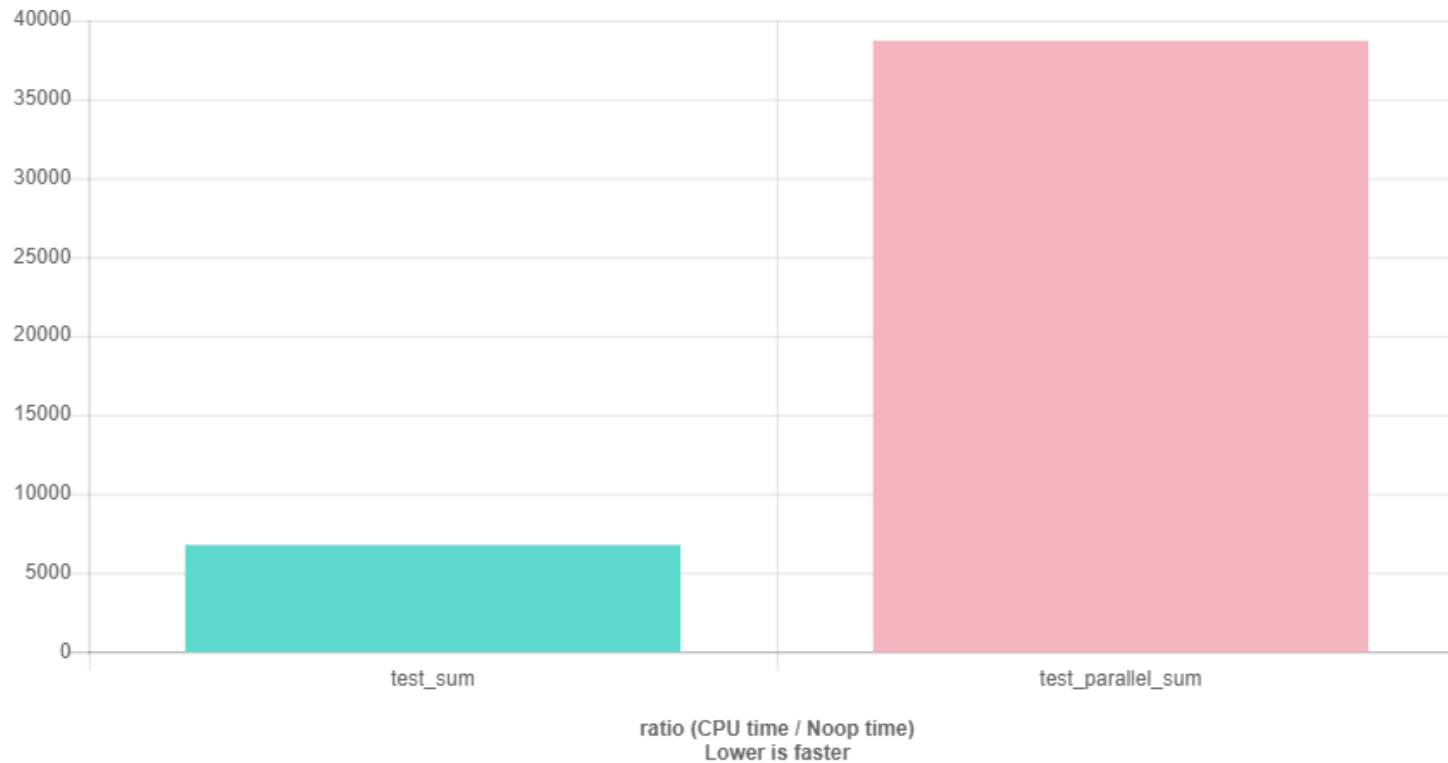
Jeden wątek

Sprzętowa liczba wątków

Ćwiczenie 13 – Sumowanie, małe dane

Jeden wątek

Sprzętowa liczba wątków



Programowanie dynamiczne

Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.

Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.
- Podproblemy są rozwiązywane tylko raz, a wynik jest zapamiętywany („memoizacja”).

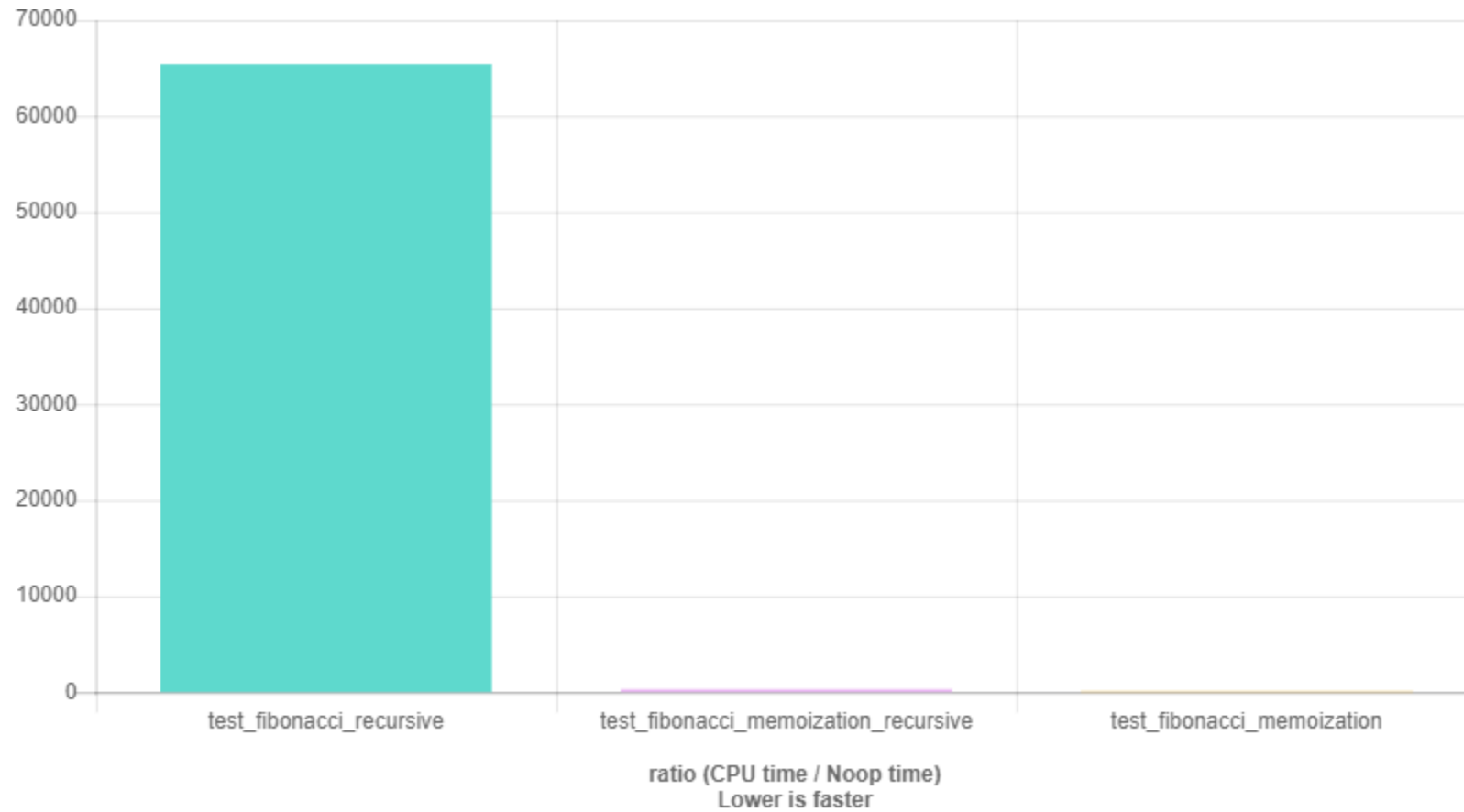
Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.
- Podproblemy są rozwiązywane tylko raz, a wynik jest zapamiętywany („memoizacja”).
- Kluczowe jest rekurencyjne rozbitcie problemu na podproblemy.

Ćwiczenie 14 – Fibonacci

1. rekursja
2. rekursja z memoizacją
3. iteracja z memoizacją

Ćwiczenie 14 – Fibonacci



Ćwiczenie 15 – Dyskretny problem plecakowy

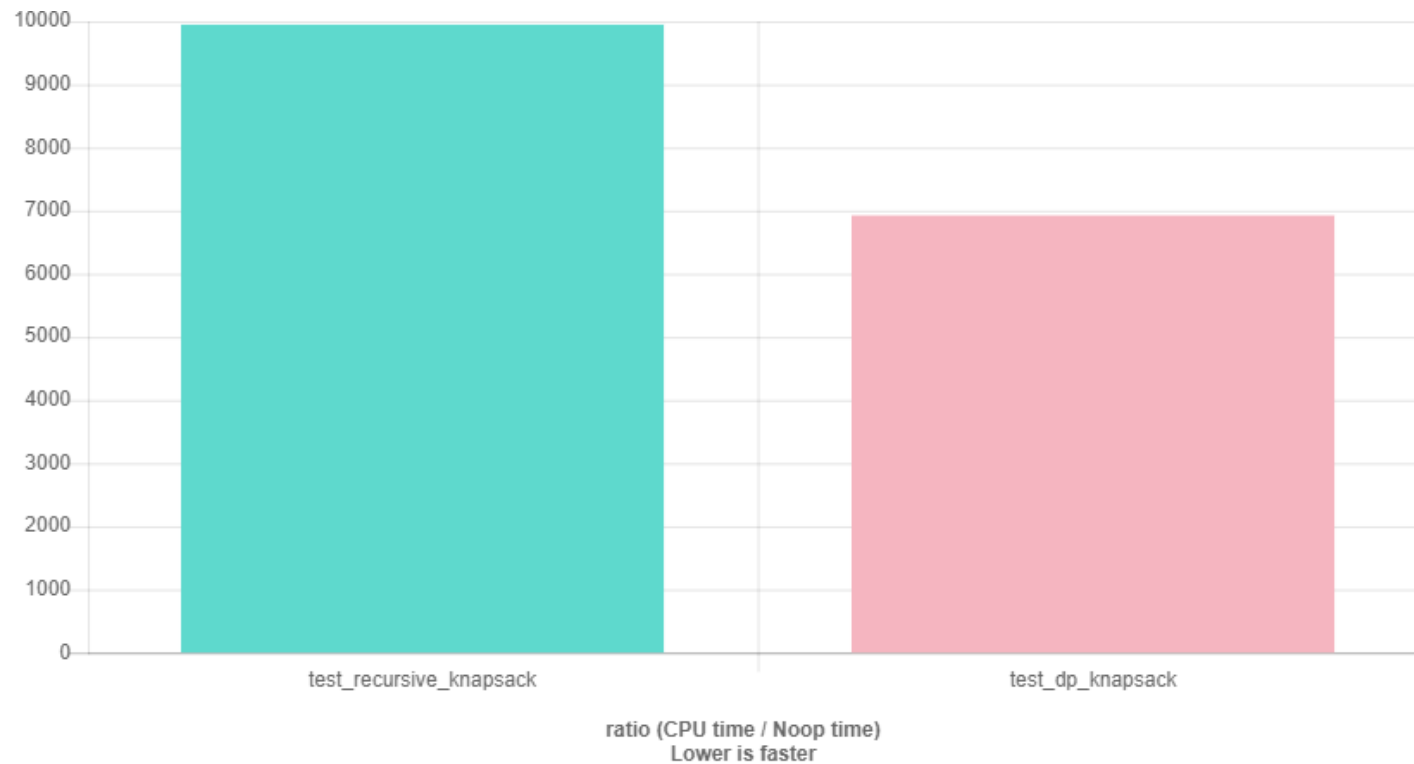
rekurencja

programowanie dynamiczne

Ćwiczenie 15 – Dyskretny problem plecakowy

rekurencja

programowanie dynamiczne



Ćwiczenie 16 – Semantyka przenoszenia

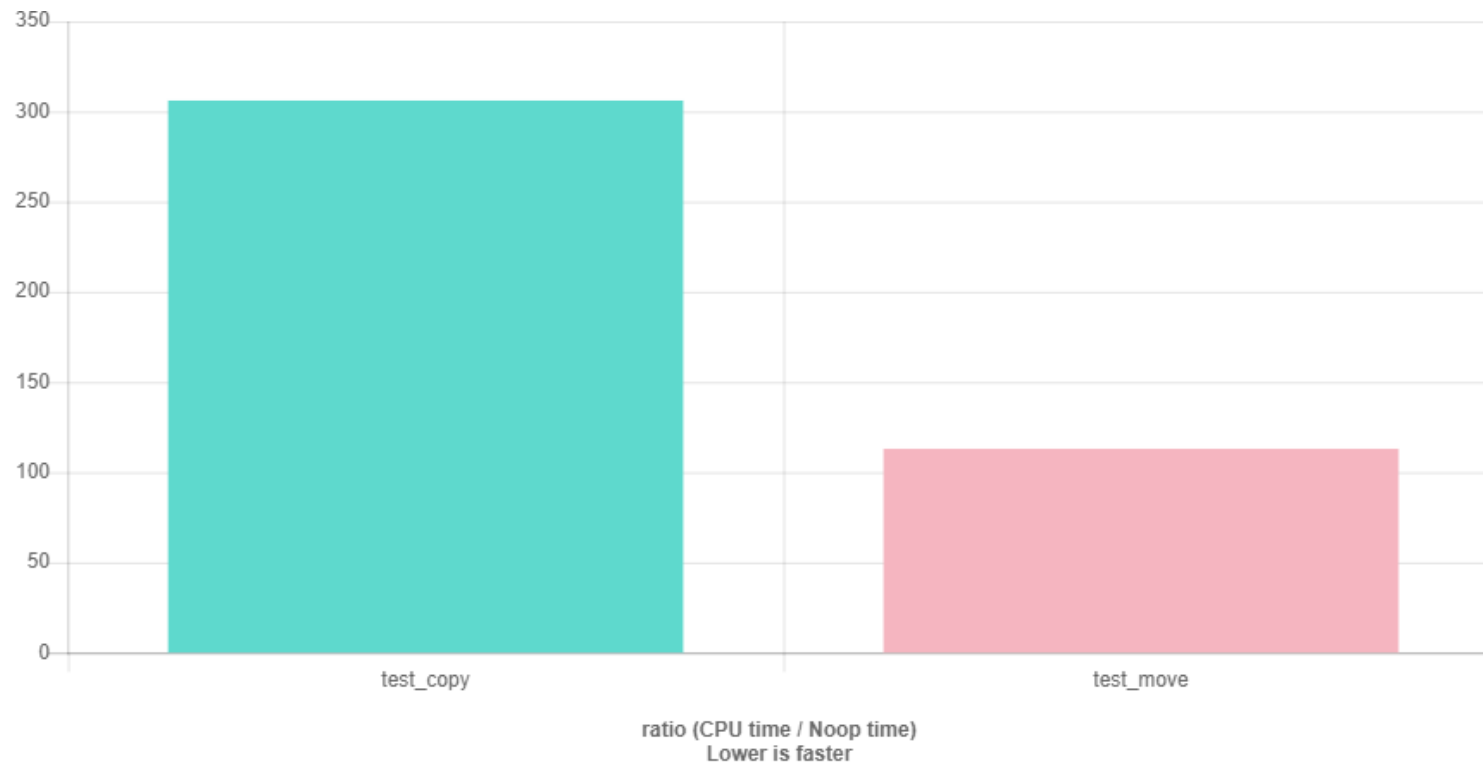
kopiowanie

przenoszenie

Ćwiczenie 16 – Semantyka przenoszenia

kopiowanie

przenoszenie



Profilowanie

Profilowanie

- Wyróżniamy 3 typy profilowania

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.
 - Profilowanie statystyczne – sprawdzanie stosu z określoną częstotliwością.

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.
 - Profilowanie statystyczne – sprawdzanie stosu z określoną częstotliwością.
 - Instrumentalizacja kodu – wprowadzenie do programu dodatkowych funkcji odpowiedzialnych za zbieranie statystyk.

Profilowanie zdarzeń

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.
- Dodatkowe środowisko znacząco wpływa na czas wykonania.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.
- Dodatkowe środowisko znacząco wpływa na czas wykonania.
- Najpopularniejszy profiler zdarzeniowy dla C i C++: valgrind.

valgrind

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.
- Najpopularniejsze narzędzie do sprawdzania wycieków pamięci.

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.
- Najpopularniejsze narzędzie do sprawdzania wycieków pamięci.
- Zapewnia narzędzia do profilowania użycia pamięci cache, profilowania wywołań funkcji, błędów wątków, zużycia pamięci dynamicznej.

Przykładowe użycie valgrind'a

Przykładowe użycie valgrind'a

- `valgrind ls -a`

Przykładowe użycie valgrind'a

- `valgrind ls -a`
- `valgrind -tool=callgrind ls -a`

Przykładowe użycie valgrind'a

- `valgrind ls -a`
- `valgrind -tool=callgrind ls -a`
- W przypadku użycia callgrind'a powstanie plik `callgrind.out.[PID]`, którego możemy użyć w narzędziach do interpretacji takich jak `kcachegrind`, `callgrind_annotate`, czy `gprof2dot`.

kcachegrind – przykład

./callgrind.out [/home/pawluch/CLionProjects/paro_profiling/cmake-build-debug/paro_profiling] — KCachegrind

File View Go Settings Help

Open... Back Forward Up Relative Cycle Detection Relative to Parent Instruction Fetch

Flat Profile

Search: Search Query (No Grouping)

Incl.	Self	Called	Function
91.74	0.00	(0)	0x00000000
59.01	0.00	1	_start
59.01	0.00	1	(below main)
58.50	0.00	1	main
38.33	1.47	1	auto make_data<int, 10000ul>() (paro_profiling: main.cpp)
32.08	0.01	1	_dl_start
32.07	0.01	1	_dl_sysdep_start (ld-2.29.so)
31.85	0.01	1	dl_main
31.17	4.15	8	_dl_relocate_static_initial (ld-2.29.so)
27.27	12.82	3 052	_dl_lookup_symbol_x (ld-2.29.so)
21.48	1.56	10 000	std::vector<int>::operator[] (std::vector<int>:1) (libstdc++.so.6.0.25)
19.33	3.42	10 000	int& std::vector<int>::operator[] (std::vector<int>:1) (libstdc++.so.6.0.25)
16.86	0.00	1	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
16.86	16.86	92 734	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
14.80	1.27	10 000	int std::uniform_int_distribution<int>::operator[] (std::uniform_int_distribution<int>:1) (libstdc++.so.6.0.25)
14.45	10.26	3 052	do_lookup_x (ld-2.29.so)
13.53	4.69	10 000	int std::uniform_int_distribution<int>::operator[] (std::uniform_int_distribution<int>:1) (libstdc++.so.6.0.25)
9.67	2.44	10 000	std::vector<int>::operator[] (std::vector<int>:1) (libstdc++.so.6.0.25)
8.26	0.03	40	start_thread (libc-2.29.so)
8.13	0.04	81	__pthread_once (libc-2.29.so)
7.95	0.01	101	pthread_once (libc-2.29.so)

main

Types Callers All Callers Callee Map Source Code

Ir	Ir per call	Count	Caller
58.50	5 990 636	1	(below main) (libc-2.29.so)

Ir	Ir per call	Count	Callee
38.33	3 924 933	1	auto make_data<int, 10000ul>() (paro_profiling: main.cpp)
16.86	1 726 539	1	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
3.14	321 292	1	auto parallel_sum<__gnu_cxx::__normal_iterator<int const*, std::vector<int>::iterator>, int> (paro_profiling: main.cpp)
0.08	2 050	4	_dl_runtime_resolve_xsave (ld-2.29.so)
0.05	4 893	1	fibonacci_memoization_recursive(unsigned int) (paro_profiling: main.cpp)
0.02	2 233	1	fibonacci_memoization(unsigned int) (paro_profiling: main.cpp)
0.01	394	3	std::ostream::operator<<(std::ostream& (*) (std::ostream&)) (libstdc++.so.6.0.25)
0.01	446	2	std::ostream::operator<<(unsigned int) (libstdc++.so.6.0.25: ostream)
0.00	301	1	std::vector<int, std::allocator<int> >::~~vector() (paro_profiling: stl_vector.h)
0.00	32	1	std::vector<int, std::allocator<int> >::~end() const (paro_profiling: stl_vector.h)
0.00	32	1	std::vector<int, std::allocator<int> >::~begin() const (paro_profiling: stl_vector.h)

Parts Callees Call Graph All Callees Caller Map Machine Code

callgrind.out [1] - Total Instruction Fetch Cost: 10 240 815

Profilowanie statystyczne

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.
- Szybkie.

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.
- Szybkie.
- **Mniej dokładne.**

perf

perf

- Dostępny od jądra 2.6.31.

perf

- Dostępny od jądra 2.6.31.
- Większość funkcjonalności jest zintegrowane z jądrem.

Przykładowe użycie perf'a

Przykładowe użycie perf'a

- Zebranie danych
 - `perf record -g -F 7500 1s`

Przykładowe użycie perf'a

- Zebranie danych

- `perf record -g -F 7500 ls`

- Analiza danych

- `perf report -g 'graph,0.5,caller'`

perf – przykład

```
Samples: 2K of event 'cycles:u', Event count (approx.): 6815962
  Children   Self  Command      Shared Object      Symbol
+ 41.89%    0.00%  paro_profiling [unknown]          [.] 0x5541f689495641d7
+ 41.89%    0.00%  paro_profiling libc-2.29.so        [.] __libc_start_main
- 41.82%    0.00%  paro_profiling paro_profiling      [.] main
- main
  + 24.73% make_data<int, 10000ul>
  + 10.23% fibonacci_recursive
  + 6.24% parallel_sum<__gnu_cxx::__normal_iterator<int const*, std::vector<int, std::
+ 24.73%    1.67%  paro_profiling paro_profiling      [.] make_data<int, 10000ul>
+ 13.94%    0.00%  paro_profiling [unknown]          [.] 0x000cea903d8d4866
+ 13.44%    0.04%  paro_profiling libpthread-2.29.so [.] __pthread_once_slow
+ 12.90%    0.02%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.88%    0.00%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.82%    0.02%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.78%    0.79%  paro_profiling paro_profiling      [.] std::uniform_int_distribut
+ 12.72%    0.04%  paro_profiling paro_profiling      [.] std::__invoke<void (std::
+ 12.69%    0.15%  paro_profiling paro_profiling      [.] std::__invoke_impl<void, vo
+ 12.68%   12.68%  paro_profiling ld-2.29.so          [.] do_lookup_x
+ 12.54%    0.08%  paro_profiling paro_profiling      [.] std::__future_base::_State
+ 11.99%    0.09%  paro_profiling paro_profiling      [.] std::function<std::unique_p
+ 11.89%    0.02%  paro_profiling paro_profiling      [.] std::_Function_handler<std:
+ 11.78%    2.22%  paro_profiling paro_profiling      [.] std::uniform_int_distribut
+ 11.30%    0.00%  paro_profiling paro_profiling      [.] std::__future_base::_Task_s
+ 10.46%    0.00%  paro_profiling paro_profiling      [.] std::thread::_Invoker<std:
```

Podsumowanie

Podsumowanie

- Najpierw algorytm.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza
 - kompilatorowi w optymalizowaniu,

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- **Za to często przeszkadza**
 - kompilatorowi w optymalizowaniu,
 - ludziom (w tym Tobie!) w rozumieniu.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza
 - kompilatorowi w optymalizowaniu,
 - ludziom (w tym Tobie!) w rozumieniu.
- **Testuj!**

Patrz też

1. [CppCon 2014: Andrei Alexandrescu "Optimization Tips - Mo' Hustle Mo' Problems"](#)
2. [code::dive conference 2015 - Andrei Alexandrescu - Writing Fast Code I](#)
3. [code::dive conference 2015 - Andrei Alexandrescu - Writing Fast Code II](#)
4. [There's Treasure Everywhere - Andrei Alexandrescu](#)
5. [Fastware - Andrei Alexandrescu](#)
6. [CppCon 2015: Chandler Carruth "Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My!,,](#)
7. [CppCon 2017: Matt Godbolt "What Has My Compiler Done for Me Lately? Unbolting the Compiler's Lid"](#)
8. [CppCon 2018: Nir Friedman "Understanding Optimizers: Helping the Compiler Help You"](#)
9. [Type punning done right - Andreas Weis - Lightning Talks Meeting C++ 2017](#)
10. [Performance Tools Developments](#)