# Class layout

Anđelković Miloš

# About me

- Use to be a teaching assistant
- Working as a C++ developer for a couple of years now
- Hoping to go back into teaching

# Creating a class

- You need some member variables and member functions
- Some public some private
- You put them in any order

# Creating a class

```
struct my_struct {
    int value_1_;
    bool is_value_1_set_;
    int value_2_;
    bool is_value_2_set_;

  private:
    double value_3_;
    bool is_value_3_set_{false};

  public:
    std::string name;

    double get_value_3() {
        if (!is_value_3_set_) {
            throw std::runtime_error("Value not set");
        }
        return value_3_;
    }

    void set_value_3(double value) {
        value_3_ = value;
        is_value_3_set_ = true;
    }
};
```

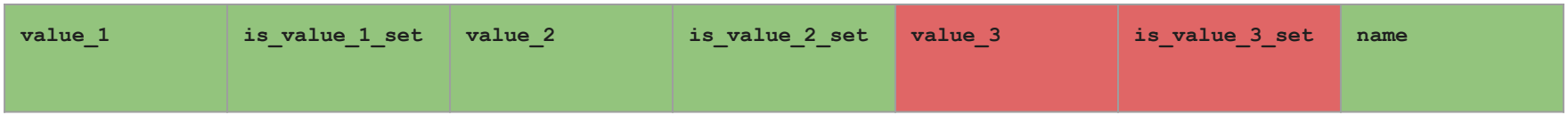
# The standard (cppref)

# Class layout

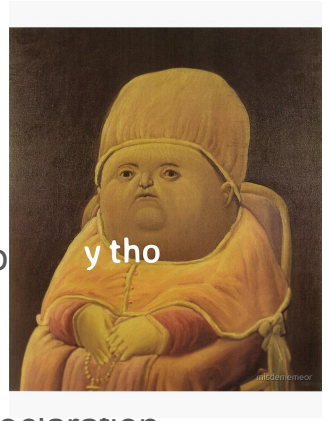
- Won’t cover the case of virtual inheritance
- Non-static data members contribute to the size of the object
- Static data members are in a different memory
- Enums have the same scope as the class itself

# Class layout

- Sequential in memory
- First variable at the 0 offset



# The standard




- Pre C++11
  - Order or member variables within a block is same as order of declaratio
  - Blocks can be rearranged in any way
- C++11 to C++23
  - Order of the member variables with same access is same as order of declaration
  - Order of member variables with different access is unspecified
- C++23
  - Make declaration order layout mandated
  - As you'd expect
- Language support for class layout control

# Subobjects

An object can have subobjects. These include

- member objects
- base class subobjects
- array elements

An object that is not a subobject of another object is called complete object

The size of an object that is neither potentially overlapping nor a bit-field is required to be non-zero

Any two objects with overlapping lifetimes (that are not bit-fields) are guaranteed to have different addresses unless one of them is nested within another, or if they are subobjects of different type within the same complete object, and one of them is a subobject of zero size.

Object - cppreference.com

# Object alignment

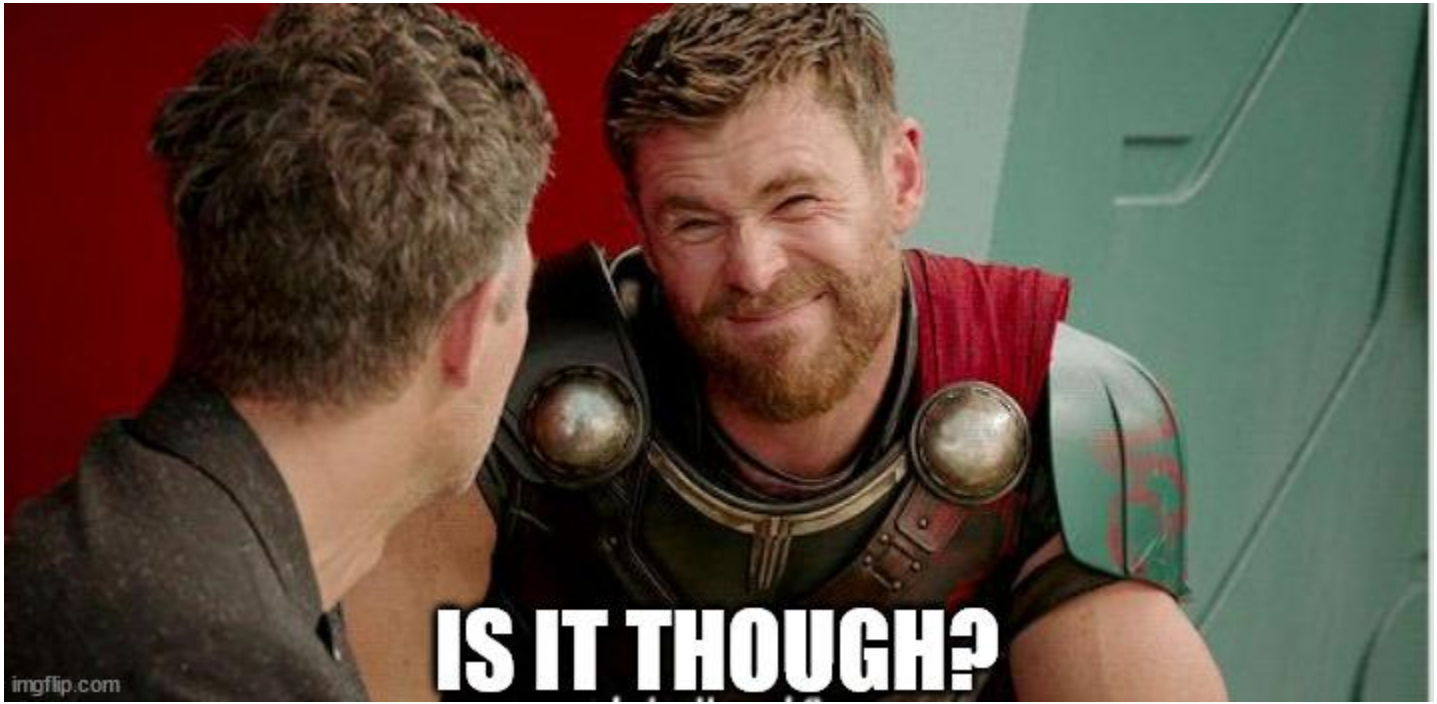
- Every object type has the property called alignment requirement, which is a nonnegative integer value (of type std::size_t, and always a power of two) representing the number of bytes between successive addresses at which objects of this type can be allocated.
- alignof or std::alignment_of can be used to get the alignment requirement of a type
- alignas can be used to set the alignment requirement of a type

# More on alignment

- std::align
- std::aligned_storage
- std::assume_aligned
- std::aligned_alloc
- #pragma push(N) / #pragma pop(N)

# Alignment OS

- On 32bit OS word size is 4 bytes
- On 64bit OS word size is 8 bytes
- In order to avoid multiple reads for single variable data needs to be aligned


IS IT THOUGH?
imgflip.com

### 4-byte integers

| offset | Core i7 | Core 2 |
|---|---|---|
| 0 | 28.1 | 34.1 |
| 1 | 28.7 | 38.1 |
| 2 | 28.7 | 38.1 |
| 3 | 28.7 | 38.1 |

### 8-byte integers

| offset | Core i7 | Core 2 |
|---|---|---|
| 0 | 33.6 | 69.1 |
| 1 | 33.6 | 77 |
| 2 | 33.6 | 77 |
| 3 | 33.6 | 77 |
| 4 | 33.6 | 77 |
| 5 | 33.6 | 77 |
| 6 | 33.6 | 77 |
| 7 | 33.6 | 77 |

https://lemire.me/blog/2012/05/31/data-alignment-for-speed-myth-or-reality/

# The Examples

# Base classes

```
struct first_base {
   int b;
};

struct second_base {};

struct third_base {
   int c;
};

struct inheritance : public first_base, second_base, third_base {};
```

https://godbolt.org/z/7b1vhzvr6

sizeof(first_base) = 4
sizeof(second_base) = 1
sizeof(third_base) = 4
sizeof(inheritance) = 8

# Base classes

- Usually the space for subobjects of base classes is before the space for derived
- Usually the space for subobjects of base classes is in the order of declaration
- Empty/Stateless classes take 1 byte

# Empty Base Class Optimization

- Compiler can optimize space for empty base classes
- Compiler can permute the order of subobjects of base classes
- EBO is compiler dependent

# EBO

```
struct second_base {};

struct first_base : public second_base{
   int b;
};

struct third_base : public second_base {
   int c;
};

struct inheritance :
     public first_base,third_base
{};
```

https://godbolt.org/z/eoT1efrGd

sizeof(second_base) = 1
sizeof(first_base) = 4
sizeof(third_base) = 4
sizeof(inheritance) =
depends on who you ask

gcc 13.2 = 8
clang 17.0.1 = 8
MSVC v19.37 = 12

# Composition

```
struct first_base {
   int b;
};

struct second_base {};

struct third_base {
   int c;
};

struct composition {
   first_base f;
   second_base s;
   third_base t;
};
```

https://godbolt.org/z/EKqz9zes6

sizeof(first_base) = 4
sizeof(second_base) = 1
sizeof(third_base) = 4
sizeof(composition) = 12

# Composition

```
struct first_base {
   int b;
};

struct second_base {};

struct third_base {
   int c;
};

struct composition {
   first_base f;
   [[no_unique_address]] second_base s;
   third_base t;
};
```

https://godbolt.org/z/EKqz9zes6

sizeof(first_base) = 4
sizeof(second_base) = 1
sizeof(third_base) = 4
sizeof(composition) = 8

# [[no_unique_address]]

- Applies to the name being declared in the declaration of a non-static data member that's not a bit-field.
- Makes this member subobject potentially-overlapping, i.e., allows this member to be overlapped with other non-static data members or base class subobjects of its class.
- If the member is not empty, any tail padding in it may be also reused to store other data members.
- C++ attribute: no_unique_address (since C++20) - cppreference.com

# [[no_unique_address]]

https://godbolt.org/z/6Y58c91GT

```
struct composition {
     first_base f;
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
     third_base t;
};
```

gcc 13.2 = 8
clang 17.0.1 = 8
MSVC v19.37 = 8

# [[no_unique_address]] https://godbolt.org/z/6Y58c91GT

```
struct composition {
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
     first_base f;
     third_base t;
};
```

gcc 13.2 = 8
clang 17.0.1 = 8
MSVC v19.37 = 8

# [[no_unique_address]]

https://godbolt.org/z/6Y58c91GT

```
struct composition {
     first_base f;
     third_base t;
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
};
```

gcc 13.2 = 12
clang 17.0.1 = 12
MSVC v19.37 = 8

# [[no_unique_address]] https://godbolt.org/z/6Y58c91GT

```
struct composition {
     first_base f;
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
     [[no_unique_address]] second_base s5;
     third_base t;
};
```

gcc 13.2 = 8
clang 17.0.1 = 8
MSVC v19.37 = 12

# [[no_unique_address]] https://godbolt.org/z/6Y58c91GT

```
struct composition {
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
     [[no_unique_address]] second_base s5;
     first_base f;
     third_base t;
};
```

gcc 13.2 = 8
clang 17.0.1 = 8
MSVC v19.37 = 12

# [[no_unique_address]]

https://godbolt.org/z/6Y58c91GT

```
struct composition {
     first_base f;
     third_base t;
     [[no_unique_address]] second_base s1;
     [[no_unique_address]] second_base s2;
     [[no_unique_address]] second_base s3;
     [[no_unique_address]] second_base s4;
     [[no_unique_address]] second_base s5;
};
```

gcc 13.2 = 12
clang 17.0.1 = 12
MSVC v19.37 = 8


But It Is Not This Day!

# Standard layout type

- No virtual functions and virtual base classes
- Zero or more base classes that are standard layout
- No two base classes are of the same type
- Zero or more non-static members such that they all have the same access and all or declared either in the most derived class or the same base class
- std::is_standard_layout
- https://godbolt.org/z/56ErveY6s

# A closer look at your class layout

offsetof - Takes a type and a member variable and expands to an integral constant of type std::size_t, which is the value of offset of subobject within the type

```
#include <cstddef>
#include <iostream>
struct S
{
   char   m0;
   double m1;
   short  m2;
   char   m3;
};
int main()
{
   std::cout
       << "offset of char   m0 = " << offsetof(S, m0) << '\n'
       << "offset of double m1 = " << offsetof(S, m1) << '\n'
       << "offset of short  m2 = " << offsetof(S, m2) << '\n'
       << "offset of char   m3 = " << offsetof(S, m3) << '\n';
}
```

```
offset of char   m0 = 0
offset of double m1 = 8
offset of short  m2 = 16
offset of char   m3 = 18
```

# A closer look at your class layout

clang++ -cc1 -fdump-record-layouts (or -Xclang -fdump-record-layouts)

g++ -fdump-lang-class (or -fdump-class-hierarchy) (dumps to a file, so not usable in Compiler explorer)

msvc /d1reportAllClassLayout

# A closer look at your class layout

```
*** Dumping AST Record Layout
         0 | struct first_base
         0 |   int b
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]

*** Dumping AST Record Layout
         0 | struct second_base (empty)
           | [sizeof=1, dsize=1, align=1,
           |  nvsize=1, nvalign=1]

*** Dumping AST Record Layout
         0 | struct third_base
         0 |   int c
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]
```

```
*** Dumping AST Record Layout
         0 | struct composition_without_no_unique_address
         0 |   struct first_base f
         0 |     int b
         4 |   struct second_base s (empty)
         8 |   struct third_base t
         8 |     int c
           | [sizeof=12, dsize=12, align=4,
           |  nvsize=12, nvalign=4]
```

# A closer look at your class layout

```
*** Dumping AST Record Layout
         0 | struct first_base
         0 |   int b
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]

*** Dumping AST Record Layout
         0 | struct second_base (empty)
           | [sizeof=1, dsize=1, align=1,
           |  nvsize=1, nvalign=1]

*** Dumping AST Record Layout
         0 | struct third_base
         0 |   int c
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]
```

```
*** Dumping AST Record Layout
         0 | struct composition_with_no_unique_adress
         0 |   struct first_base f
         0 |     int b
         0 |   struct second_base s (empty)
         4 |   struct third_base t
         4 |     int c
           | [sizeof=8, dsize=8, align=4,
           |  nvsize=8, nvalign=4]
```

# A closer look at your class layout

```
*** Dumping AST Record Layout
         0 | struct first_base
         0 |   int b
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]

*** Dumping AST Record Layout
         0 | struct second_base (empty)
           | [sizeof=1, dsize=1, align=1,
           |  nvsize=1, nvalign=1]

*** Dumping AST Record Layout
         0 | struct third_base
         0 |   int c
           | [sizeof=4, dsize=4, align=4,
           |  nvsize=4, nvalign=4]
```

```
*** Dumping AST Record Layout
         0 | struct inheritance
         0 |   struct first_base (base)
         0 |     int b
         0 |   struct second_base (base) (empty)
         4 |   struct third_base (base)
         4 |     int c
           | [sizeof=8, dsize=8, align=4,
           |  nvsize=8, nvalign=4]
```

# A closer look at your class layout

```
*** Dumping AST Record Layout
         0 | struct my_struct
         0 |   int value_1_
         4 |   _Bool is_value_1_set_
         8 |   int value_2_
        12 |   _Bool is_value_2_set_
        16 |   double value_3_
        24 |   _Bool is_value_3_set_
        32 |   class std::basic_string<char> name
        32 |     struct std::basic_string<char>::_Alloc_hider _M_dataplus
        32 |       class std::allocator<char> (base) (empty)
        32 |         class std::__new_allocator<char> (base) (empty)
        32 |       std::basic_string<char>::pointer _M_p
        40 |     std::basic_string<char>::size_type _M_string_length
        48 |     union std::basic_string<char>::(anonymous at /usr/bin/…/basic_string.h:217:7)
        48 |       char[16] _M_local_buf
        48 |       std::basic_string<char>::size_type _M_allocated_capacity
           | [sizeof=64, dsize=64, align=8,
           |  nvsize=64, nvalign=8]
```

# A closer look at your class layout

```
class inheritance    size(12):
      +---
 0    | +--- (base class first_base)
 0    | | +--- (base class second_base)
      | | +---
 0    | | b
      | +---
 8    | +--- (base class third_base)
 8    | | +--- (base class second_base)
      | | +---
 8    | | c
      | +---
      +---
```

```
*** Dumping AST Record Layout
         0 | struct inheritance
         0 |   struct first_base (base)
         0 |     struct second_base (base) (empty)
         0 |     int b
         4 |   struct third_base (base)
         4 |     struct second_base (base) (empty)
         4 |     int c
           | [sizeof=8, dsize=8, align=4,
           |  nvsize=8, nvalign=4]
```

# A closer look at your class layout

```
class composition    size(8):
    +---
0   | first_base f
0   | second_base s1
1   | second_base s2
2   | second_base s3
3   | second_base s4
4   | third_base t
    +---
```

```
*** Dumping AST Record Layout
        0 | struct composition
        0 |   struct first_base f
        0 |     int b
        0 |   struct second_base s1 (empty)
        4 |   struct second_base s2 (empty)
        5 |   struct second_base s3 (empty)
        6 |   struct second_base s4 (empty)
        4 |   struct third_base t
        4 |     int c
          | [sizeof=8, dsize=8, align=4,
          |  nvsize=8, nvalign=4]
```


But It Is Not This Day!

# Don’t assume, assert

```
static_assert(offsetof(inheritance, c) == sizeof(first_base));
  static_assert(sizeof(inheritance) == sizeof(first_base) +
                                       sizeof(third_base));
```

# The Performance

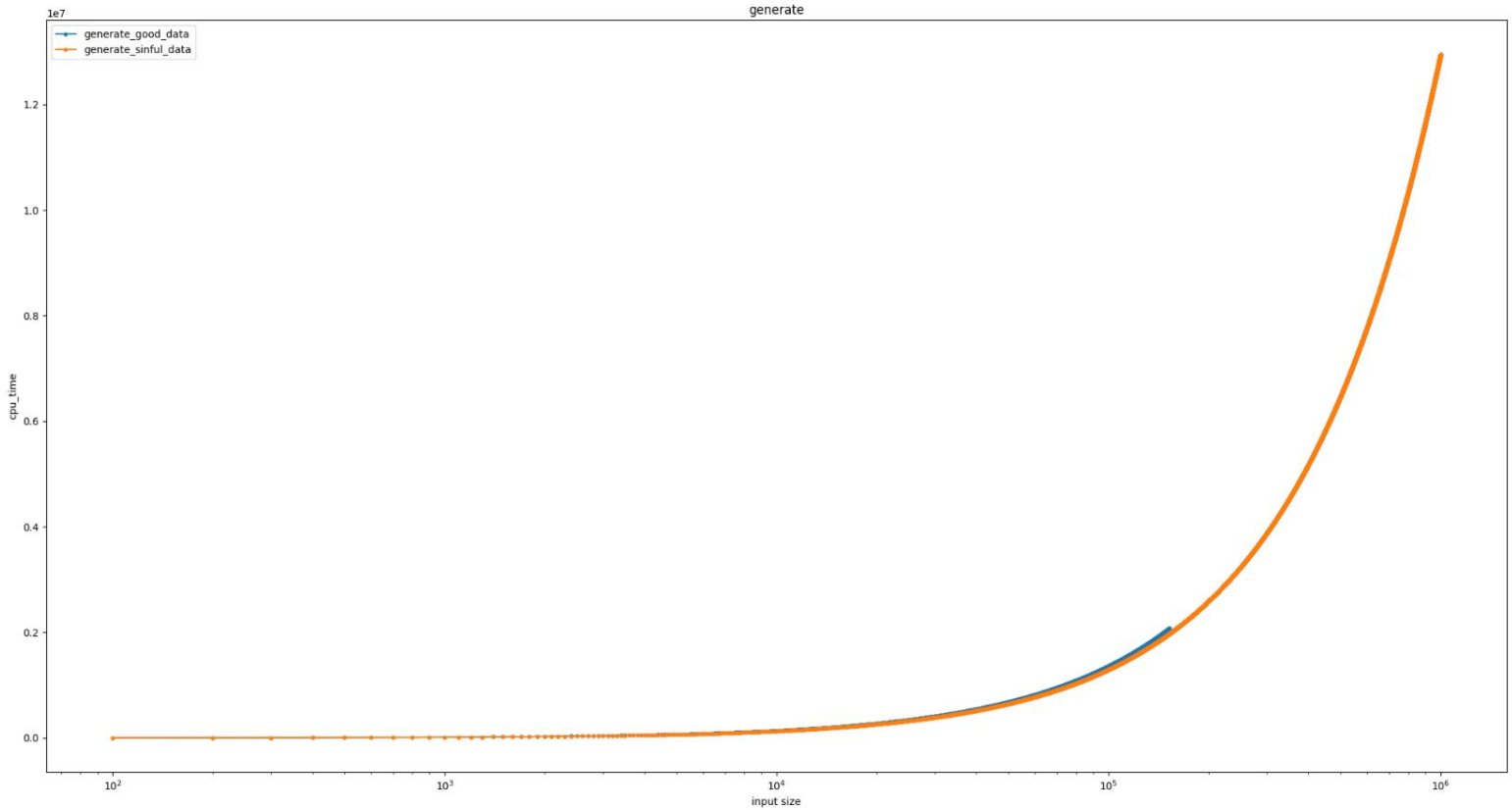
# Benchmark

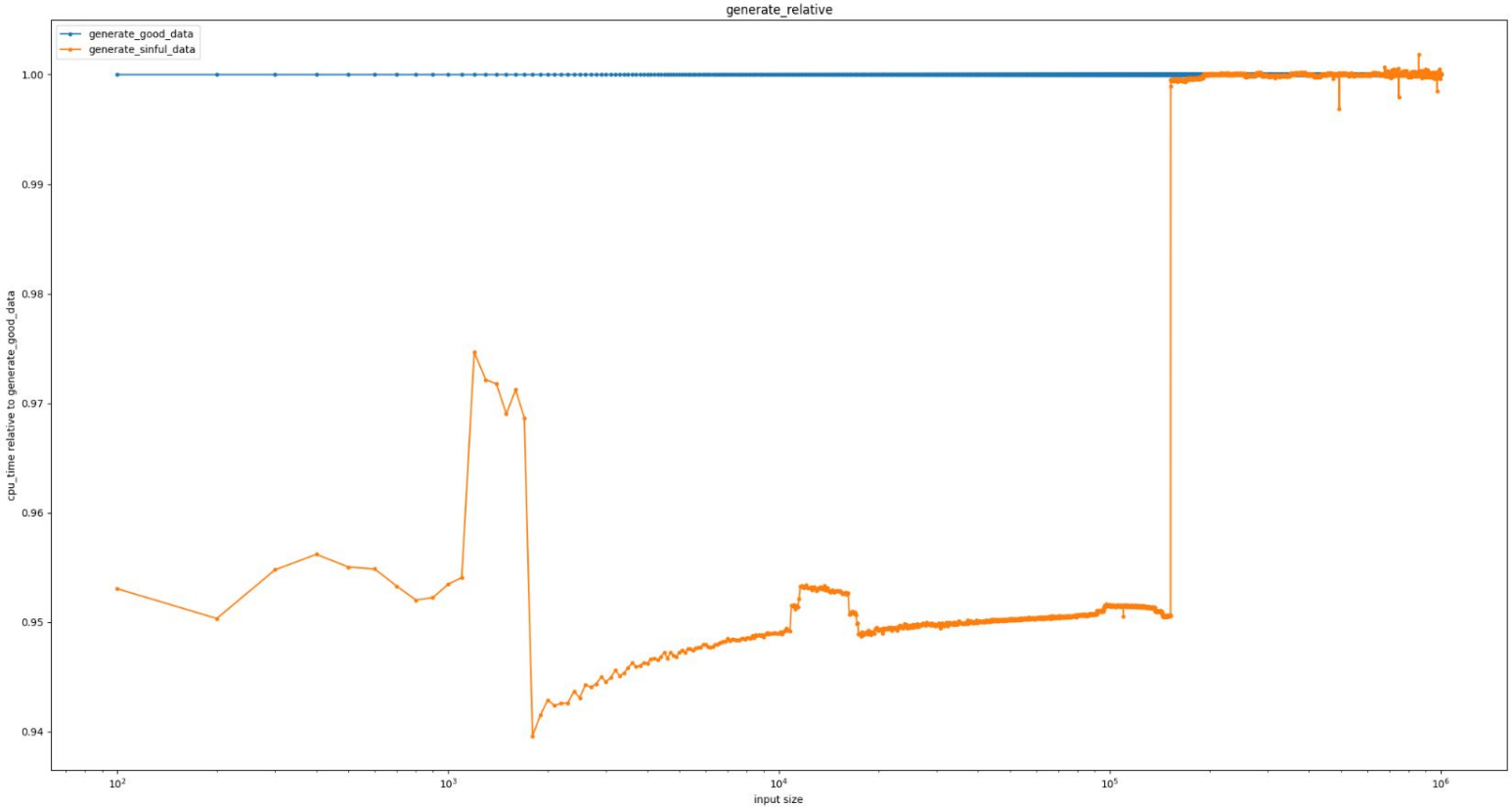
```
struct good_data {
   int value_;
   bool b1_, b2_;

   constexpr good_data(int val, bool b1, bool b2)
       : value_{val}, b1_{b1}, b2_{b2} {}

   bool operator<(good_data const& rhs) const {
       return value_ < rhs.value_ ||
              b1_ < rhs.b1_ ||
              b2_ < rhs.b2_;
   }
};
```

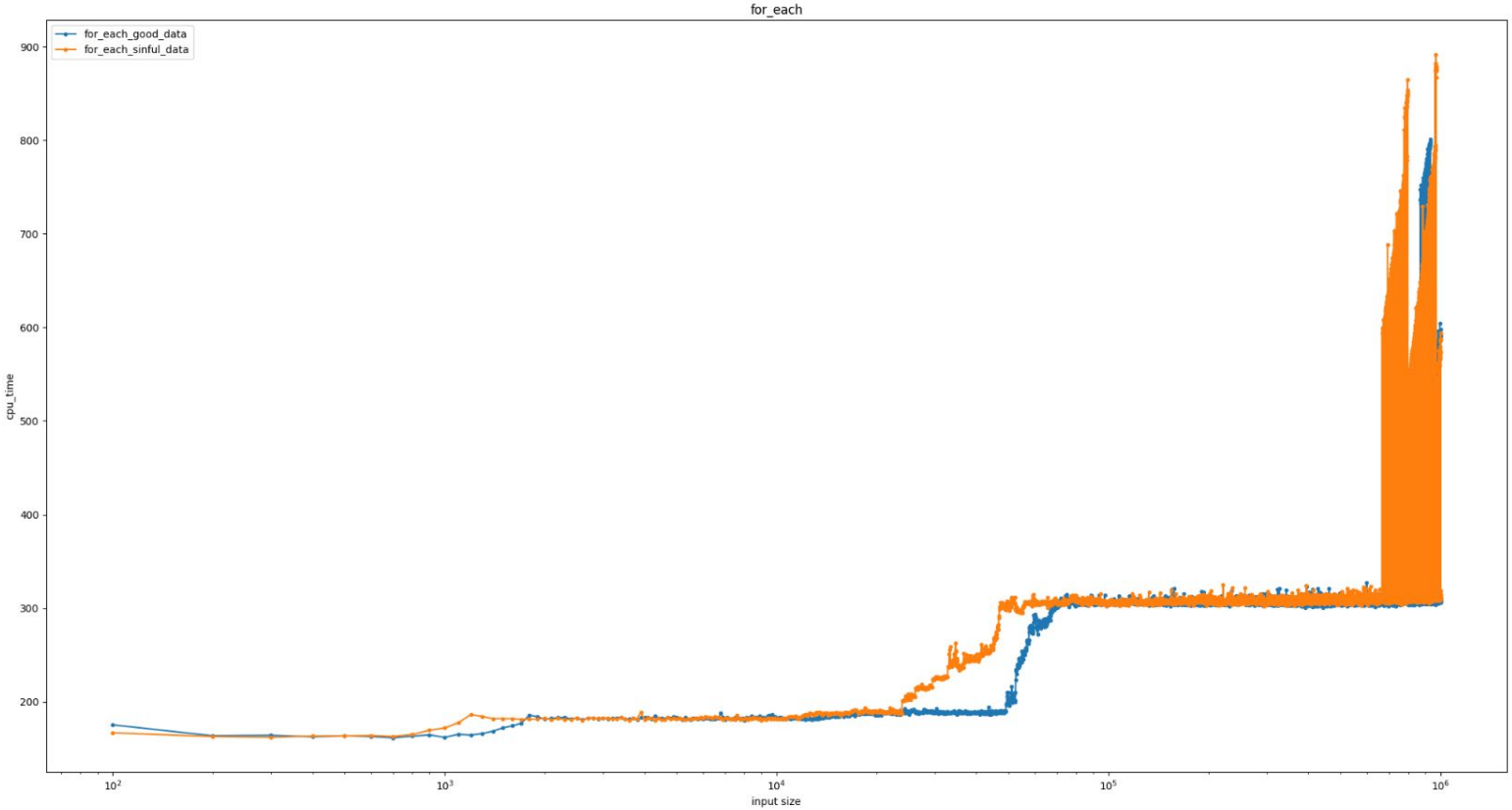
```
struct sinful_data {
   bool b1_;
   int value_;
   bool b2_;

   constexpr sinful_data(int val, bool b1, bool b2)
       : b1_{b1}, value_{val}, b2_{b2} {}

   bool operator<(sinful_data const& rhs) const {
       return value_ < rhs.value_ ||
              b1_ < rhs.b1_ ||
              b2_ < rhs.b2_;
   }
};
```

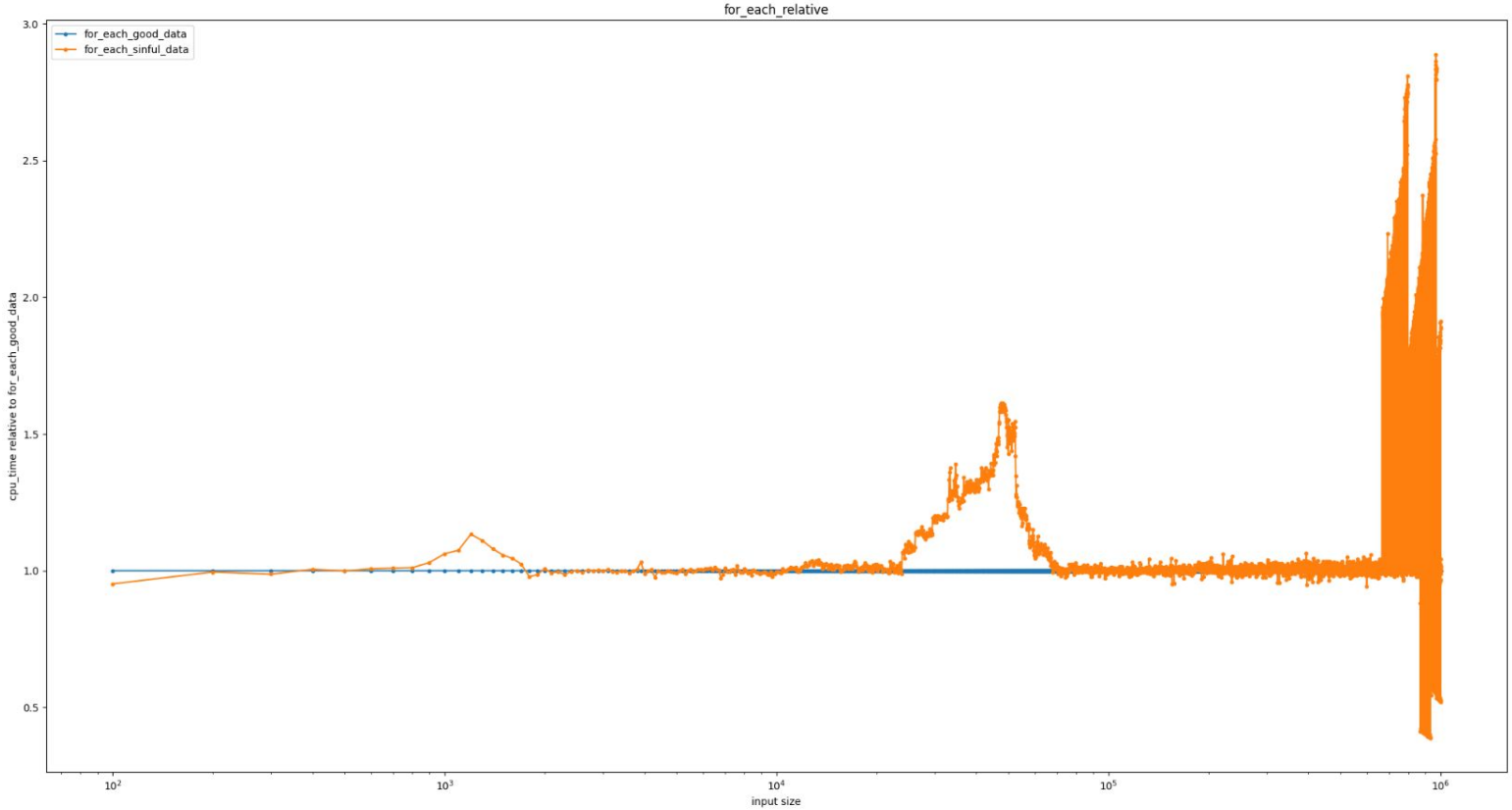
# Setup

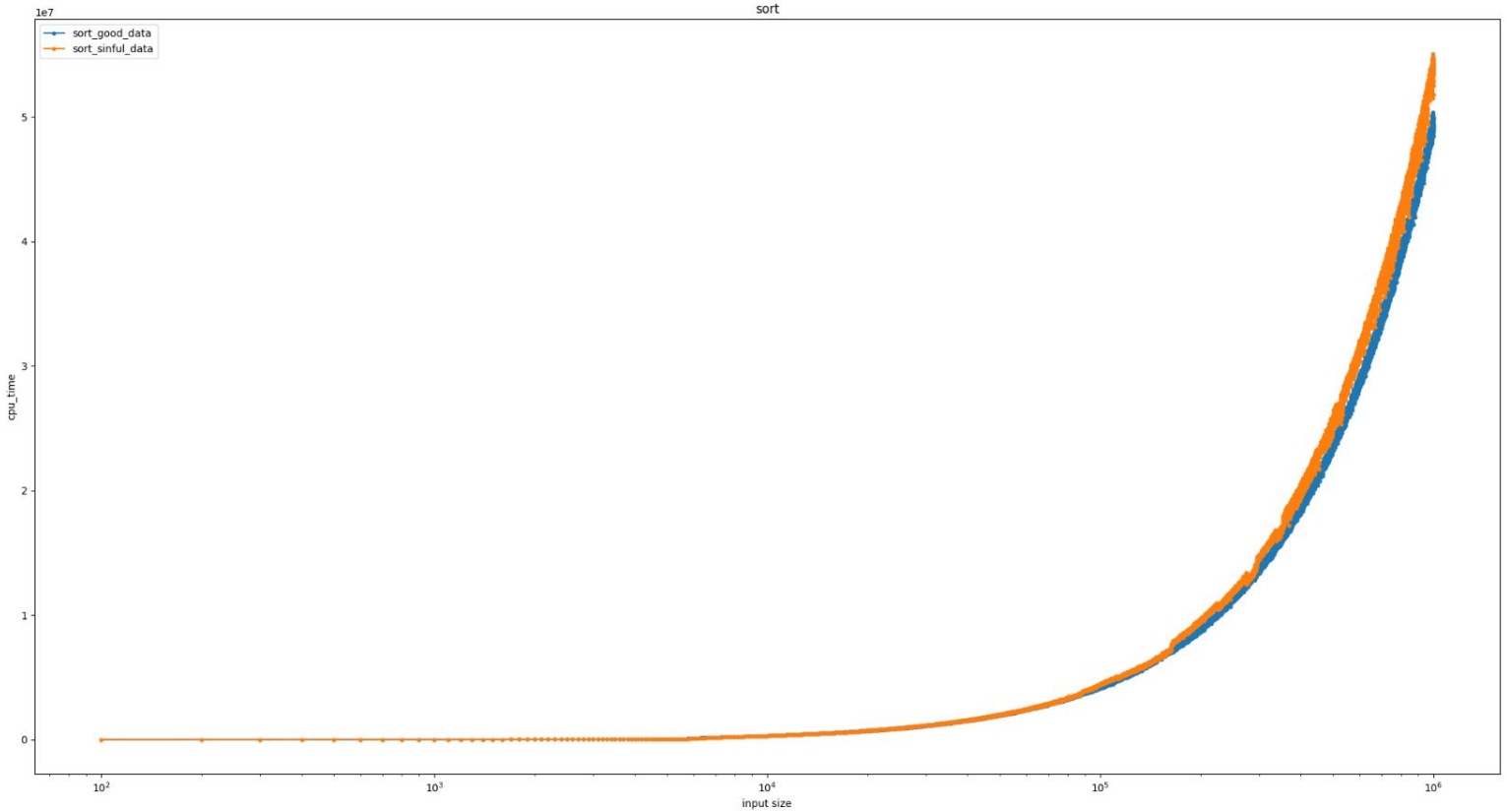
- Processor: AMD Ryzen™ 9 7950X
- OS: Manjaro
- Google Benchmark https://github.com/google/benchmark
- Num iterations: 100
- Data sizes: 100 to 1`000`000, with 100 increment
- Functions: generate_data, for_each, sort, accumulate
- Github repo

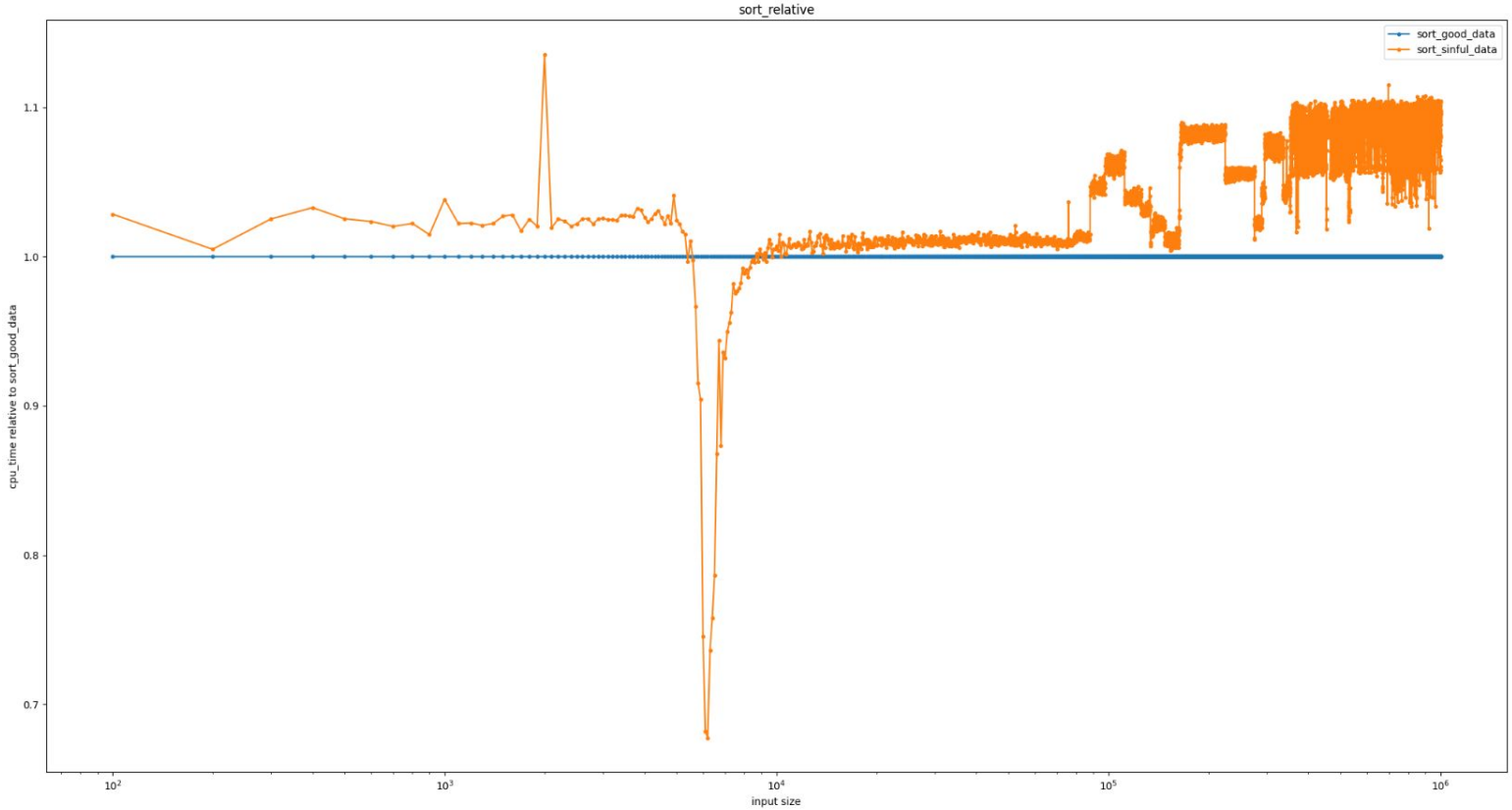

generate
1e7
generate_good_data
generate_sinful_data
1.2
1.0
0.8
0.6
0.4
0.2
0.0
cpu_time
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$
input size

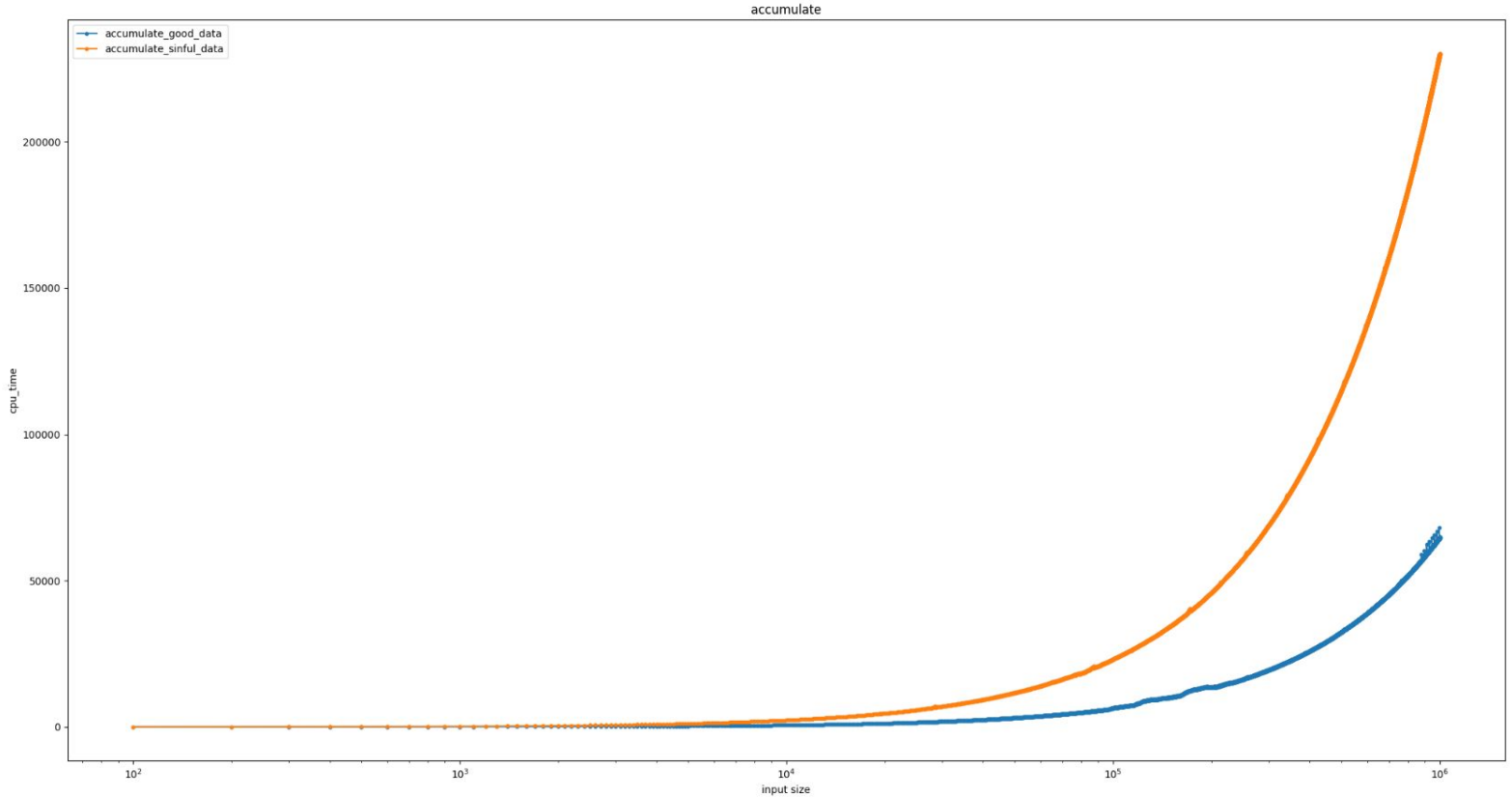

generate_relative
generate_good_data
generate_sinful_data
1.00
0.99
0.98
0.97
0.96
0.95
0.94
cpu_time relative to generate_good_data
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$
input size

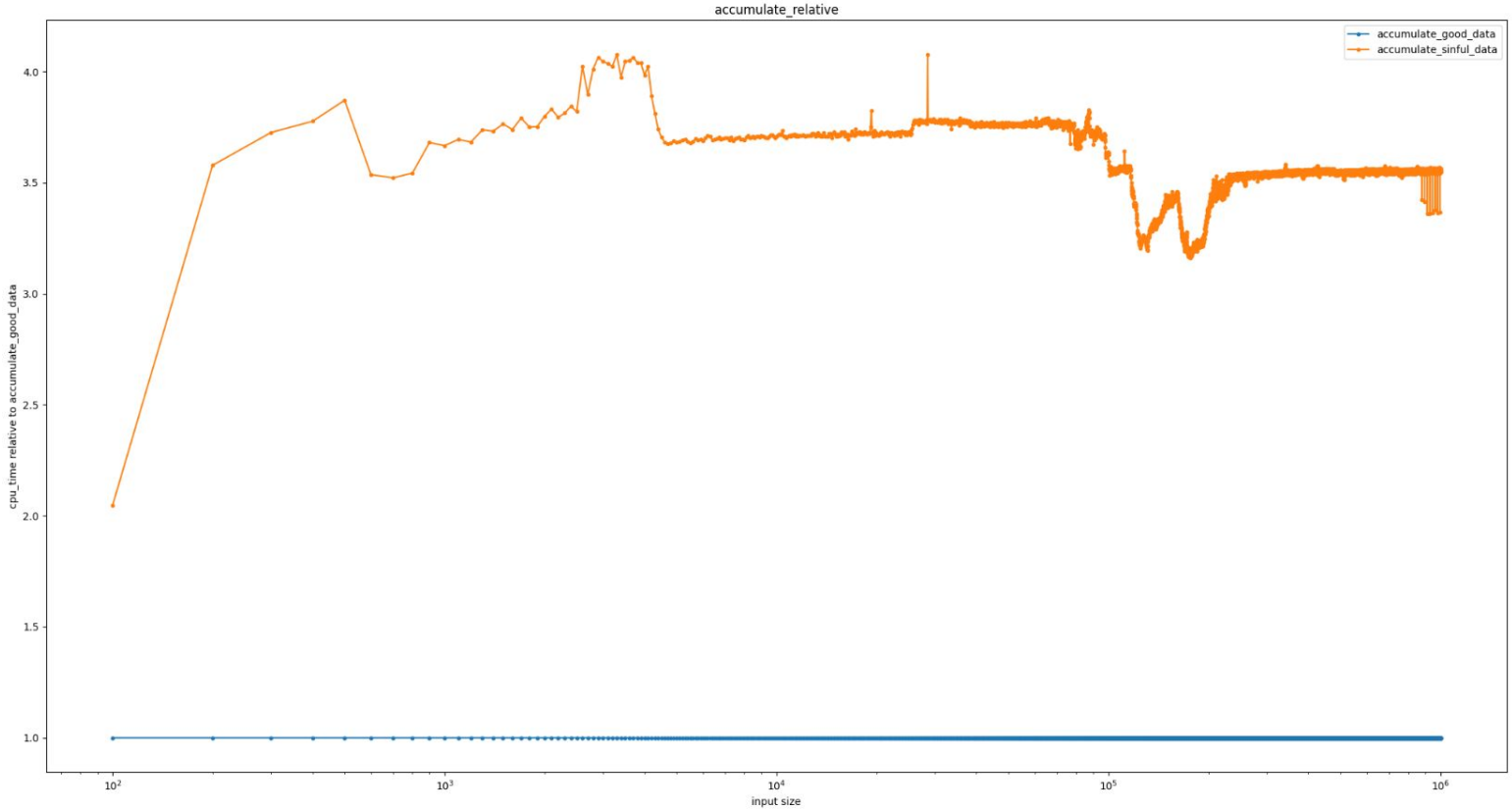

for_each
for_each_good_data
for_each_sinful_data
cpu_time
input size
900
800
700
600
500
400
300
200
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$


for_each_relative
for_each_good_data
for_each_sinful_data
cpu_time relative to for_each_good_data
input size
3.0
2.5
2.0
1.5
1.0
0.5
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$


sort
1e7
sort_good_data
sort_sinful_data
cpu_time
input size
0
1
2
3
4
5
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$


sort_relative
sort_good_data
sort_sinful_data
cpu_time relative to sort_good_data
1.1
1.0
0.9
0.8
0.7
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$
input size


accumulate
accumulate_good_data
accumulate_sinful_data
200000
150000
100000
50000
0
cpu_time
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$
input size


accumulate_relative
accumulate_good_data
accumulate_sinful_data
cpu_time relative to accumulate_good_data
input size
1.0
1.5
2.0
2.5
3.0
3.5
4.0
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$

# References


- Back to Basics: Class Layout - Stephen Dewhurst - CppCon 2020
- Object - cppreference.com
- Classes - cppreference.com
- Non-static data members - cppreference.com
- Software Performance and Class Layout
- Dumping a C++ object's memory layout with Clang - Eli Bendersky's website
- Diagnosing Hidden ODR Violations in Visual C++ (and fixing LNK2022) - C++ Team Blog
- /d1reportAllClassLayout – Dumping Object Memory Layout | Ofek's Visual C++ stuff

# Questions?

misasedam