ALGORITHM INTUITION Reloaded

Conor Hoekstra

code_report

codereport

“I’m not an expert,
I’m just a dude.”

- Scott Schurr, CppCon 2015

“The more I learn, the less I know.”

- Albert Einstein

“The larger my island of knowledge, the longer my shore of ignorance.”

- Ben Deane, ADSP: The Podcast, Episode 24

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

ode_report

YouTube

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

ode_report

YouTube

C++

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

Code_report

YouTube

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

ode_report

YouTube

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

ode_report

YouTube

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

code_report

YouTube

Algorithms +
Data
Structures =
Programs

NVIDIA

RAPIDS

ode_report

YouTube

http://rapids.ai

https://www.youtube.com/codereport

https://www.adspthepodcast.com

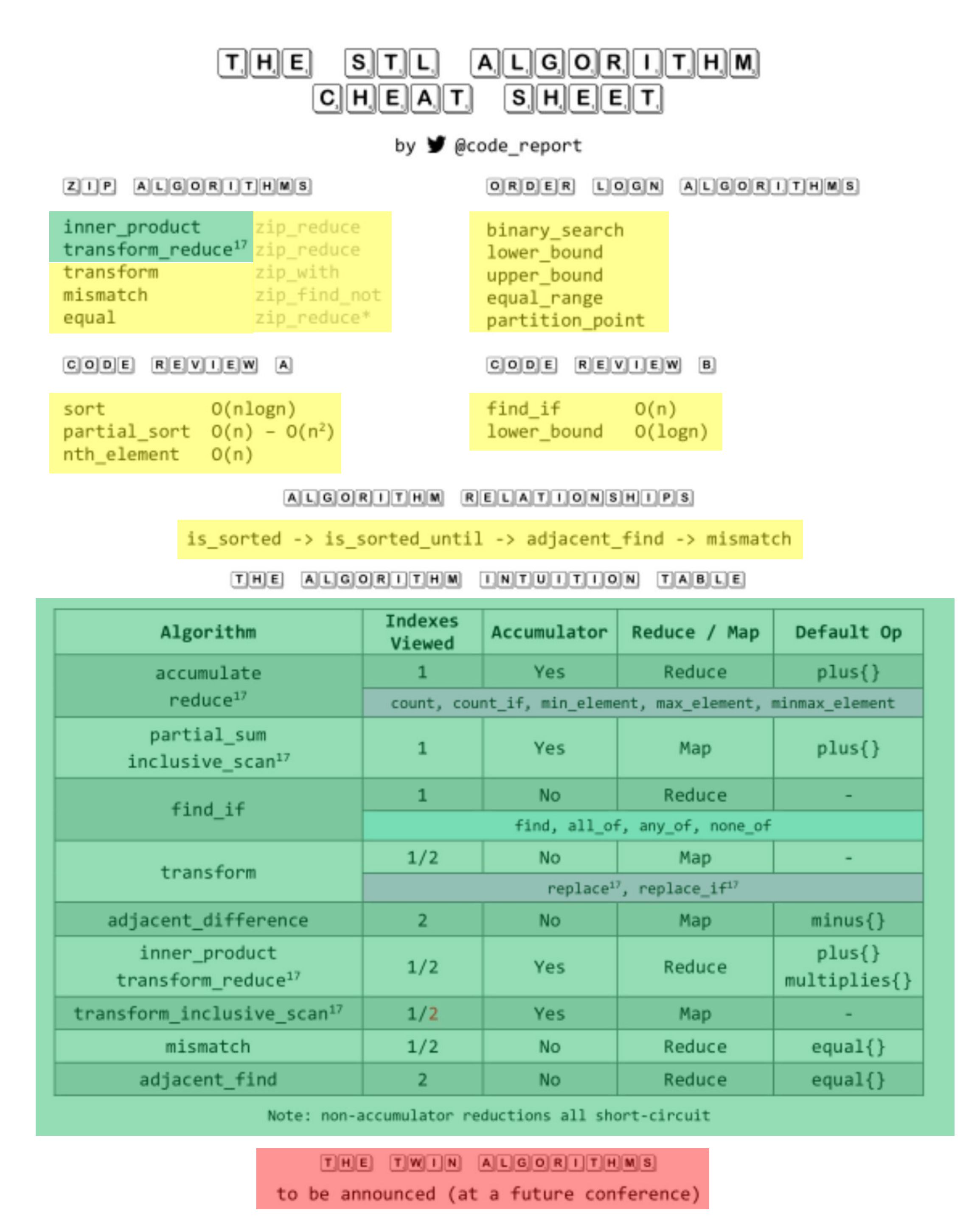
https://www.meetup.com/Programming-Languages-Toronto-Meetup/

ALGORITHM INTUITION Reloaded

Conor Hoekstra

code_report

codereport



THE STL ALGORITHM CHEAT SHEET

by @code_report

ZIP ALGORITHMS

inner_product	zip_reduce
transform_reduce^{17}	zip_reduce
transform	zip_with
mismatch	zip_find_not
equal	zip_reduce*

ORDER LOGN ALGORITHMS

binary_search
lower_bound
upper_bound
equal_range
partition_point

CODE REVIEW A

sort	O(nlogn)
partial_sort	O(n) - $O(n^2)$
nth_element	O(n)

CODE REVIEW B

find_if	O(n)
lower_bound	O(logn)

ALGORITHM RELATIONSHIPS

is_sorted -> is_sorted_until -> adjacent_find -> mismatch

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce^{17}	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
partial_sum inclusive_scan^{17}	1	Yes	Map	plus{}
find_if	1	No	Reduce	-
	find, all_of, any_of, none_of			
transform	1/2	No	Map	-
	replace^{17}, replace_if^{17}			
adjacent_difference	2	No	Map	minus{}

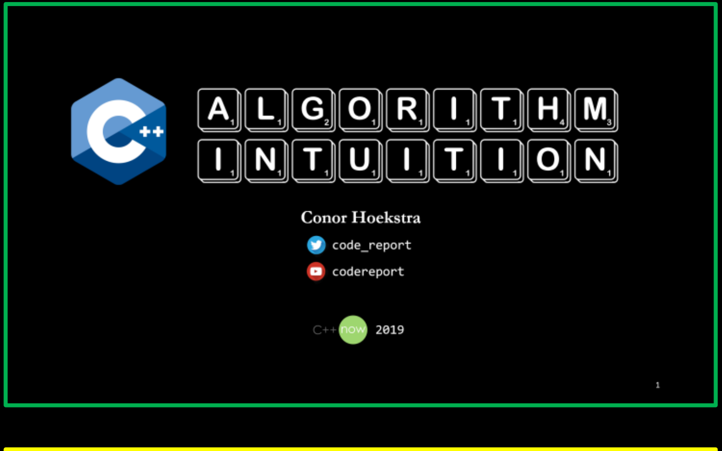
inner_product
transform_reduce^{17}
	1/2	Yes	Reduce	plus{}
				multiplies{}

transform_inclusive_scan^{17}	1/2	Yes	Map	-
mismatch	1/2	No	Reduce	equal{}
adjacent_find	2	No	Reduce	equal{}

Note: non-accumulator reductions all short-circuit

THE TWIN ALGORITHMS

to be announced (at a future conference)



ALGORITHM INTUITION

Conor Hoekstra

code_report

codereport

C++now 2019

1

Better Algorithm Intuition

Conor Hoekstra (he/him)

code_report

codereport

NVIDIA. RAPIDS

https://rapids.ai

the Twin Algorithms

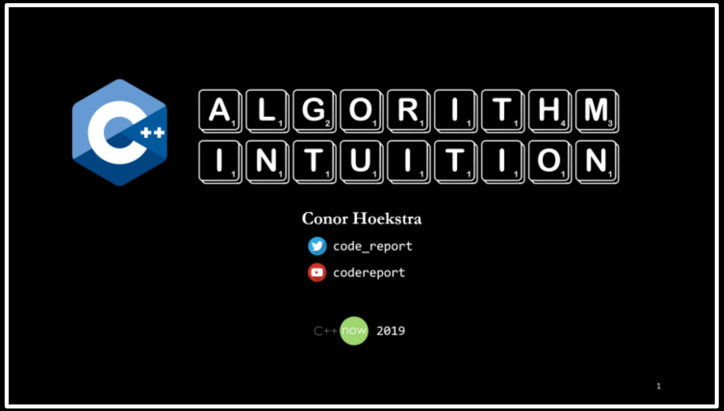
Conor Hoekstra

code_report

NVIDIA. RAPIDS

#include

The Algorithm Intuition Trilogy



ALGORITHM INTUITION

Conor Hoekstra

code_report

codereport

C++now 2019

1

Better

Algorithm Intuition

Conor Hoekstra (he/him)

code_report

codereport

NVIDIA RAPIDS

https://rapids.ai

the

Twin Algorithms

Conor Hoekstra

code_report

NVIDIA RAPIDS

#include

https://github.com/codereport/Algorithms
https://github.com/codereport/Talks

Let’s go back in time…

Native2013

"… and just as you can say, that would be a good use of a linked list, we don't have that **intuition** about **algorithms** yet, and we need to."

"… and just as you can say, that would be a good use of a linked list, we don't have that **intuition** about **algorithms** yet, and we need to."

- Kate Gregory

CppCast

auto CppCast = pod_cast<C++>("http://cppcast.com");

Episode 30

Goal (for you)

- Get you excited about algorithms
- Learn a new algorithm
- Start to develop some **algorithm intuition**

Interview Warm-up Question

Given an array of integers, find the difference between the minimum and maximum?

Guaranteed to have non-empty list

```
// Solution 4c (C++20)

auto solve() -> int {
   vector v = { 2, 1, 3, 5, 4 };
   auto [a, b] = minmax_element(v);
   return *b - *a;
}
```

[[Slideware Disclaimer]]

- Should be using std:: namespace
- solve is a terrible function name
- a and b are terrible variable names
- Failing SOLID

```cpp
// Solution 4c (C++20)

auto solve() -> int {
   vector v = { 2, 1, 3, 5, 4 };
   auto [a, b] = minmax_element(v);
   return *b - *a;
}
```

```cpp
// Solution 4c (C++20)

auto solve() -> int {
   auto v = vector{ 2, 1, 3, 5, 4 };
   auto [a, b] = minmax_element(v);
   return *b - *a;
}
```

```
// Solution 4c (C++20)

auto solve() -> int {
   auto v      = vector{ 2, 1, 3, 5, 4 };
   auto [a, b] = minmax_element(v);
   return *b - *a;
}
```

```cpp
// Solution 4c (C++20)

auto solve() -> int {
   auto const v      = vector{ 2, 1, 3, 5, 4 };
   auto const [a, b] = minmax_element(v);
   return *b - *a;
}
```

```cpp
// Solution 4c (C++20)

auto solve() -> int {
   auto const v      = std::vector{ 2, 1, 3, 5, 4 };
   auto const [a, b] = std::minmax_element(v);
   return *b - *a;
}
```

```cpp
// Solution 4c (C++20)

auto solve() -> int {
    auto const v      = std::vector{ 2, 1, 3, 5, 4 };
    auto const [a, b] = std::ranges::minmax_element(v);
    return *b - *a;
}
```

```
// Solution 5

auto solve() -> int {
    auto const v = std::vector{ 2, 1, 3, 5, 4 };
    auto const r = std::ranges::minmax_element(v);
    return *r.max - *r.min;
}
```

Library	Pre-C++11	C++11	C++17	Grand Total
`<algorithm>`	66	19	3, -1	**87***
`<numeric>`	4	1	6	**11**
`<memory>`	3	1	9	**13**
Grand Total	**73**	**21**	**17***	**111**

Can anyone name one of the **four original numeric** algorithms?
Can anyone name the **one C++11 numeric** algorithms?

Library	Pre-C++11	C++11	C++17	Grand Total
`<algorithm>`	66	19	3, -1	**87***
`<numeric>`	4	1	6	**11**
`<memory>`	3	1	9	**13**
Grand Total	**73**	**21**	**17***	**111**

🟦 accumulate	🟦 partial_sum
🟦 adjacent_difference	🟩 reduce
🟩 exclusive_scan	🟩 transform_exclusive_scan
🟩 inclusive_scan	🟩 transform_inclusive_scan
🟦 inner_product	🟩 transform_reduce
🟨 iota	

🟦 Pre-C++11	🟨 C++11	🟩 C++17

```
vector<int> v(10);
iota(begin(v), end(v), 1);

// 1 2 3 4 5 6 7 8 9 10
```

```cpp
vector<int> v(10);
iota(rbegin(v), rend(v), 1);

// 10 9 8 7 6 5 4 3 2 1
```

accumulate	partial_sum
adjacent_difference	reduce
exclusive_scan	transform_exclusive_scan
inclusive_scan	transform_inclusive_scan
inner_product	transform_reduce
iota ☑	

Pre-C++11	C++11	C++17

```
vector v = { 1, 2, 3 };

auto x = accumulate(cbegin(v), cend(v), 0);
```

```
vector v = { 1, 2, 3 };

auto x = accumulate(cbegin(v), cend(v), 0);
auto y = accumulate(cbegin(v), cend(v), 0, plus{});
```

Not the best name …

```
vector v = { 1, 2, 3 };

auto x = accumulate(cbegin(v), cend(v), 0);
auto y = accumulate(cbegin(v), cend(v), 1, multiplies{});
```

```
vector v = { 1, 2, 3 };

auto x = reduce(cbegin(v), cend(v));
auto y = reduce(cbegin(v), cend(v), 0, plus{});
auto z = reduce(cbegin(v), cend(v), 1, multiplies{});
```

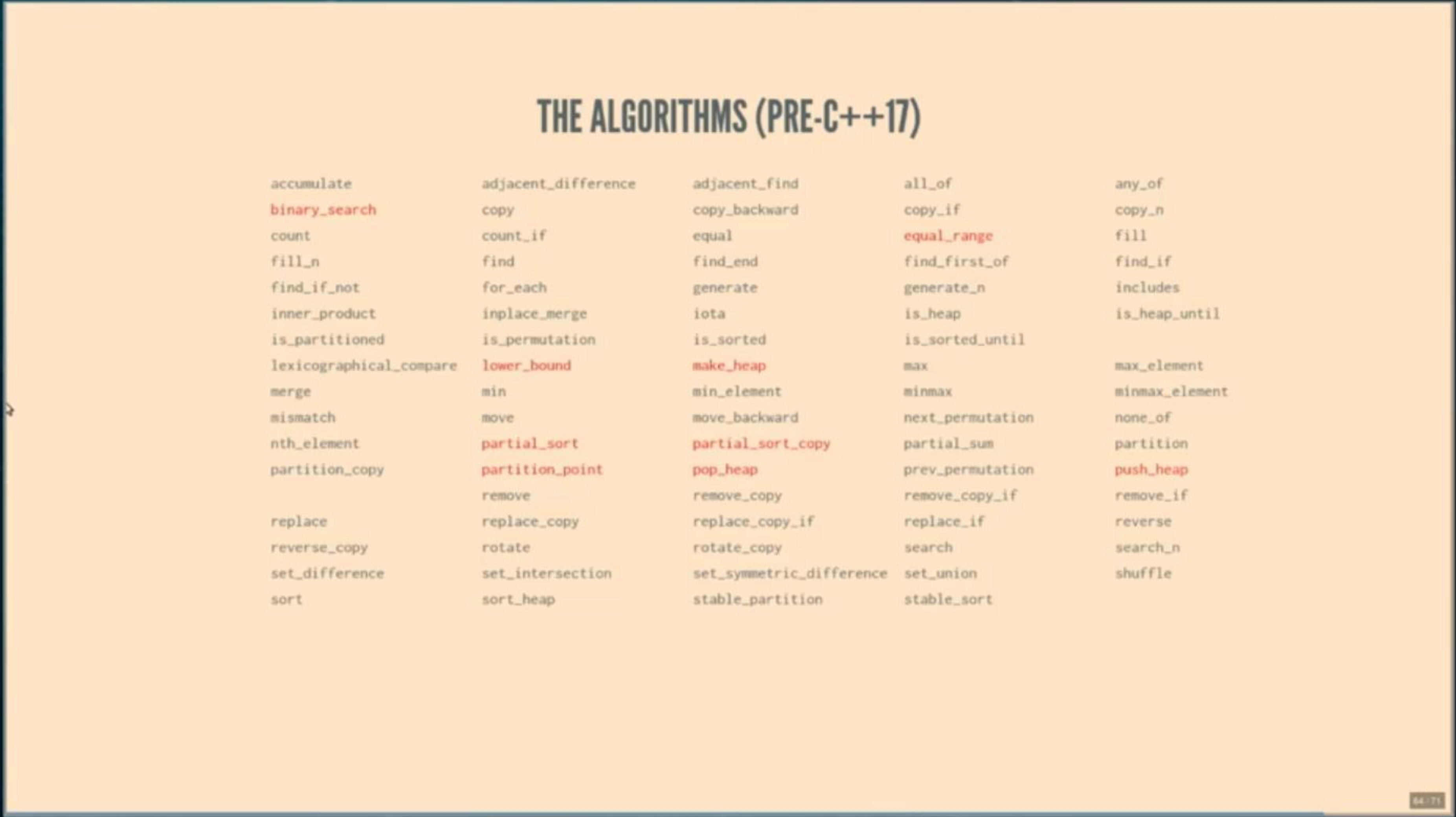
cppcon | 2016

THE C++ CONFERENCE • BELLEVUE, WASHINGTON

BEN DEANE

std::accumulate: Exploring an Algorithmic Empire

CppCon.org



THE ALGORITHMS (PRE-C++17)

accumulate
adjacent_difference
adjacent_find
all_of
any_of
binary_search
copy
copy_backward
copy_if
copy_n
count
count_if
equal
equal_range
fill
fill_n
find
find_end
find_first_of
find_if
find_if_not
for_each
generate
generate_n
includes
inner_product
inplace_merge
iota
is_heap
is_heap_until
is_partitioned
is_permutation
is_sorted
is_sorted_until
lexicographical_compare
lower_bound
make_heap
max
max_element
merge
min
min_element
minmax
minmax_element
mismatch
move
move_backward
next_permutation
none_of
nth_element
partial_sort
partial_sort_copy
partial_sum
partition
partition_copy
partition_point
pop_heap
prev_permutation
push_heap
remove
remove_copy
remove_copy_if
remove_if
replace
replace_copy
replace_copy_if
replace_if
reverse
reverse_copy
rotate
rotate_copy
search
search_n
set_difference
set_intersection
set_symmetric_difference
set_union
shuffle
sort
sort_heap
stable_partition
stable_sort

accumulate ☑	partial_sum
adjacent_difference	reduce ☑
exclusive_scan	transform_exclusive_scan
inclusive_scan	transform_inclusive_scan
inner_product	transform_reduce
iota ☑	

Pre-C++11	C++11	C++17

David is going to give Vittorio and Jon each one coin. David has N coins with different values. David wants the absolute difference between the value of the coins to be minimized so Vittorio and Jon don't fight with each other. Given an array of coin values, help David find this minimum.

https://www.codechef.com/problems/EID

Find the minimum difference between two values in a list.

```
1 4 2   ->    1

1 3 3   ->    0
```

https://www.codechef.com/problems/EID

```cpp
auto min_value(vector<int>& coins) -> int {
   sort(begin(coins), end(coins));
   vector<int> diff(coins.size());
   adjacent_difference(cbegin(coins), cend(coins), begin(diff));
   return *min_element(cbegin(diff) + 1, cend(diff));
}
```

https://www.codechef.com/problems/EID

What’s wrong with 2nd line?
Do we O(n) space?

```
auto min_value(vector<int>& c) -> int {
   sort(begin(c), end(c));
   vector<int> d(c.size());
   adjacent_difference(cbegin(c), cend(c), begin(d));
   return *min_element(cbegin(d) + 1, cend(d));
}
```

https://www.codechef.com/problems/EID

```
auto min_value(vector<int>& c) -> int {
   sort(begin(c), end(c));
   return reduce(cbegin(c) + 1, cend(c), numeric_limits<int>::max(),
      [prev = c.front()](auto a, auto b) mutable {
         auto d = b - prev;
         prev = b;
         return min(a, d);
      });
}
```

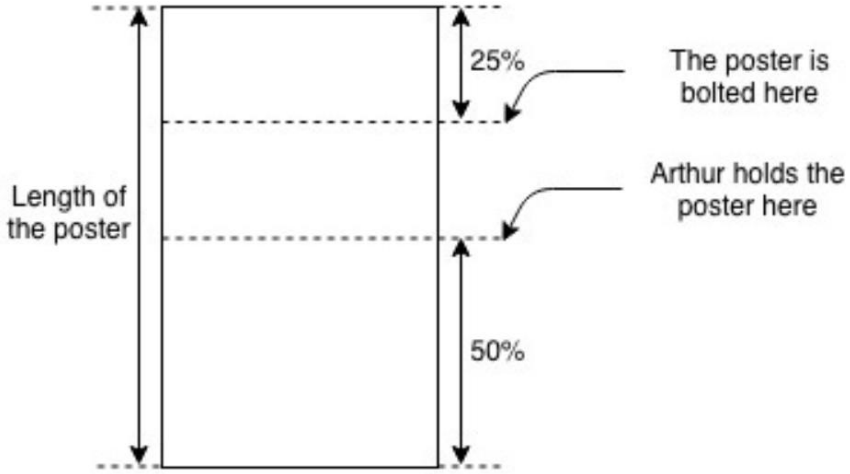
https://www.codechef.com/problems/EID

🟦 accumulate ☑	🟦 partial_sum
🟦 adjacent_difference ☑	🟩 reduce ☑
🟩 exclusive_scan	🟩 transform_exclusive_scan
🟩 inclusive_scan	🟩 transform_inclusive_scan
🟦 inner_product	🟩 transform_reduce
🟨 iota ☑	

🟦 Pre-C++11	🟨 C++11	🟩 C++17

HackerRank

You are given the length (l) of N posters, and the wall heights (w) at which they will be hung. They are hung at they 75% mark of the poster. Given David has height h, how tall a ladder does he need?

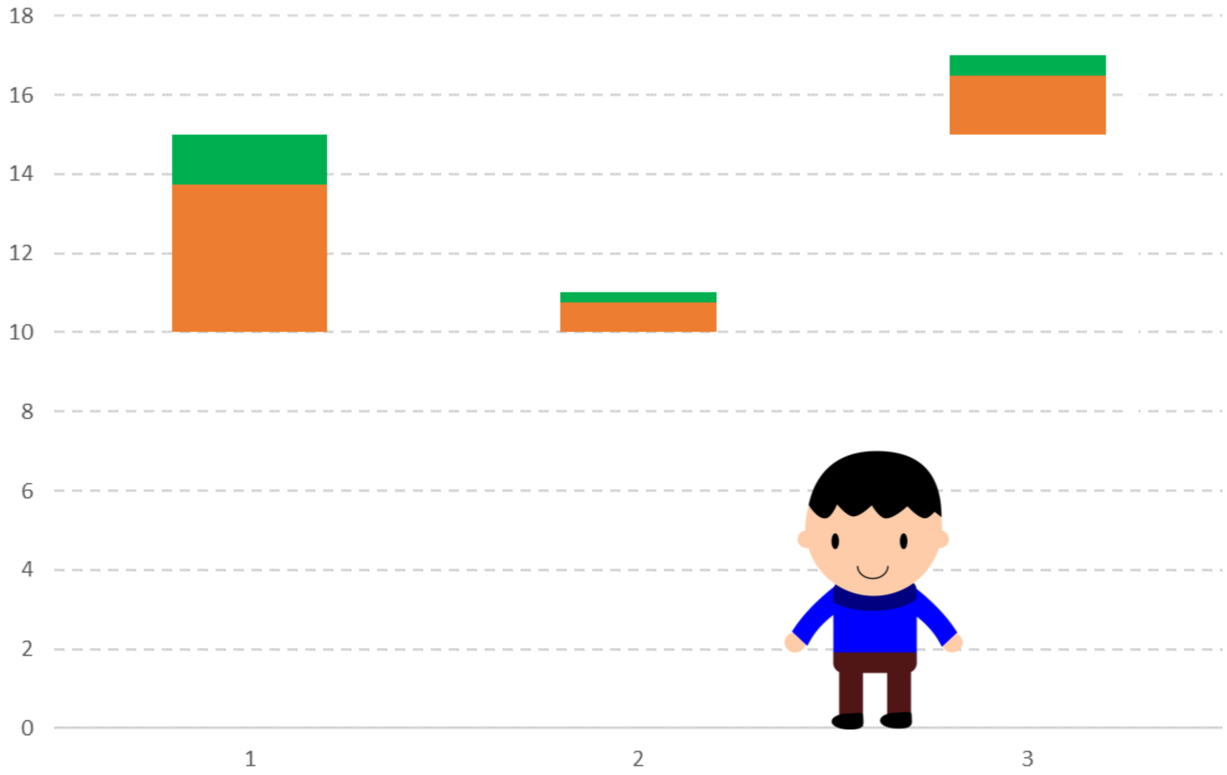


https://www.hackerrank.com/contests/hourrank-31/challenges/hanging-posters/problem

HourRank 31: Problem 1 – Hanging Posters

3 5
15 11 17
5 1 2

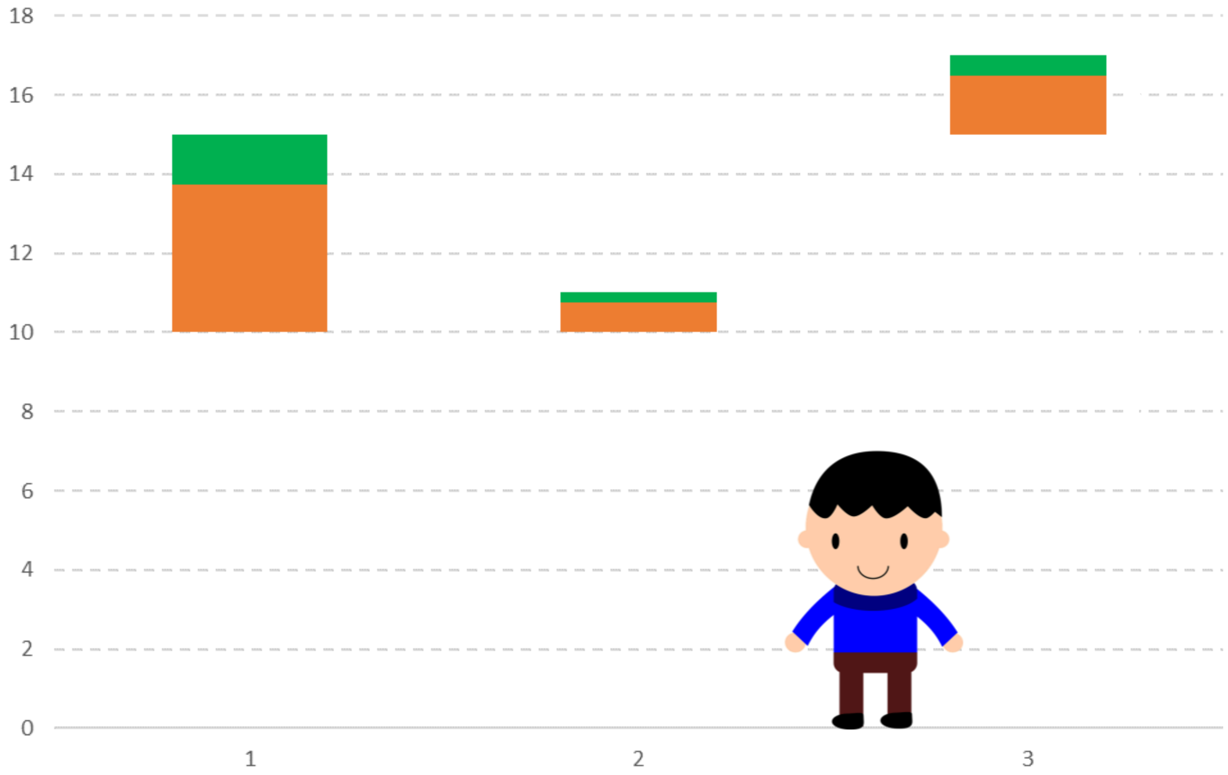
$$ceil\left(w - \frac{l}{4}\right) - h = 12$$

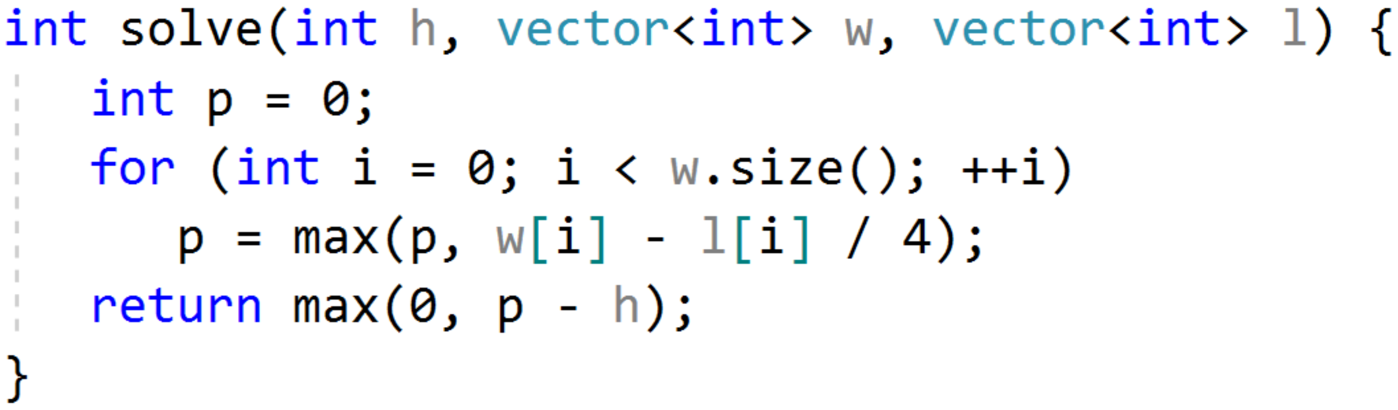


HourRank 31: Problem 1 – Hanging Posters

3 5
15 11 17
5 1 2

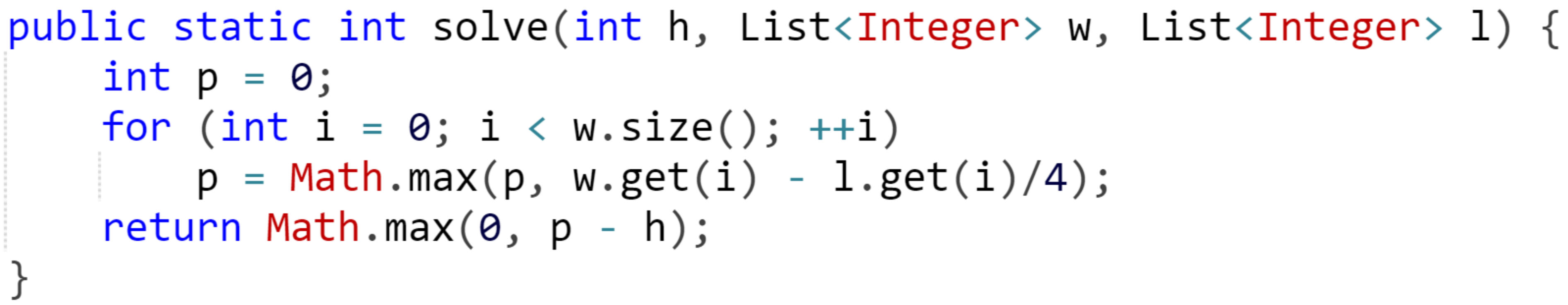
$$ceil\left(w-\frac{l}{4}\right)-h = 12$$





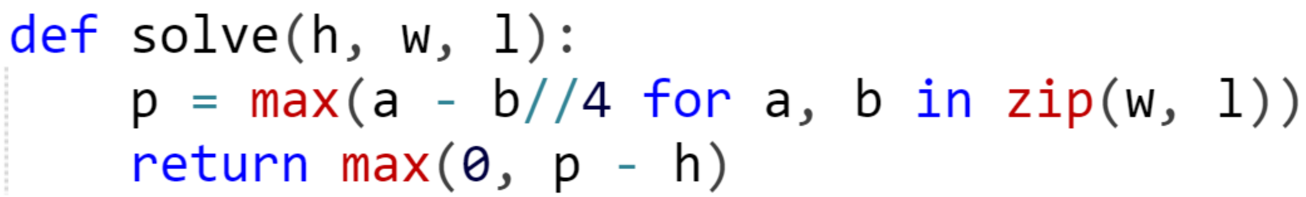
```
int solve(int h, vector<int> w, vector<int> l) {
    int p = 0;
    for (int i = 0; i < w.size(); ++i)
        p = max(p, w[i] - l[i] / 4);
    return max(0, p - h);
}
```

HourRank 31: Problem 1 – Hanging Posters

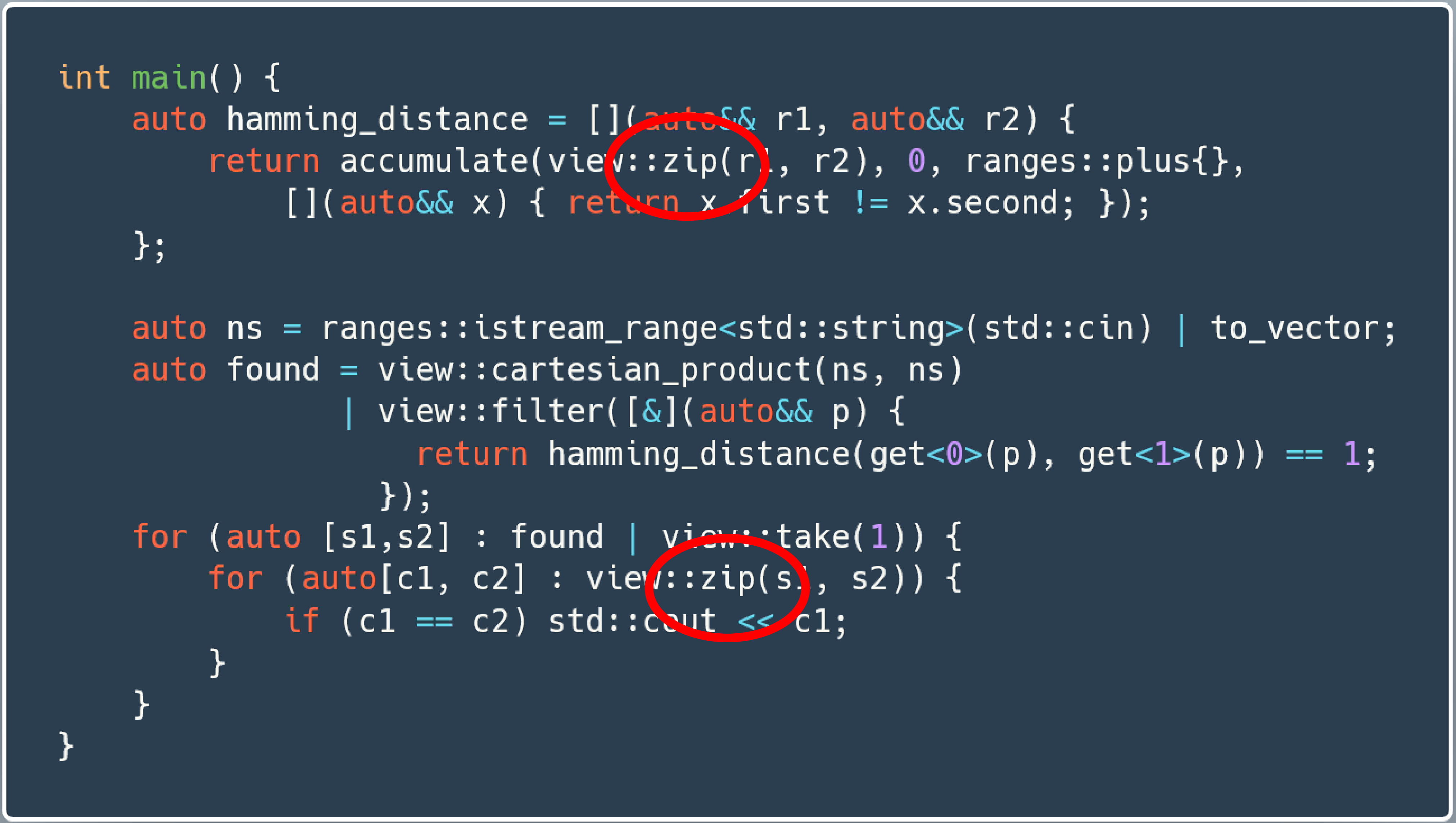


```java
public static int solve(int h, List<Integer> w, List<Integer> l) {
    int p = 0;
    for (int i = 0; i < w.size(); ++i)
        p = Math.max(p, w.get(i) - l.get(i)/4);
    return Math.max(0, p - h);
}
```

Then Sy Brand tweeted…

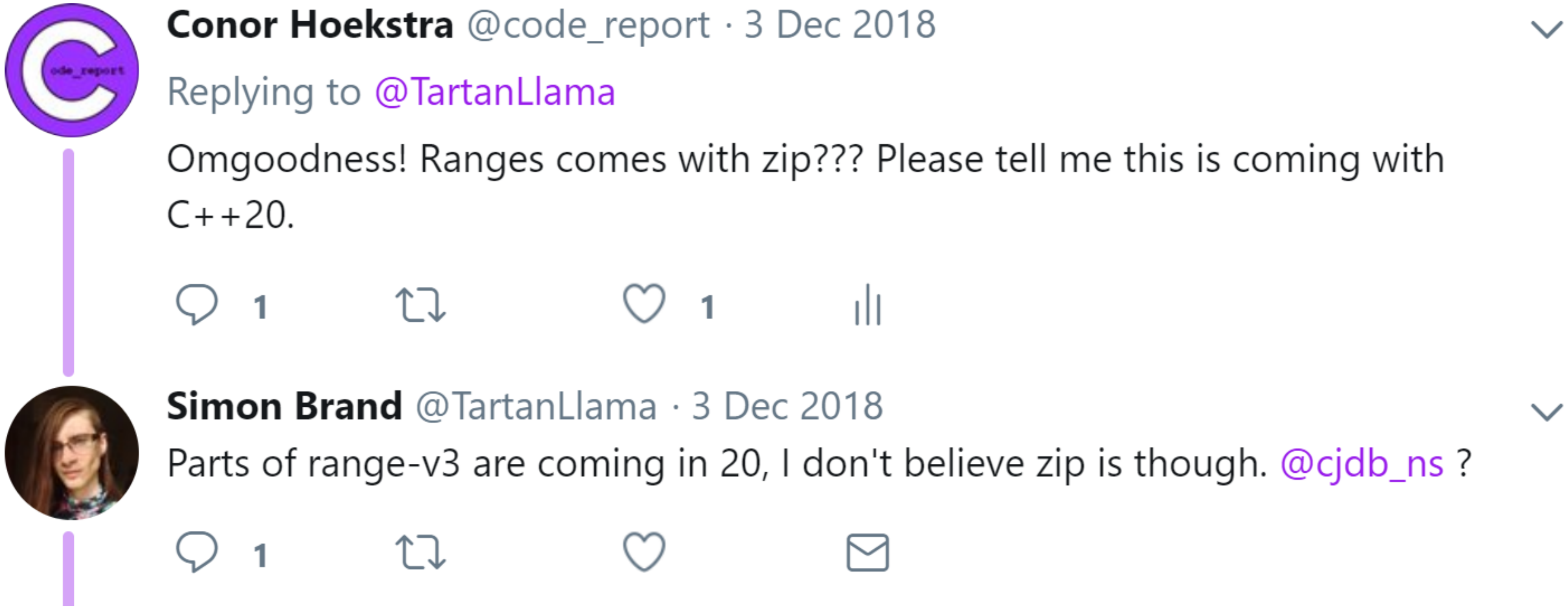


```python
def solve(h, w, l):
    p = max(a - b//4 for a, b in zip(w, l))
    return max(0, p - h)
```



```
int main() {
    auto hamming_distance = [](auto&& r1, auto&& r2) {
        return accumulate(view::zip(r1, r2), 0, ranges::plus{},
            [](auto&& x) { return x.first != x.second; });
    };

    auto ns = ranges::istream_range<std::string>(std::cin) | to_vector;
    auto found = view::cartesian_product(ns, ns)
               | view::filter([&](auto&& p) {
                   return hamming_distance(get<0>(p), get<1>(p)) == 1;
                 });
    for (auto [s1,s2] : found | view::take(1)) {
        for (auto[c1, c2] : view::zip(s1, s2)) {
            if (c1 == c2) std::cout << c1;
        }
    }
}
```



Conor Hoekstra @code_report · 3 Dec 2018

Replying to @TartanLlama

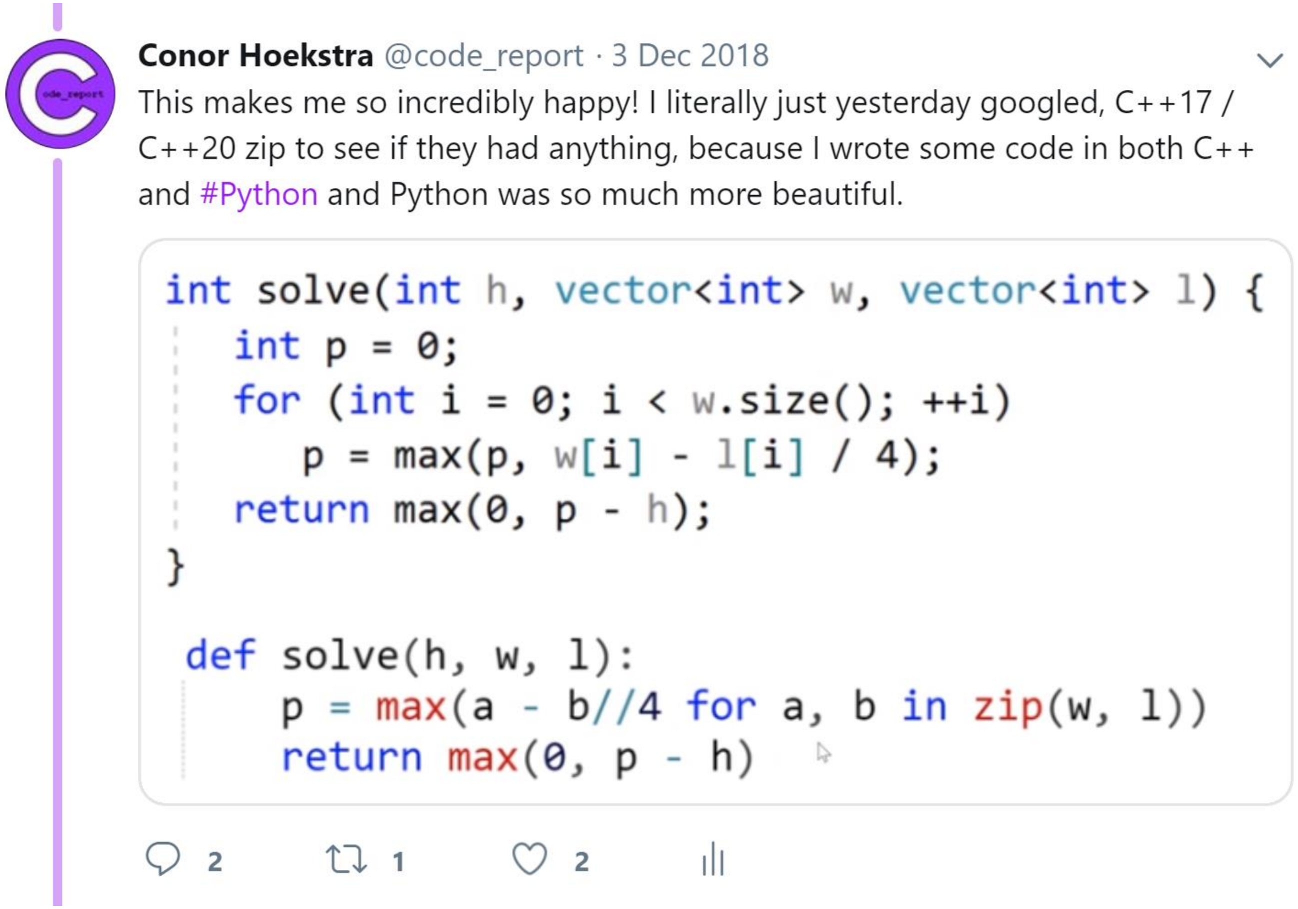
Omgoodness! Ranges comes with zip??? Please tell me this is coming with C++20.

1 1

Simon Brand @TartanLlama · 3 Dec 2018

Parts of range-v3 are coming in 20, I don't believe zip is though. @cjdb_ns ?

1

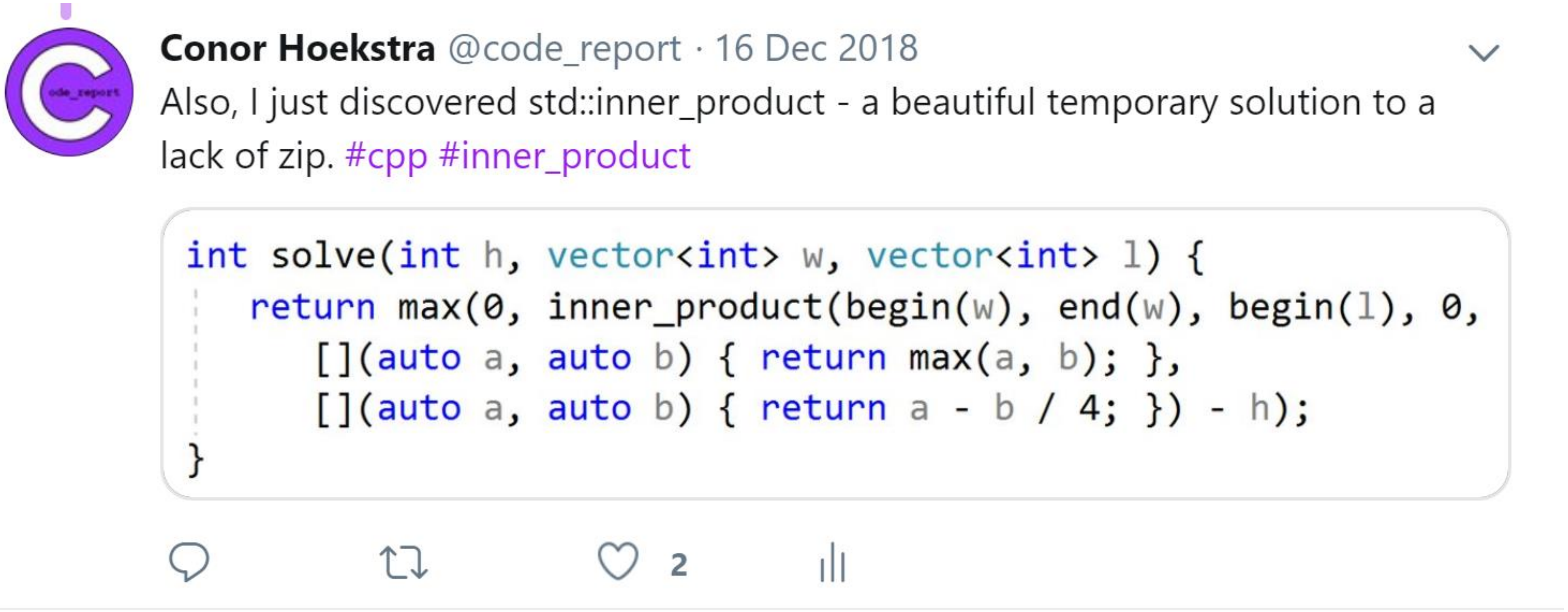


Conor Hoekstra @code_report · 3 Dec 2018

This makes me so incredibly happy! I literally just yesterday googled, C++17 / C++20 zip to see if they had anything, because I wrote some code in both C++ and #Python and Python was so much more beautiful.

```
int solve(int h, vector<int> w, vector<int> l) {
   int p = 0;
   for (int i = 0; i < w.size(); ++i)
      p = max(p, w[i] - l[i] / 4);
   return max(0, p - h);
}
```

```
def solve(h, w, l):
    p = max(a - b//4 for a, b in zip(w, l))
    return max(0, p - h)
```

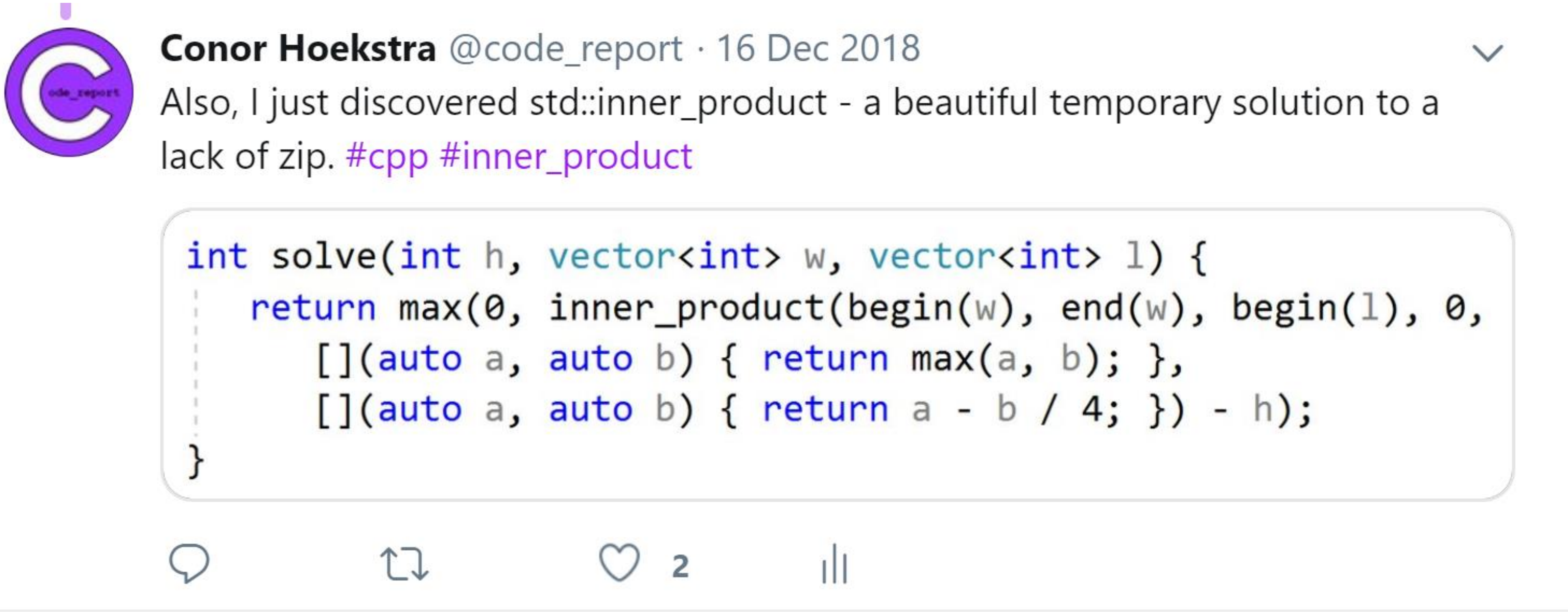


Conor Hoekstra @code_report · 16 Dec 2018

Also, I just discovered std::inner_product - a beautiful temporary solution to a lack of zip. #cpp #inner_product

```
int solve(int h, vector<int> w, vector<int> l) {
    return max(0, inner_product(begin(w), end(w), begin(l), 0,
        [](auto a, auto b) { return max(a, b); },
        [](auto a, auto b) { return a - b / 4; }) - h);
}
```

Not the best name …



Conor Hoekstra @code_report · 16 Dec 2018

Also, I just discovered std::inner_product - a beautiful temporary solution to a lack of zip. #cpp #inner_product

```
int solve(int h, vector<int> w, vector<int> l) {
    return max(0, inner_product(begin(w), end(w), begin(l), 0,
        [](auto a, auto b) { return max(a, b); },
        [](auto a, auto b) { return a - b / 4; }) - h);
}
```

And then …

transform_reduce

```
vector v = { 1, 2, 3 };
vector u = { 2, 3, 4 };

auto x = transform_reduce(cbegin(v), cend(v), cbegin(u), 0);
```

```
vector v = { 1, 2, 3 };
vector u = { 2, 3, 4 };

auto x = transform_reduce(cbegin(v), cend(v), cbegin(u), 0);
auto y = transform_reduce(cbegin(v), cend(v), cbegin(u), 0, plus{}, multiplies{});
```

```
vector v = { 1, 2, 3 };
vector u = { 2, 3, 4 };

auto x = transform_reduce(cbegin(v), cend(v), cbegin(u), 0);
auto y = transform_reduce(cbegin(v), cend(v), cbegin(u), 0, plus{}, multiplies{});
auto z = transform_reduce(cbegin(v), cend(v), cbegin(u), 0,
   plus{},
   multiplies{});
```

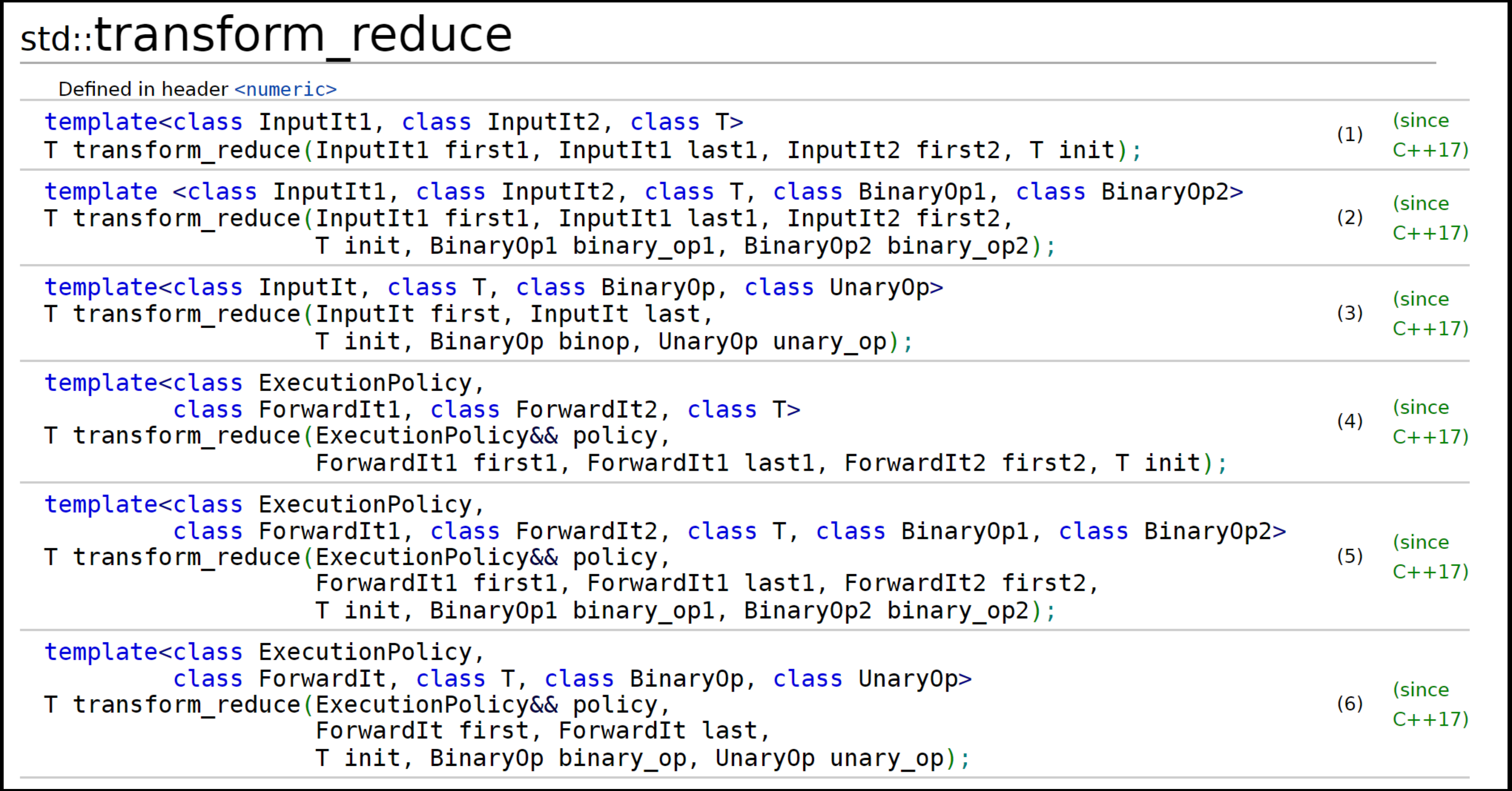
```
vector v = { 1, 2, 3 };
vector u = { 2, 3, 4 };

auto x = transform_reduce(cbegin(v), cend(v), cbegin(u), 0);
auto y = transform_reduce(cbegin(v), cend(v), cbegin(u), 0, plus{}, multiplies{});
auto z = transform_reduce(cbegin(v), cend(v), cbegin(u), 0,
   [](auto a, auto b) { return max(a, b); },
   multiplies{});
```

Why not call it zip_reduce?
Doesn’t need to zip!!

```cpp
vector v = { 1, 2, 3 };
vector u = { 2, 3, 4 };

auto x = transform_reduce(cbegin(v), cend(v), cbegin(u), 0);
auto y = transform_reduce(cbegin(v), cend(v), cbegin(u), 0, plus{}, multiplies{});
auto z = transform_reduce(cbegin(v), cend(v), cbegin(u), 0,
   [](auto a, auto b) { return max(a, b); },
   [](auto a, auto b) { return a + b * b; });
```



std::transform_reduce

Defined in header `<numeric>`

```
template<class InputIt1, class InputIt2, class T>
T transform_reduce(InputIt1 first1, InputIt1 last1, InputIt2 first2, T init);
```

(1) (since C++17)

```
template <class InputIt1, class InputIt2, class T, class BinaryOp1, class BinaryOp2>
T transform_reduce(InputIt1 first1, InputIt1 last1, InputIt2 first2,
                   T init, BinaryOp1 binary_op1, BinaryOp2 binary_op2);
```

(2) (since C++17)

```
template<class InputIt, class T, class BinaryOp, class UnaryOp>
T transform_reduce(InputIt first, InputIt last,
                   T init, BinaryOp binop, UnaryOp unary_op);
```

(3) (since C++17)

```
template<class ExecutionPolicy,
         class ForwardIt1, class ForwardIt2, class T>
T transform_reduce(ExecutionPolicy&& policy,
                   ForwardIt1 first1, ForwardIt1 last1, ForwardIt2 first2, T init);
```

(4) (since C++17)

```
template<class ExecutionPolicy,
         class ForwardIt1, class ForwardIt2, class T, class BinaryOp1, class BinaryOp2>
T transform_reduce(ExecutionPolicy&& policy,
                   ForwardIt1 first1, ForwardIt1 last1, ForwardIt2 first2,
                   T init, BinaryOp1 binary_op1, BinaryOp2 binary_op2);
```

(5) (since C++17)

```
template<class ExecutionPolicy,
         class ForwardIt, class T, class BinaryOp, class UnaryOp>
T transform_reduce(ExecutionPolicy&& policy,
                   ForwardIt first, ForwardIt last,
                   T init, BinaryOp binary_op, UnaryOp unary_op);
```

(6) (since C++17)

```
vector v = { 1, 2, 3 };
```

✗

```
auto x = reduce(cbegin(v), cend(v), 0,
    [](auto a, auto b) { return a + b * b; });
```

✓

```
auto y = transform_reduce(cbegin(v), cend(v), 0,
    std::plus{},
    [](auto e) { return e * e; });
```

Let’s revisit our adjacent_difference / reduce question

```
auto min_value(vector<int>& c) {
   sort(begin(c), end(c));
   return reduce(cbegin(c) + 1, cend(c), numeric_limits<int>::max(),
      [prev = c.front()](auto a, auto b) mutable {
      auto d = abs(b - prev);
      prev = b;
      return min(a, d);
   });
}
```

https://www.codechef.com/problems/EID

Can anyone see how to improve this?
Hint: make use of a STL function object

```
auto min_value(vector<int>& c) {
   sort(begin(c), end(c));
   return transform_reduce(cbegin(c), --cend(c), ++cbegin(c),
      numeric_limits<int>::max(),
      [](auto a, auto b) { return min(a, b); },
      [](auto a, auto b) { return abs(a - b); });
}
```

https://www.codechef.com/problems/EID

```cpp
auto min_value(vector<int>& c) {
   sort(begin(c), end(c));
   return transform_reduce(++cbegin(c), cend(c), cbegin(c),
      numeric_limits<int>::max(),
      [](auto a, auto b) { return min(a, b); },
      std::minus{});
}
```

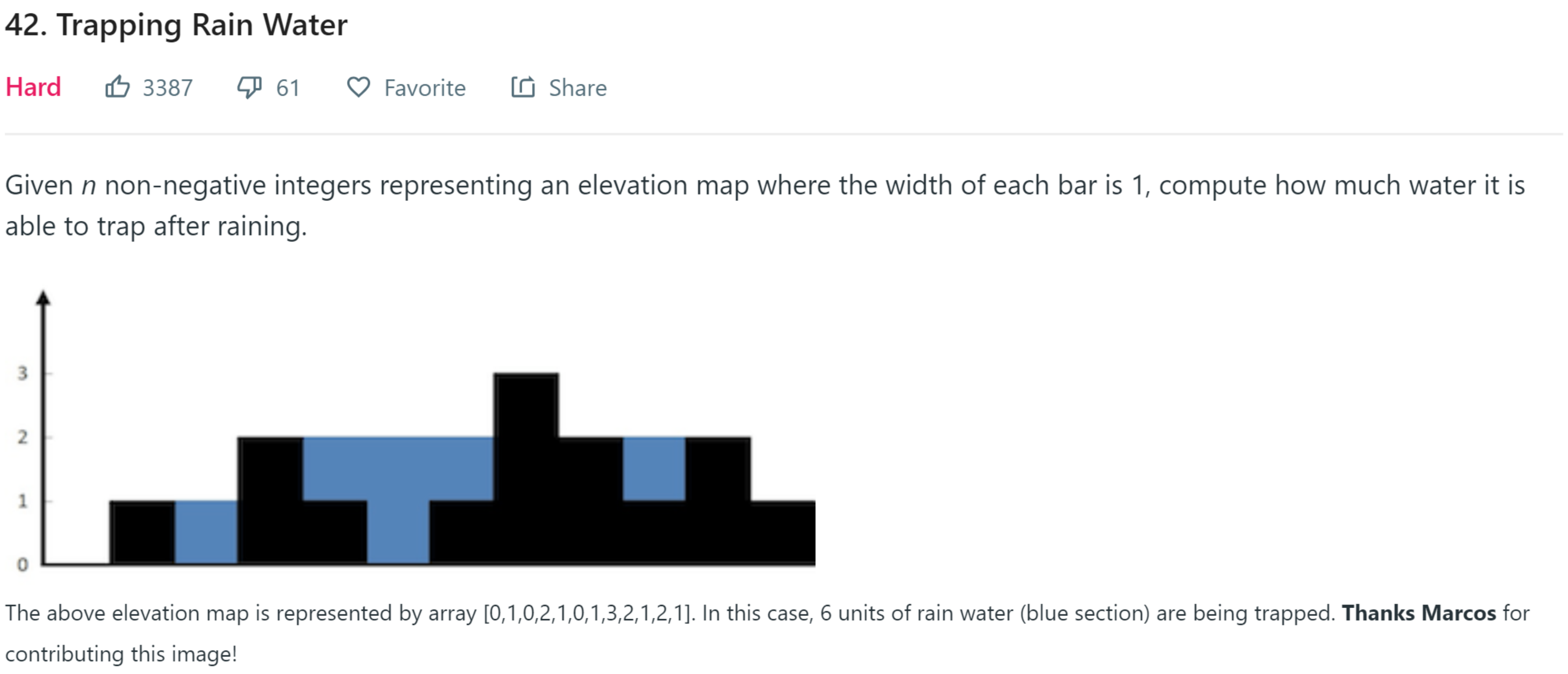
https://www.codechef.com/problems/EID

transform_reduce

🟦 accumulate ☑	🟦 partial_sum
🟦 adjacent_difference ☑	🟩 reduce ☑
🟩 exclusive_scan	🟩 transform_exclusive_scan
🟩 inclusive_scan	🟩 transform_inclusive_scan
🟦 inner_product ☑	🟩 transform_reduce ☑
🟨 iota ☑	

🟦 Pre-C++11	🟨 C++11	🟩 C++17

LeetCode



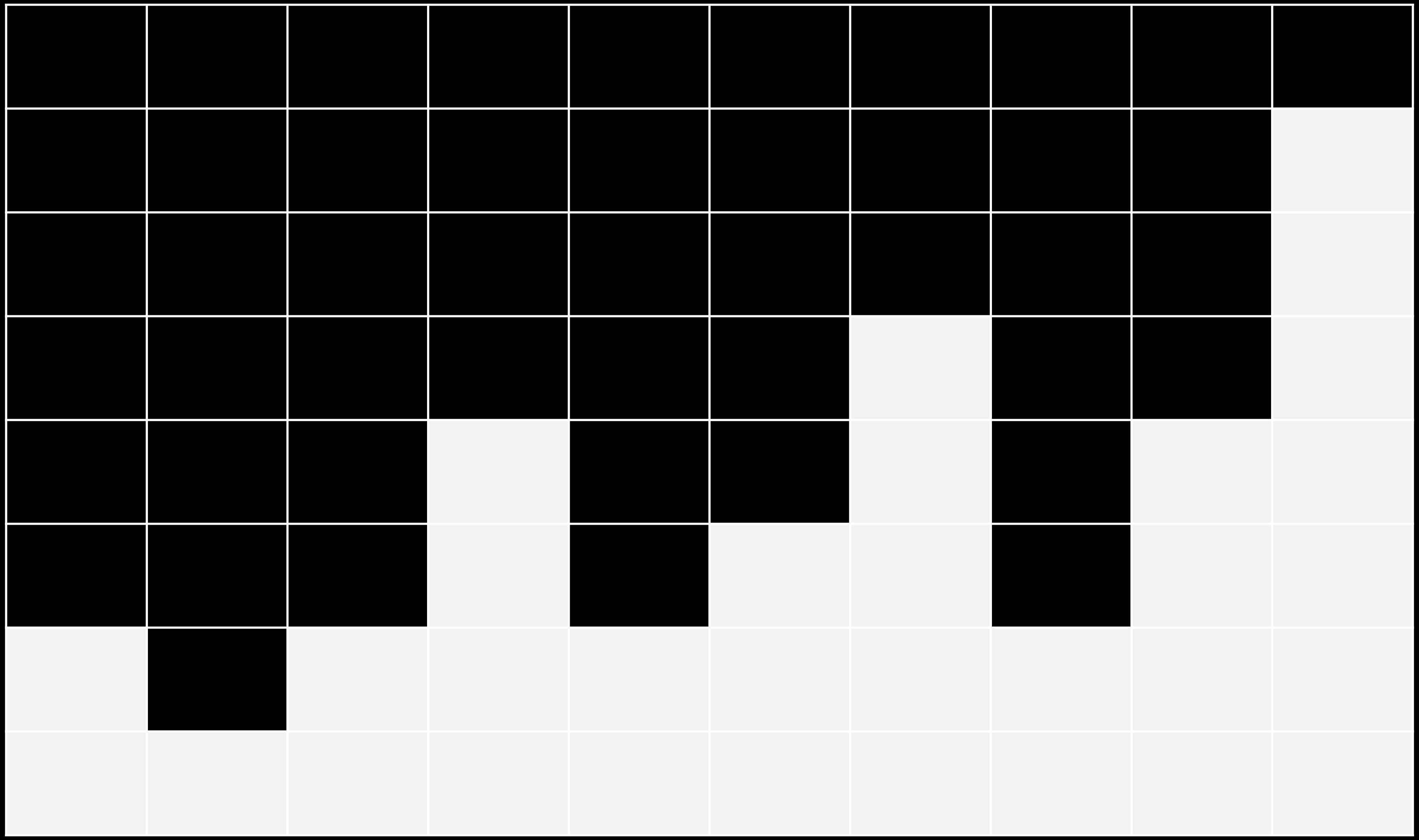
42. Trapping Rain Water

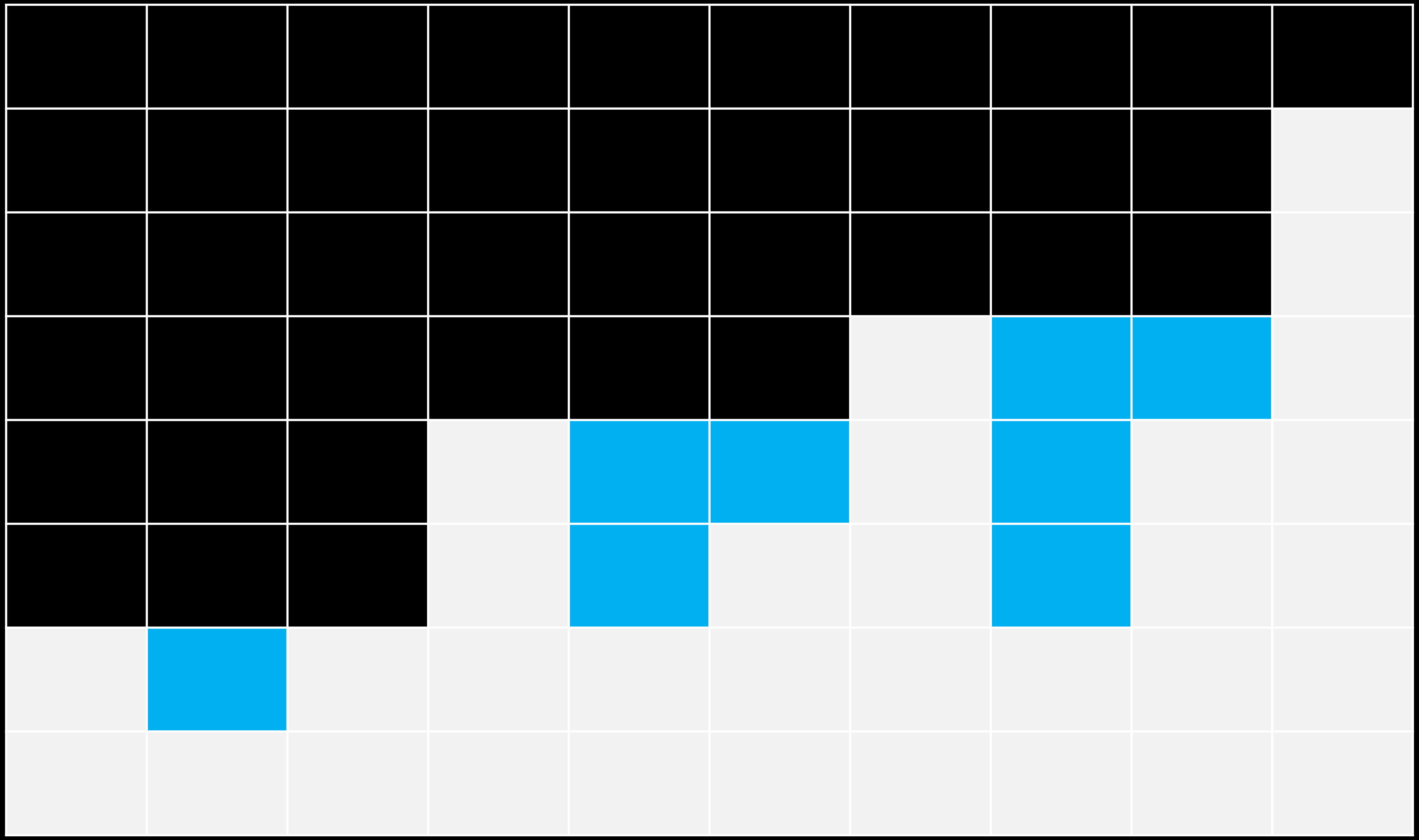
Hard 3387 61 Favorite Share

Given *n* non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. **Thanks Marcos** for contributing this image!

https://leetcode.com/problems/trapping-rain-water/





```cpp
auto solve() {
   vector v = { 2, 1, 2, 4, 2, 3, 5, 2, 4, 7 };
   auto m   = v.front(); // max so far
   auto ans = 0;
   for (auto e : v) {
      m    = max(m, e);
      ans += m - e;
   }
   return ans;
}
```

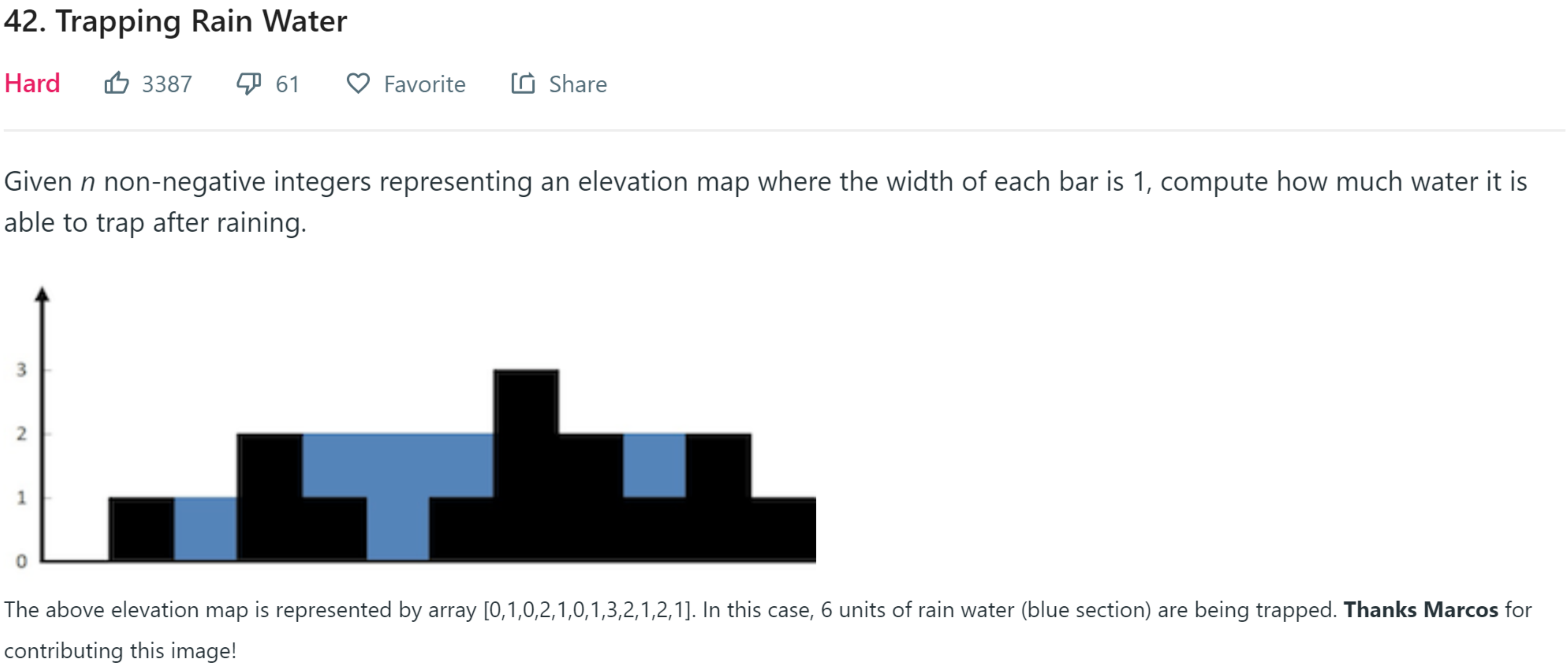
```
auto solve() {
   vector v = { 2, 1, 2, 4, 2, 3, 5, 2, 4, 7 };
   vector u(v.size(), 0);
   partial_sum(cbegin(v), cend(v), begin(u),
      [](auto a, auto b) { return max(a, b); });
   return transform_reduce(cbegin(v), cend(v), cbegin(u), 0,
      std::plus{},
      [](auto a, auto b) { return abs(a - b); });
}
```

Not the best name …

```
auto solve() {
   vector v = { 2, 1, 2, 4, 2, 3, 5, 2, 4, 7 };
   vector u(v.size(), 0);
   partial_sum(cbegin(v), cend(v), begin(u), ufo::max{});
   return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
      std::plus{},
      std::minus{});
}
```

```cpp
auto solve() {
   vector v = { 2, 1, 2, 4, 2, 3, 5, 2, 4, 7 };
   vector u(v.size(), 0);
   inclusive_scan(cbegin(v), cend(v), begin(u), ufo::max{});
   return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
      std::plus{},
      std::minus{});
}
```

LeetCode



42. Trapping Rain Water

Hard 👍 3387 👎 61 ♡ Favorite Share

Given *n* non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. **Thanks Marcos** for contributing this image!

https://leetcode.com/problems/trapping-rain-water/

```
int trap(vector<int>& v) {
    vector u(v.size(), 0);
    inclusive_scan(begin(v),  end(v),   begin(u),  ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

```cpp
int trap(vector<int>& v) {
    vector u(v.size(), 0);
    auto it = max_element(begin(v), end(v));
    inclusive_scan(begin(v),  end(v),   begin(u),  ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

```cpp
int trap(vector<int>& v) {
    vector u(v.size(), 0);
    auto it = max_element(begin(v), end(v));
    inclusive_scan(begin(v),  next(it), begin(u),  ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

```
int trap(vector<int>& v) {
    vector u(v.size(), 0);
    auto it = max_element(begin(v), end(v));
    inclusive_scan(begin(v),  next(it), begin(u),  ufo::max{});
    inclusive_scan(rbegin(v), rev(it),  rbegin(u), ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

```
template<class T>
using rev = reverse_iterator<T>;

int trap(vector<int>& v) {
    vector u(v.size(), 0);
    auto it = max_element(begin(v), end(v));
    inclusive_scan(begin(v),  next(it), begin(u),  ufo::max{});
    inclusive_scan(rbegin(v), rev(it),  rbegin(u), ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

🟦 accumulate	☑	🟦 partial_sum	☑
🟦 adjacent_difference	☑	🟩 reduce	☑
🟩 exclusive_scan	☑	🟩 transform_exclusive_scan	□
🟩 inclusive_scan	☑	🟩 transform_inclusive_scan	□
🟦 inner_product	☑	🟩 transform_reduce	☑
🟨 iota	☑		

🟦 Pre-C++11	🟨 C++11	🟩 C++17

The Algorithm Intuition Table

Algorithm	Indexes Viewed	Accumulator	Reduce / Transform
accumulate / reduce	1	Yes Init = **Specified**	Reduce
inner_product / transform_reduce	1*	Yes Init = **Specified**	Reduce
partial_sum / inclusive_scan	1	Yes Init = **First elem**	Transform
exclusive_scan	1	Yes Init = **Specified**	Transform
adjacent_difference	2	No	Transform
iota	N / A	N / A	Transform

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce^{17}	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
partial_sum inclusive_scan^{17}	1	Yes	Map	plus{}
find_if	1	No	Reduce	-
	find, all_of, any_of, none_of			
transform	1/2	No	Map	-
	replace^{17}, replace_if^{17}			
adjacent_difference	2	No	Map	minus{}

inner_product
transform_reduce^{17}
	1/2	Yes	Reduce	plus{}
				multiplies{}

transform_inclusive_scan^{17}	1/2	Yes	Map	-
mismatch	1/2	No	Reduce	equal{}
adjacent_find	2	No	Reduce	equal{}

Note: non-accumulator reductions all short-circuit

Conclusion

1. Algorithms are awesome! And fun!
2. Especially `transform_reduce`
    1. `minmax_element, min_element, max_element, sort, iota, count_if,  inner_product, adjacent_difference, partial_sum, accumulate, reduce, inclusive_scan, exclusive_scan`
3. Know the default operations that algorithms come with
4. Leverage algorithms with function objects / lambdas

QUESTIONS?

Conor Hoekstra

code_report

codereport