

Better Algorithm Intuition



Conor Hoekstra



code_report



codereport

Better Algorithm Intuition



Conor Patrick Hoekstra



code_report



codereport

It is awesome to be in Belfast!



**“I’m not an expert,
I’m just a dude.”**

- Scott Schurr, CppCon 2015

constexpr: Applications

By Scott Schurr for Ripple Labs at CppCon September 2015



SCOTT SCHURR

constexpr:
Applications



About Me

- I'm a Senior Library Software Engineer for  NVIDIA.
 - Working on the  RAPIDS AI team (<http://rapids.ai>)
- I am a programming language enthusiast
- I've been coding in C++ for 5+ years*
- I love **auto** (AAA)
- I prefer **east const** (1 const west)
- I love algorithms and beautiful code
- I have a  YouTube channel





Conor Hoekstra

Algorithm Intuition

Video Sponsorship
Provided By:

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

int trap(vector<int>& v) {
    vector<int> u(v.size);
    auto it = max_element(v.begin(), v.end());
    inclusive_scan(it, v.end(), u.begin(), std::plus<int>());
    inclusive_scan(v.begin(), it, u.begin() + 1, std::minus<int>());
}
```



```
auto dangerous_teams(std::string const& s) -> bool {
    return s.size() >= 7;
}
```



>

<https://brevzin.github.io/c%2B%2B/2019/08/22/ufcs-custom-extension/>

22

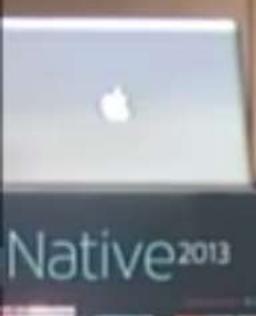
Conor Hoekstra

Algorithm Intuition
(part 2 of 2)

Video Sponsorship Provided By:







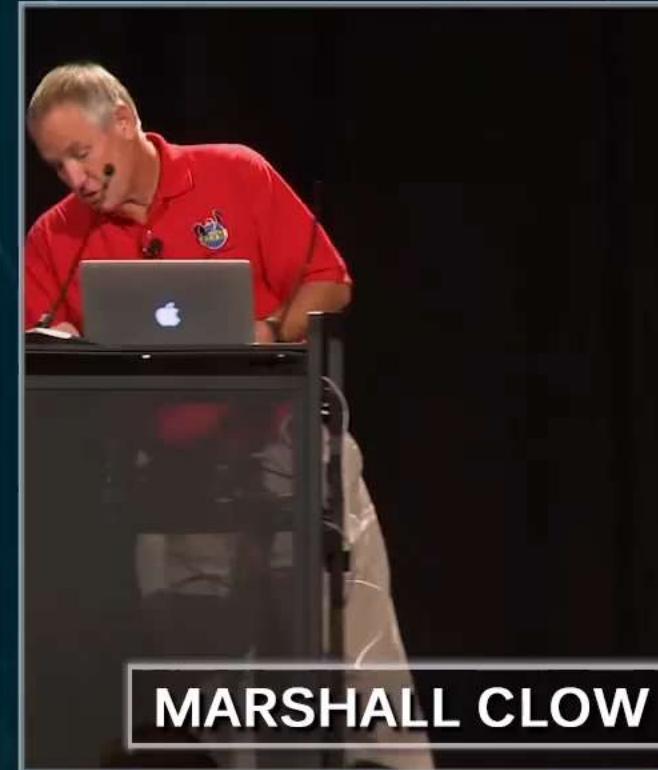
STL Algorithms - How to use them; how to write your own

Marshall Clow
Qualcomm, Inc.

CppCon 2016

mclow@qti.qualcomm.com

@mclow



MARSHALL CLOW

**STL Algorithms -
How you should use them;
how to write your own**

CppCon.org

“... 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



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

Episode 30



Goal

(once again)

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

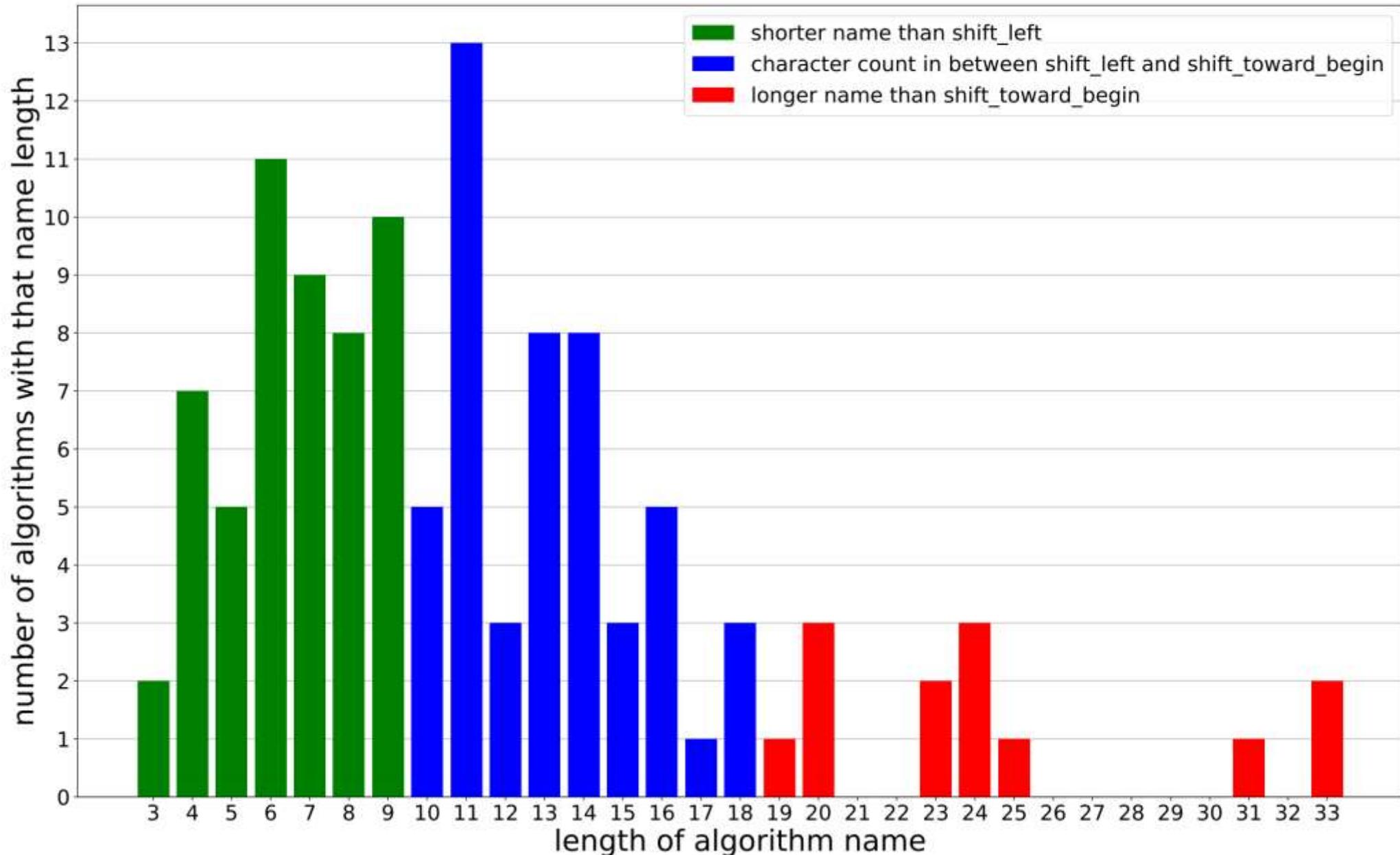
Prologue

<numeric> vs <algorithm>

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

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

Distribution of algorithm names character count



3	max	7	bsearch	11	equal_range	15	is_sorted_until
3	min	7	copy_if	11	find_if_not	15	partition_point
		7	destroy	11	lower_bound	15	replace_copy_if
		7	find_if	11	max_element		
		7	is_heap	11	min_element	16	next_permutation
		7	none_of	11	nth_element	16	prev_permutation
4	copy	7	replace	11	partial_sum	16	set_intersection
4	fill	7	reverse	11	remove_copy	16	stable_partition
4	find	7	shuffle	11	rotate_copy	16	transform_reduce
4	iota	8	count_if	11	stable_sort		
4	move	8	find_end	11	swap_ranges	17	partial_sort_copy
4	sort	8	for_each	11	unique_copy		
4	swap	8	generate	11	upper_bound	18	uninitialized_copy
5	clamp	8	includes	12	partial_sort	18	uninitialized_fill
5	count	8	mismatch	12	replace_copy	18	uninitialized_move
5	equal	8	pop_head	12	reverse_copy	19	adjacent_difference
5	merge	8	search_n	13	adjacent_find		
5	qsort	9	destroy_n	13	binary_search	20	uninitialized_copy_n
6	all_of	9	is_sorted	13	copy_backward	20	uninitialized_fill_n
6	any_of	9	iter_swap	13	find_first_of	20	uninitialized_move_n
6	copy_n	9	make_heap	13	inner_product	23	lexicographical_compare
6	fill_n	9	partition	13	inplace_merge	23	uninitialized_construct
6	minmax	9	push_heap	13	is_heap_until		
6	reduce	9	remove_if	13	move_backward	24	set_symmetric_difference
6	remove	9	set_union	14	exclusive_scan	24	transform_exclusive_scan
6	rotate	9	sort_heap	14	inclusive_scan	24	transform_inclusive_scan
6	sample	9	transform	14	is_partitioned	25	uninitialized_construct_n
6	search	10	accumulate	14	is_permutation		
6	unique	10	destroy_at	14	minmax_element	31	uninitialized_default_construct
		10	for_each_n	14	partition_copy		
		10	generate_n	14	remove_copy_if	33	lexicographical_compare_three_way
		10	replace_if	14	set_difference	33	uninitialized_default_construct_n

Chapter 1

LeetCode Level 1



344. Reverse String

Easy 908 581 Favorite Share

Write a function that reverses a string.



```
void reverseString(std::string& s) {  
    std::reverse(  
        std::begin(s),  
        std::end(s));  
}
```



[thrust::reverse](#)

344. Reverse String

Hot Newest to Oldest **Most Votes** Most Posts Recent Activity Oldest to Newest



Simple C++ solution cpp

xz2210 created at: April 21, 2016 9:32 PM | Last Reply: yellomellofello August 28, 2019 8:25 PM



One Line C++ Code (with Help of STL) cpp

krrk2102 created at: April 22, 2016 12:53 PM | Last Reply: nicho December 12, 2018 8:53 AM



Share my C++ solution,very easy to understand cpp easy-understand

vdvvdd created at: April 23, 2016 7:21 PM | Last Reply: lidz March 16, 2019 10:48 PM



Easy C++ 2 Line Solution [beats 92%, 100%] 2 lines c++ clean-code

Pooja0406 created at: September 4, 2019 12:44 AM | No replies yet.



[C++] One-Liner c++ cpp

KasraGhodsi created at: June 23, 2019 2:18 PM | No replies yet.



C++ solution without extra memory cpp

magiceye07 created at: May 25, 2019 2:12 PM | No replies yet.



Simple C++ code cpp solution solution-cpp

neelneelpurk created at: May 11, 2019 9:36 AM | No replies yet.

 ★ 567

Last Edit: October 2, 2018 10:22 AM

54

```
class Solution {  
public:  
    string reverseString(string s) {  
        int i = 0, j = s.size() - 1;  
        while(i < j){  
            swap(s[i++], s[j--]);  
        }  
  
        return s;  
    }  
};
```

★ 479 April 23, 2016 7:21 PM

5

```
class Solution {
public:
    string reverseString(string s) {
        int start = 0;
        int end = s.length() - 1;
        char ch = 0;

        for (; start < end; start++, end--)
        {
            ch = s[start];
            s[start] = s[end];
            s[end] = ch;
        }

        return s;
    };
}
```

< Back

Easy C++ 2 Line Solution [beats 92%, 100%]



★ 83

Last Edit: September 4, 2019 12:44 AM

0

Runtime: 44 ms, faster than 92.12% of C++ online submissions for Reverse String.



Memory Usage: 15.2 MB, less than 96.34% of C++ online submissions for Reverse String.

```
void reverseString(vector<char>& s) {  
  
    for(int start=0, end = s.size()-1; start < end; start++, end--)  
        swap(s[start], s[end]);  
}
```

< Back

[C++] One-Liner



★ 74

June 23, 2019 2:18 PM

0



< Back

One Line C++ Code (with Help of STL)



★ 8

April 22, 2016 12:53 PM

8





709. To Lower Case

Easy

341

1188

Favorite

Share

Implement function `ToLowerCase()` that has a string parameter `str`, and returns the same string in lowercase.



```
std::string toLowerCase(string s) {  
    std::transform(  
        std::begin(s),  
        std::end(s),  
        std::begin(s),  
        ::tolower);  
  
    return s;  
}
```



[thrust::transform](#)



```
toLowerCase :: String -> String  
toLowerCase = map toLower
```



905. Sort Array By Parity

Easy 586 62 Favorite Share

Given an array A of non-negative integers, return an array consisting of all the even elements of A , followed by all the odd elements of A .

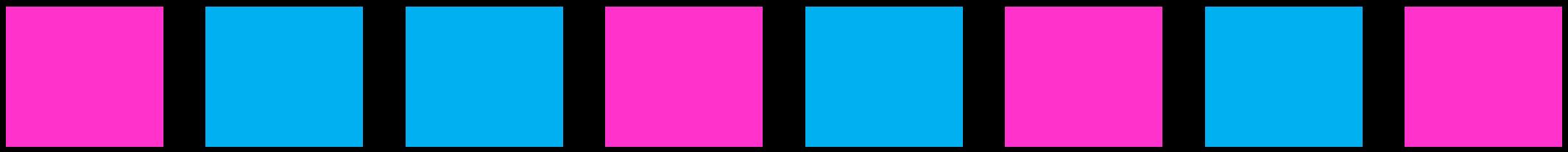
You may return any answer array that satisfies this condition.



```
auto sortArrayByParity(vector<int>& A) {  
    std::partition(  
        std::begin(A),  
        std::end(A),  
        [](auto e) {  
            return e % 2 == 0;  
        });  
  
    return A;  
}
```



[thrust::partition](#)





```
sortArrayByParity :: [Int] -> ([Int], [Int])
sortArrayByParity = partition odd
```



[1,2,3,4,5,6]
([1,3,5],[2,4,6])
[1,3,5,2,4,6]

```
sortArrayByParity :: [Int] -> [Int]
sortArrayByParity = uncurry (++)
| | | | . partition odd
```



283. Move Zeroes

Easy

2463

90

Favorite

Share

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.



```
void moveZeroes(vector<int>& nums) {  
    std::stable_partition(  
        std::begin(nums),  
        std::end(nums),  
        [] (auto e) {  
            return e != 0;  
        });  
}
```



[thrust::stable_partition](#)



[1,0,2,0,3,0]
([1,2,3],[0,0,0])
[1,2,3,0,0,0]

```
moveZeroes :: [Int] -> [Int]
moveZeroes = uncurry (++)
| | |
| . partition (/=0)
```



973. K Closest Points to Origin

Medium

782

75

Favorite

Share

We have a list of `points` on the plane. Find the `K` closest points to the origin $(0, 0)$.

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)



```
vector<vector<int>> kClosest(
    vector<vector<int>>& points, int K) {

    std::sort(
        std::begin(points),
        std::end(points),
        [] (auto const& a, auto const& b) {
            return sqrt(a[0] * a[0] + a[1] * a[1]) <
                   sqrt(b[0] * b[0] + b[1] * b[1]);
        });

    return vector(
        std::begin(points),
        std::begin(points) + K);
}
```



[thrust::sort](#)



```
vector<vector<int>> kClosest(
    vector<vector<int>>& points, int K) {

    std::sort(
        std::begin(points),
        std::end(points),
        [] (auto const& a, auto const& b) {
            return (a[0] * a[0] + a[1] * a[1]) <
                (b[0] * b[0] + b[1] * b[1]);
        });

    return vector(
        std::begin(points),
        std::begin(points) + K);
}
```



[thrust::sort](#)



```
vector<vector<int>> kClosest(
    vector<vector<int>>& points, int K) {

    std::partial_sort(
        std::begin(points),
        std::begin(points) + K,
        std::end(points),
        [] (auto const& a, auto const& b) {
            return (a[0] * a[0] + a[1] * a[1]) <
                (b[0] * b[0] + b[1] * b[1]);
        });

    return vector(
        std::begin(points),
        std::begin(points) + K);
}
```



```
vector<vector<int>> kClosest(
    vector<vector<int>>& points, int K) {

    std::nth_element(
        std::begin(points),
        std::begin(points) + K,
        std::end(points),
        [] (auto const& a, auto const& b) {
            return (a[0] * a[0] + a[1] * a[1]) <
                (b[0] * b[0] + b[1] * b[1]);
        });

    return vector(
        std::begin(points),
        std::begin(points) + K);
}
```



`std::sort`



`std::partial_sort`



`std::nth_element`



Kate Gregory

Naming is Hard:
Let's Do Better

An <algorithm> story

- sort
- partial_sort

1 5 4 2 9 7

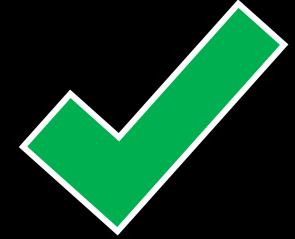
Video Sponsorship Provided By:

ansatz



partial_sort_copy

top_n



nth_element_copy

top_n



`set`

`unordered_set`



`nth_element_copy`

`top_n`

`partial_sort_copy`

`top_n_sorted`



reverse

stable_partition

transform

nth_element

for_each

sort

partition

partial_sort



std::swap ->
std::move

reverse

stable_partition

transform

nth_element

for_each

sort

partition

partial_sort



std::swap ->
std::move

reverse

stable_partition

transform

nth_element

for_each

sort

partition

partial_sort



move

swap

stable_partition

Chapter 2

Twitter Followers



Kate Gregory

Naming is Hard:
Let's Do Better

An <algorithm> story

Video Sponsorship Provided By:

ansatz



Kate Gregory

Naming is Hard:
Let's Do Better

An <algorithm> story

Video Sponsorship Provided By:

ansatz



Kate Gregory

Naming is Hard:
Let's Do Better

An <algorithm> story

Video Sponsorship Provided By:

ansatz



Algorithm Love Club

0 Tweets



Edit profile

Algorithm Love Club

@algo_love_club

For those that want to join the Algorithm Love Club! Our leaders are @SeanParent and Kate Gregory (@gregcons)!

**JOIN
THE
CLUB**

@algo_love_club

Join the club on Twitter



#include



Videos

Scott Schurr

Sean Parent

Marshall Clow

Kate Gregory

Jonathon Bocarra

Odin Holmes

Ben Deane

Matt Godbolt

Ivan Čukić

Code/Slides

Jonathon Bocarra

Sy Brand

Matt Godbolt

Barry Revzin

#include



```
auto make_cpp_ppl() {
    return vector{ make_tuple("Scott Schurr",      "-",
                             "0.0),
                  make_tuple("Sean Parent",     "@SeanParent",
                             "4.2),
                  make_tuple("Marshall Clow",  "@mclow",
                             "0.9),
                  make_tuple("Kate Gregory",   "@gregcons",
                             "6.8),
                  make_tuple("Jonathon Bocarra", "@jobocarra",
                             "4.4),
                  make_tuple("Odin Holmes",    "@odinthenerd",
                             "4.6),
                  make_tuple("Ben Deane",      "@ben_deane",
                             "2.2),
                  make_tuple("Matt Godbolt",   "@mattgodbolt",
                             "7.4),
                  make_tuple("Ivan Cukic",     "@ivan_cukic",
                             "1.5),
                  make_tuple("Sy Brand",       "@TartanLlama",
                             "5.8),
                  make_tuple("Barry Revzin",   "@BarryRevzin",
                             "0.8),
                  make_tuple("Jonathon Bocarra", "@jobocarra",
                             "4.4),
                  make_tuple("Matt Godbolt",   "@mattgodbolt",
                             "7.4) };
}
```



```
auto make_cpp_ppl() {
    return vector{ make_tuple("Scott Schurr",      "-",          0.0),
                  make_tuple("Sean Parent",     "@SeanParent",  4.2),
                  make_tuple("Marshall Clow",  "@mclow",       0.9),
                  make_tuple("Kate Gregory",   "@gregcons",   6.8),
                  make_tuple("Jonathon Bocarra", "@jobocarra", 4.4),
                  make_tuple("Odin Holmes",    "@odinthenerd", 4.6),
                  make_tuple("Ben Deane",      "@ben_deane",   2.2),
                  make_tuple("Matt Godbolt",   "@mattgodbolt", 7.4),
                  make_tuple("Ivan Cukic",     "@ivan_cukic",  1.5),
                  make_tuple("Sy Brand",       "@TartanLlama", 5.8),
                  make_tuple("Barry Revzin",   "@BarryRevzin", 0.8),
                  make_tuple("Jonathon Bocarra", "@jobocarra",  4.4),
                  make_tuple("Matt Godbolt",   "@mattgodbolt", 7.4),
                  make_tuple("Bryce Adelstain Lelbach", "@blelbach", 4.35)};
}
```



```
auto make_cpp_ppl() {
    return vector{ make_tuple("Scott Schurr",                 "-",
                             "Sean Parent",          "@SeanParent",   0.0),
                  make_tuple("Marshall Clow",      "@mclow",        4.2),
                  make_tuple("Kate Gregory",     "@gregcons",    0.9),
                  make_tuple("Jonathon Bocarra", "@jobocarra",   6.8),
                  make_tuple("Odin Holmes",       "@odinthenerd", 4.4),
                  make_tuple("Ben Deane",         "@ben_deane",   4.6),
                  make_tuple("Matt Godbolt",      "@mattgodbolt", 2.2),
                  make_tuple("Ivan Cukic",        "@ivan_cukic",  7.4),
                  make_tuple("Sy Brand",          "@TartanLlama", 1.5),
                  make_tuple("Barry Revzin",      "@BarryRevzin", 5.8),
                  make_tuple("Jonathon Bocarra", "@jobocarra",  0.8),
                  make_tuple("Matt Godbolt",      "@mattgodbolt", 4.4),
                  make_tuple("Bryce Adelstein Lelbach", "@blelbach",  7.4),
                                         "@blelbach",  4.35) };
}
```



```
auto cpp_ppl = make_cpp_ppl();

std::sort(
    std::begin(cpp_ppl),
    std::end(cpp_ppl));

auto last = std::unique(
    std::begin(cpp_ppl),
    std::end(cpp_ppl));

cpp_ppl.erase(
    last,
    std::end(cpp_ppl));
```



[thrust::unique thrust::sort](#)



```
auto cpp_ppl = make_cpp_ppl();

std::sort(
    std::begin(cpp_ppl),
    std::end(cpp_ppl));

cpp_ppl.erase(
    std::unique(
        std::begin(cpp_ppl),
        std::end(cpp_ppl)),
    std::end(cpp_ppl));
```



[thrust::unique thrust::sort](#)

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell		
	Elixir		
	Ruby		
	D		
	Rust		
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir		
	Ruby		
	D		
	Rust		
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby		
	D		
	Rust		
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D		
	Rust		
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D	uniq	squeeze*
	Rust		
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D	uniq	squeeze*
	Rust	unique	dedup
	F#		
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D	uniq	squeeze*
	Rust	unique	dedup
	F#	distinct	-
	Clojure		
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D	uniq	squeeze*
	Rust	unique	dedup
	F#	distinct	-
	Clojure	distinct	dedupe
	Kotlin		
	Scala		

	Language	remove duplicates	std::unique
	C++	sort + unique	unique
	Haskell	sortUniq	-
	Elixir	uniq	dedup
	Ruby	uniq	-
	D	uniq	squeeze*
	Rust	unique	dedup
	F#	distinct	-
	Clojure	distinct	dedupe
	Kotlin	distinct	-
	Scala	distinct	-



```
std::sort(  
    std::begin(cpp_ppl),  
    std::end(cpp_ppl),  
    [] (auto const& person_a,  
        auto const& person_b) {  
        return std::get<2>(person_a) >  
               std::get<2>(person_b);  
    } );
```



[thrust::sort](#)



```
std::for_each(
    std::begin(cpp_ppl),
    std::end(cpp_ppl),
    [](auto const& person) {
        auto [name, twitter_handle, num_followers] = person;
        std::cout << ' ' << name
                    << ' ' << twitter_handle
                    << ' ' << num_followers << '\n';
    });
}
```



[thrust::for_each](#)



std::copy(...)

```
std::transform(  
    std::begin(cpp_ppl),  
    std::end(cpp_ppl),  
    std::ostream_iterator<std::string>(std::cout, "\n"),  
    [](auto const& person) {  
        auto [name, twitter_handle, num_followers] = person;  
        return " "s + name +  
               " "s + twitter_handle +  
               " "s + std::to_string(num_followers);  
    });
```



[thrust::transform](#) [thrust::copy](#)



Matt Godbolt	@mattgodbolt	7.5
Kate Gregory	@gregcons	6.9
Sy Brand	@TartanLlama	6.0
Odin Holmes	@odinthenerd	4.7
Jonathon Bocarra	@jobocarra	4.5
Bryce Adelstein Lelbach	@blelbach	4.5
Sean Parent	@SeanParent	4.3
Ben Deane	@ben_deane	2.2
Ivan Cukic	@ivan_cukic	1.5
Marshall Clow	@mcclow	0.9
Barry Revzin	@BarryRevzin	0.8
Scott Schurr	-	0.0



Scott Schurr	@Ripple	942.1
Matt Godbolt	@mattgodbolt	7.5
Kate Gregory	@gregcons	6.9
Sy Brand	@TartanLlama	6.0
Odin Holmes	@odinthenerd	4.7
Jonathon Bocarra	@jobocarra	4.5
Bryce Adelstein Lelbach	@blelbach	4.5
Sean Parent	@SeanParent	4.3
Ben Deane	@ben_deane	2.2
Ivan Cukic	@ivan_cukic	1.5
Marshall Clow	@mcclow	0.9
Barry Revzin	@BarryRevzin	0.8

Chapter 3

LeetCode Level 2



977. Squares of a Sorted Array

Easy

543

57

Favorite

Share

Given an array of integers A sorted in non-decreasing order, return an array of the squares of each number, also in sorted non-decreasing order.

Note:

1. $1 \leq A.length \leq 10000$
2. $-10000 \leq A[i] \leq 10000$
3. A is sorted in non-decreasing order.



```
vector<int> sortedSquares(vector<int>& A) {  
    std::transform(  
        std::begin(A),  
        std::end(A),  
        std::begin(A),  
        [] (auto e) {  
            return e * e;  
        });  
  
    std::sort(  
        std::begin(A),  
        std::end(A));  
  
    return A;  
}
```



[thrust::transform](#) [thrust::sort](#)



```
vector<int> sortedSquares(vector<int>& A) {
    transform(begin(A), end(A), begin(A),
              [] (auto e) { return e * e; });
    sort(begin(A), end(A));
    return A;
}
```



[thrust::transform](#) [thrust::sort](#)



20 Ranges

<https://godbolt.org/z/4wQbNY>

```
namespace rv = ranges::views;
namespace ra = ranges::actions;

auto sortedSquares(std::vector<int> A) {
    return A
        | rv::all
        | ra::transform([] (int e) { return e * e; })
        | ra::sort
        | ranges::to<std::vector<int>>;
}
```



[thrust::transform](#) [thrust::sort](#)



[1,3,2,5,4]
[1,9,4,25,16]
[1,4,9,16,25]

```
square :: Int -> Int  
square n = n * n
```

```
sortedSquares :: [Int] -> [Int]  
sortedSquares = sort . map square
```



961. N-Repeated Element in Size 2N Array

Easy 271 167 Favorite Share

In a array `A` of size `2N`, there are `N+1` unique elements, and exactly one of these elements is repeated `N` times.

Return the element repeated `N` times.



961. Element Repeated Once

Easy

271

167

Favorite

Share

In an array A of size N , there are $N - 1$ unique values (therefore exactly one of those values appears twice).

Return the value that is repeated 2 times.



```
int repeatedNTimes(vector<int>& A) {  
    std::sort(  
        std::begin(A),  
        std::end(A));  
  
    return *std::adjacent_find(  
        std::cbegin(A),  
        std::cend(A));  
}
```



[thrust::sort](#)



```
[2,1,2,2,3,4]
[(1,1),(2,3),(3,1),(4,1)]
[(2,3)]
(2,3)
2
```

```
import Data.List.Unique (count)

repeatedNTimes :: [Int] -> Int
repeatedNTimes = fst
  . head
  . filter ((>1) . snd)
  . count x
```



1051. Height Checker

Easy

90

708

Favorite

Share

Students are asked to stand in non-decreasing order of heights for an annual photo.

Return the minimum number of students not standing in the right positions. (This is the number of students that must move in order for all students to be standing in non-decreasing order of height.)



```
int heightChecker(vector<int>& heights) {  
  
    std::vector<int> h_copy(heights.size());  
  
    std::partial_sort_copy(  
        std::begin(heights),  
        std::end(heights),  
        std::begin(h_copy),  
        std::end(h_copy));  
  
    return std::inner_product(  
        std::begin(heights),  
        std::end(heights),  
        std::begin(h_copy),  
        0,  
        std::plus{},  
        std::not_equal_to{});  
}
```



[thrust::inner_product](#)



```
[3,2,1,4]
[(3,1),(2,2),(1,3),(4,4)]
[2,0,-2,0]
[0,0]
2
```

```
heightChecker :: [Int] -> Int
heightChecker = length
| | | | . filter (==0)
| | | | . (zipWith (-) <*> sort)
```



33. Search in Rotated Sorted Array

Medium

3093

368

Favorite

Share

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

If found in the array return `true`, otherwise return `false`.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.



```
bool search(vector<int>& nums, int target) {
    auto f = std::cbegin(nums);
    auto l = std::cend(nums);
    auto p = std::is_sorted_until(f, l);
    return target >= *f ? std::binary_search(f, p, target)
                         : std::binary_search(p, l, target);
}
```



[thrust::is_sorted_until](#) [thrust::binary_search](#)



```
bool search(vector<int>& nums, int target) {
    auto f = std::cbegin(nums);
    auto l = std::cend(nums);
    auto p = std::partition_point(f, l,
        [x = *f] (auto const& e) {
            return e >= x;
        });
    return target >= *f ? std::binary_search(f, p, target)
                        : std::binary_search(p, l, target);
}
```



[thrust::partition_point](#) [thrust::binary_search](#)



```
bool search(vector<int>& nums, int target) {
    auto f = std::cbegin(nums);
    auto l = std::cend(nums);
    auto p = std::partition_point(f, l,
        std::bind(std::greater_equal{}, _1, *f));
    return target >= *f ? std::binary_search(f, p, target)
                         : std::binary_search(p, l, target);
}
```



[thrust::partition_point](#) [thrust::binary_search](#)

Chapter 4

Haskell Engineering

The Four Horseshoers of the



@incomputable



@WalterBright



@ericniebler



@BartoszMilewski





@odinthenerd

”...most programming languages are grossly inferior to the enormous, but not strong, von Neumann computer.”

Inherent constraints at the most basic level cause them to be both slow and weak.

their intuitive hand-at-a-time style of programming inherited from their common ancestor --the von Neumann computer, [...] and their lack of useful mathematical properties for reasoning about programs.”

Auto-Intern GmbH

13

Is this good?

Can Programming Be Liberated from the von Neumann Style?
A Functional Style and Its Algebra of Programs

-- John Backus 1978



Odin Holmes

Tacit DSL All the Things

Video Sponsorship
Provided By:



[https://github.com/graninas/cpp functional programming](https://github.com/graninas/cpp_functional_programming)

- Bartosz Milewski talks
 - Functional Patterns in C++, 1. Functors, 2012
 - Functional Patterns in C++, 2. Currying, Applicative, 2012
 - Functional Patterns in C++, 3. Async API, Monoid, Monad, 2012
 - Compile-Time/Run-Time Functional Programming in C++, Bartosz Milewski, Eric Niebler, BoostCon, 2012
 - Haskell -- The Pseudocode Language for C++ Template Metaprogramming (Part 1), BoostCon, 2013
 - Haskell -- The Pseudocode Language for C++ Template Metaprogramming (Part 2), BoostCon, 2013
 - Re-discovering monads in C++, C++ User Group Novosibirsk, 2014
 - Functional techniques in C++, CDays14, 2014
 - Categories for the Working C++ Programmer, C++ Russia, 2015
 - Monads for C++, itCppCon17, 2017

“Even if you program in C++, it is a good idea to learn some Haskell.”



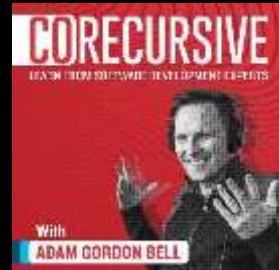
- Bartosz Milewski
@BartoszMilewski



“... if you solve your problem first in Haskell
and then you translate it into C++ you will
probably get better quality code.”



- Bartosz Milewski
@BartoszMilewski





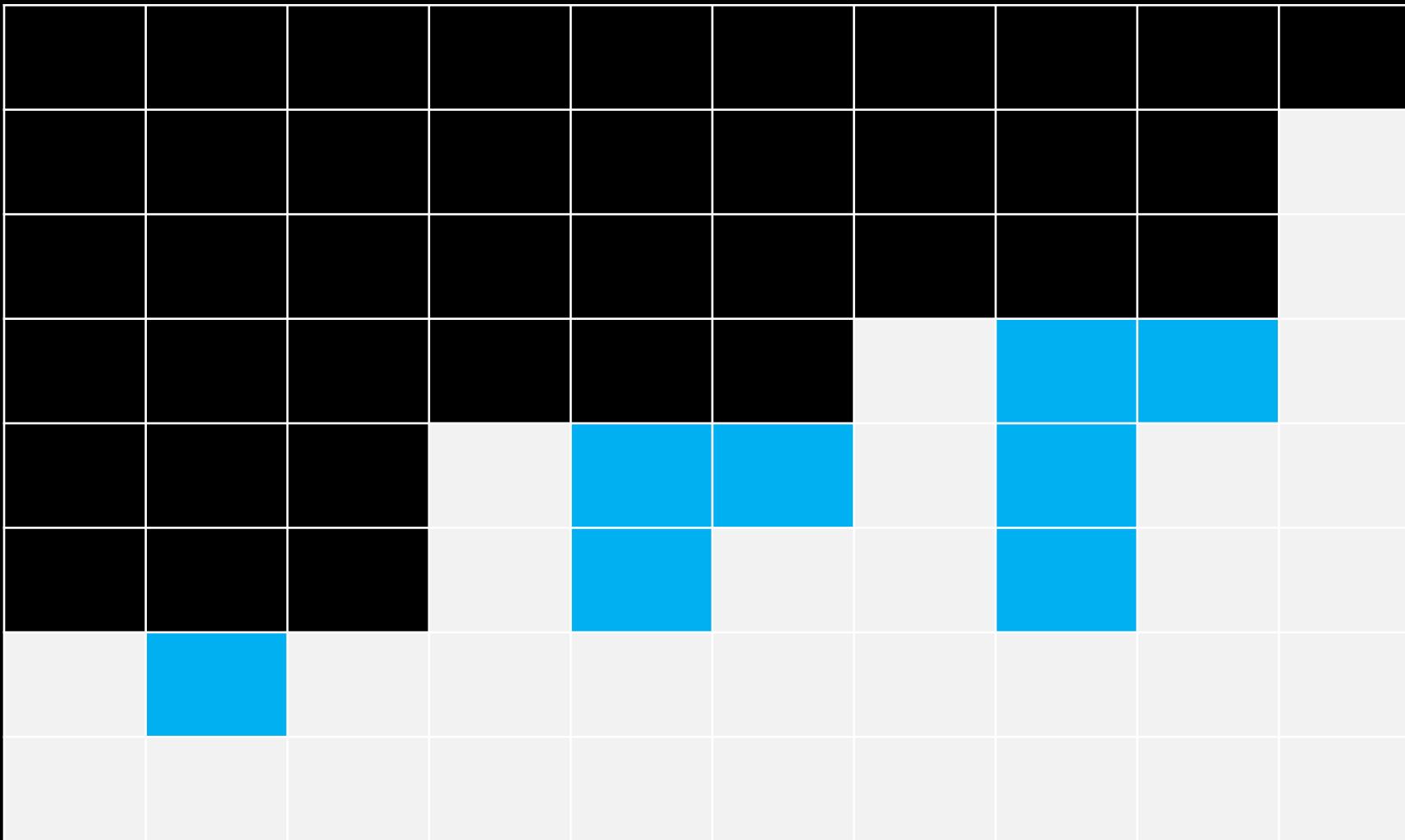
42. Trapping Rain Water

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!





```
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<>());
}
```



[thrust::inclusive_scan](#) [thrust::transform_reduce](#)



```
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<>());
}
```



[thrust::inclusive_scan](#) [thrust::transform_reduce](#) [thrust::max_element](#)



```
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<>());
}
```



[thrust::inclusive_scan](#) [thrust::transform_reduce](#) [thrust::max_element](#)



```
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<>());
}
```



[thrust::inclusive_scan](#) [thrust::transform_reduce](#) [thrust::max_element](#)



```
template<class T>
using rev = reverse_iterator<T>;\n\nint 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<>());
}
```



[thrust::inclusive_scan](#) [thrust::transform_reduce](#) [thrust::max_element](#)



```
trappingRain :: [Int] -> Int
trappingRain xs =
    where
```



```
trappingRain :: [Int] -> Int
trappingRain xs = sum $ zipWith (-) ys xs
    where m      = maximum xs
          i      = length $ takeWhile (/=m) xs
          a      = take i xs
          b      = drop i xs
          ys    = scanl1 max a ++ scanr1 max b
```



```
trappingRain :: [Int] -> Int
trappingRain xs = sum $ zipWith (-) ys xs
    where m      = maximum xs
          i      = length $ takeWhile (/=m) xs
          (a,b) = splitAt i xs
          ys    = scanl1 max a ++ scanr1 max b
```



```
trappingRain :: [Int] -> Int
trappingRain xs = sum $ zipWith (-) ys xs
  where m      = maximum xs
        (a,b) = break (==m) xs
        ys    = scanl1 max a ++ scanr1 max b
```



```
trappingRain :: [Int] -> Int
trappingRain xs = sum $ zipWith (-) ys xs
  where (a,b) = break (==(maximum xs)) xs
        | ys    = scanl1 max a ++ scanr1 max b
```



```
trappingRain :: [Int] -> Int
trappingRain xs = sum $ zipWith (-) ys xs
  where m      = maximum xs
        (a,b) = break (==m) xs
        ys    = scanl1 max a ++ scanr1 max b
```

```
toLowerCase :: String -> String
toLowerCase = map toLower
```

transform_reduce

reduce

transform

?

max_element

inclusive_scan



```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
map   :: (a -> b) -> [a] -> [b]
```



```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
map      :: (a -> b)        -> [a] -> [b]
```



```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
map      :: (a -> b)        -> [a]           -> [b]
```



```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
map      :: (a -> c)        -> [a]           -> [c]
```



std::transform

Chapter 5

LeetCode Level 3



1053. Previous Permutation With One Swap

Medium

62

161

Favorite

Share

Given an array A of positive integers (not necessarily distinct), return the lexicographically largest permutation that is smaller than A , that can be **made with one swap** (A swap exchanges the positions of two numbers $A[i]$ and $A[j]$). If it cannot be done, then return the same array.

[3,2,1] -> [3,1,2]

[1,1,5] -> [1,1,5]

[1,9,4,6,7] -> [1,7,4,6,9]

How many STL algorithms will it take to solve?

1

2

3

4



[1,1,5]



```
vector<int> prevPermOpt1(vector<int>& A) {  
    prev_permutation(A.begin(), A.end());  
    return A;  
}
```



[1,9,4,6,7]

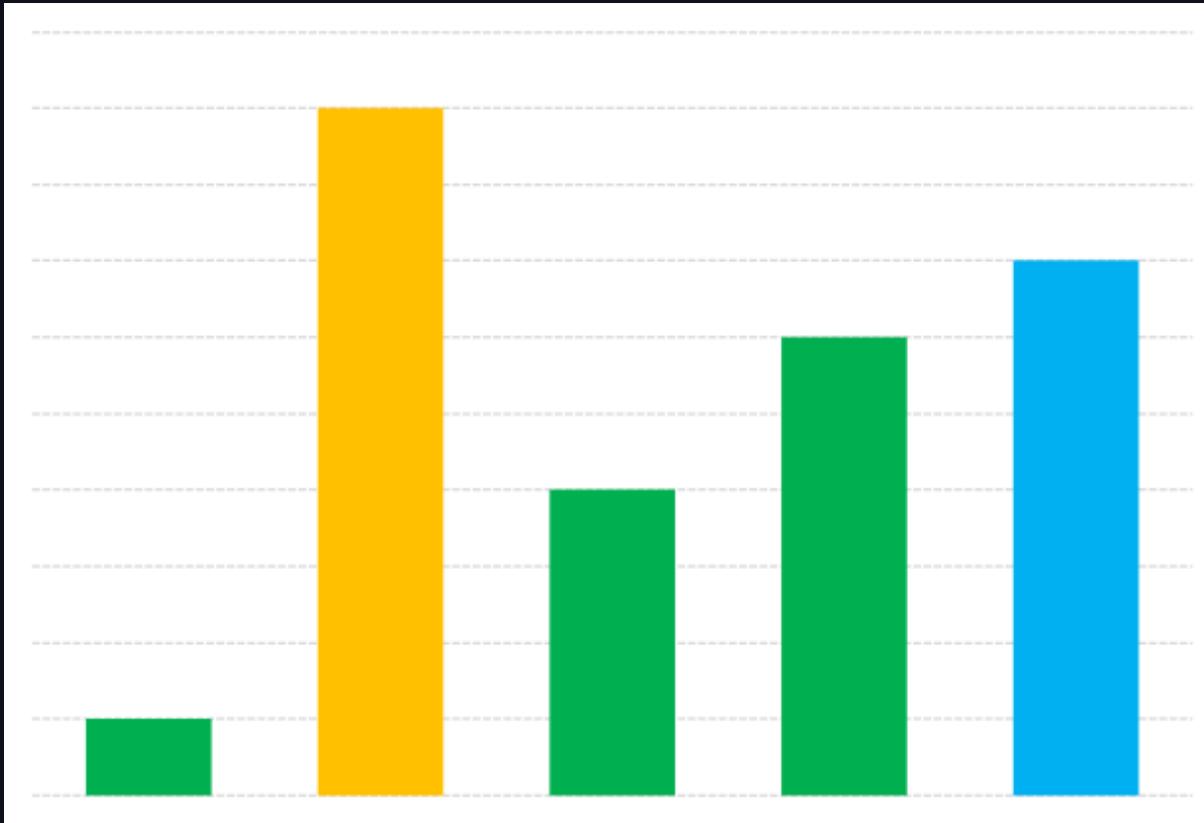


[1,7,4,6,9]

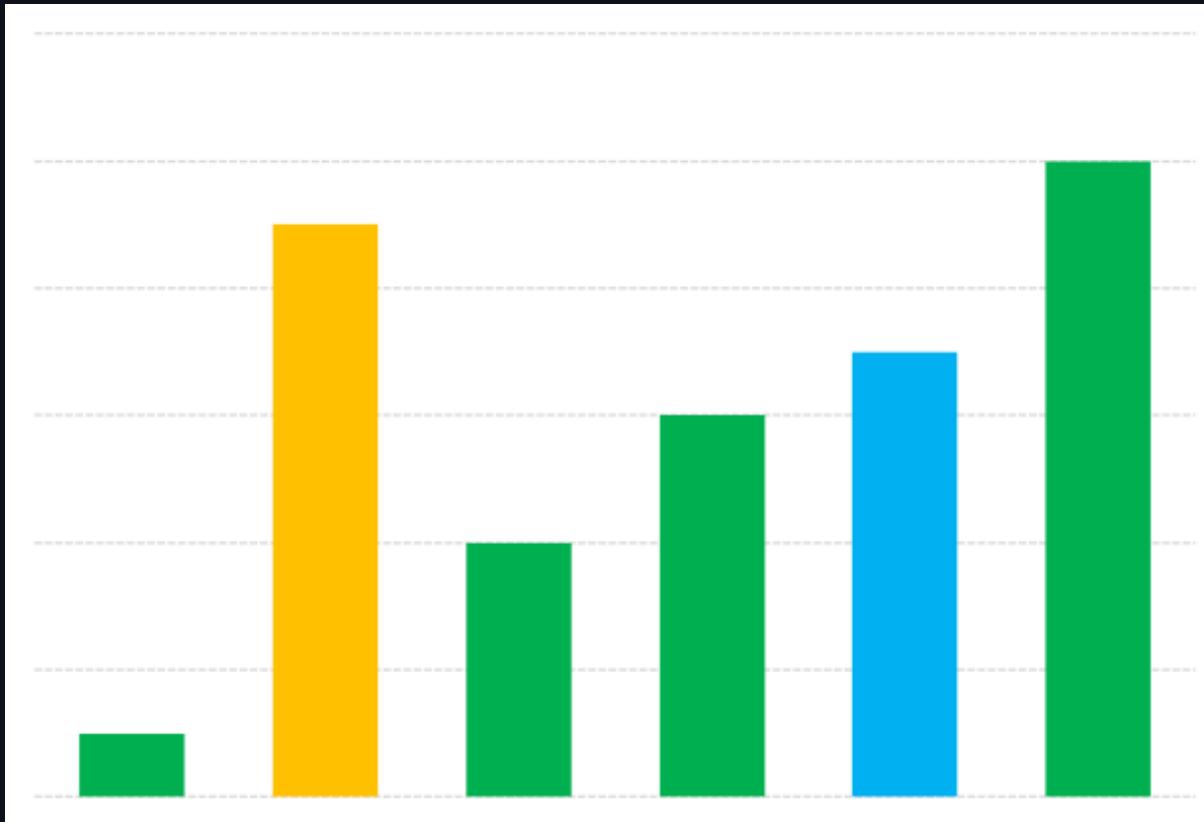
```
vector<int> prevPermOpt1(vector<int>& A) {  
    if (!is_sorted(A.begin(), A.end()))  
        prev_permutation(A.begin(), A.end());  
    return A;  
}
```



[thrust::is_sorted](#)



[1,9,4,6,7]



[1,9,4,6,7,10]



[3,1,1,3]



```
vector<int> prevPermOpt1(vector<int>& A) {
    if (is_sorted(A.begin(), A.end())) return A;
    auto i = is_sorted_until(A.rbegin(), A.rend(), greater<>());
    auto j = find_if(A.rbegin(), i, [&](auto e) { return e < *i; });
    iter_swap(i, j);
    return A;
}
```



[thrust::is_sorted](#) [thrust::is_sorted_until](#) [thrust::find_if](#)

1,1,3,3

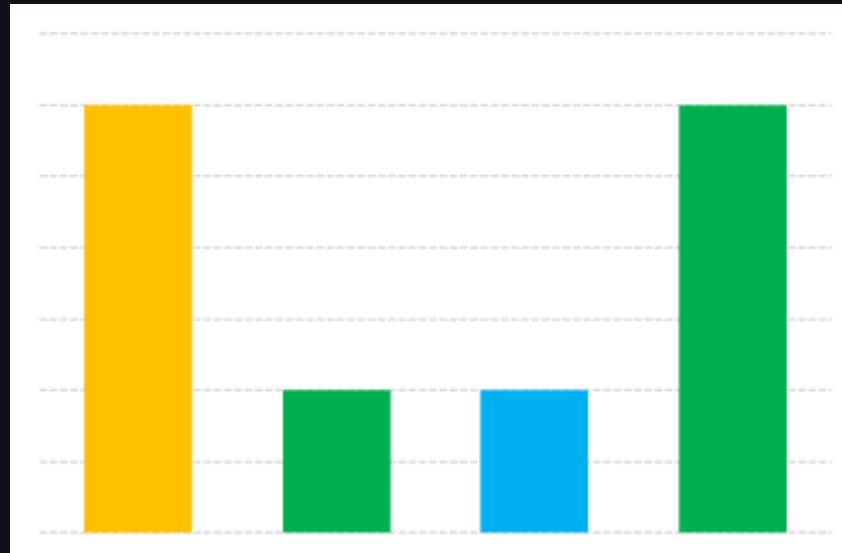
1,3,1,3

1,3,3,1

3,1,1,3

3,1,3,1

3,3,1,1



[3,1,1,3]

1,1,3,3

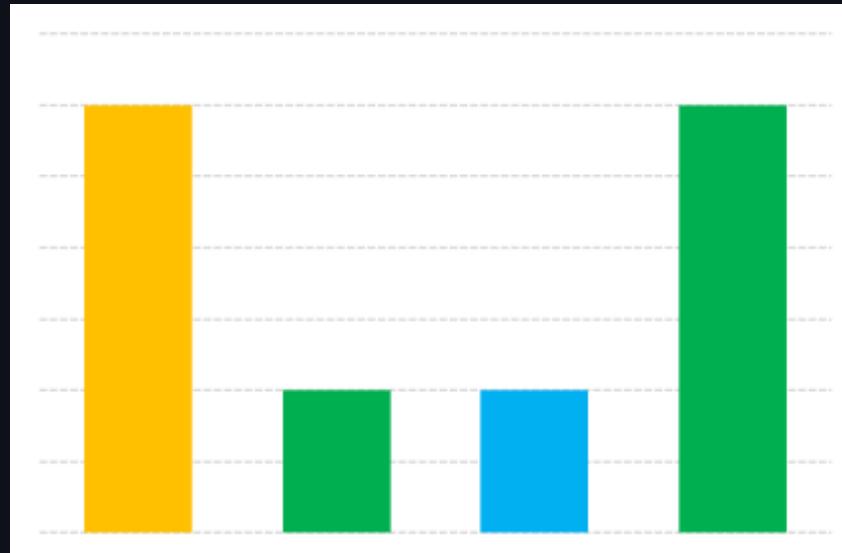
1,3,1,3

1,3,3,1

3,1,1,3

3,1,3,1

3,3,1,1



[3,1,1,3]

1,1,3,3

1,3,1,3

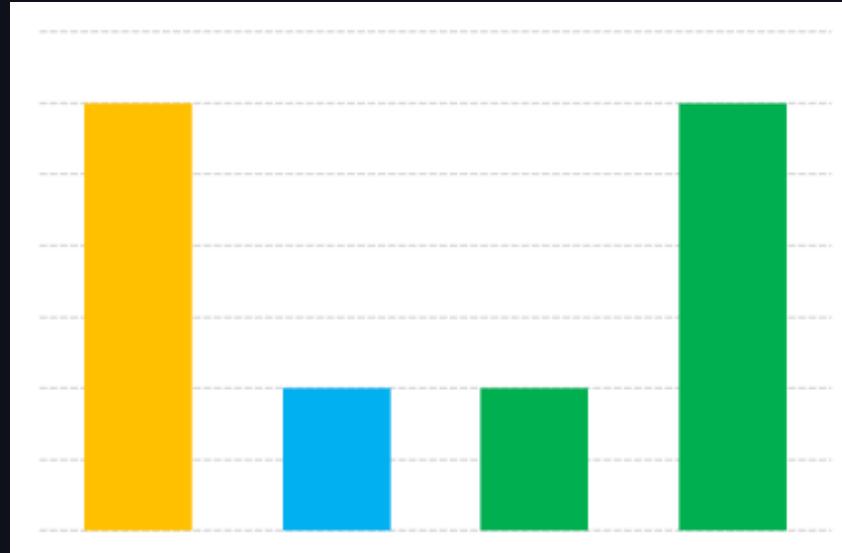
1,3,3,1

3,1,1,3

3,1,3,1

3,3,1,1

1. `is_sort_until` from back -> **i**
2. `find_if < i` from back -> **x**
3. find “range of all values equal to **x**”
4. swap furthest **x** (from back) with **i**



[3,1,1,3]



```
vector<int> prevPermOpt1(vector<int>& A) {
    if (is_sorted(A.begin(), A.end())) return A;
    auto i = is_sorted_until(A.rbegin(), A.rend(), greater<>());
    auto j = find_if(A.rbegin(), i, [&](auto e) { return e < *i; });
    auto p = equal_range(i.base(), A.end(), *j);
    iter_swap(i, p.first);
    return A;
}
```



[thrust::is_sorted](#) [thrust::is_sorted_until](#) [thrust::find_if](#) [thrust::equal_range](#)



```
vector<int> prevPermOpt1(vector<int>& A) {
    auto i = is_sorted_until(A.rbegin(), A.rend(), greater<>());
    if (i != A.rend()) {
        auto j = find_if(A.rbegin(), i, [&](auto e) { return e < *i; });
        auto p = equal_range(i.base(), A.end(), *j);
        iter_swap(i, p.first);
    }
    return A;
}
```



[thrust::is_sorted_until](#) [thrust::find_if](#) [thrust::equal_range](#)

```
// Next, check if the panel has moved to the other side of another panel.  
  
auto p = find_if(begin(expanded_panels_), end(expanded_panels_),  
    [&](const ref_ptr<Panel>& e){ return center_x <= e->cur_panel_center(); });  
  
// Fix this code - panel is the panel found above.  
  
if (panel != fixed_panel) {  
    // If it has, then we reorder the panels.  
    ref_ptr<Panel> ref = expanded_panels_[fixed_index];  
    expanded_panels_.erase(expanded_panels_.begin() + fixed_index);  
    expanded_panels_.insert(expanded_panels_.begin() + i, ref);  
}
```



```
template<typename F, typename T>
F first_less_than(F f, F l, T val) {
    auto it = std::lower_bound(f, l, val);
    return it == f ? l : --it;
}

vector<int> prevPermOpt1(vector<int>& A) {
    auto i = is_sorted_until(A.rbegin(), A.rend(), greater<>());
    if (i != A.rend()) {
        auto j = first_less_than(i.base(), A.end(), *i);
        auto p = equal_range(i.base(), A.end(), *j);
        iter_swap(i, p.first);
    }
    return A;
}
```



[thrust::is_sorted_until](#) [thrust::lower_bound](#) [thrust::equal_range](#)

How many STL algorithms will it take to solve?

1

2

3

4



917. Reverse Only Letters

Easy

321

30

Favorite

Share

Given a string `s`, return the "reversed" string where all characters that are not a letter stay in the same place, and all letters reverse their positions.

1ab2cd -> 1dc2ba

How many STL algorithms will it take to solve?

1

2

3

4



```
string reverseOnlyLetters(string S) {  
    std::string letters;  
  
    std::copy_if(  
        std::begin(S),  
        std::end(S),  
        std::back_inserter(letters),  
        ::isalpha);  
  
    std::reverse(  
        std::begin(letters),  
        std::end(letters));  
  
    std::transform(  
        std::begin(S),  
        std::end(S),  
        std::begin(S),  
        [c = std::cbegin(letters)] (auto e) mutable {  
            return isalpha(e) ? *c++ : e;  
        });  
  
    return S;  
}
```



[transform](#) [reverse](#) [copy_if](#)



```
string reverseOnlyLetters(string S) {  
    std::string letters;  
  
    std::copy_if(  
        std::crbegin(S),  
        std::crend(S),  
        std::back_inserter(letters),  
        ::isalpha);  
  
    std::transform(  
        std::begin(S),  
        std::end(S),  
        std::begin(S),  
        [c = std::cbegin(letters)] (auto e) mutable {  
            return isalpha(e) ? *c++ : e;  
        });  
  
    return S;  
}
```



[transform](#) [copy_if](#)



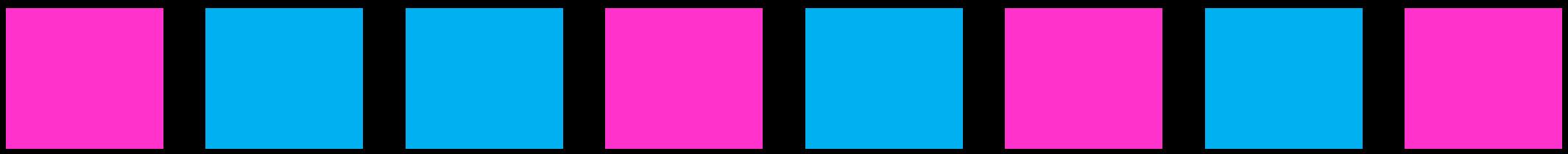
```
string reverseOnlyLetters(string S) {  
    std::reverse_if(  
        std::begin(S),  
        std::end(S),  
        ::isalpha);  
  
    return S;  
}
```

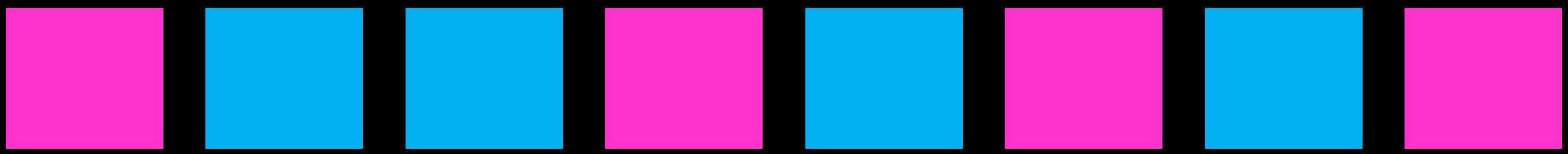


```
template <class B, class P>
void reverse_if(B first, B last, P pred) {
    while (first != last) {
        while (!pred(*first)) ++first;
        while (!pred(*last)) --last;
        iter_swap(first++, last++);
    }
}
```



```
template <class B, class P>
void reverse_if(B f, B l, P pred) {
    while (f != l) {
        while (!pred(*f)) { ++f; if (f == l) goto done; }
        while (!pred(*l)) { --l; if (f == l) goto done; }
        iter_swap(f++, l);
        if (f == l) break;
        ++l;
    }
done: ;
}
```







```
std::string reverseOnlyLetters(std::string S) {  
    std::partition(  
        std::begin(S),  
        std::end(S),  
        [b = true] (auto e) mutable {  
            if (isalpha(e)) b = !b;  
            return b;  
        });  
  
    return S;  
}
```



[thrust::partition](#)

How many STL algorithms will it take to solve?

1

2

3

4

Chapter 6

The Sean Parent Game



`std::is_sorted`



`std::is_sorted_until`



```
template<class _FwdIt,
         class _Pr>
_NODISCARD inline bool is_sorted(_FwdIt _First, _FwdIt _Last, _Pr _Pred)
{ // test if range is ordered by predicate
    _Adl_verify_range(_First, _Last);
    const auto _UFirst = _Get_unwrapped(_First);
    const auto _ULast = _Get_unwrapped(_Last);
    return (_STD is_sorted_until(_UFirst, _ULast, _Pass_fn(_Pred)) == _ULast);
}
```



[thrust::is_sorted](#) [thrust::is_sorted_until](#)



`std::is_sorted`



`std::is_sorted_until`



`std::adjacent_find`



```
template<class _FwdIt,
         class _Pr>
_NODISCARD inline _FwdIt is_sorted_until(const _FwdIt _First, _FwdIt _Last, _Pr _Pred)
{   // find extent of range that is ordered by predicate
    _Adl_verify_range(_First, _Last);
    auto _UFirst = _Get_unwrapped(_First);
    auto _ULast = _Get_unwrapped(_Last);
    if (_UFirst != _ULast)
    {
        for (auto _UNext = _UFirst; ++_UNext != _ULast; ++_UFirst)
        {
            if (_DEBUG_LT_PRED(_Pred, *_UNext, *_UFirst))
            {
                _ULast = _UNext;
                break;
            }
        }
        _Seek_wrapped(_Last, _ULast);
    }
    return (_Last);
}
```



```
template<class _FwdIt,
         class _Pr>
_NODISCARD inline _FwdIt adjacent_find(const _FwdIt _First, _FwdIt _Last, _Pr _Pred)
{   // find first satisfying _Pred with successor
    _Adl_verify_range(_First, _Last);
    auto _UFirst = _Get_unwrapped(_First);
    auto _ULast = _Get_unwrapped(_Last);
    if (_UFirst != _ULast)
    {
        for (auto _UNext = _UFirst; ++_UNext != _ULast; _UFirst = _UNext)
        {
            if (_Pred(*_UFirst, *_UNext))
            {
                _ULast = _UFirst;
                break;
            }
        }
    }

    _Seek_wrapped(_Last, _ULast);
    return (_Last);
}
```



```
1 template<class _FwdIt,
2      class _Pr>
3     _NODISCARD inline _FwdIt adjacent_find(const
4         _FwdIt _First, _FwdIt _Last, _Pr _Pred)
5     { // find first satisfying _Pred with successor
6         _Adl_verify_range(_First, _Last);
7         auto _UFirst = _Get_unwrapped(_First);
8         auto _ULast = _Get_unwrapped(_Last);
9         if (_UFirst != _ULast)
10            {
11                for (auto _UNext = _UFirst; ++_UNext != _ULast; _UFirst = _UNext)
12                    {
13                        if (_Pred(*_UFirst, *_UNext))
14                            {
15                                _ULast = _UFirst;
16                                break;
17                            }
18            }
19
20         _Seek_wrapped(_Last, _ULast);
21         return (_Last);
22     }
```

```
1 template<class _FwdIt,
2      class _Pr>
3     _NODISCARD inline _FwdIt is_sorted_until(const
4         _FwdIt _First, _FwdIt _Last, _Pr _Pred)
5     { // find extent of range that is ordered by predicate
6         _Adl_verify_range(_First, _Last);
7         auto _UFirst = _Get_unwrapped(_First);
8         auto _ULast = _Get_unwrapped(_Last);
9         if (_UFirst != _ULast)
10            {
11                for (auto _UNext = _UFirst; ++_UNext != _ULast; ++_UFirst)
12                    {
13                        if (_DEBUG_LT_PRED(_Pred, *_UNext, *_UFirst))
14                            {
15                                _ULast = _UNext;
16                                break;
17                            }
18            }
19
20         _Seek_wrapped(_Last, _ULast);
21         return (_Last);
22     }
```



```
#include <boost/hana/functional/flip.hpp>

template<class F, class P>
F is_sorted_until(F first, F last, P pred) {
    return std::adjacent_find(
        first,
        last,
        boost::hana::flip(pred));
}
```





`std::is_sorted`



`std::is_sorted_until`



`std::adjacent_find`



`std::is_sorted`



`std::is_sorted_until`



`std::adjacent_find`



`std::mismatch`



```
template <class I, class P>
I adjacent_find(I first, I last, P pred) {
    return std::mismatch(
        first,
        std::prev(last),
        std::next(first),
        [pred] (auto const& a, auto const& b) {
            return !pred(a, b);
        }).first;
}
```



```
#include <boost/hana/functional/compose.hpp>
#include <boost/hana/fwd/not.hpp>

template <class I, class P>
I adjacent_find(I first, I last, P pred) {
    return std::mismatch(
        first,
        std::prev(last),
        std::next(first),
        boost::hana::compose(
            boost::hana::not_,
            pred))
    .first;
}
```



```
#include <boost/hana/functional/demux.hpp>
#include <boost/hana/not.hpp>

template <class I, class P>
I adjacent_find(I first, I last, P pred) {
    return std::mismatch(
        first,
        std::prev(last),
        std::next(first),
        boost::hana::demux(
            boost::hana::not_)(pred))
    .first;
}
```





```
template <class I, class P>
I adjacent_find(I first, I last, P pred) {
    return std::mismatch(
        first,
        std::prev(last),
        std::next(first),
        std::not_fn(pred)).first;
}
```





zip_find

transform_find



zip_find

```
auto mismatch(F f, F l, F f2, P p) {
    std::zip_find(f, l, f2,
                  [p] (auto a, auto b) { return !p(a, b) });
}
```



```
template<class _InIt1,
         class _RightTy,
         size_t _RightSize>
_NODISCARD inline pair<_InIt1, _RightTy *> mismatch(const _InIt1 _First1, const _InIt1 _Last1,
[_RightTy (&_First2)[_RightSize])
{ // return [_First1, _Last1)/[_First2, ...) mismatch, array source
return (_STD mismatch(_First1, _Last1, _First2, equal_to<>())));
}
```



mismatch -> **zip_find**

transform -> **zip_with**

inner_product -> **zip_reduce**

transform_reduce -> **zip_reduce**

equal -> **zip_reduce(logical_and, equal_to)**

includes, **merge**, **search**, **set_***, **swap_range**



```
template<class _InIt1,
         class _InIt2,
         class _Pr> inline
bool _Equal_unchecked1(_InIt1 _First1, const _InIt1 _Last1,
                      _InIt2 _First2, _Pr _Pred, false_type)
{   // compare [_First1, _Last1) to [_First2, ...) using _Pred,
for (; _First1 != _Last1; ++_First1, (void)++_First2)
{
    if (!_Pred(*_First1, *_First2))
    {
        return (false);
    }
}

return (true);
```



```
template <class InputIterator1,
          class InputIterator2,
          class BinaryPredicate>
inline bool equal(InputIterator1 first1,
                  InputIterator1 last1,
                  InputIterator2 first2,
                  BinaryPredicate binary_pred) {
    return mismatch(
        first1,
        last1,
        first2,
        binary_pred).first == last1;
}
```



<http://stepanovpapers.com/butler.hpl.hp/stl/stl/ALGOBASE.H>

```
template <class InputIterator1,
          class InputIterator2,
          class BinaryPredicate>
inline bool equal(InputIterator1 first1,
                  InputIterator1 last1,
                  InputIterator2 first2,
                  BinaryPredicate binary_pred) {
    return mismatch(
        first1,
        last1,
        first2,
        binary_pred).first == last1;
}
```





<http://stepanovpapers.com/butler.hpl.hp/stl/stl/ALGOBASE.H>

```
template <class InputIterator1, class InputIterator2, class BinaryPredicate>
inline bool equal(InputIterator1 first1, InputIterator1 last1,
                  InputIterator2 first2, BinaryPredicate binary_pred) {
    return mismatch(first1, last1, first2, binary_pred).first == last1;
}
```



mismatch -> **zip_find**

transform -> **zip_with**

inner_product -> **zip_reduce**

transform_reduce -> **zip_reduce**

equal -> **zip_reduce(logical_and, equal_to)**

equal -> **mismatch() == end()**

includes, merge, search, set_*, swap_range



`mismatch` -> `zip_find`

`adjacent_difference` -> `adjacent_transform`



reverse
transform
partition
stable_partition
sort
partial_sort
nth_element
remove_if

unique
for_each
copy
adjacent_find
partial_sort_copy
partition_point
binary_search
prev_permutation

is_sorted
is_sorted_until
find_if
equal_range
lower_bound
copy_if
mismatch
equal

Conclusion

Chapter 1:

sort | partial_sort | nth_element
remove_if | stable_partition

Chapter 2:

sort + unique = dedup
follow [@algo_love_club](#) + [@BenDeane](#), [@gregcons](#), [@SeanParent](#) ...

Chapter 3: is_sorted_until | partition_point (... is log n)

Chapter 4: transform is both zipWith and map

Chapter 5:

find_if | lower_bound / upper_bound
partition | reverse_if

Chapter 6: is_sorted | is_sorted_until | adjacent_find | mismatch



code_report



codereport

The screenshot shows a video player interface with a purple background. At the top, there are logos for various platforms: HackerRank, topcoder, hackerearth, CODECHEF, LeetCode, and CODEFORCES. Below the logos, two video thumbnails are visible:

- STL Algorithms 13: std::unique & std::unique_copy**
Duration: 13:57
- STL Algorithms 12: min, max, minmax, min_element, max_element & std::minmax**
Duration: 6:10

In the center of the screen, a woman with short blonde hair is smiling. She is wearing a dark t-shirt with the text "T 2014" printed on it. The background behind her is also purple and features the same platform logos.

At the bottom of the video player, there are two sets of social media links:

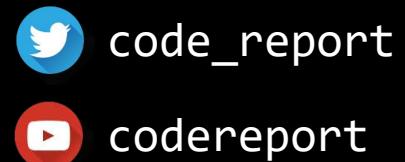
- /codereport** (with a YouTube icon)
- /code_report** (with a Twitter icon)



Thank you!

<https://github.com/codereport/Talks/>

Conor Hoekstra





Questions?

<https://github.com/codereport/Talks/>

Conor Hoekstra

 code_report

 codereport