105 STL ALGORITMS IN LESS THAN AN HOUR

Jonathan Boccara

@JoBoccara

Fluent {C++}

Hi, I'm Jonathan Boccara!

@JoBoccara

105 STL ALGORITHMS IN LESS THAN AN HOUR

Agenda

- Brief chat about STL algorithms
- The STL algorithms themselves

@JoBoccara

We need to know them. All. Well. Very well.

STL algorithms can make code more expressive

Raising levels of abstraction

Sometimes, it can be spectacular.

```
void PanelBar::RepositionExpandedPanels(Panel* fixed_panel) {
  CHECK(fixed_panel);

  // First, find the index of the fixed panel.
  int fixed_index = GetPanelIndex(expanded_panels_, *fixed_panel);
  CHECK_LT(fixed_index, expanded_panels_.size());

  // Next, check if the panel has moved to the other side of another panel.
  const int center_x = fixed_panel->cur_panel_center();
  for (size_t i = 0; i < expanded_panels_.size(); ++i) {
    Panel* panel = expanded_panels_[i].get();
    if (center_x <= panel->cur_panel_center() ||
        i == expanded_panels_.size() - 1) {
      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);
        if (i < expanded_panels_.size()) {
          expanded_panels_.insert(expanded_panels_.begin() + i, ref);
        } else {
          expanded_panels_.push_back(ref);
        }
      }
      break;
    }
  }
```

```
// Next, check if the panel has moved to the left side of another panel.

auto f = begin(expanded_panels_) + fixed_index;

auto p = lower_bound(begin(expanded_panels_), f, center_x,
  [](const ref_ptr<Panel>& e, int x){ return e->cur_panel_center() < x; });

// If it has, then we reorder the panels.
rotate(p, f, f + 1);
```

@JoBoccara

We need to know them. All. Well. Very well.

STL algori[illegible] make code

R[illegible]tractio[illegible]

Som[illegible]mes, it [illegible]spectac[illegible]la

Avoi[illegible] mistakes:

We need to know them. All. Well. Very well.

STL algori[illegible] make code

- R[illegible]tractio[illegible]
- Som[illegible]mes, it [illegible]spectac[illegible]

Avoi[illegible] mistakes:

- ot[illegible]
- em[illegible]

We need to know them. All. Well. Very well.

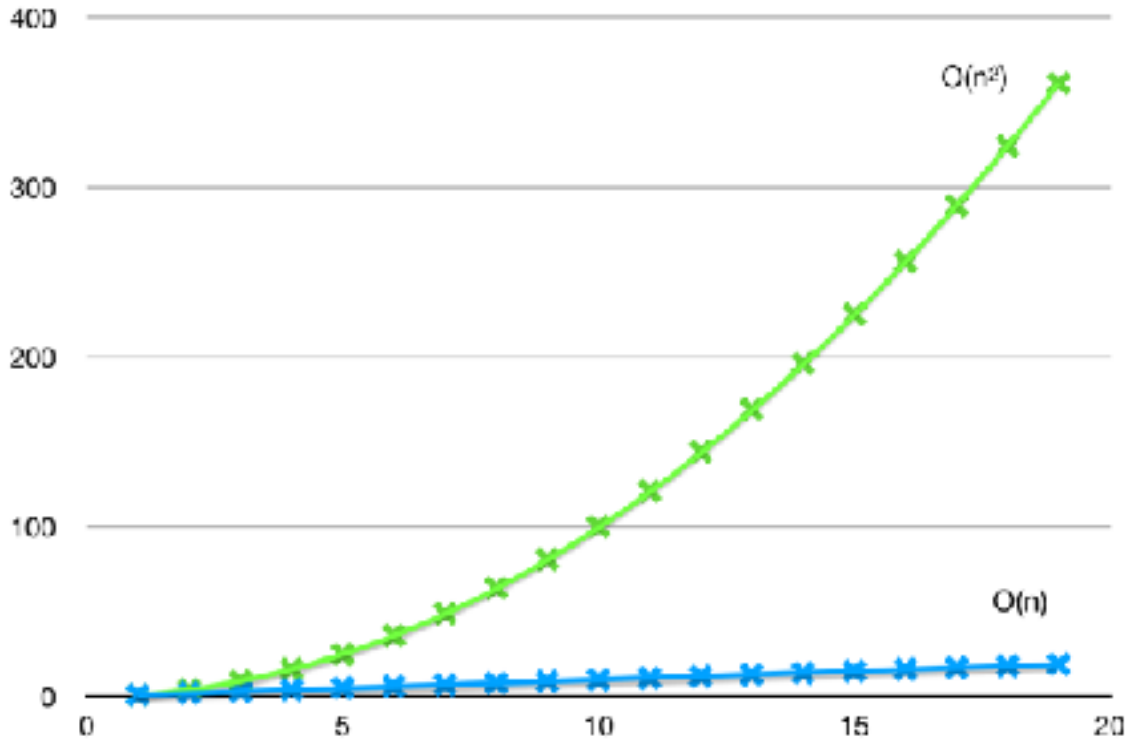
STL algorithms can make code more expressive

Raising levels of abstraction

Sometimes, it can be spectacular.

Avoid common mistakes:

off-by-one
empty loops
naive complexity



We need to know them. All. Well. Very well.

STL algorithms can make code more expressive

- Raising levels of abstraction
- Sometimes, it can be spectacular.

Avoid common mistakes:

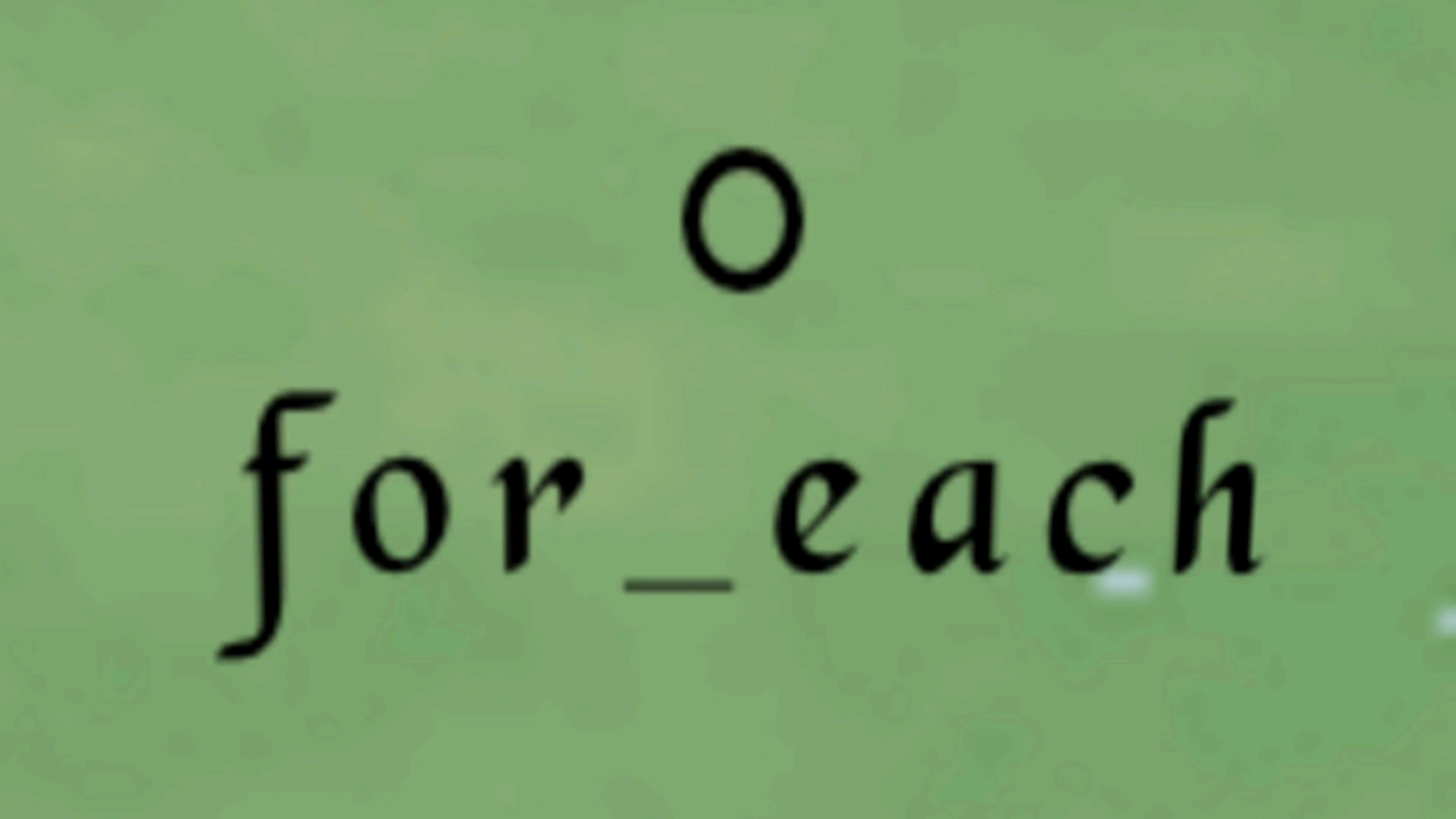
- off-by-one
- empty loops
- naive complexity

Used by lots of people

- A common vocabulary
- Whatever the version of your compiler

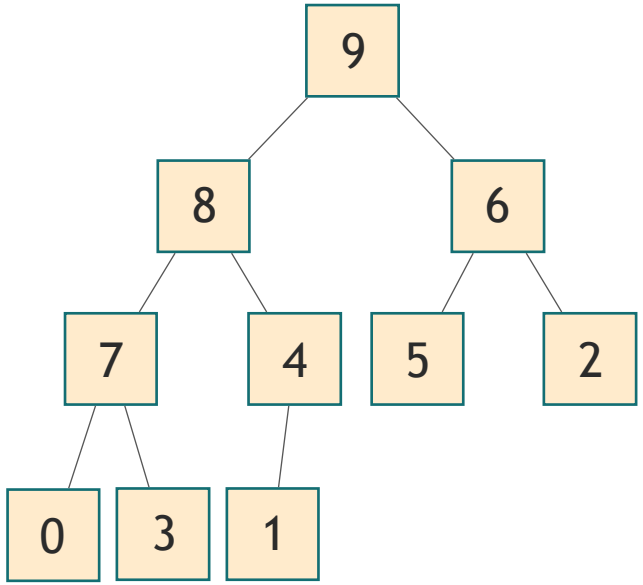
It's not just

for_each

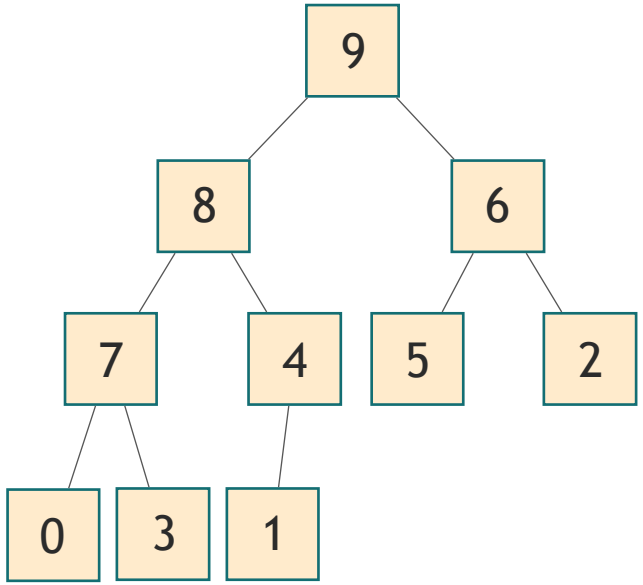

for_each


Fluent {C++}
The World of C++ STL Algorithms
Province of Value Queries
Province of Property Queries
Province of 2-Ranges Properties
Province of Research
Lands of Queries
Glorious County of Algos on Sets
Peninsula of Raw Memory
Territory of Movers
Island of Structure Changers
Land of Value Modifiers
Lands of Permutations
Lonely Islands
Secret Runes
Value Searchers
Relative Value Searchers
Valley of Partial Sums
Lagoon of Folds
Straits of Sorting
Province of Heaps
Bay of Range Searchers
count
all_of
any_of
none_of
transform_inclusive_scan
transform_exclusive_scan
inclusive_scan
exclusive_scan
partial_sum
transform_reduce
inner_product
accumulate
reduce
temple
adjacent_difference
find
adjacent_find
equal_range
binary_search
lower_bound
upper_bound
max_element
min_element
minmax_element
find_first_of
find_end
search
mismatch
lexicographical_compare
equal
is_permutation
set_difference
set_symmetric_difference
set_intersection
set_union
includes
merge
for_each
transform
uninitialized_value_construct
uninitialized_default_construct
uninitialized_copy
uninitialized_fill
uninitialized_move
destroy
copy_backward
copy
move
move_backward
swap_ranges
fill
generate
remove
unique
replace
partition_point
partition
sort
partial_sort
sort_heap
nth_element
inplace_merge
make_heap
pop_heap
push_heap
next_permutation
prev_permutation
shuffle
random_shuffle
reverse
rotate

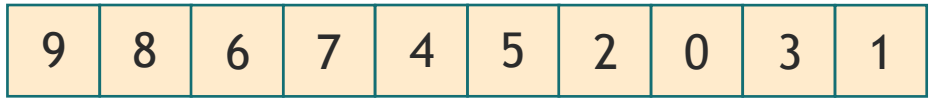
HEAPS



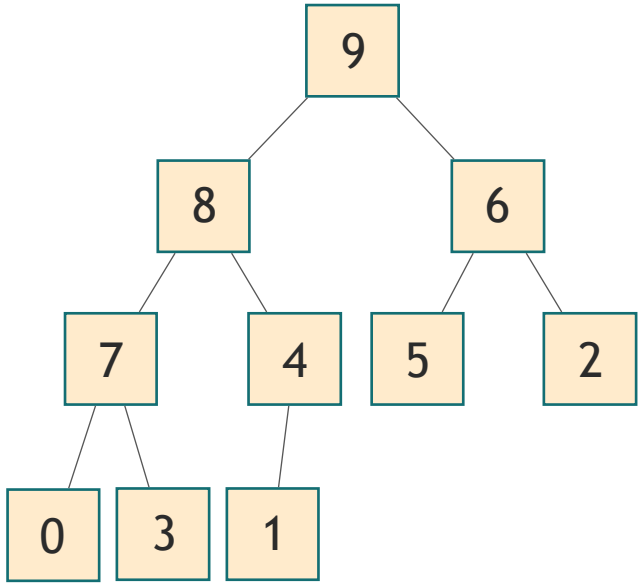
HEAPS



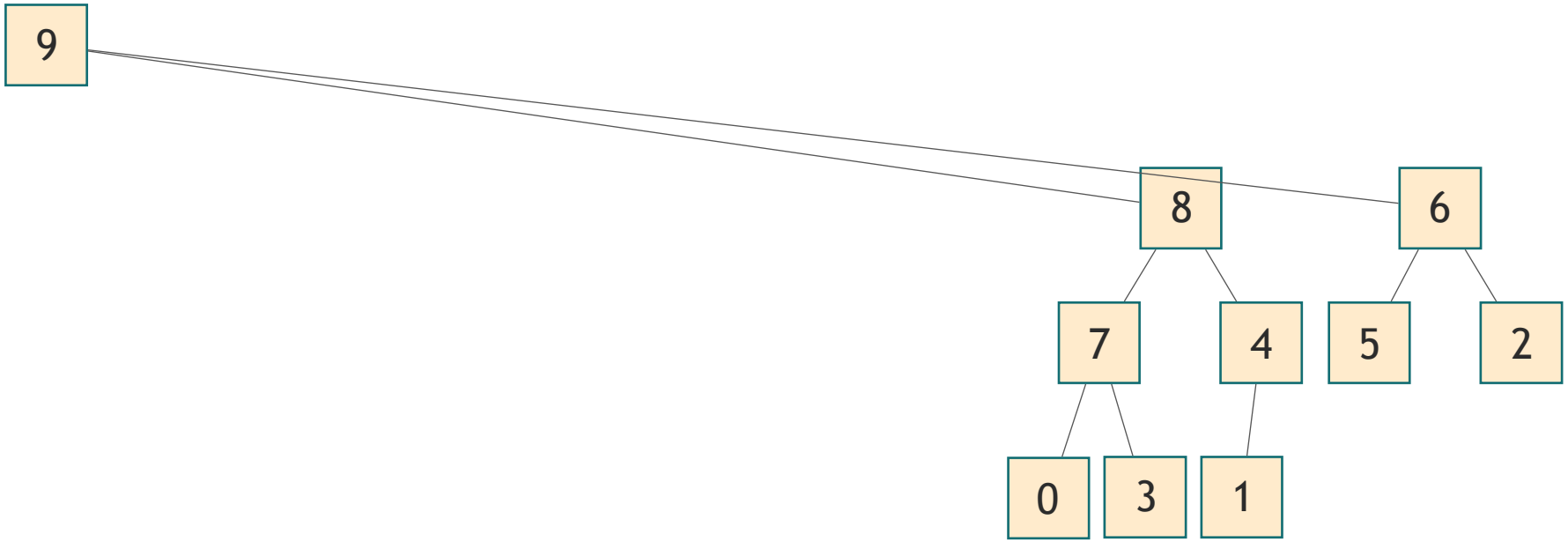
HEAPS



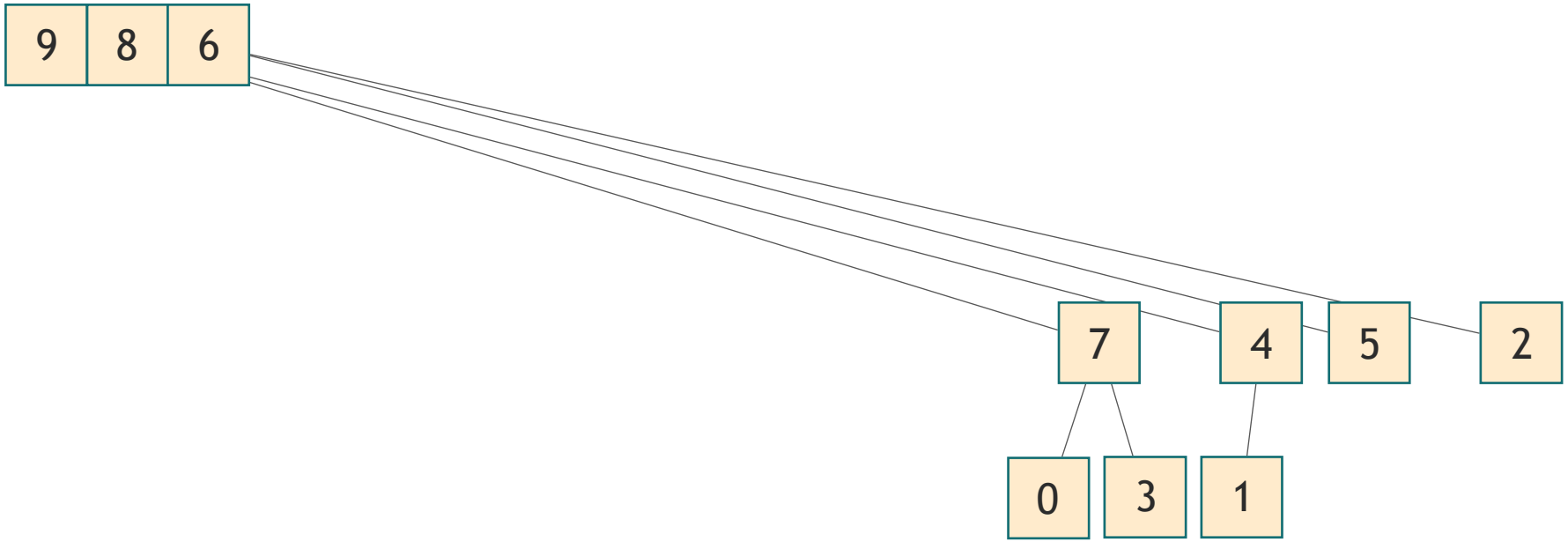
HEAPS



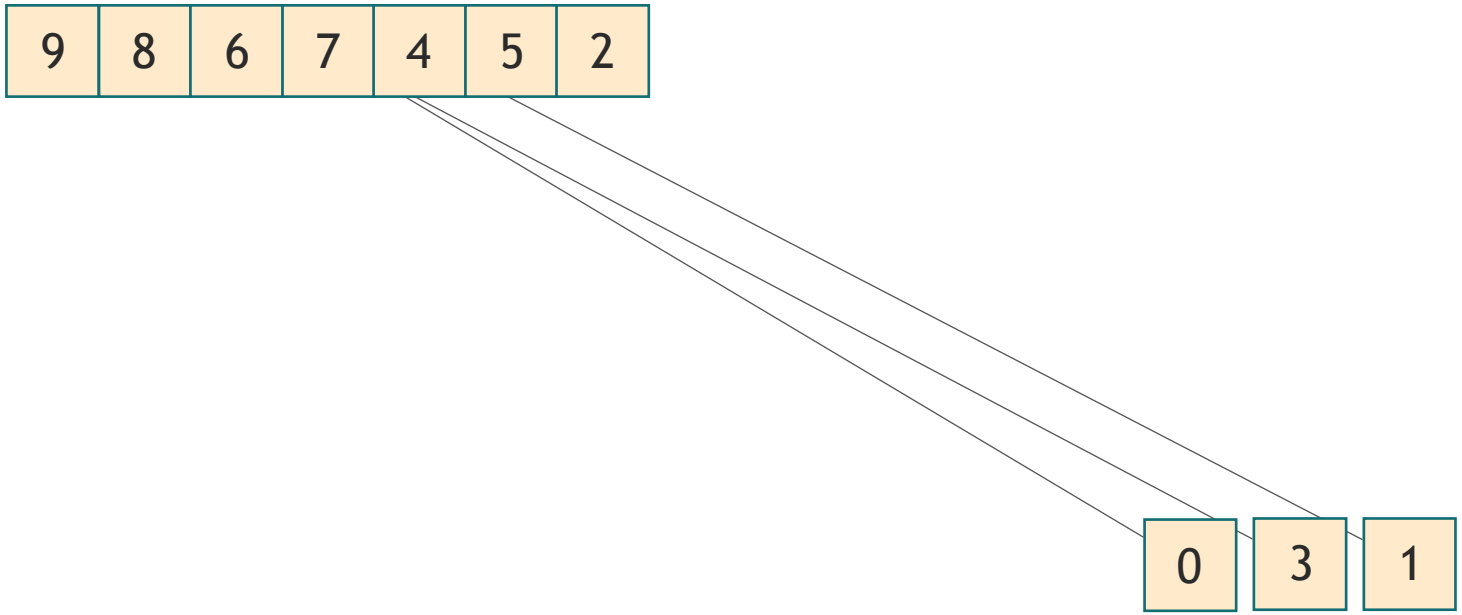
HEAPS



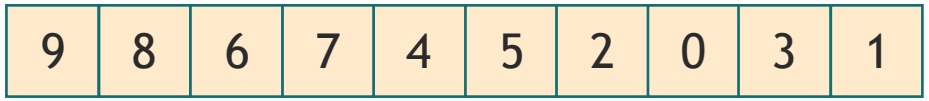
HEAPS



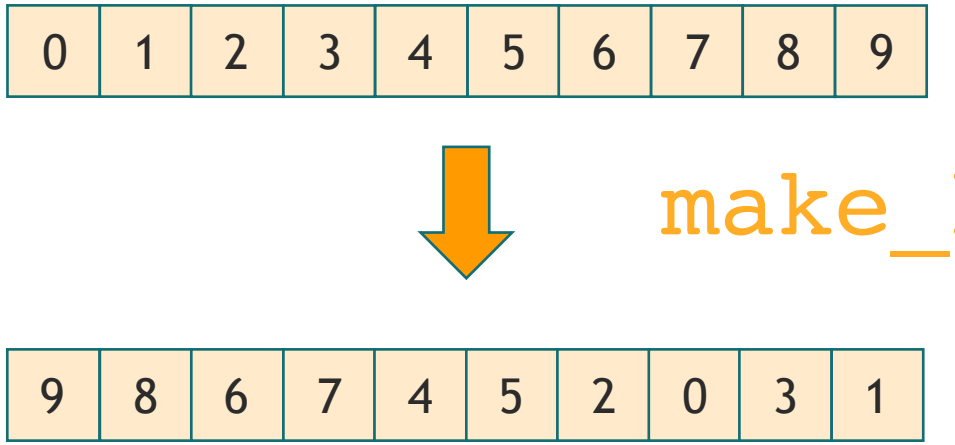
HEAPS



HEAPS

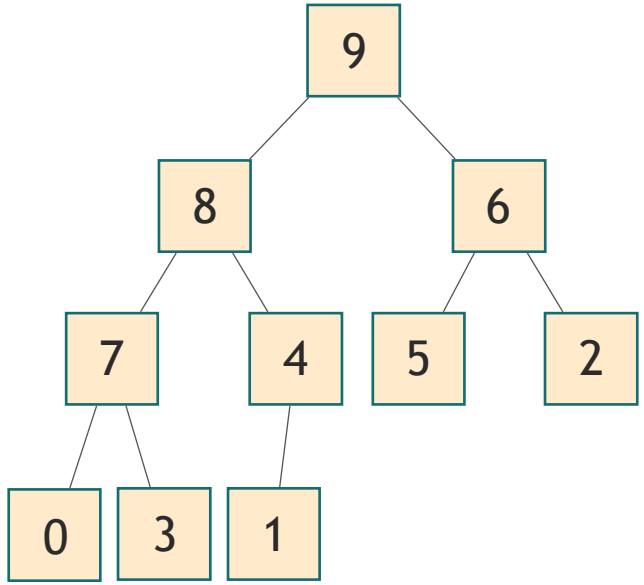


HEAPS

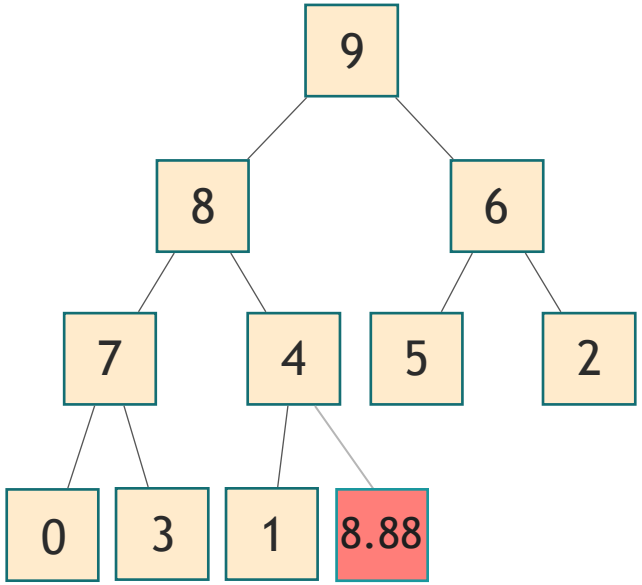



```
std::make_heap(begin(numbers), end(numbers));
```

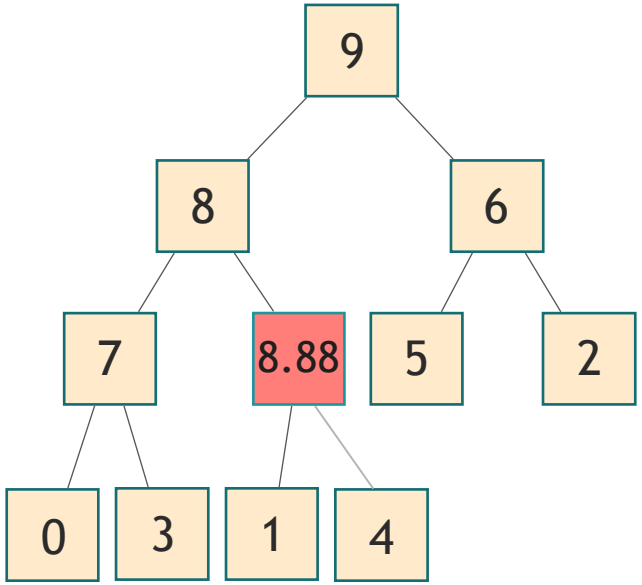
HEAPS



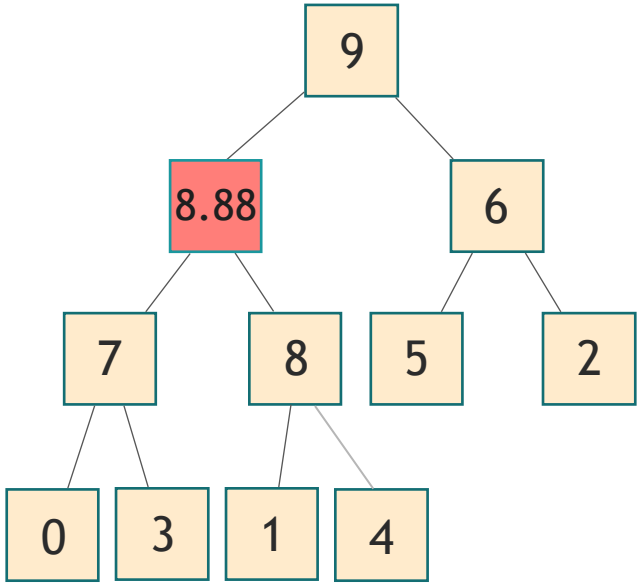
HEAPS



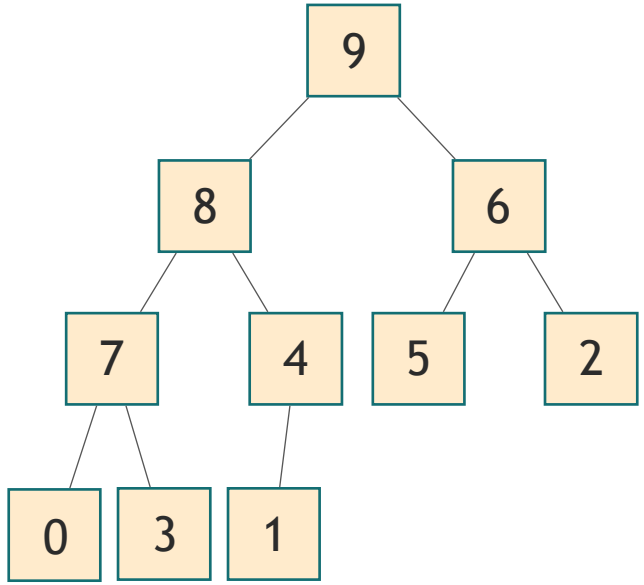
HEAPS



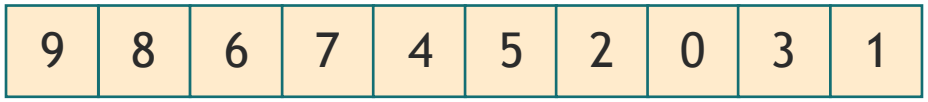
HEAPS



HEAPS



HEAPS



HEAPS

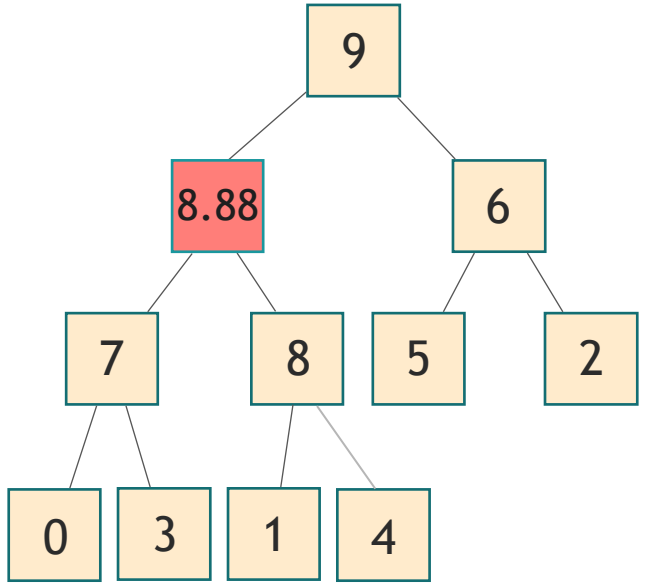

```
numbers.push_back(8.88);

std::push_heap(begin(numbers), end(numbers));
```

HEAPS

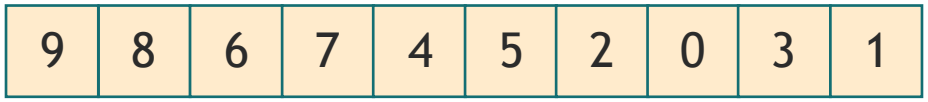

```
std::push_heap(begin(numbers), end(numbers));
```

HEAPS

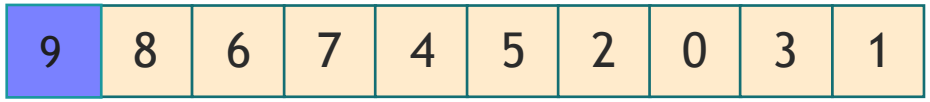



```
std::push_heap(begin(numbers), end(numbers));
```

HEAPS

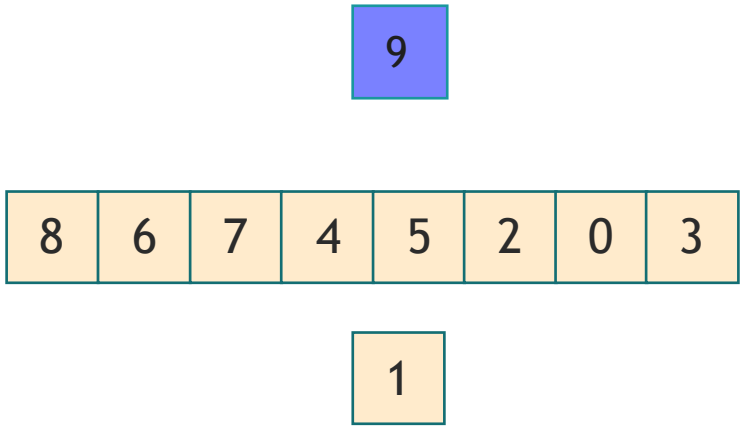


HEAPS



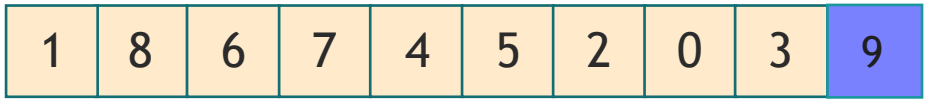

```
std::pop_heap(begin(numbers), end(numbers));
```

HEAPS



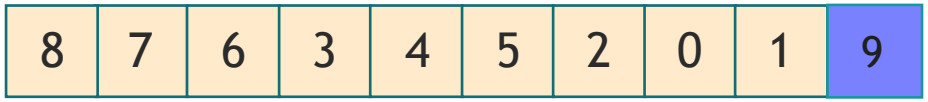

```
std::pop_heap(begin(numbers), end(numbers));
```

HEAPS




```
std::pop_heap(begin(numbers), end(numbers));
```

HEAPS




```
std::pop_heap(begin(numbers), end(numbers));
```

HEAPS


```
std::pop_heap(begin(numbers), end(numbers));

numbers.pop_back();
```

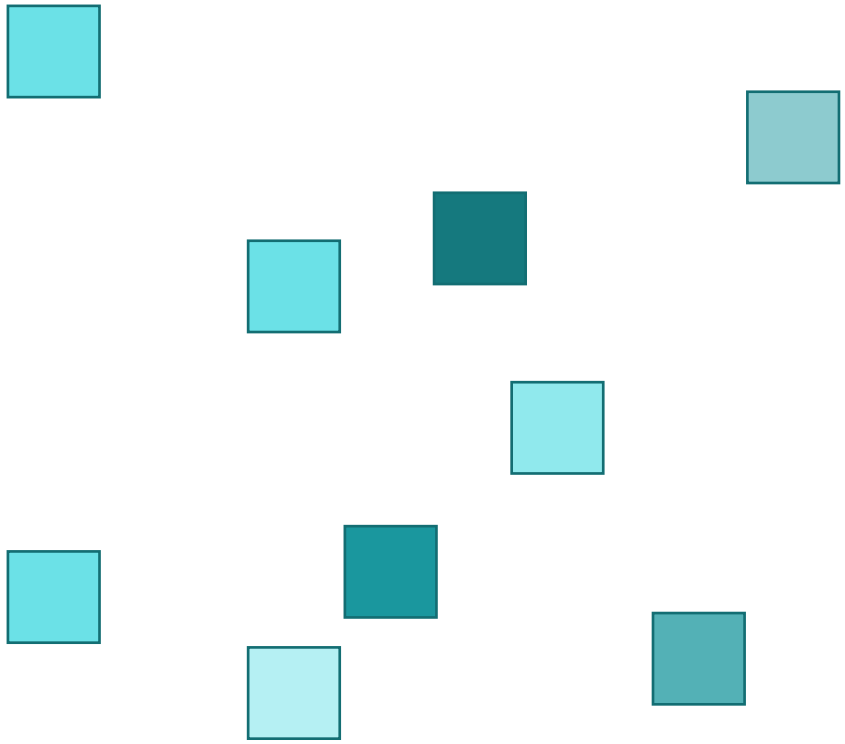

NS
make_heap
inplac
tation
pop_heap
push_heap
PROVINCE OF HEAPS

SORTING

sort

SORTING

sort



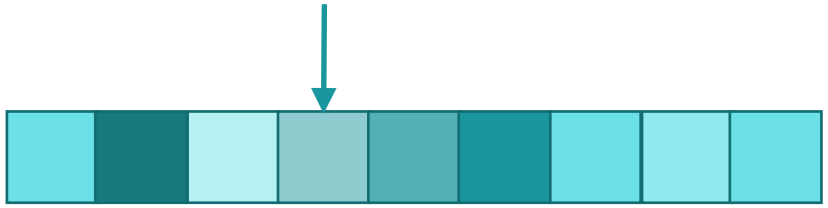
SORTING

sort

SORTING

sort

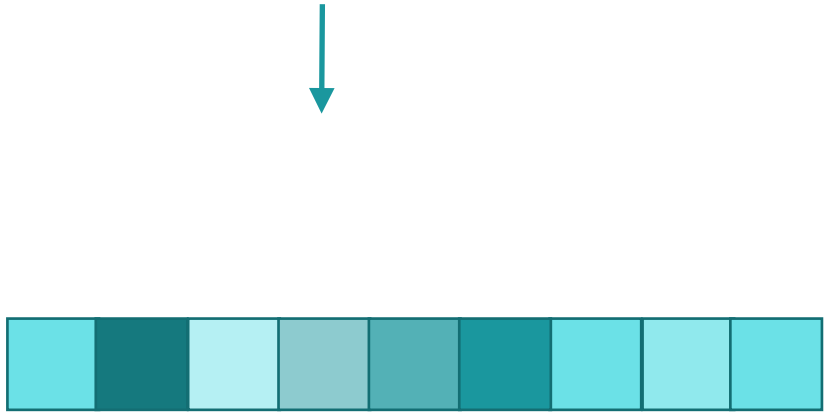
partial_sort



SORTING

sort

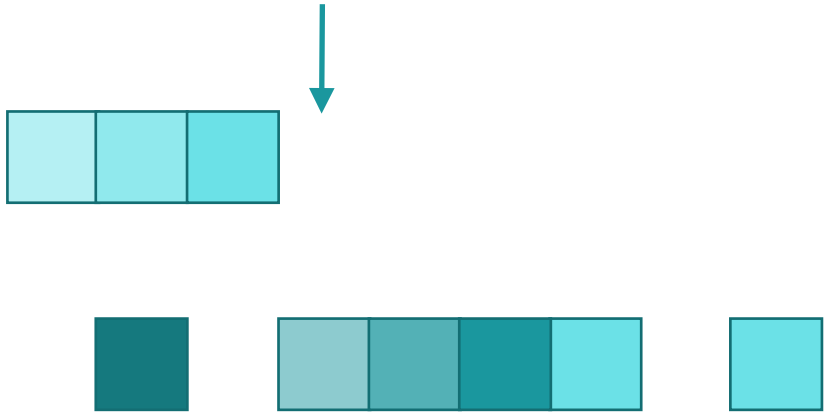
partial_sort



SORTING

sort

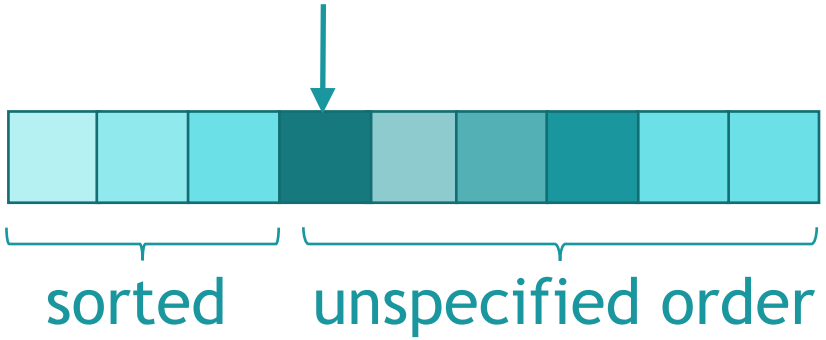
partial_sort



SORTING

sort

partial_sort

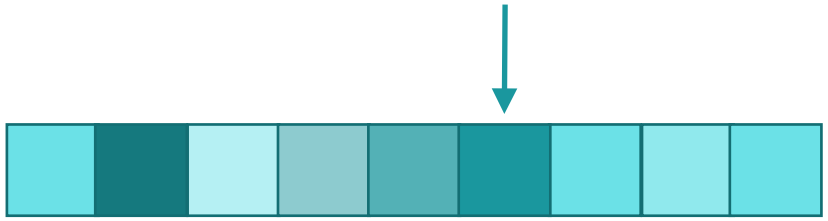


SORTING

sort

partial_sort

nth_element

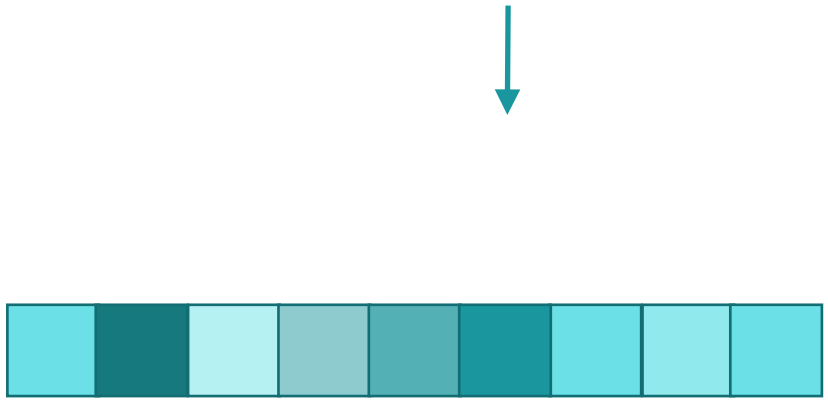


SORTING

sort

partial_sort

nth_element

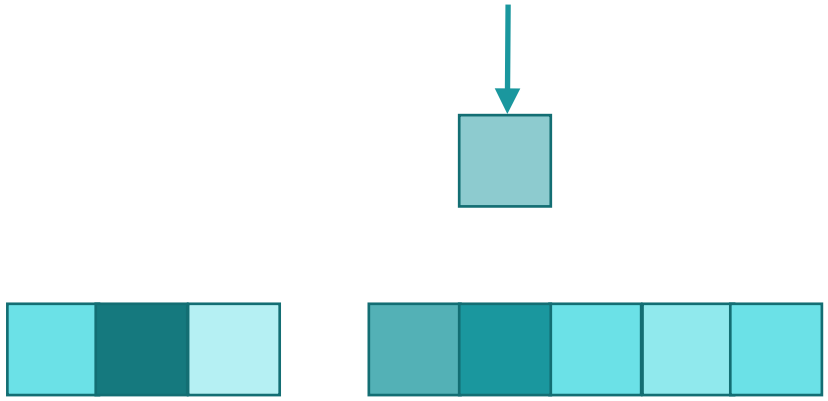


SORTING

sort

partial_sort

nth_element

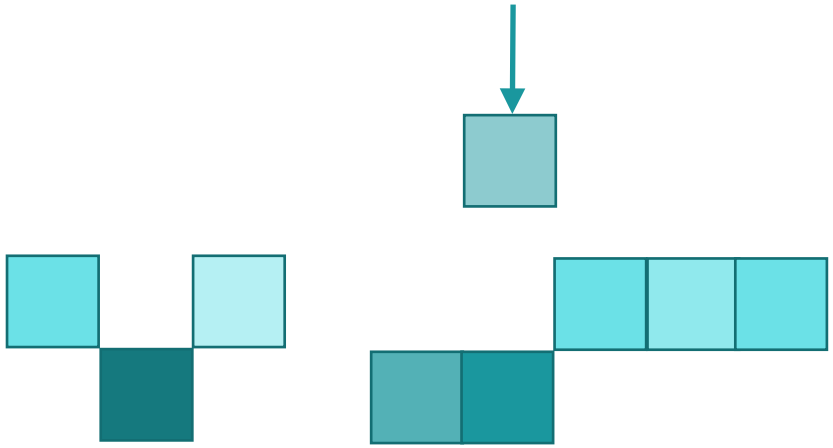


SORTING

sort

partial_sort

nth_element

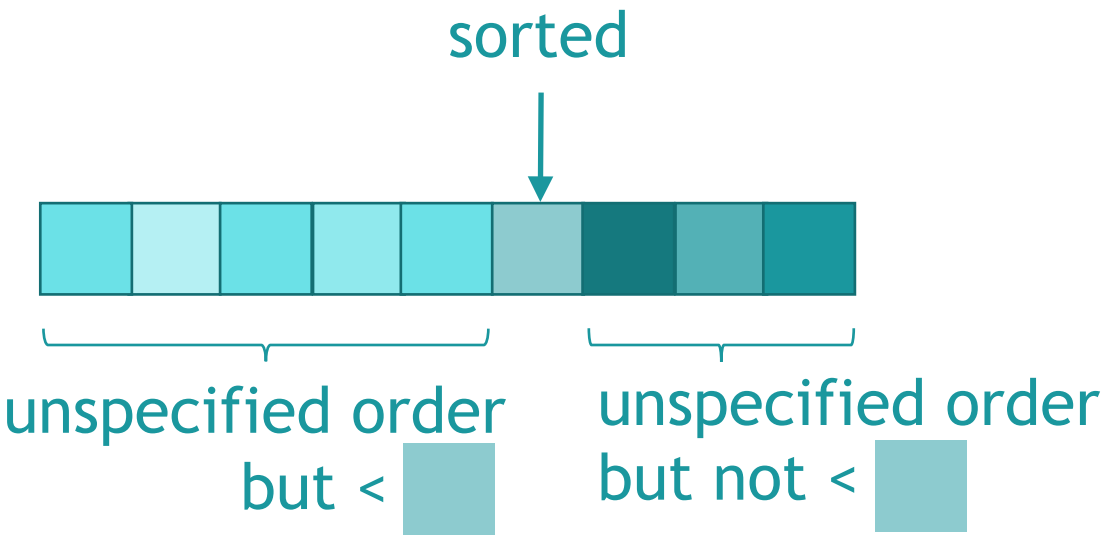


SORTING

sort

partial_sort

nth_element



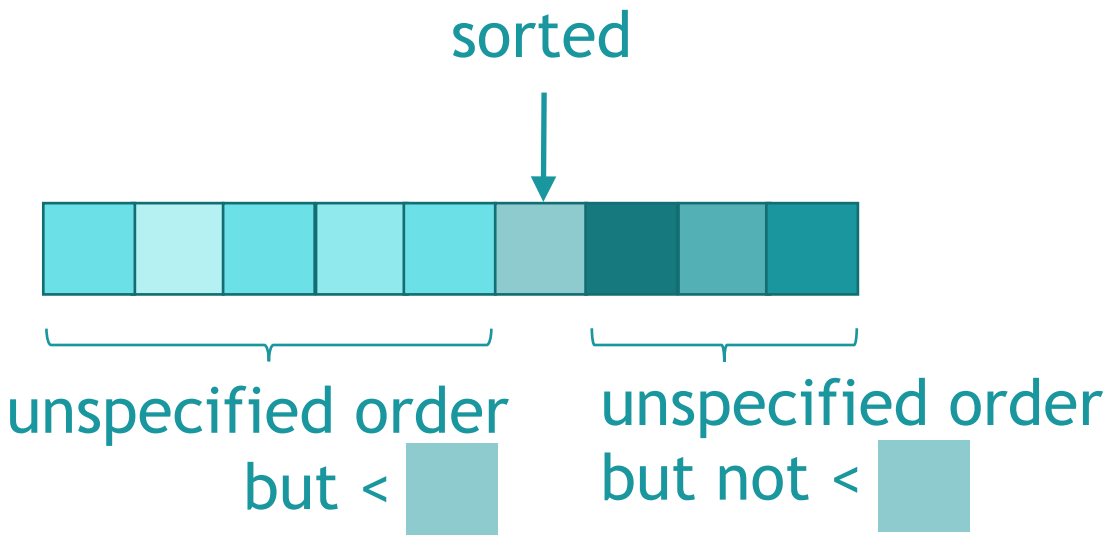
SORTING

sort

partial_sort

nth_element

sort_heap



SORTING

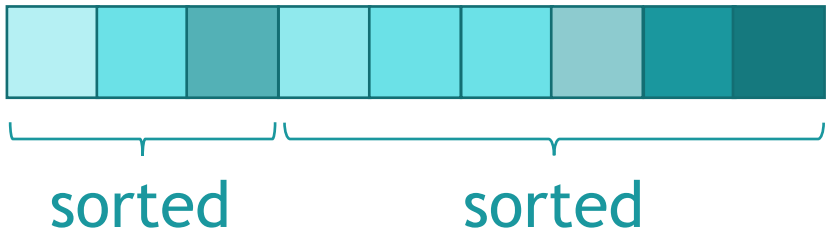
sort

partial_sort

nth_element

sort_heap

inplace_merge



SORTING

sort

partial_sort

nth_element

sort_heap

inplace_merge

SORTING

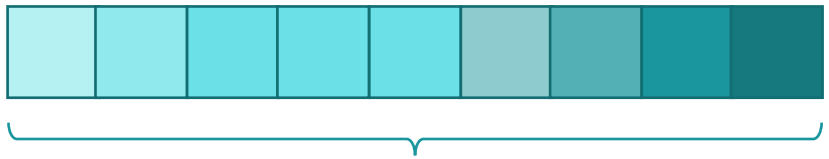
sort

partial_sort

nth_element

sort_heap

inplace_merge

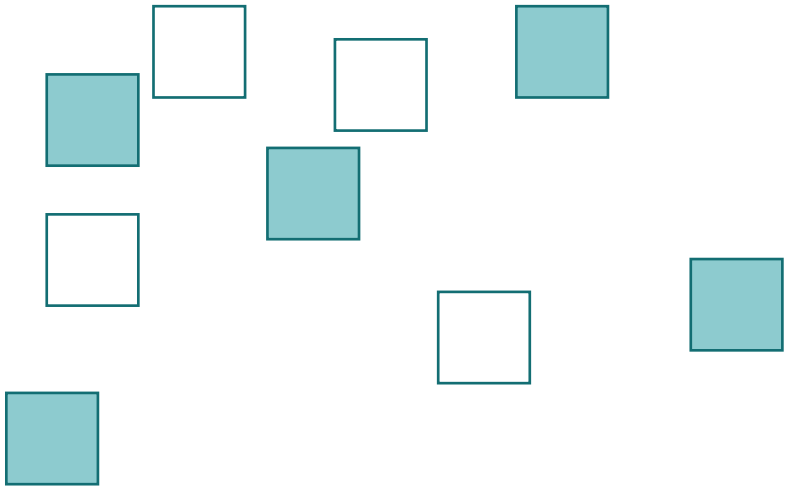



SHORE OF SORTING
sort
partial sort
on
sort_heap
nth_element
inplace_merge
ke_heap
pop_heap

PARTITIONING

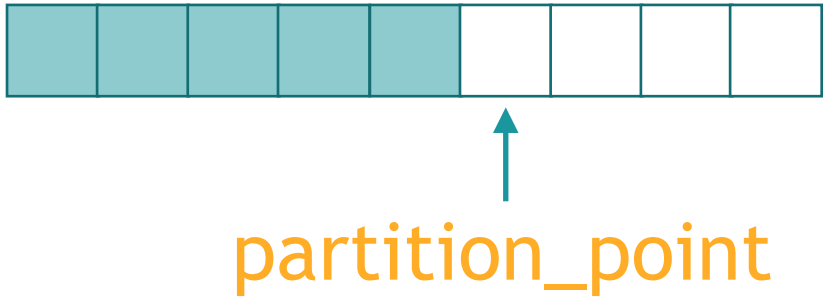
partition

PARTITIONING



partition

PARTITIONING



partition

OTHER PERMUTATIONS

rotate

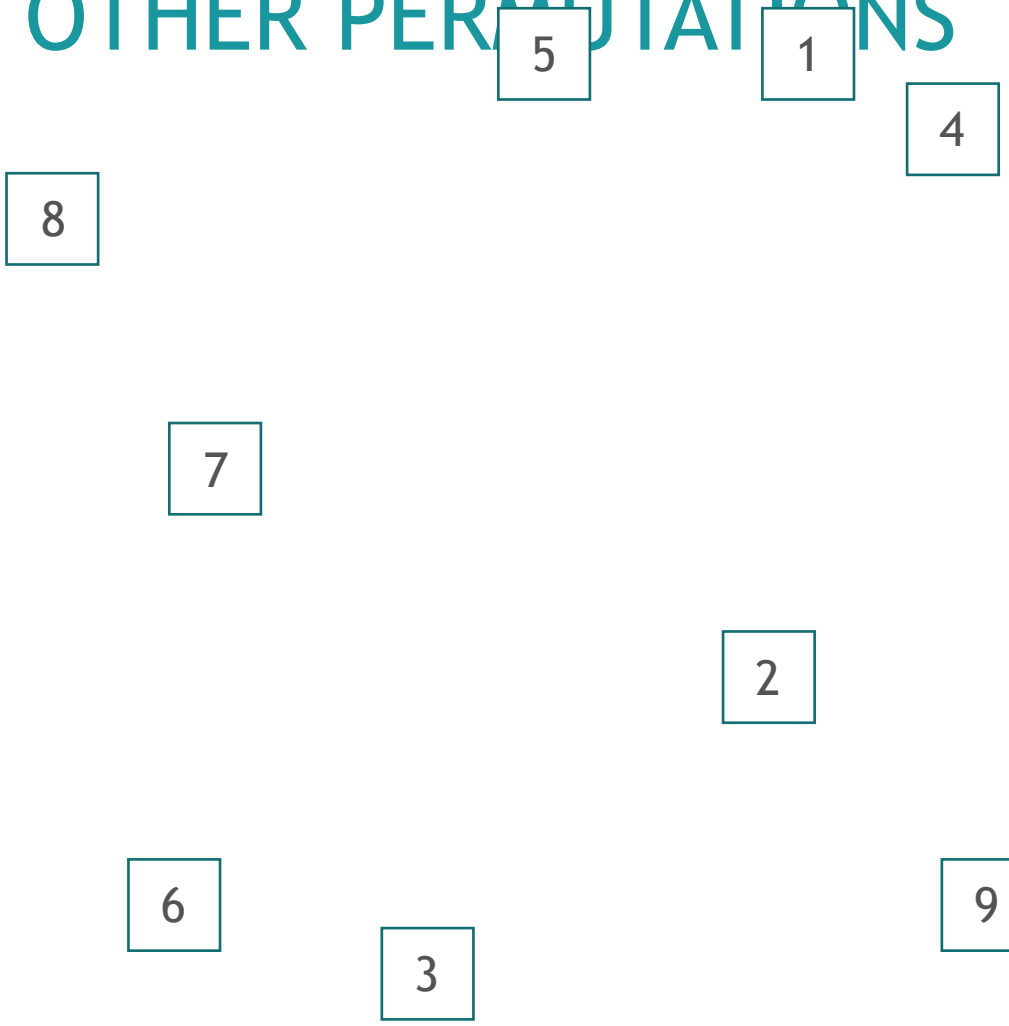
2	1	5	2	7	10	2	5	6

OTHER PERMUTATIONS

rotate

shuffle

OTHER PERMUTATIONS



rotate

5	6	2	1	5	2	7	10	2

shuffle

OTHER PERMUTATIONS

rotate

5	6	2	1	5	2	7	10	2

9	5	8	1	3	7	6	4	2

shuffle

OTHER PERMUTATIONS

rotate

5	6	2	1	5	2	7	10	2

9	5	8	1	3	7	6	4	2

shuffle

reverse

1	2	3	4	5	6	7	8	9

next_permutation

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	9	8
1	2	3	4	5	6	8	7	9
...								
9	8	7	6	5	4	3	2	1

prev_permutation

OTHER PERMUTATIONS

rotate

5	6	2	1	5	2	7	10	2

9	5	8	1	3	7	6	4	2

shuffle

reverse

9	8	7	6	5	4	3	2	1

next_permutation

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	9	8
1	2	3	4	5	6	8	7	9
...								
9	8	7	6	5	4	3	2	1

prev_permutation

PARTITIONING-SORT-HEAP

stable_*

stable_sort
stable_partition

is_*

is_sorted
is_partitioned
is_heap

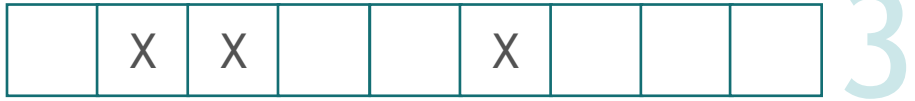
is_*_until

is_sorted_until
is_partitioned_until
is_heap_until

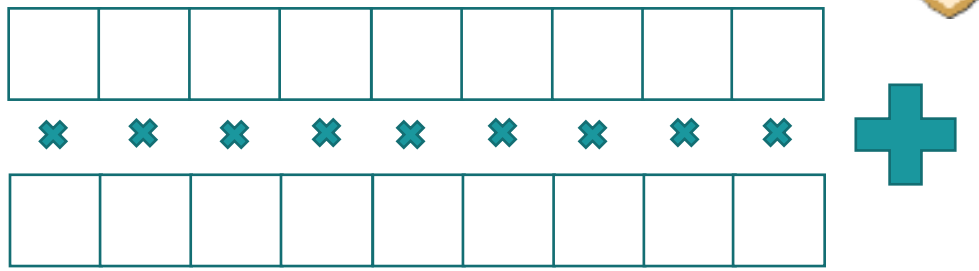

Fluent {C++}
Province of Value Queries
Province of Property Queries
Province of 2-Ranges Properties
Glorious County of Algos on Sets
Lonely Islands
Lands of Queries
Peninsula of Raw Memory
Territory of Movers
Province of Research
Island of Structure Changers
Land of Value Modifiers
Lands of Permutations
Secret Runes
The World of C++ STL Algorithms

NUMERIC ALGORITHMS

count

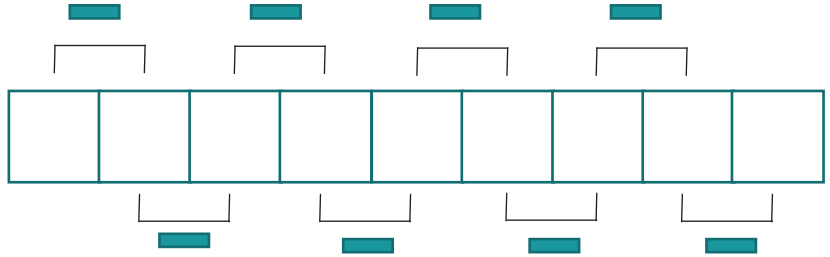


inner_product

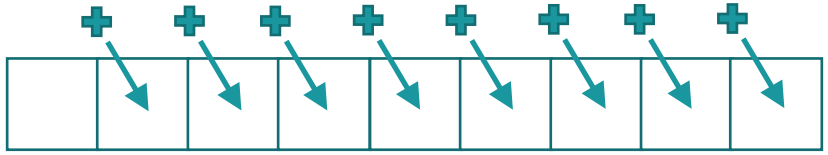


accumulate/(transform_)reduce

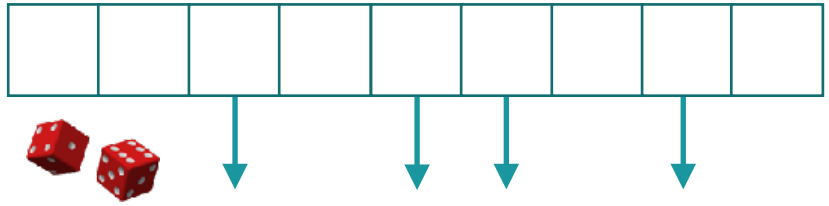
adjacent_difference



partial_sum
(transform_)inclusive_scan
(transform_)exclusive_scan



sample



@JoBoccara

QUERYING A PROPERTY

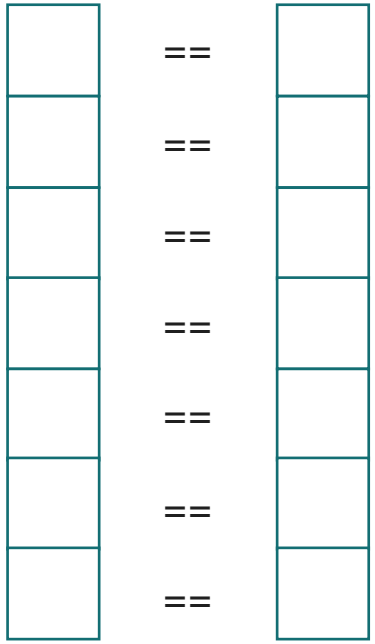
all_of true

any_of false

none_of true

QUERYING A PROPERTY ON 2 RANGES

equal

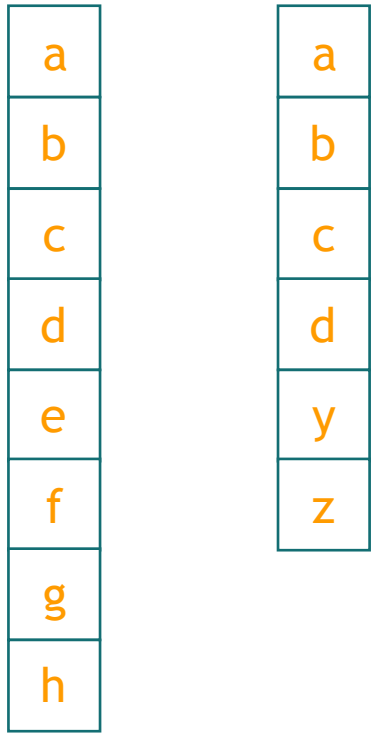


&& same size

→ bool

is_permutation → bool

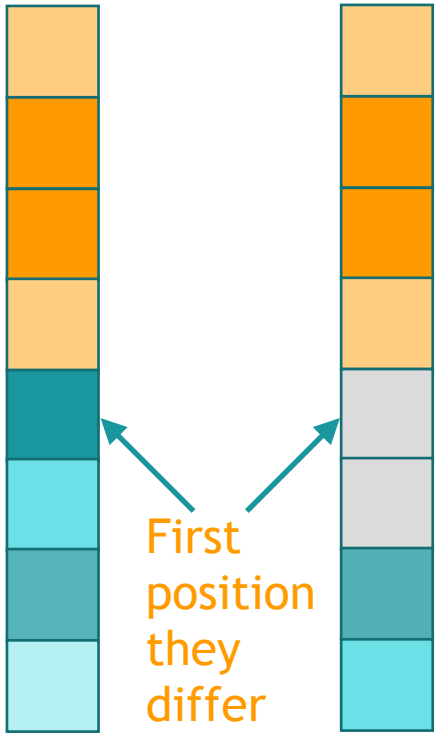
lexicographical_compare



First one smaller?

→ bool

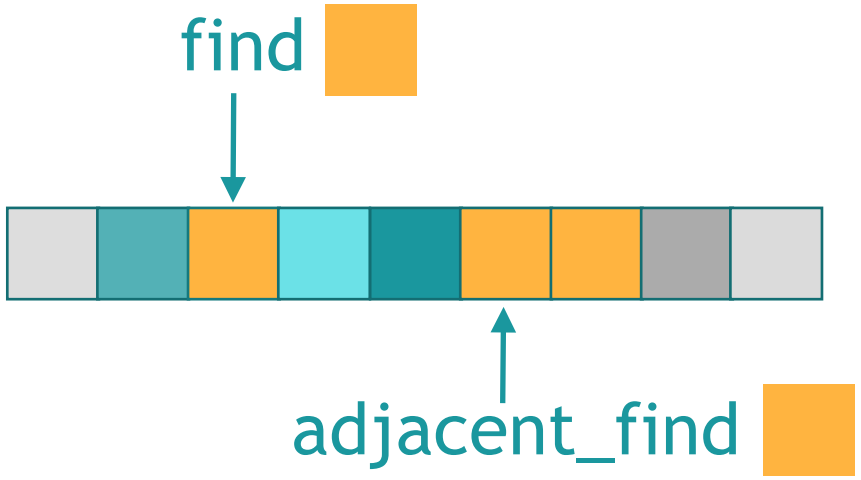
mismatch



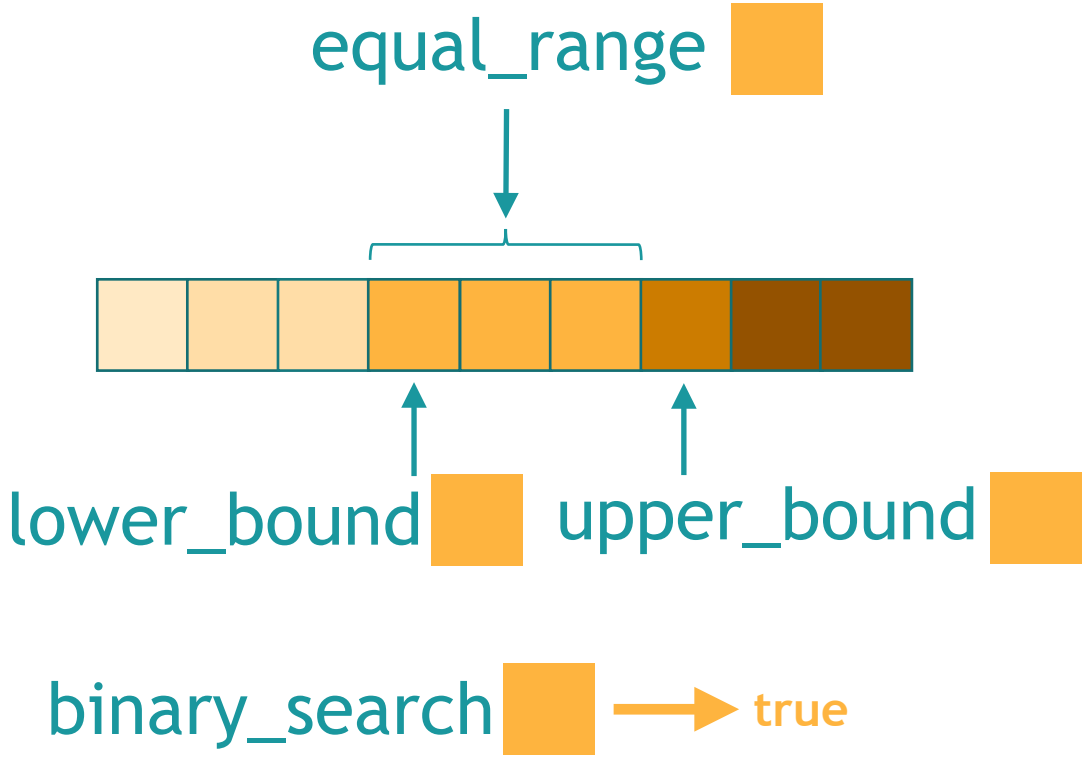
→ std::pair<Iterator, Iterator>

SEARCHING A VALUE

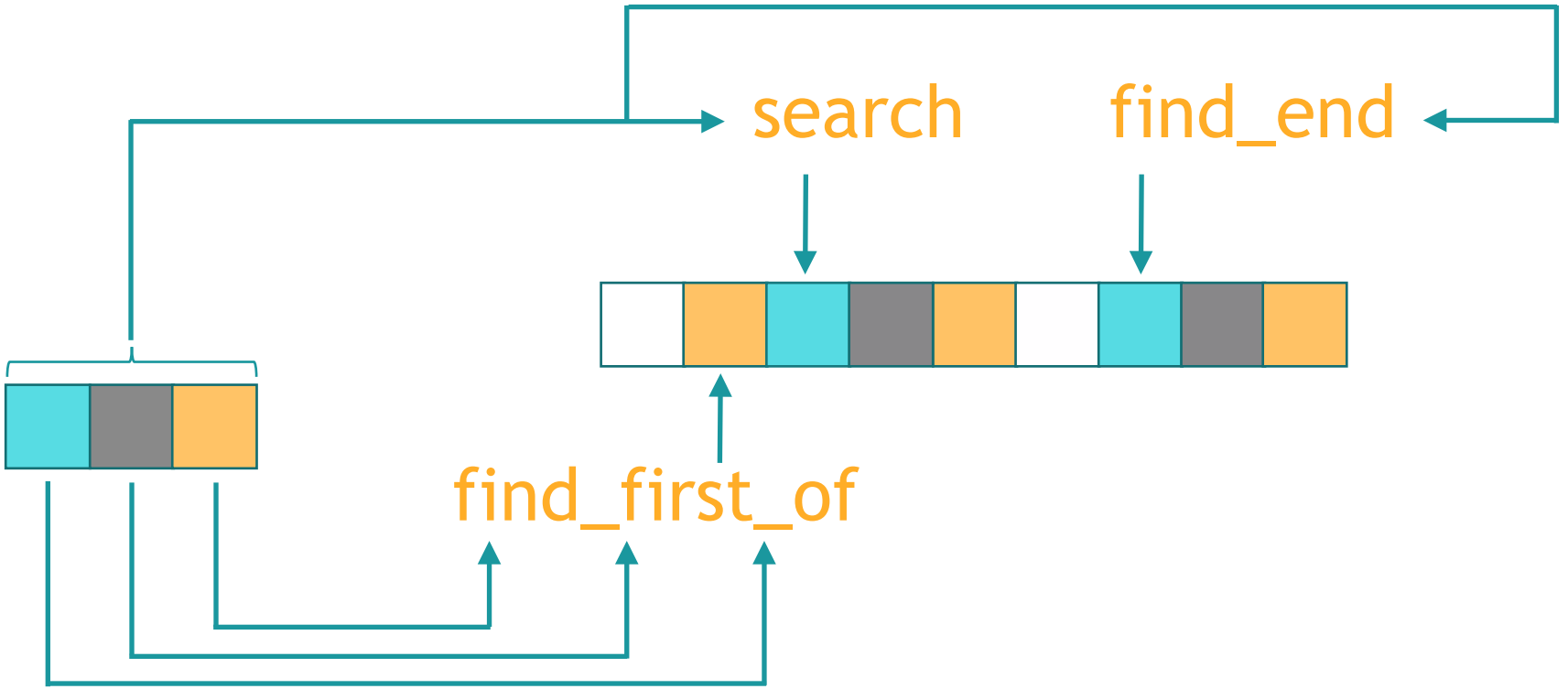
NOT SORTED



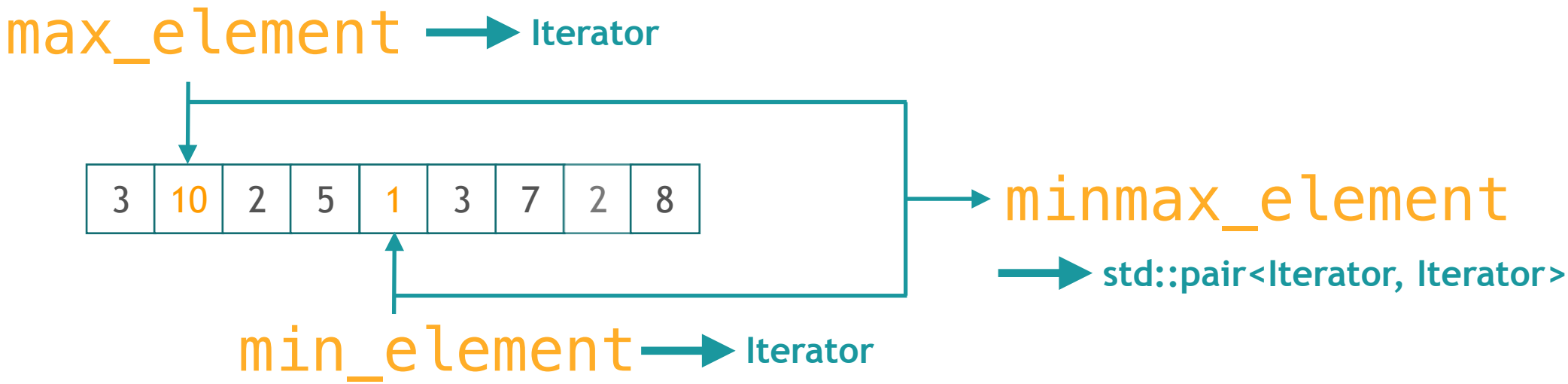
SORTED



SEARCHING A RANGE

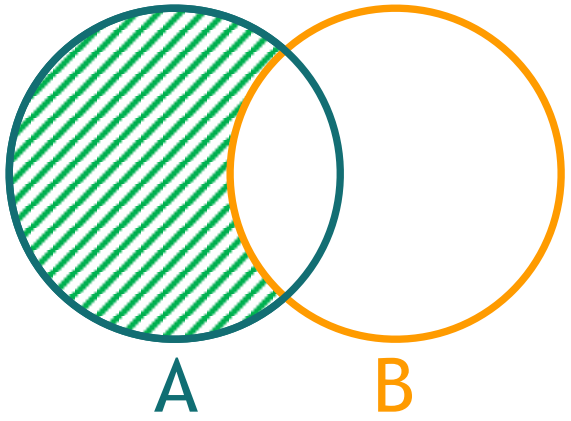


SEARCHING A RELATIVE VALUE



ALGOS ON SETS

A: sorted B: sorted

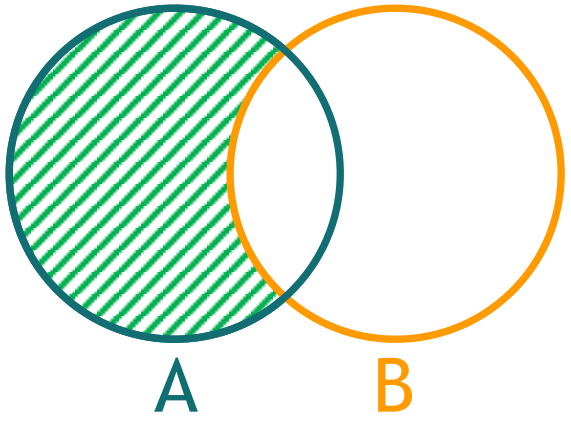


`std::set_difference`

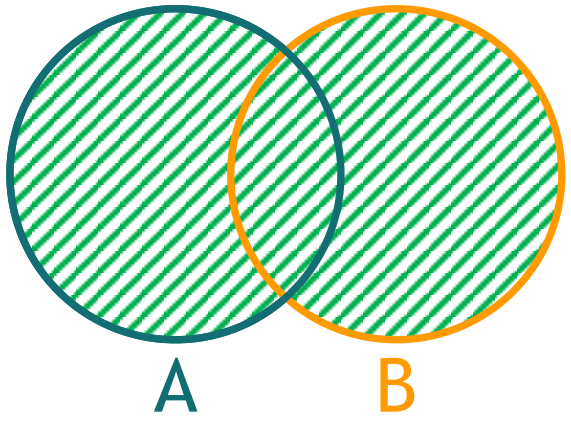
```
std::set_difference(begin(A), end(A);
                    begin(B), end(B),
                    std::back_inserter(results));
```

ALGOS ON SETS

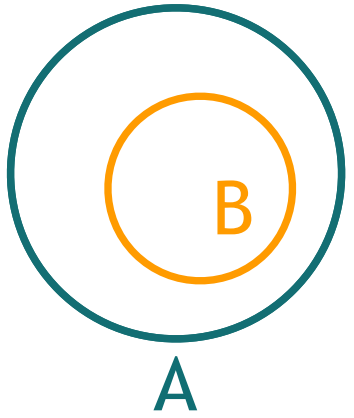
A: sorted B: sorted



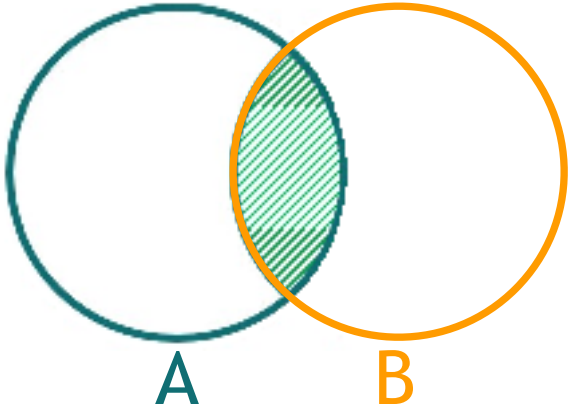
`std::set_difference`



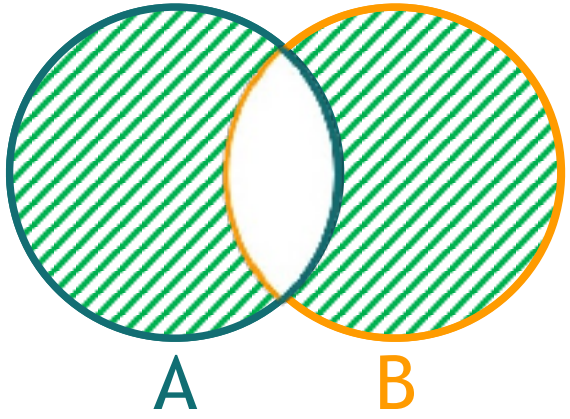
`std::set_union`



`std::includes`



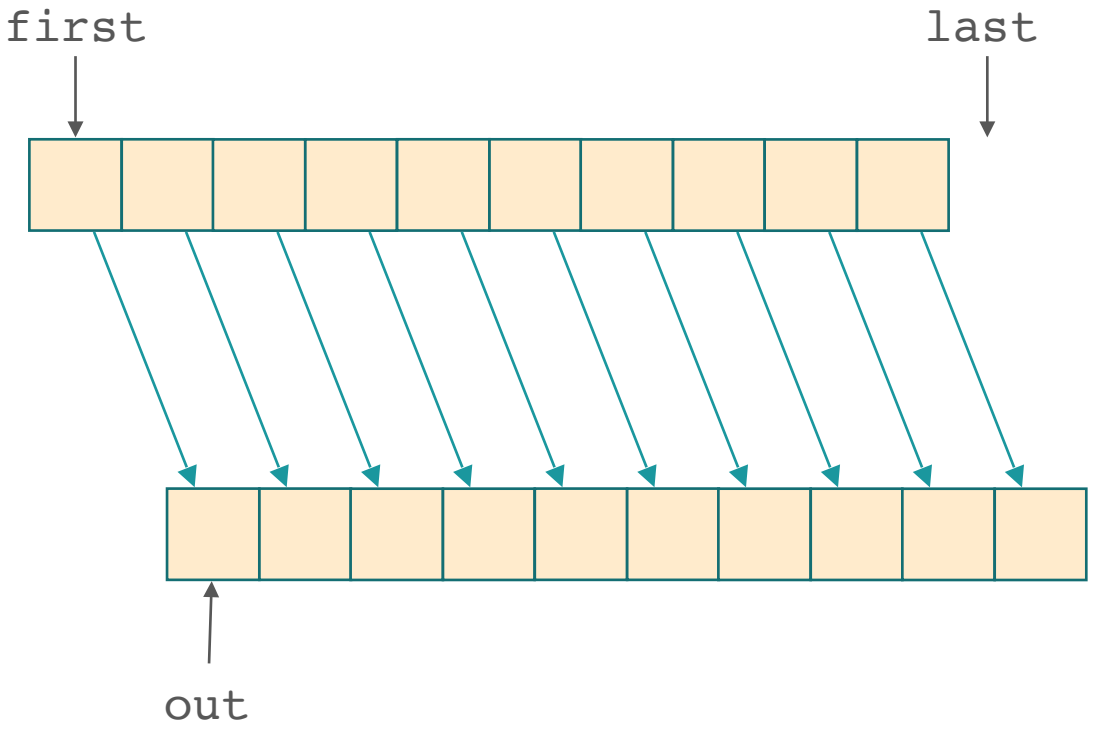
`std::set_intersection`



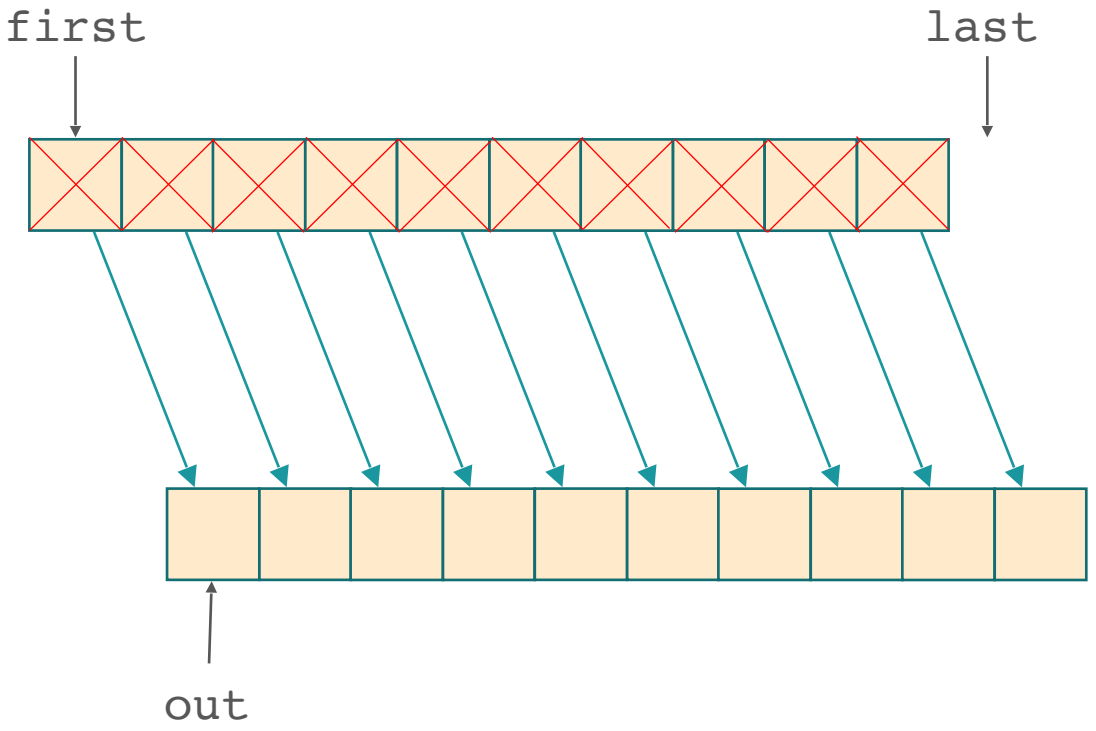
`std::set_symmetric_difference`

`std::merge`

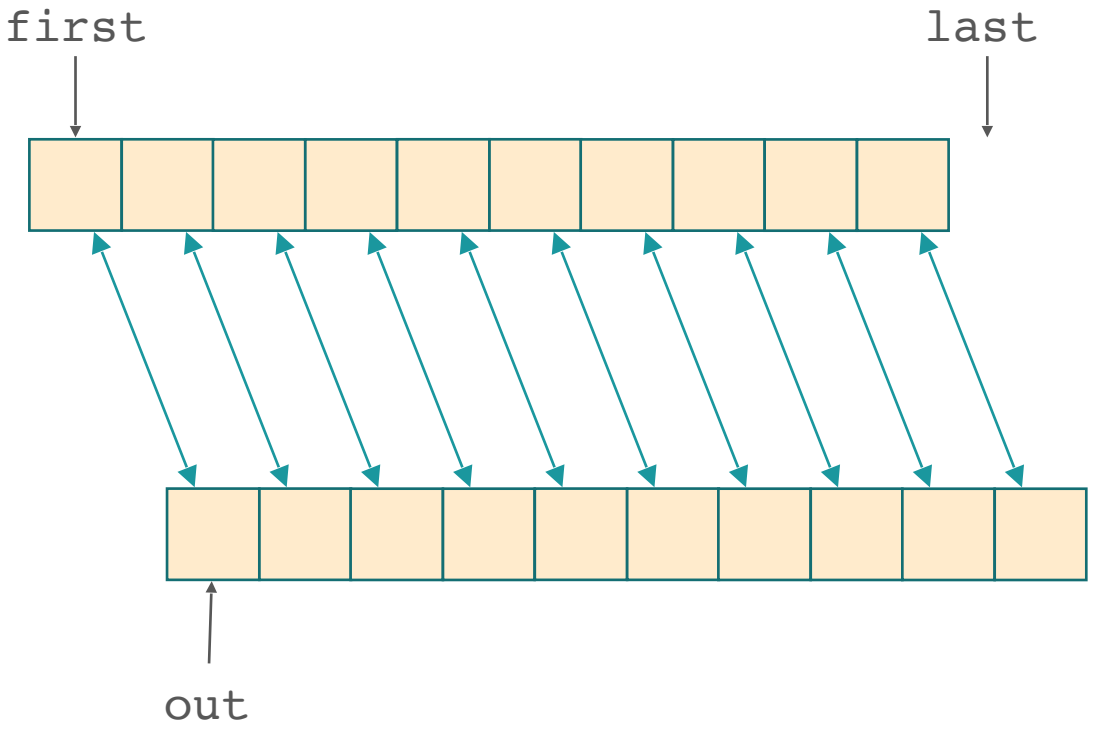

Fluent {C++}
Province of Value Queries
Province of Property Queries
Province of 2-ranges Properties
Province of Research
Glorious County of Algos on Sets
Lonely Islands
Lands of Queries
Peninsula of Raw Memory
Territory of Movers
Island of Structure Changers
Land of Value Modifiers
Lands of Permutations
Secret Runes
Value Searchers
Relative Value Searchers
The World of C++ STL Algorithms



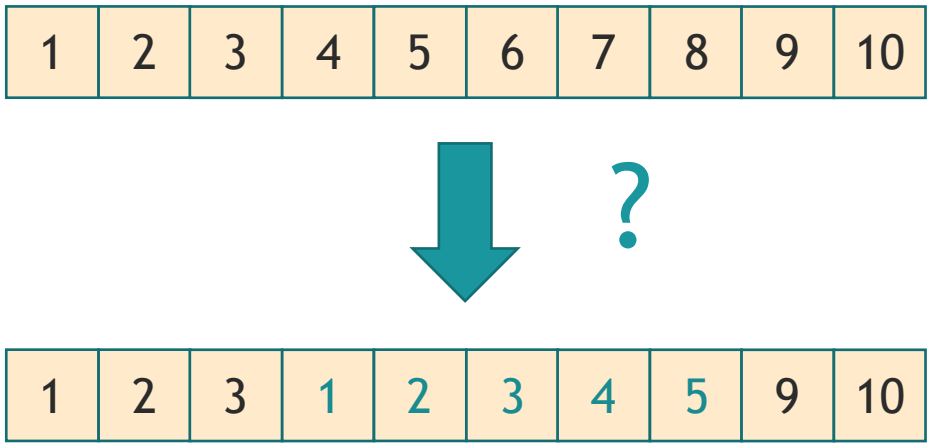

```
std::copy(first, last, out);
```

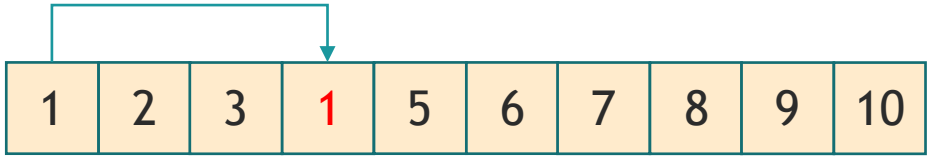


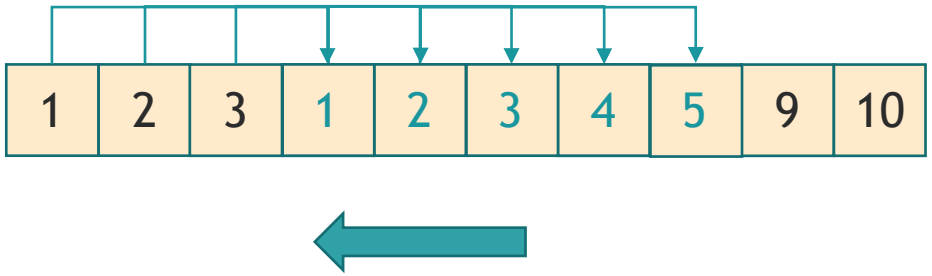
```
std::move(first, last, out);
```




```
std::swap_ranges(first, last, out);
```


1 2 3 4 5 6 7 8 9 10
?
1 2 3 1 2 3 4 5 9 10

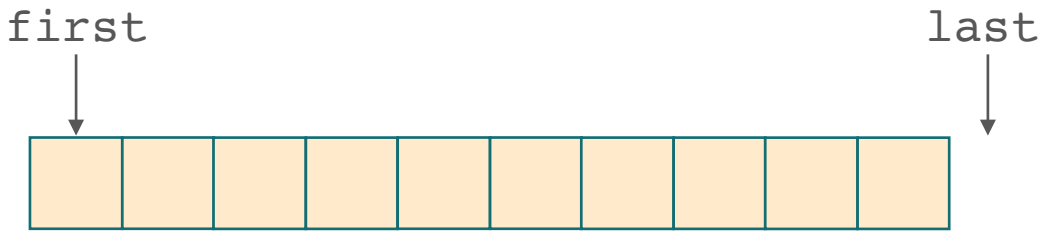

1
2
3
1
5
6
7
8
9
10



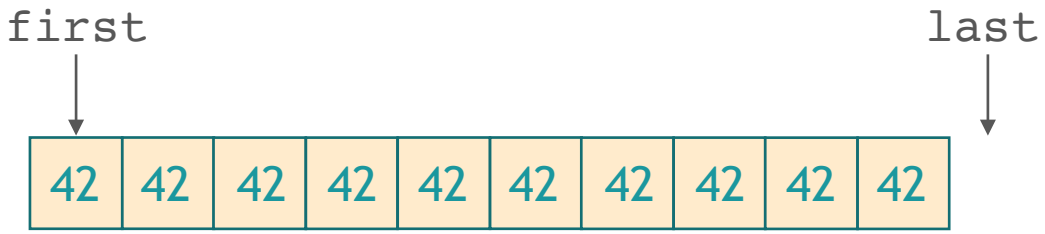
`std::copy_backward`

`std::move_backward`

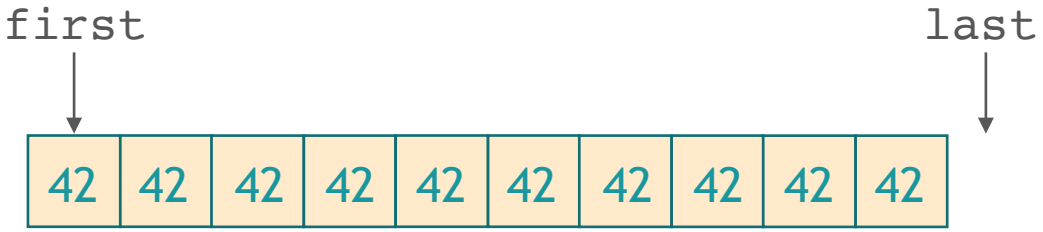
```
std::fill(first, last, 42);
```



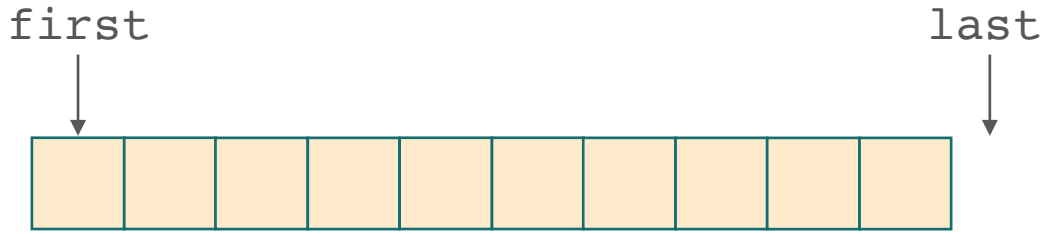
```
std::fill(first, last, 42);
```



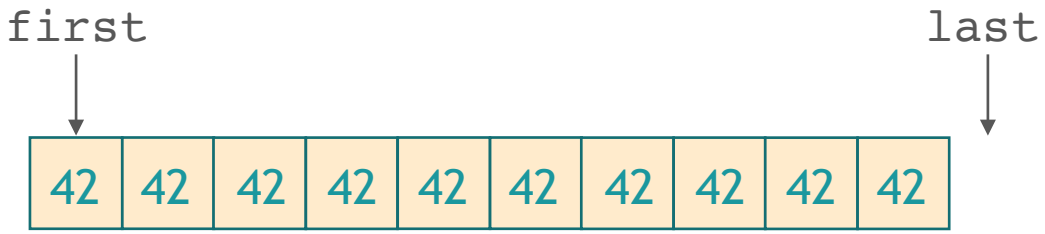
```
std::fill(first, last, 42);
```



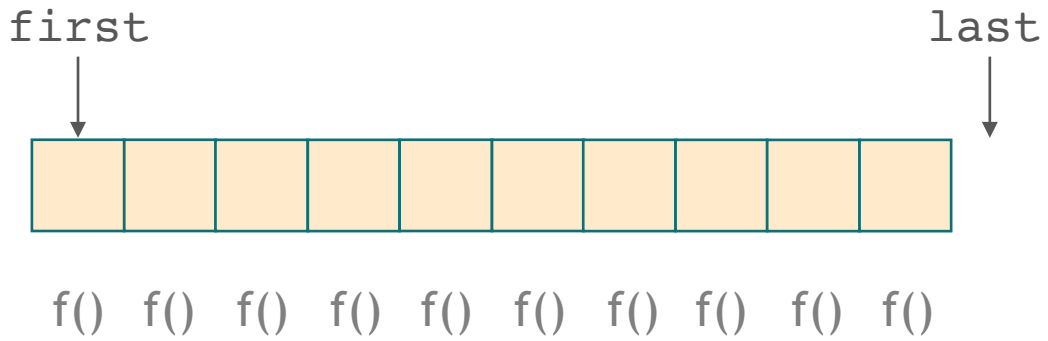

```
std::generate(first, last, f);
```



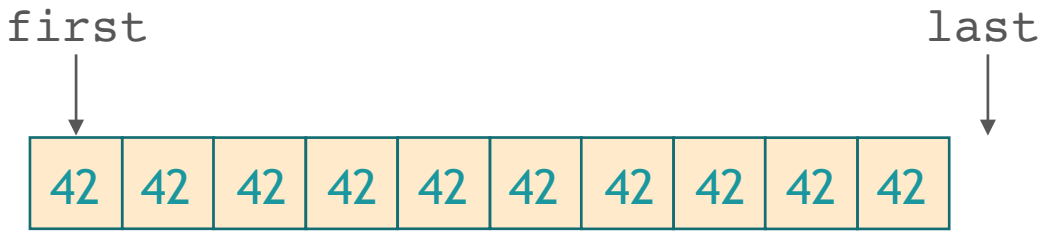
```
std::fill(first, last, 42);
```



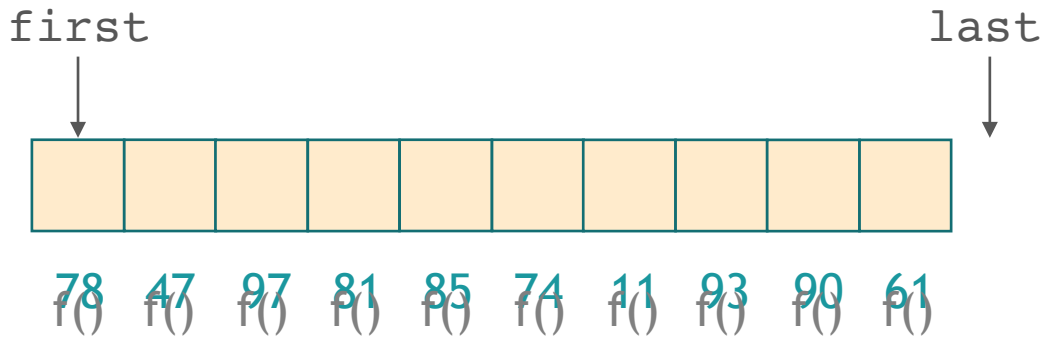

```
std::generate(first, last, f);
```



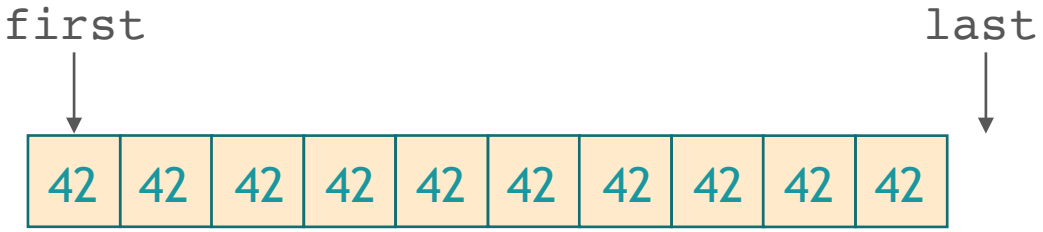
```
std::fill(first, last, 42);
```



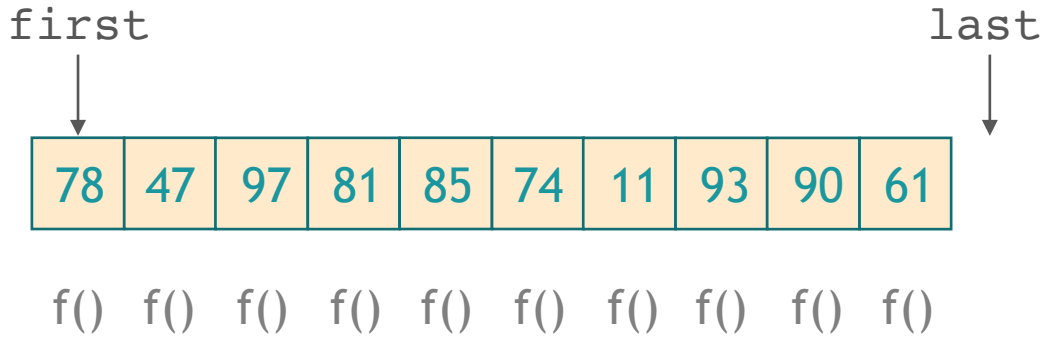

```
std::generate(first, last, f);
```



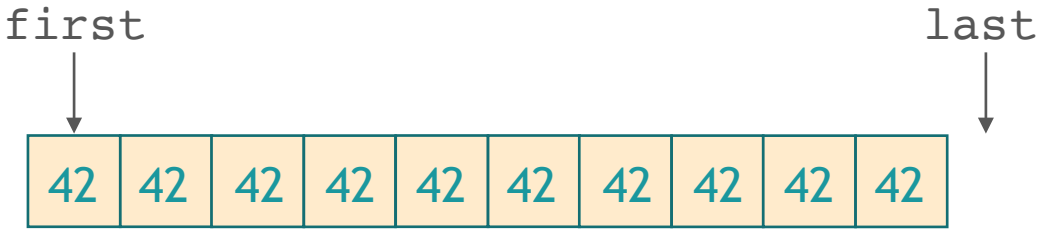
```
std::fill(first, last, 42);
```



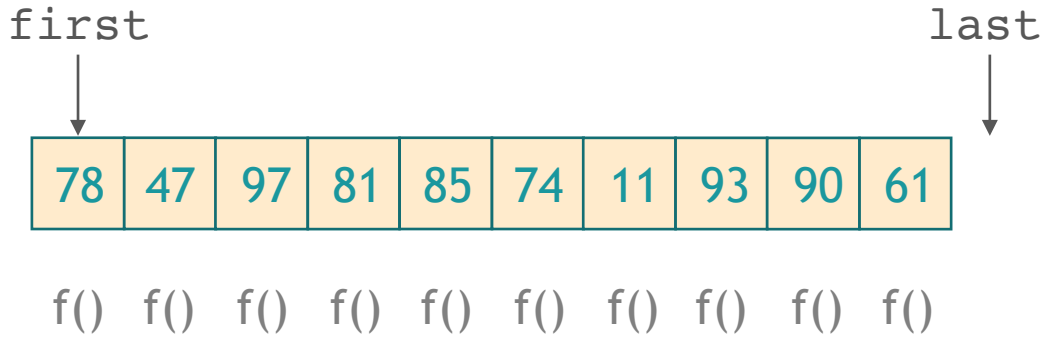

```
std::generate(first, last, f);
```



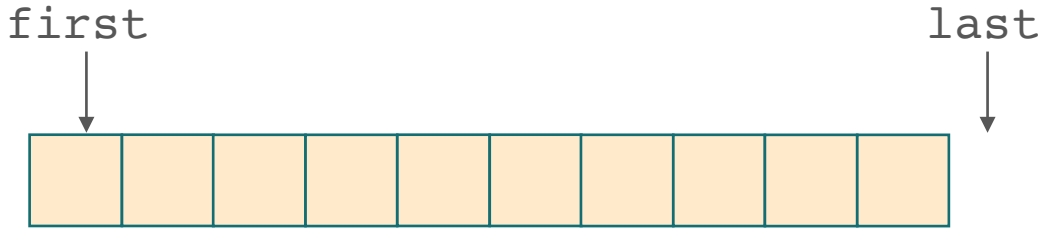
```
std::fill(first, last, 42);
```



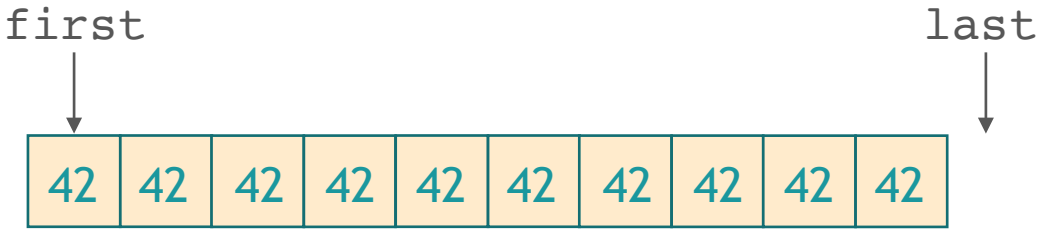

```
std::generate(first, last, f);
```



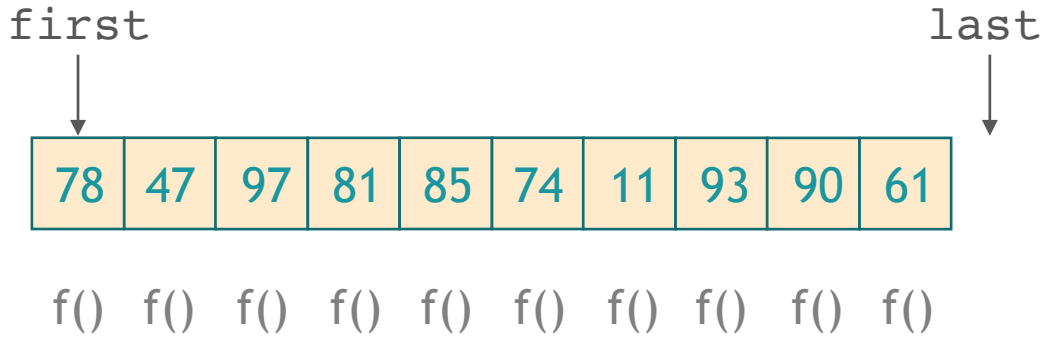

```
std::iota(first, last, 42);
```



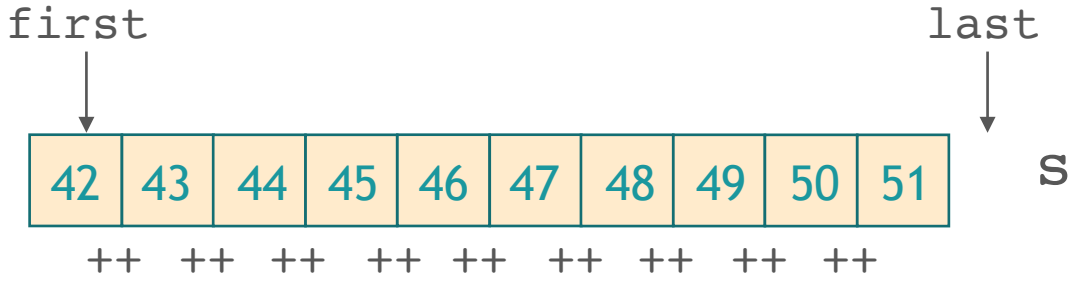
```
std::fill(first, last, 42);
```




```
std::generate(first, last, f);
```



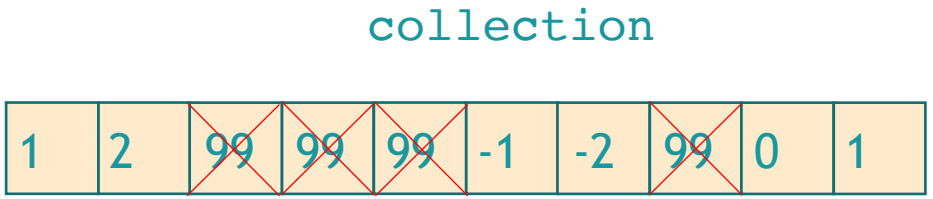

```
std::iota(first, last, 42);
```




```
std::replace(first, last, 42, 43);
```

CHANGING STRUCTURE

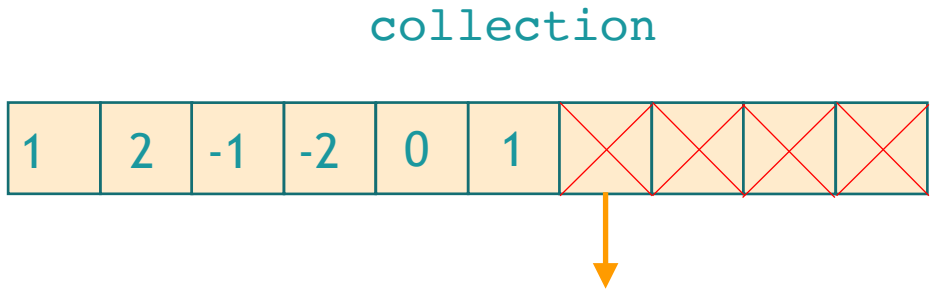
remove




```
std::remove(begin(collection), end(collection), 99);
```

CHANGING STRUCTURE

`remove`




```
std::remove(begin(collection), end(collection), 99);
```

CHANGING STRUCTURE

`remove`

collection

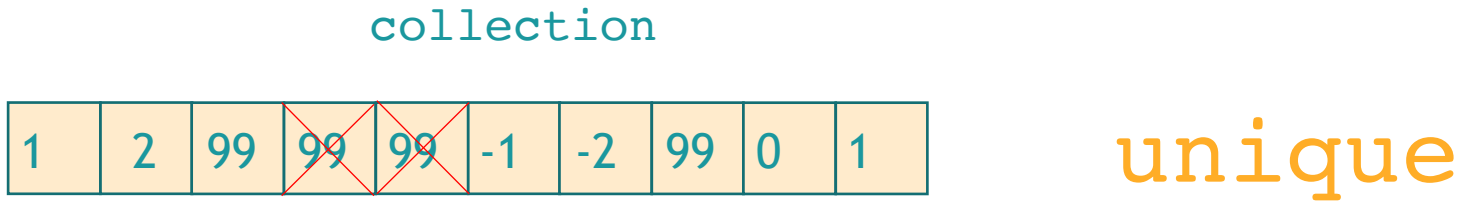
1	2	-1	-2	0	1

```
collection.erase(std::remove(begin(collection), end(collection), 99), end(collection));
```

→ `erase(collection, 99);`

@JoBoccara

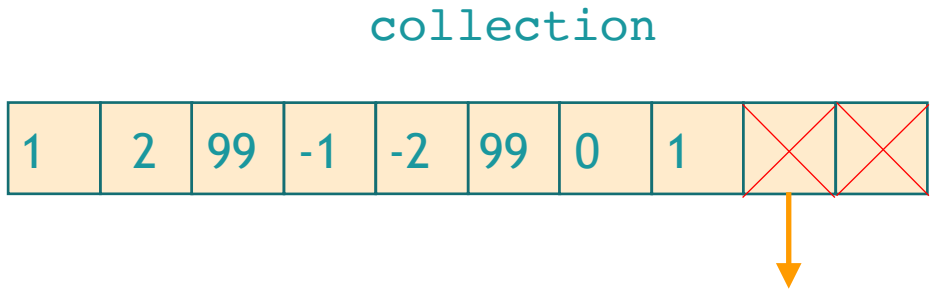
CHANGING STRUCTURE



unique

```
std::unique(begin(collection), end(collection));
```

CHANGING STRUCTURE

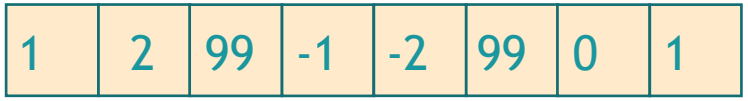


unique

```
std::unique(begin(collection), end(collection));
```

CHANGING STRUCTURE

collection



unique

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

→ `unique(collection);`

@JoBoccara

_COPY

*_copy

remove_copy

unique_copy

reverse_copy

rotate_copy

replace_copy

partition_copy

partial_sort_copy

_IF

*_if

find_if

find_if_not

count_if

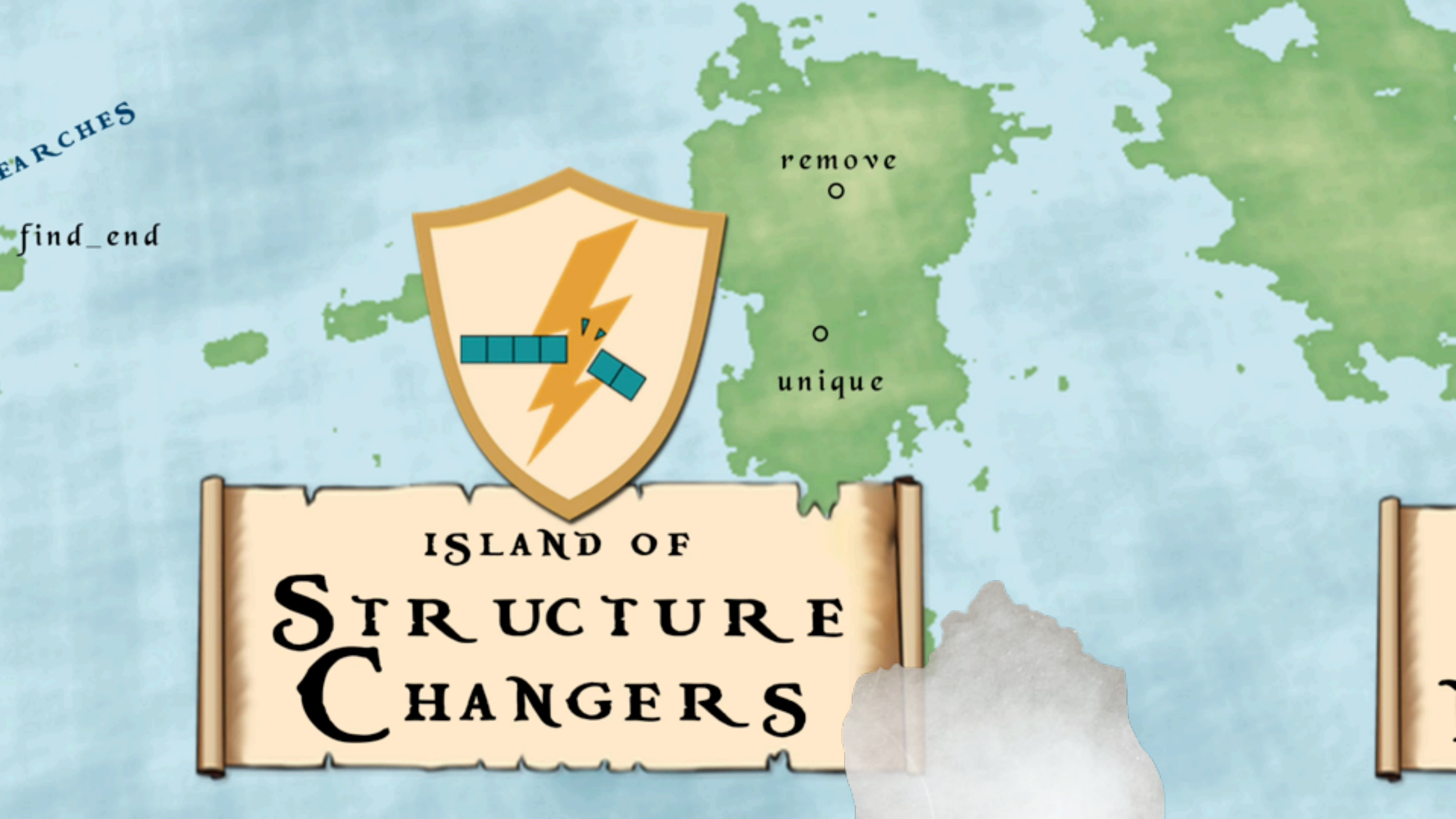
remove_if

remove_copy_if

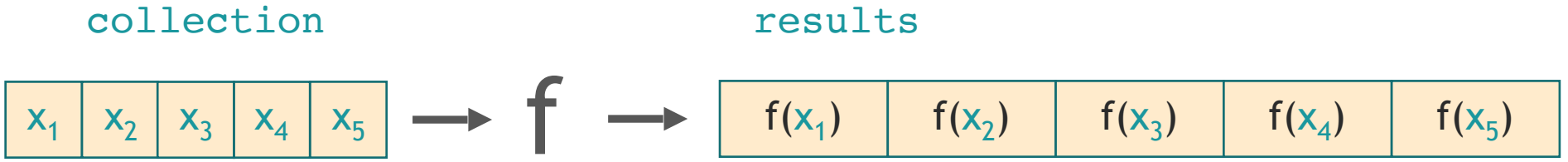
replace_if

replace_copy_if

copy_if

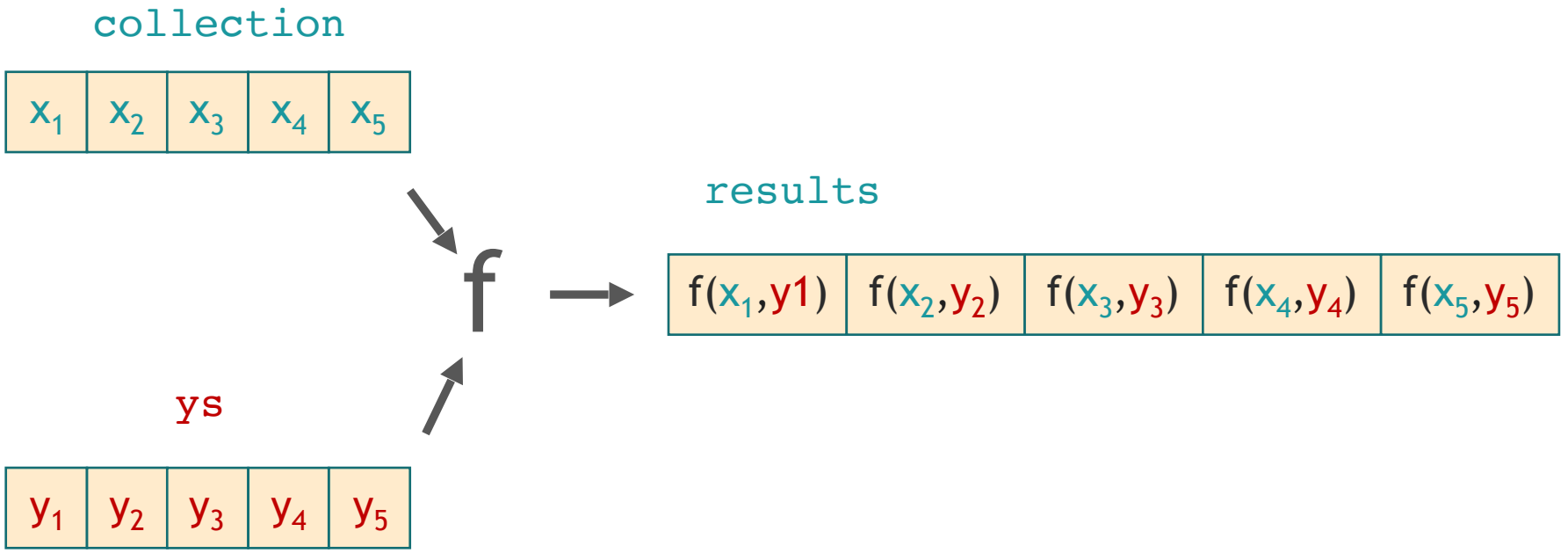

EARCHES
find_end
remove
unique
ISLAND OF
STRUCTURE
CHANGERS

TRANSFORM




```
std::transform(begin(collection), end(collection),
               std::back_inserter(results),
               f);
```

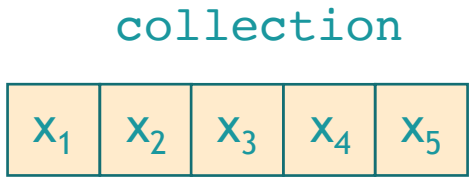
TRANSFORM




```
std::transform(begin(collection), end(collection),
               begin(ys),
               std::back_inserter(results),
               f);
```

FOR_EACH

```
std::for_each(begin(collection), end(collection), f);
```



Calls:

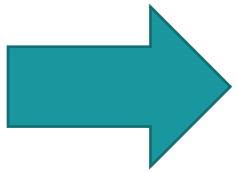
$f(x_1)$

$f(x_2)$

$f(x_3)$

$f(x_4)$

$f(x_5)$



Side-effects


Fluent {C++}
for_each
transform
Lonely Islands
nstruct
ized_copy
copy_backward


Fluent {C++}
Province of Value Queries
Province of Property Queries
Province of 2-Ranges Properties
Glorious County of Algos on Sets
Lonely Islands
Lands of Queries
Peninsula of Raw Memory
Territory of Movers
Province of Research
Island of Structure Changers
Land of Value Modifiers
Lands of Permutations
Secret Runes
The World of C++ STL Algorithms

RAW MEMORY

`fill` → operator=

`copy` → operator=

`move` → operator=

`uninitialized_fill` → ctor

`uninitialized_copy` → copy ctor

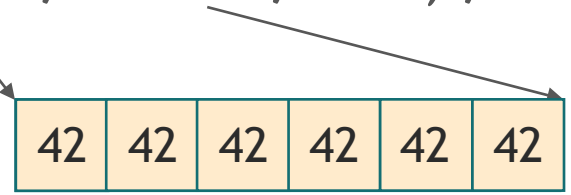
`uninitialized_move` → move ctor

```
std::uninitialized_fill(first, last, 42);

std::destroy(first, last);

uninitialized_default_construct

uninitialized_value_construct
```



_N

```
copy_n
fill_n
generate_n
search_n
for_each_n
uninitialized_copy_n
uninitialized_fill_n
uninitialized_move_n
uninitialized_default_construct_n
uninitialized_value_construct_n
destroy_n
```

_N

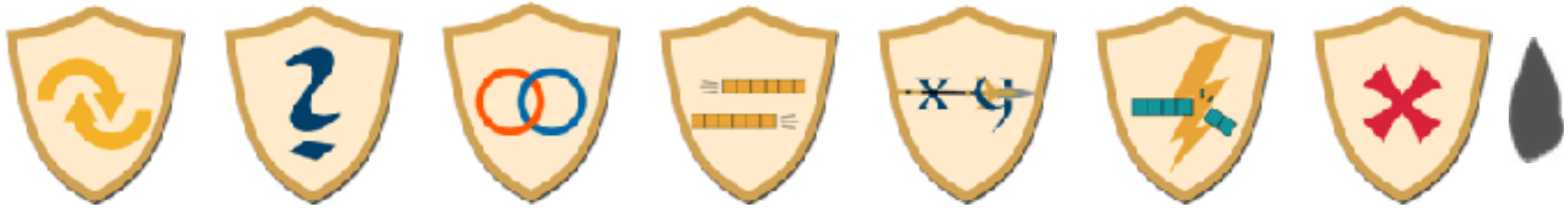
```
std::fill(begin(collection), end(collection), 42);
```

copy_n
fill_n
generate_n
search_n
for_each_n
uninitialized_copy_n
uninitialized_fill_n
uninitialized_move_n
uninitialized_default_construct_n
uninitialized_value_construct_n
destroy_n

```
std::fill_n(begin(collection), 5, 42);
```

```
std::fill_n(std::back_inserter(collection), 5, 42);
```

NOW WHAT?



STL algorithms can make code more expressive

- Replace your `for`-loops by the right algorithm
- Happy to lend a hand if it helps
  jonathan@fluentcpp.com

Want more algorithms?

- Boost.Algorithms
  - gather
  - sort_subrange
  - is_palindrome
  - boyer_moore
  - one_of
  - ...

Understand their technical aspects

- Algorithmic complexity
- Pre/post - requisites
- Look at the implementation

Get inspired

- Understand what abstractions work well
- Write your own algorithms
  - Combine an algorithm with a rune sort_copy
  - Enrich a family set_segregate
  - Start a new family

fluentcpp.com/getTheMap

@JoBoccara

Fluent {C++}

EXPRESSIVE CODE IN C++

fluentcpp.com/getTheMap

@JoBoccara

Fluent {C++}

EXPRESSIVE CODE IN C++