New Algorithms in C++23

Conor Hoekstra

code_report

Algorithms + Data Structures = Programs

RUN FOR THE FUN OF IT!

2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023

MOODY'S ANALYTICS

NVIDIA

Java

GO

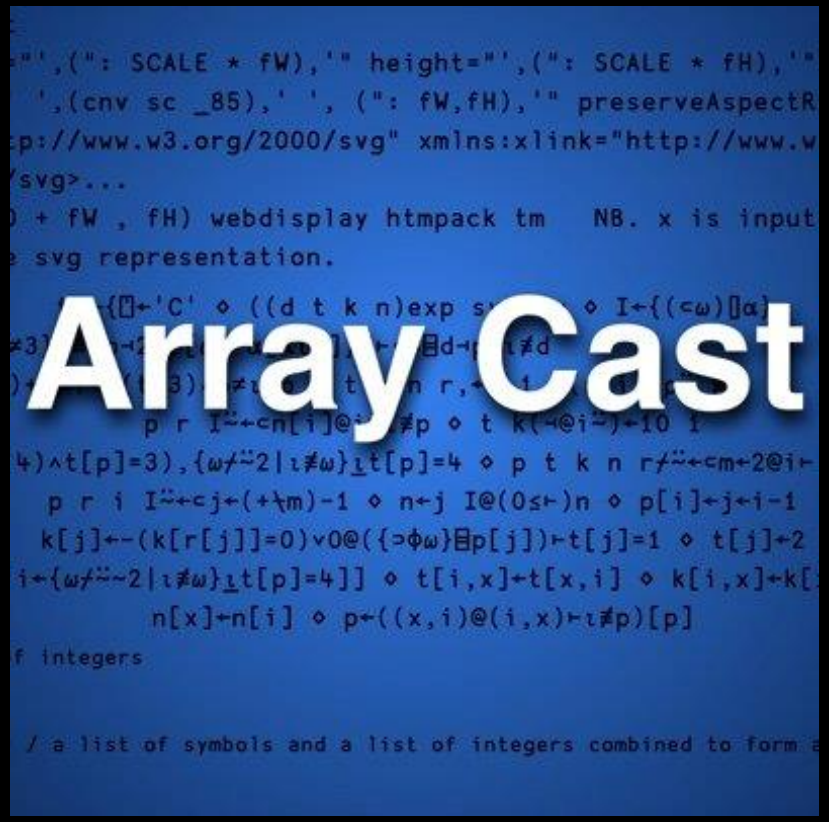
About Me

Conor Hoekstra / @code_report

APL

J

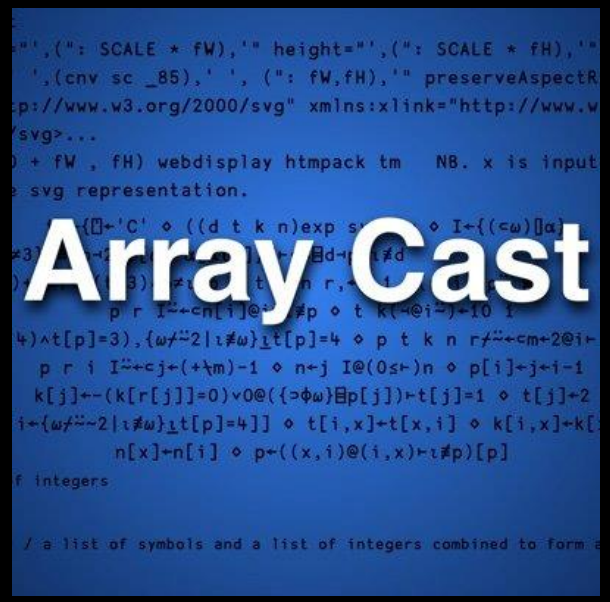
Algorithms + Data Structures = Programs



319 Videos

32 (20) Talks

Algorithms + Data Structures = Programs



RUN 👟 FOR THE FUN 🙂 OF IT!

132 Episodes
@adspthepodcast

54 Episodes
@arraycast

9 Episodes
@conorhoekstra

https://github.com/codereport/Content

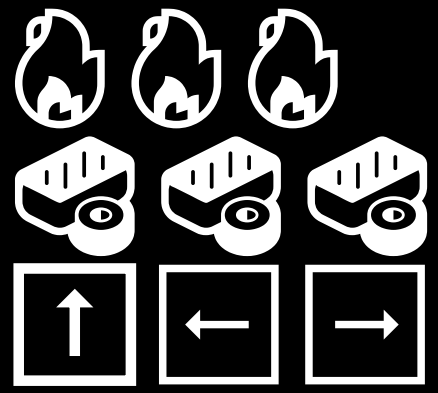
Code Links

Example	Language	GitHub Link	Godbolt Link
Warm Up: Negatives	C++	Link	https://godbolt.org/z/8TEevabqd
Sushi for Two	C++	Link	https://godbolt.org/z/69Kh8baz3
Sushi for Two	Circle	Link	https://godbolt.org/z/P6PxMnhMz
Max Gap	C++	Link	https://godbolt.org/z/P43EhYcsj
Max Gap	Circle	Link	https://godbolt.org/z/3K598jMMa

Algorithm Land Overview

Problems:

- Warm Up
- Sushi for Two
- Max Gap



C++ Algorithm Land Overview

C++98 Iterator Algorithms
std::find_if(a.begin(), a.end(), f)
std::count(b.begin(), b.end(), 3)
std::all_of(c.begin(), c.end(), f)
std::sort(d.begin(), d.end())
...

C++ Algorithm Land Overview

C++98 Iterator Algorithms	C++20 Range Algorithms
`std::find_if(a.begin(), a.end(), f)`	`std::ranges::find_if(a, f)`
`std::count(b.begin(), b.end(), 3)`	`std::ranges::count(b, 3)`
`std::all_of(c.begin(), c.end(), f)`	`std::ranges::all_of(c, f)`
`std::sort(d.begin(), d.end())`	`std::ranges::sort(d)`
...	...

C++ Algorithm Land Overview

C++98 Iterator Algorithms	C++20 Range Algorithms	C++20/23 Range Adaptors & Factories
`std::find_if(a.begin(), a.end(), f)`	`std::ranges::find_if(a, f)`	`std::views::take`
`std::count(b.begin(), b.end(), 3)`	`std::ranges::count(b, 3)`	`std::views::drop`
`std::all_of(c.begin(), c.end(), f)`	`std::ranges::all_of(c, f)`	`std::views::transform`
`std::sort(d.begin(), d.end())`	`std::ranges::sort(d)`	`std::views::filter`
...	...	`std::views::chunk_by`
		...

C++ Algorithm Land Overview

C++98 Iterator Algorithms	C++20 Range Algorithms	C++20/23 Range Adaptors & Factories
std::find_if(a.begin(), a.end(), f)	std::ranges::find_if(a, f)	std::views::take
std::count(b.begin(), b.end(), 3)	std::ranges::count(b, 3)	std::views::drop
std::all_of(c.begin(), c.end(), f)	std::ranges::all_of(c, f)	std::views::transform
std::sort(d.begin(), d.end())	std::ranges::sort(d)	std::views::filter
...	...	std::views::chunk_by
		...

C++20/23 Range Adaptors & Factories

std::views::take

std::views::drop

std::views::transform

std::views::filter

std::views::chunk_by

C++20/23 Range Adaptors & Factories

std::views::take
std::views::drop
std::views::transform
std::views::filter

std::views::chunk_by

C++20/23 Range Adaptors & Factories

drop
drop_while
elements (keys | values)
filter
iota
join
reverse
split
take
take_while
transform

adjacent (pairwise)
adjacent_transform (pairwise_transform)
cartesian_product
chunk
chunk_by
enumerate
join_with
slide
stride
zip
zip_transform

[[digression]]

Different Programming Paradigms

- Collection Oriented Programming ☆
- Function Programming
- Object-Oriented Programming
- Imperative Programming

Libraries	Languages
Ranges	APL
Iterators	λ
Streams	Pharo

[[end of digression]]

Warm Up

```cpp
int num_negatives(std::vector<int> nums) {
    int count = 0;
    for (int i = 0; i < nums.size(); ++i) {
        if (nums[i] < 0) ++count;
    }
    return count;
}
```

```cpp
int num_negatives(std::vector<int> nums) {
    int count = 0;
    for (auto const num : nums) {
        if (num < 0) ++count;
    }
    return count;
}
```

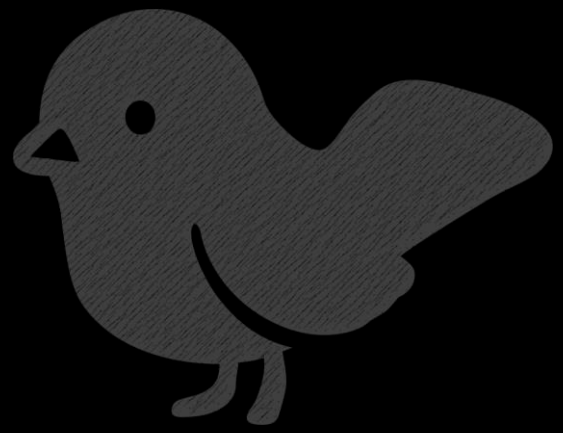
```cpp
auto num_negatives(std::vector<int> nums) {
    int count = 0;
    for (auto const num : nums) {
        if (num < 0) ++count;
    }
    return count;
}
```

```cpp
auto num_negatives(std::vector<int> nums) -> int {
    int count = 0;
    for (auto const num : nums) {
        if (num < 0) ++count;
    }
    return count;
}
```

```cpp
auto num_negatives(std::vector<int> nums) -> int {
    return std::count_if(nums.cbegin(), nums.cend(),
        [](auto e) { return e < 0; });
}
```

```cpp
auto num_negatives(std::vector<int> nums) -> int {
    return std::ranges::count_if(nums,
        [](auto e) { return e < 0; });
}
```

```cpp
auto num_negatives(std::vector<int> nums) -> int {
    return std::ranges::count_if(nums, lt_(0));
}
```



```
using namespace combinators;

auto num_negatives(std::vector<int> nums) -> int {
    return std::ranges::count_if(nums, lt_(0));
}
```

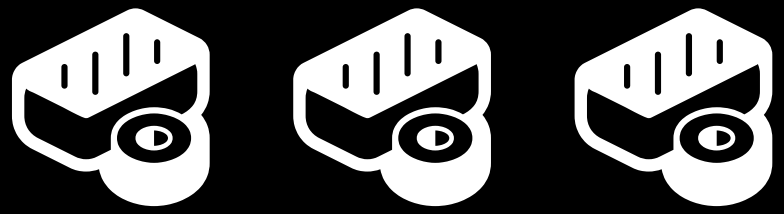
inevitably someone says…

Loops are easier to read / understand … everyone knows them

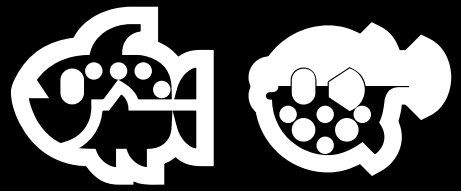
```
int num_negatives(std::vector<int> nums) {
    int count = 0;
    for (auto const num : nums) {
        if (num < 0) ++count;
    }
    return count;
}
```

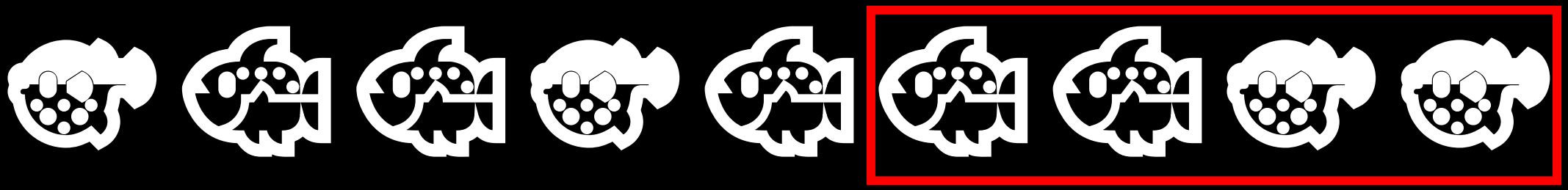
introducing one of my favorite problems of all time…

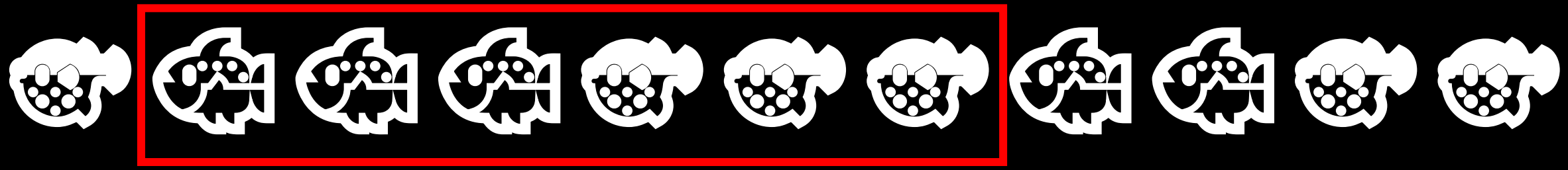
Sushi for Two



https://codeforces.com/contest/1138/problem/A







```cpp
template <int N>
constexpr auto sushi_for_two(std::array<int, N> sushi) {
    int current_sushi       = 0;
    int sushi_in_a_row      = 0;
    int prev_sushi_in_a_row = 0;
    int max_of_mins         = 0;
    for (auto const s : sushi) {
        if (current_sushi != s) {
            current_sushi = s;
            if (prev_sushi_in_a_row == 0) {
                prev_sushi_in_a_row = sushi_in_a_row;
                sushi_in_a_row      = 1;
            } else {
                auto const min = std::min(sushi_in_a_row, prev_sushi_in_a_row);
                max_of_mins         = std::max(max_of_mins, min);
                prev_sushi_in_a_row = sushi_in_a_row;
                sushi_in_a_row      = 1;
            }
        } else {
            sushi_in_a_row += 1;
        }
    }
    auto const min = std::min(sushi_in_a_row, prev_sushi_in_a_row);
    max_of_mins    = std::max(max_of_mins, min);
    return max_of_mins * 2;
}
```

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return 2 * std::ranges::max(sushi
        | chunk_by(_eq_)
        | transform(std::ranges::distance)
        | adjacent_transform<2>(_min_));
}
```

q

```
sf2: { x }

[1, 2, 2, 1, 2, 2, 2, 1, 1]
```

```
sf2: { chunk x }

[[1], [2, 2], [1], [2, 2, 2], [1, 1]]
```

```
sf2: { count each chunk x }

[1, 2, 1, 3, 2]
```

```
sf2: { (&) prior count each chunk x }
```

```
[1, 1, 1, 1, 2]
```

Hoogle Translate

prior

CUDA	adjacent_difference	Thrust	Doc
C++	adjacent_difference	<numeric>	Doc
APL	/ (n-wise reduce)	-	Doc
Haskell	mapAdjacent	Data.List.HT	Doc
Kotlin	zipWithNext	collections	Doc
q	prior	-	Doc
C++	adjacent_transform	<ranges>	Doc

```
sf2: { (&) prior count each chunk x }
```

```
[1, 1, 1, 1, 2]
```

```
sf2: { 1 _ (&) prior count each chunk x }
```

```
[1, 1, 1, 2]
```

```
sf2: { max 1 _ (&) prior count each chunk x }
```

2

```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
```

2

```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
```

4

```
sf2: 2 * max 1 _ (&) prior count each chunk ::
```

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi;
}
```

[1, 2, 2, 1, 2, 2, 2, 1, 1]

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        | chunk_by(std::equal_to{});
}
```

[[1], [2, 2], [1], [2, 2, 2], [1, 1]]

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        | chunk_by(std::equal_to{})
        | transform(std::ranges::distance);
}
```

[1, 2, 1, 3, 2]

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        | chunk_by(std::equal_to{})
        | transform(std::ranges::distance)
        | adjacent_transform<2>(
            [](auto a, auto b) { return std::min(a, b); });
}
```

[1, 1, 1, 2]

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return std::ranges::max(sushi
        | chunk_by(std::equal_to{})
        | transform(std::ranges::distance)
        | adjacent_transform<2>(
            [](auto a, auto b) { return std::min(a, b); }));
}
```

2

2 very irritating things about this code

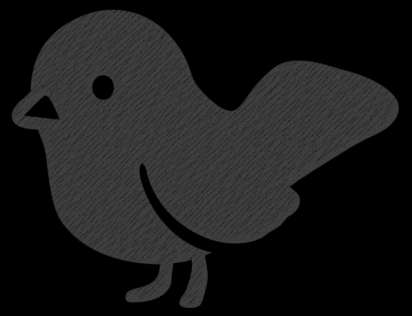
```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) -> int {
    return 2 * std::ranges::max(sushi
        | chunk_by(std::equal_to{})
        | transform(std::ranges::distance)
        | adjacent_transform<2>(
            [](auto a, auto b) { return std::min(a, b); }));
}
```

```
using namespace std::views;

auto sushi_for_two(std::vector<int> sushi) {
    return 2 * std::ranges::max(sushi
            | chunk_by(std::equal_to{})
            | transform(std::ranges::distance)
            | adjacent_transform<2>(_min_));
}
```

```
using namespace std::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return 2 * std::ranges::max(sushi
            | chunk_by(std::equal_to{})
            | transform(std::ranges::distance)
            | adjacent_transform<2>(_min_));
}
```

```
using namespace std::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return 2 * std::ranges::max(sushi
            | chunk_by(_eq_)
            | transform(std::ranges::distance)
            | adjacent_transform<2>(_min_));
}
```

CIRCLE

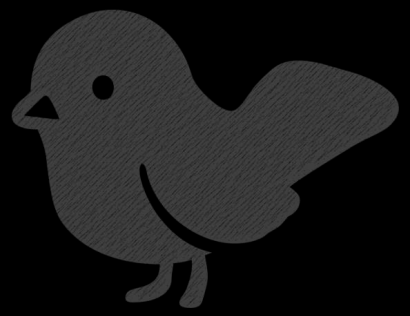


```
using namespace std::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> chunk_by($, _eq_)
            |> transform($, std::ranges::distance)
            |> adjacent_transform<2>($, _min_)
            |> std::ranges::max($) * 2;
}
```

CIRCLE

COMPILER EXPLORER

```
-std=c++2b --gcc-toolchain /opt/compiler-explorer/gcc-13.1.0/ -Wl,-rpath,/opt/compiler-explorer/gcc-13.1.0/lib64/
```



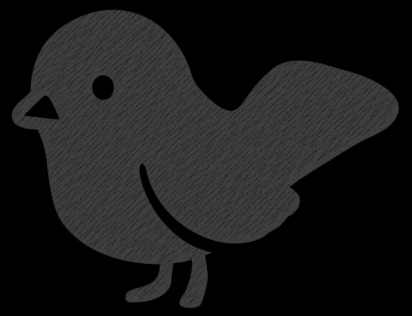
```
using namespace std::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> chunk_by($, _eq_)
            |> transform($, std::ranges::distance)
            |> adjacent_transform<2>($, _min_)
            |> std::ranges::max($) * 2;
}
```

Code Links

Example	Language	GitHub Link	Godbolt Link
Warm Up: Negatives	C++	Link	https://godbolt.org/z/8TEevabqd
Sushi for Two	C++	Link	https://godbolt.org/z/69Kh8baz3
Sushi for Two	Circle	Link	https://godbolt.org/z/P6PxMnhMz
Max Gap	C++	Link	https://godbolt.org/z/P43EhYcsj
Max Gap	Circle	Link	https://godbolt.org/z/3K598jMMa

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

CIRCLE



```
using namespace std::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> chunk_by($, _eq_)
            |> transform($, std::ranges::distance)
            |> adjacent_transform<2>($, _min_)
            |> std::ranges::max($) * 2;
}
```

CIRCLE

```
auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> chunk_by($, _eq_)
            |> transform($, std::ranges::distance)
            |> adjacent_transform<2>($, _min_)
            |> std::ranges::max($) * 2;
}
```

```
sushiForTwo :: [Int] -> Int
sushiForTwo = (*2)
            . maximum
            . mapAdjacent min
            . map length
            . group
```

CIRCLE

```
auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> chunk_by($, _eq_)
            |> transform($, std::ranges::distance)
            |> adjacent_transform<2>($, _min_)
            |> std::ranges::max($) * 2;
}
```

```
sushiForTwo :: [Int] -> Int
sushiForTwo = (*2)
            . maximum
            . mapAdjacent min
            . map length
            . group
```

q

```
sf2: 2 * max 1 _ (&) prior
    count each chunk ::
```

```
q
sf2: 2 * max 1 _ (&) prior
    count each chunk ::
```

```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
```

```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
chunk: { (where differ x) cut x }
```

```
sf2: { 2 * max 1 _ (&) prior deltas where differ x }
```

```
sf2: { 2 * max 1 _ (&) prior deltas (where differ x) , count x }
```

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi;
}
```

[1, 2, 2, 1, 2, 2, 2, 1, 1]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1));
}
```

[1, 0, 1, 1, 0, 0, 1, 0]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1));
}
```

```
[(1,1), (0,2), (1,3), (1,4), (0,5), (0,6), (1,7), (0,8)]
```

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst);
}
```

[(1,1), (1,3), (1,4), (1,7)]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> transform($, _snd);
}
```

[1, 3, 4, 7]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($);
}
```

[1, 3, 4, 7]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()));
}
```

[0, 1, 3, 4, 7, 9]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $);
}
```

[1, 2, 1, 3, 2]

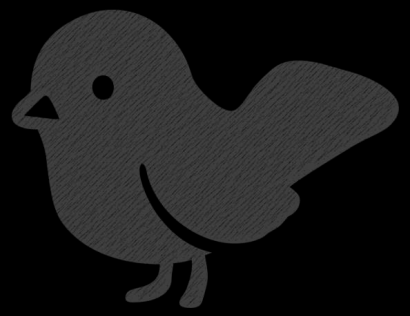
```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $)
        |> zip_with(_min_, $, $ | drop(1));
}
```

[1, 1, 1, 2]

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $)
        |> zip_with(_min_, $, $ | drop(1))
        |> std::ranges::max($) * 2;
}
```

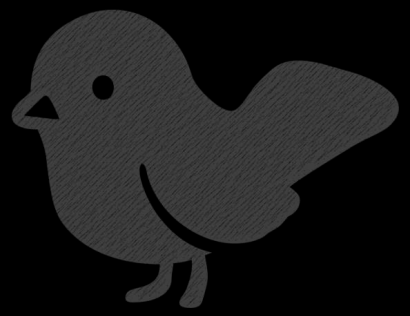
4

CIRCLE



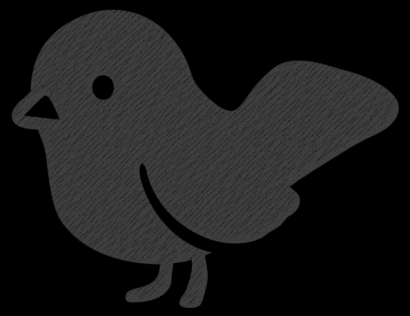
```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $)
        |> zip_with(_min_, $, $ | drop(1))
        |> std::ranges::max($) * 2;
}
```

CIRCLE



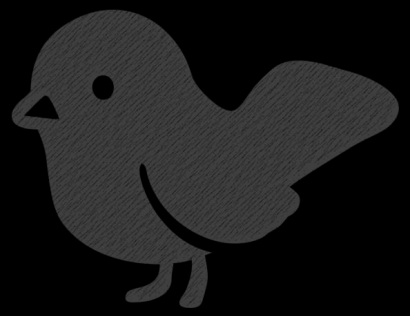
```
auto adjacent_transform_2(auto&& rng, auto op) {
    return rng |> zip_with(op, $, $ | drop(1));
}
auto differ(auto&& rng) { return adjacent_transform_2(rng, _neq_); }
auto deltas(auto&& rng) { return adjacent_transform_2(rng, _c(_sub_)); }

auto indices(auto&& rng) {
    return rng
           |> zip($, iota(1))
           |> filter($, _fst)
           |> values($)
           |> concat(single(0), $);
}
```

CIRCLE



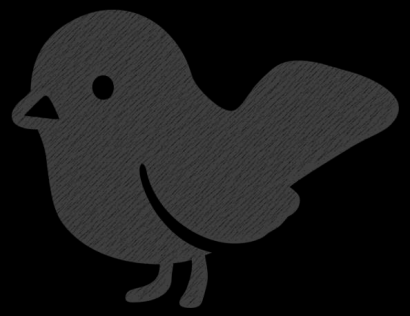
```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $)
        |> zip_with(_min_, $, $ | drop(1))
        |> std::ranges::max($) * 2;
}
```

CIRCLE



```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
            |> differ($)
            |> indices($)
            |> concat($, single(sushi.size()))
            |> deltas($)
            |> adjacent_transform_2($, _min_)
            |> std::ranges::max($) * 2;
}
```

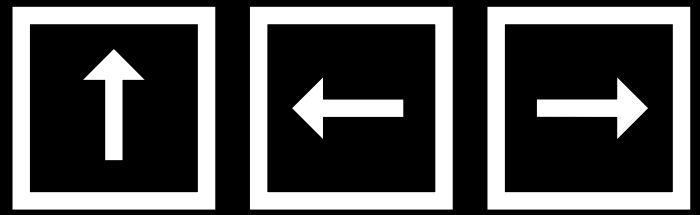
CIRCLE



```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
    return sushi
        |> zip_with(_neq_, $, $ | drop(1))
        |> zip($, iota(1))
        |> filter($, _fst)
        |> values($)
        |> concat(single(0), $, single(sushi.size()))
        |> zip_with(_sub_, $ | drop(1), $)
        |> zip_with(_min_, $, $ | drop(1))
        |> std::ranges::max($) * 2;
}
```

```
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
  auto indices = concat(                                  //
      concat(single(0),                                   //
             zip(zip_with(_neq_, sushi, sushi | drop(1)),  //
                 iota(1))                                 //
                 | filter(_fst)                           //
                 | values),                               //
      single(sushi.size()));                              //
  auto deltas = zip_with(_sub_, indices | drop(1), indices);
  return 2 * ranges::max(zip_with(_min_, deltas, deltas | drop(1)));
}
```

Max Gap



https://leetcode.com/problems/maximum-gap/

[8, 4, 1, 3, 10]

[1, 3, 4, 8, 10]

[1, 3, 4, 8, 10]
[2, 1, 4, 2]

[1, 3, 4, 8, 10]

[2, 1, **4**, 2]

```cpp
using namespace std::views;

auto max_gap(std::vector<int> nums) {
    return nums;
}
```

[8, 4, 1, 3, 10]

```cpp
using namespace std::views;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return nums;
}
```

[1, 3, 4, 8, 10]

```cpp
using namespace std::views;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return nums
      | adjacent_transform<2>(std::minus{});
}
```

[2, 1, 4, 2]

anyone notice the bug?

```
using namespace std::views;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return std::ranges::max(nums
      | adjacent_transform<2>(std::minus{}));
}
```

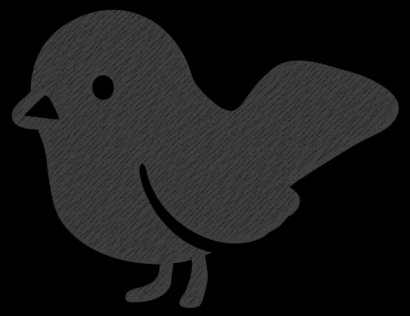
4

```cpp
using namespace std::views;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return std::ranges::max(nums
      | reverse
      | adjacent_transform<2>(std::minus{}));
}
```

4

```
using namespace std::views;
using namespace combinators;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return std::ranges::max(nums
      | adjacent_transform<2>(_c(_sub_)));
}
```

CIRCLE



```
using namespace std::views;
using namespace combinators;

auto max_gap(std::vector<int> nums) {
    std::ranges::sort(nums);
    return nums
            |> adjacent_transform<2>($, _c(_sub_))
            |> std::ranges::max($);
}
```

Thank You

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

Conor Hoekstra

code_report

codereport

Questions?

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

Conor Hoekstra

code_report

codereport