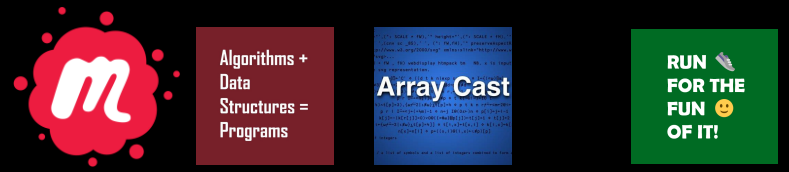
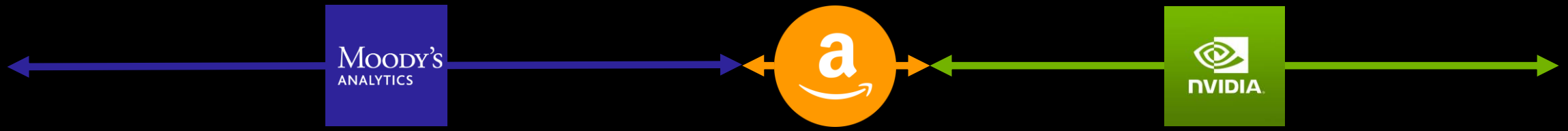
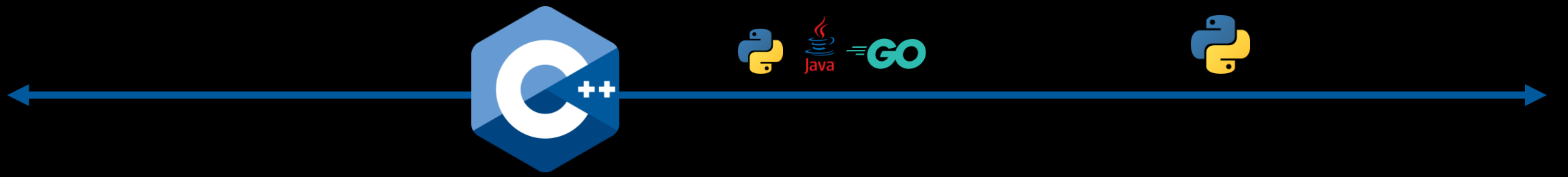
Algorithms in q

Conor Hoekstra

 code_report 

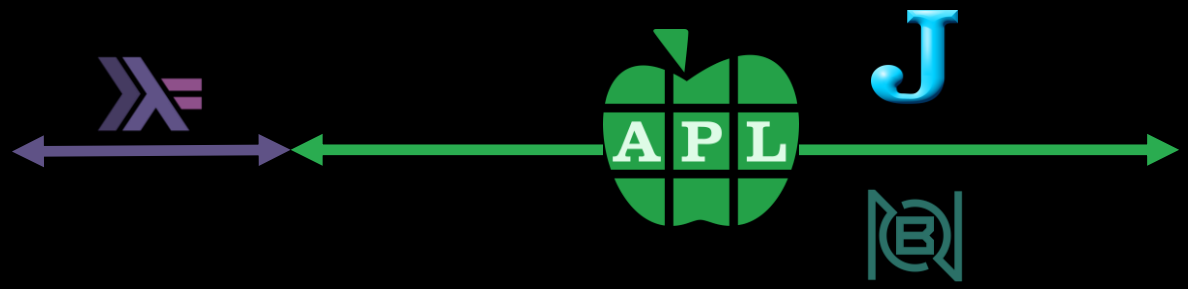




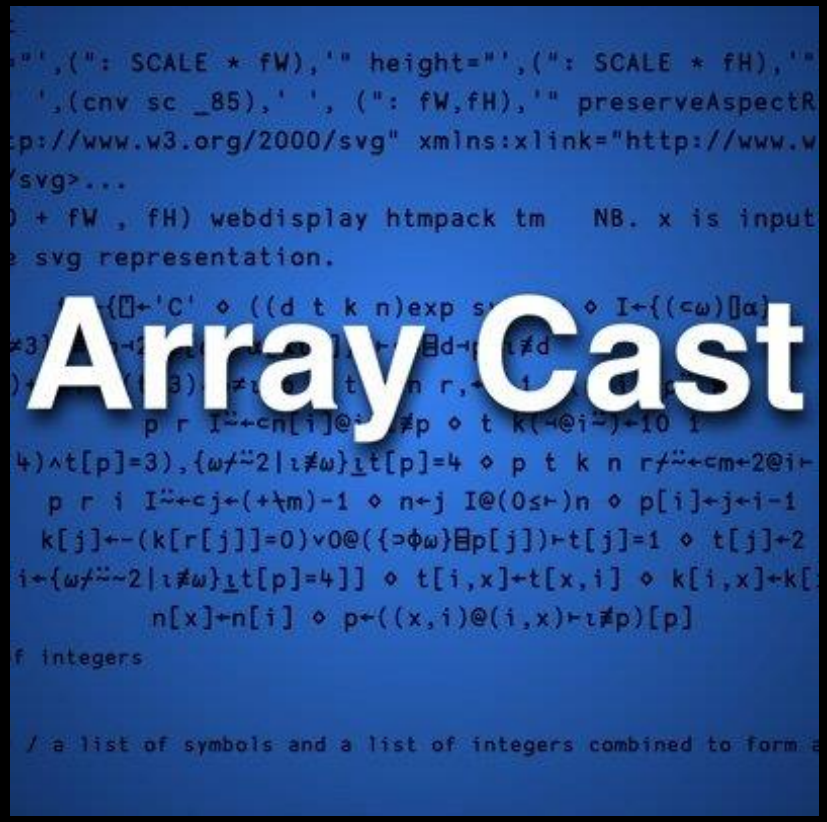


About Me

Conor Hoekstra / @code_report



Algorithms + Data Structures = Programs

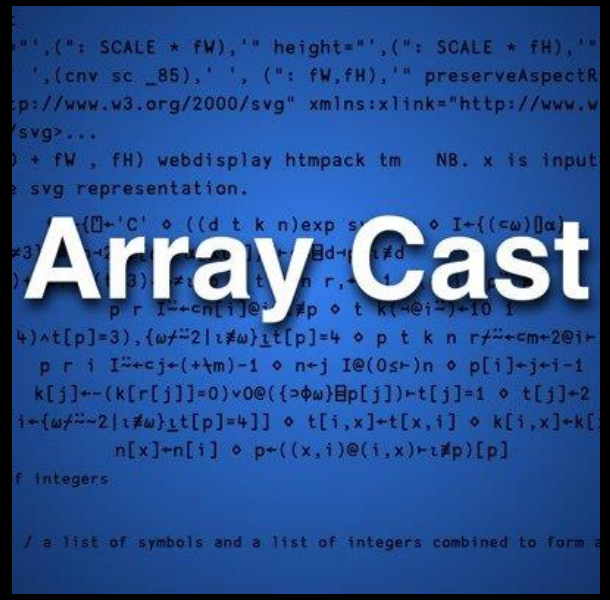


318 Videos

31 (20) Talks

Algorithms + Data Structures = Programs

131 Episodes



52 Episodes

RUN FOR THE FUN OF IT!

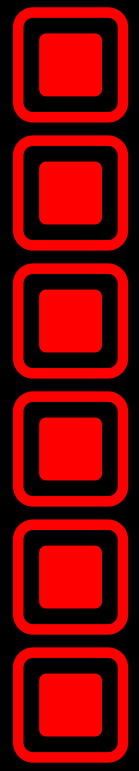
8 Episodes

https://github.com/codereport/Content

https://github.com/codereport/top10

	Problem	Solution
1	Rain Water	`rainWater: { sum abs x - (reverse maxs reverse x) & maxs x }`
2	MCO	1) `mco: { max sum each chunk x }` 2) `mco: { max { y * y + x } scan x }`
3	LCIS	1) `lcis: { 1 + max { y * y + x } scan (>) prior x }` 2) `lcis: { 1 + max sum each chunk 1 _ (>) prior x }`
4	Kadanes	`kadanes: { max { y | y + x } scan x }`
5	SF2	`sf2: { 2 * max (&) prior count each chunk x }`
6	Max Gap	`maxgap: { max 1 _ deltas asc x }`
7	Max Gap Count	`maxgapcount: { sum { x = max x } 1 _ deltas asc x }`
8	TCO	`tco: { 3 <= max sum each chunk x mod 2 }`
9	Skyscraper	`skyscraper: { count distinct maxs x }`
10	OceanView	`oceanview: { where reverse differ maxs reverse x }`

Viewers can expect to learn the utility and importance of the fundamental built-in functions that come with q such as **scans**, **reductions**, **where**, **cut**, **prior** and **more**.



scans
reductions
where
cut
prior
more

Problems:

5. Sushi for Two
6. Max Gap
9. Skyscraper

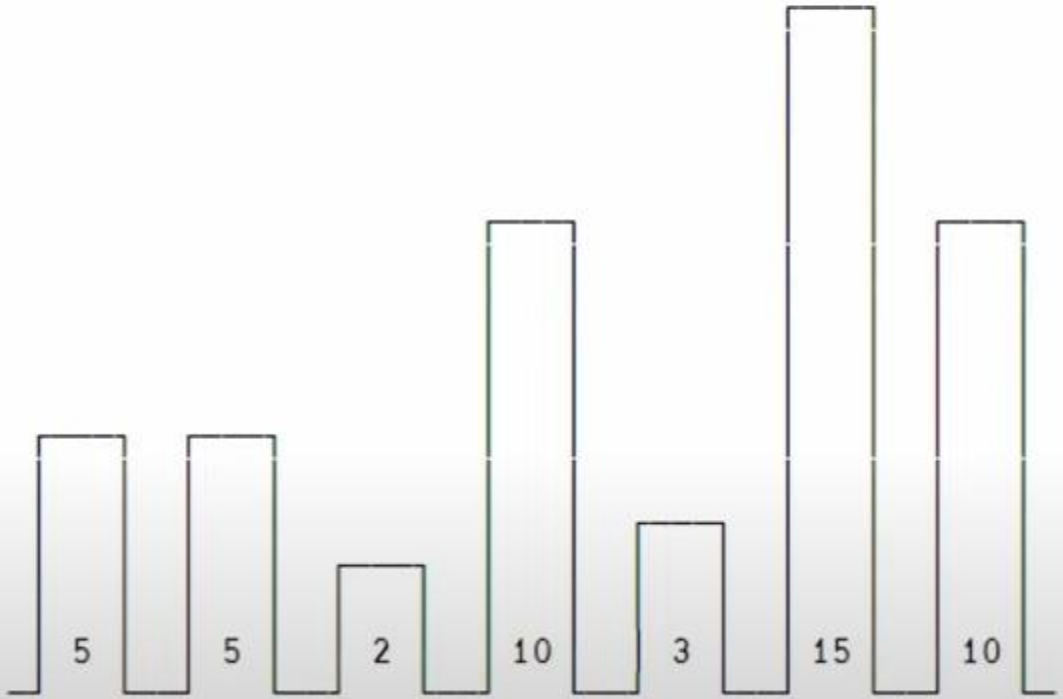
Problems:

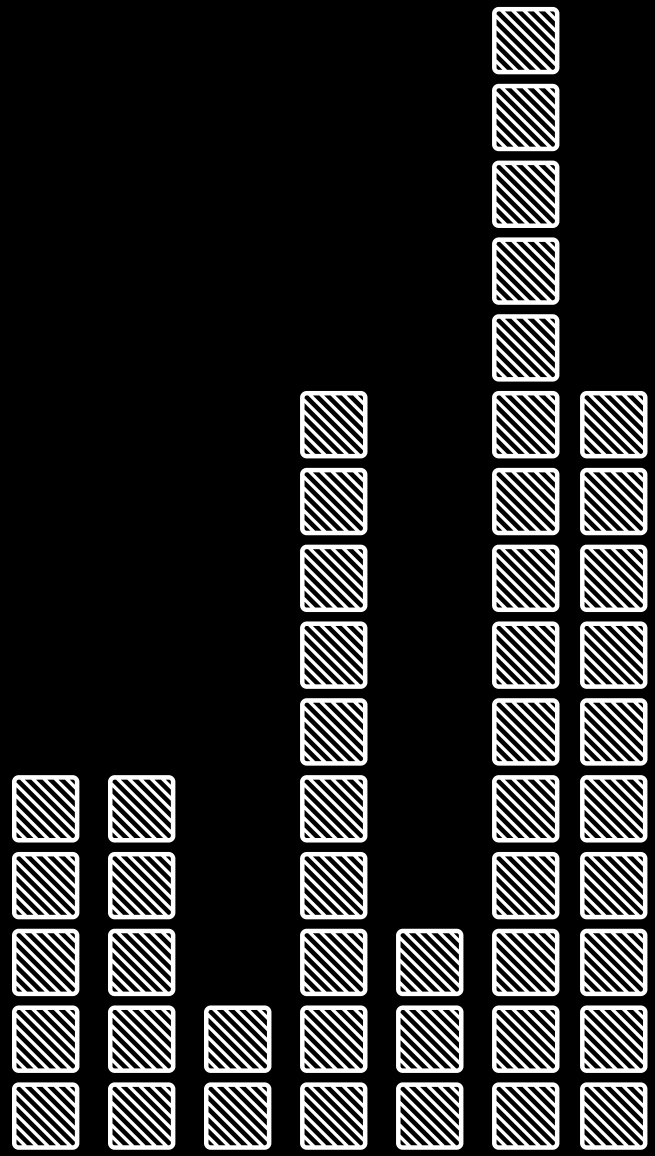
9. Skyscraper
6. Max Gap
5. Sushi for Two

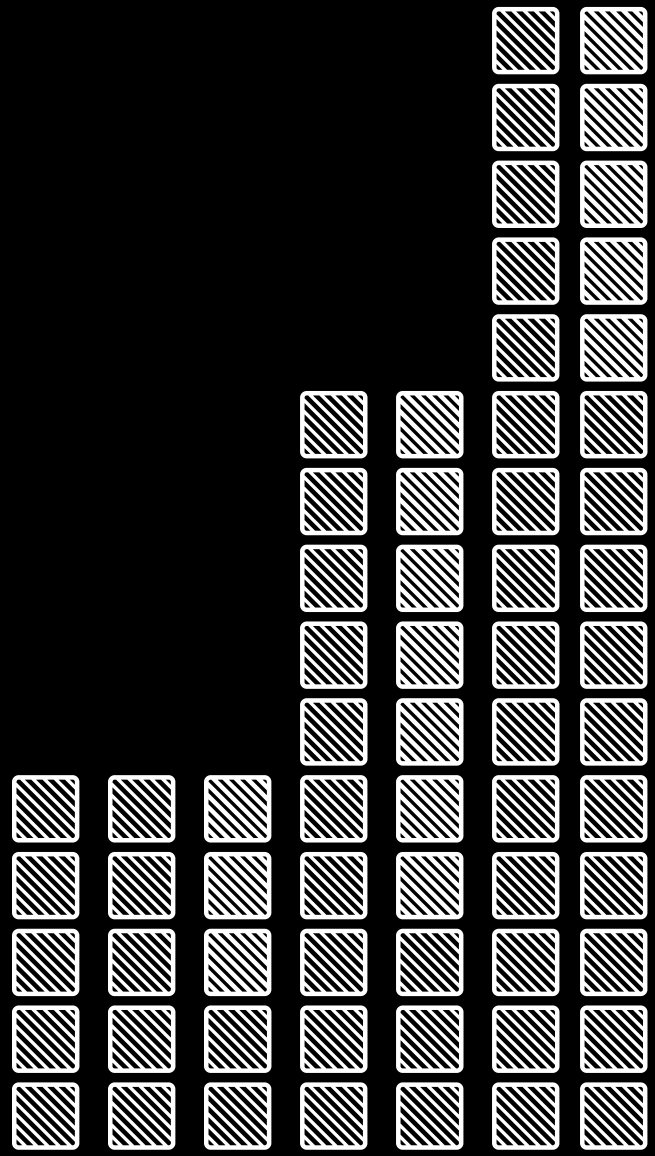
Skyline

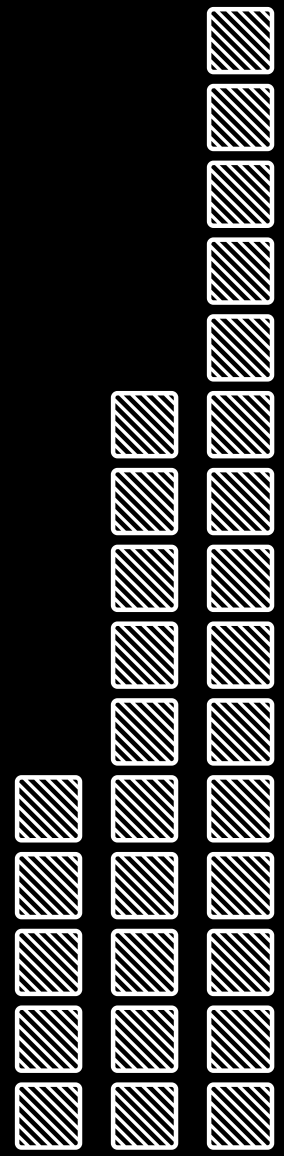
https://youtu.be/6AWSPC6qQB4?t=560

APL Practice Problem









[5, 5, 2, 10, 3, 15, 10]

```
skyline: { x }

[5, 5, 2, 10, 3, 15, 10]
```

```
skyline: { (|) scan x }

[5, 5, 5, 10, 10, 15, 15]
```

```
skyline: { maxs x }

[5, 5, 5, 10, 10, 15, 15]
```

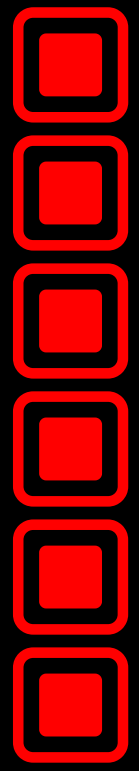
```
skyline: { distinct maxs x }
```

```
[5, 10, 15]
```

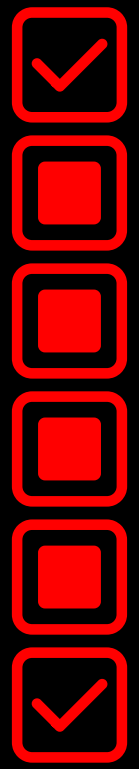
```
skyline: { count distinct maxs x }
```

3

```
skyline: count distinct maxs ::
```

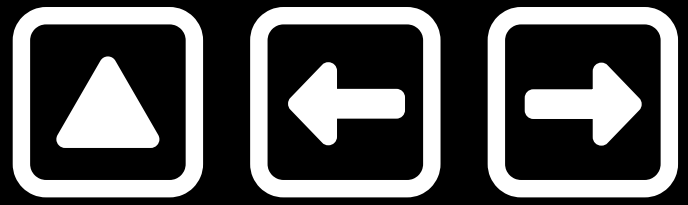


scans
reductions
where
cut
prior
more



scans
reductions
where
cut
prior
more (count, distinct)

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]

maxgap: { x }

[8, 4, 1, 3, 10]

```
maxgap: { asc x }

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

```
maxgap: { deltas asc x }

[1, 2, 1, 4, 2]
```

```
maxgap: { max deltas asc x }
```

4

```
maxgap: max deltas asc ::
```

```
maxgap: { deltas asc x }

[1, 2, 1, 4, 2]
```

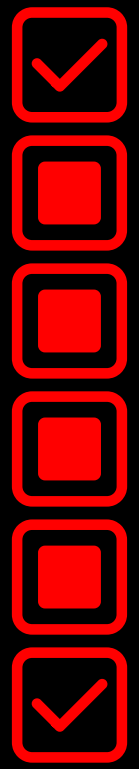
```
maxgap: { deltas asc x }

[10, 2, 1, 4, 2]
```

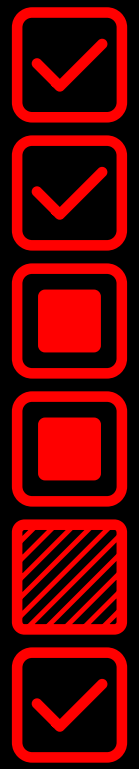
```
maxgap: { 1 _ deltas asc x }

[2, 1, 4, 2]
```

```
maxgap: max 1 _ deltas asc ::
```

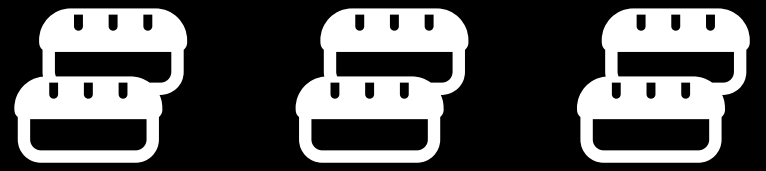


scans
reductions
where
cut
prior
more (count, distinct)

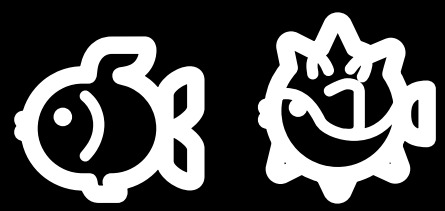


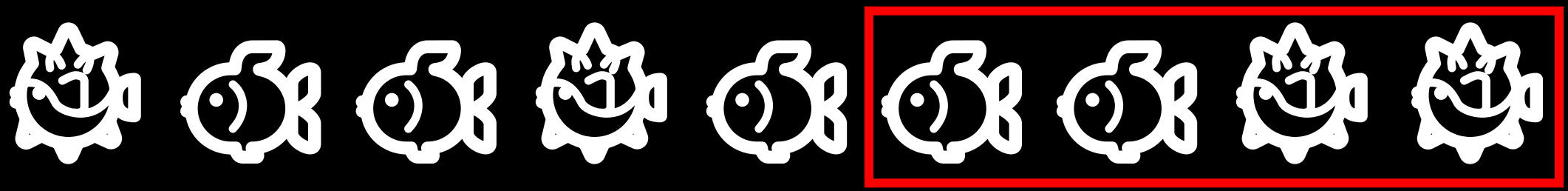
scans
reductions
where
cut
prior (deltas)
more (count, distinct, _, asc)

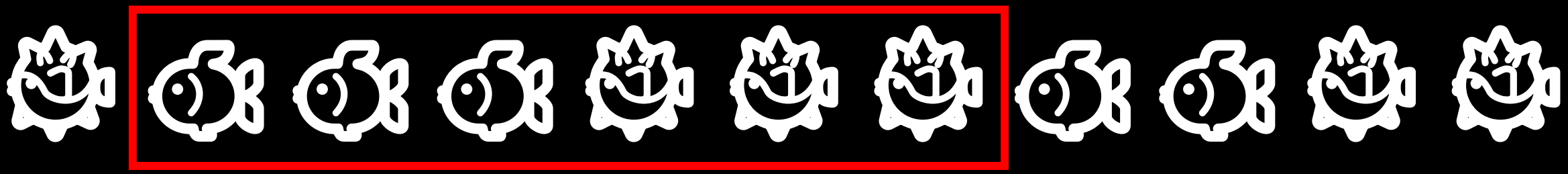
Sushi for Two



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







```
sf2: { x }

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

```
sf2: { x }
chunk: { (where differ x) cut x }
```

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

```
chunk: { x }

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

```
chunk: { differ x }

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

```
chunk: { where differ x }

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

```
chunk: { (where differ x) cut x }
```

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

```
sf2: { chunk x }
chunk: { (where differ x) cut x }

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

```
sf2: { count each chunk x }
chunk: { (where differ x) cut x }
```

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

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

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

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

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

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

2

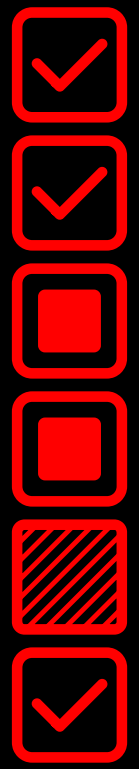
```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
chunk: { (where differ x) cut x }
```

2

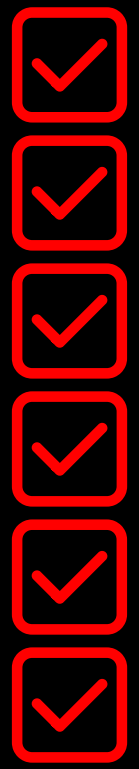
```
sf2: { 2 * max 1 _ (&) prior count each chunk x }
chunk: { (where differ x) cut x }
```

4

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



scans
reductions
where
cut
prior (deltas)
more (count, distinct, _, asc)



scans
reductions
where
cut
prior (deltas, differ)
more (count, distinct, _, asc, each)

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