Monotonicity testing

Ronitt Rubinfeld

6.5240 Sublinear Time Algorithms

(slides on testing monotonicity of functions $f: \{0,1\}^n \to \{0,1\}$ from Sofya Raskhodnikova)

Alphabetized?

https://petapixel.com/2017/11/14/photos-new-futuristic-library-china-1-2-million-books/

Sortedness of a sequence

- Given: list $y_1\, y_2 \ldots\, y_n$
- Question: is the list sorted?

- Clearly requires n steps – must look at each y_i

Sortedness of a sequence

- Given: list $y_1\, y_2 \ldots\, y_n$

- Question: can we quickly test if the list close to sorted?

What do we mean by “close”?

Definition: a list of size n is ε-close to sorted if can delete at most εn values to make it sorted. Otherwise, ε-far.

(ε is given as input, e.g., ε=1/5)

Sorted:	1	2	4	5	7	11	14	19	20	21	23	38	39	45
Close:	1	4	2	5	7	11	14	19	20	39	23	21	38	45
	1	4		5	7	11	14	19	20		23		38	45
Far:	45	39	23	1	38	4	5	21	20	19	2	7	11	14
				1		4	5					7	11	14

Requirements for algorithm:

- Pass sorted lists
- Fail lists that are ε-far.
  - Equivalently: if list likely to pass test, can change (delete) at most ε fraction of list to make it sorted

What if list not sorted, but not far?

Probability of success > ¾

(as usual, can boost it arbitrarily high by repeating several times and outputting "fail" if ever see a "fail", "pass" otherwise)

A first try for an algorithm:

Pick *random entry* and test that *entry and its right neighbor* are in the *correct order*

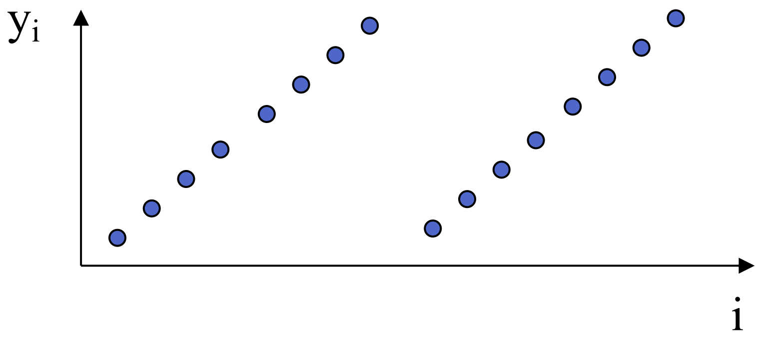
Good input type:

1 2 4 5 7 11 14 19 20 21 23 38 39 45 46 50 57 60 61 80

GOOD: Always passes!

First try (cont.):

- Proposed algorithm:
  - Pick random *i* and test that $y_i \leq y_{i+1}$
- Bad input type:
  - 1,2,3,4,5,…i, 1,2,….n-i,
  - Difficult for this algorithm to find "breakpoint"
  - But other tests work well on this input…

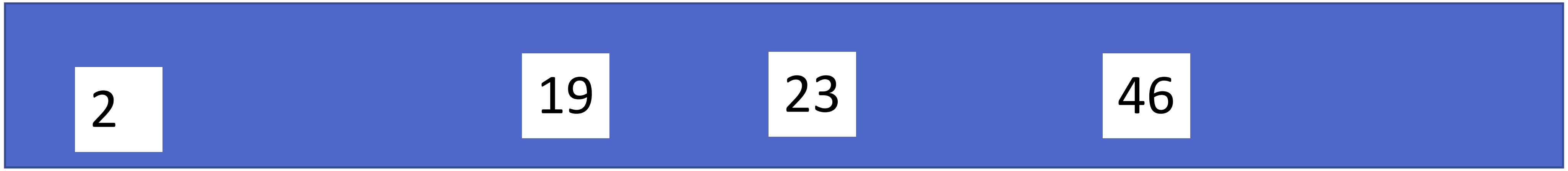


A second try for an algorithm:

Pick lots of random entries and pass if all in right order

Good input type:

1 2 4 5 7 11 14 19 20 21 23 38 39 45 46 50 57 60 61 80

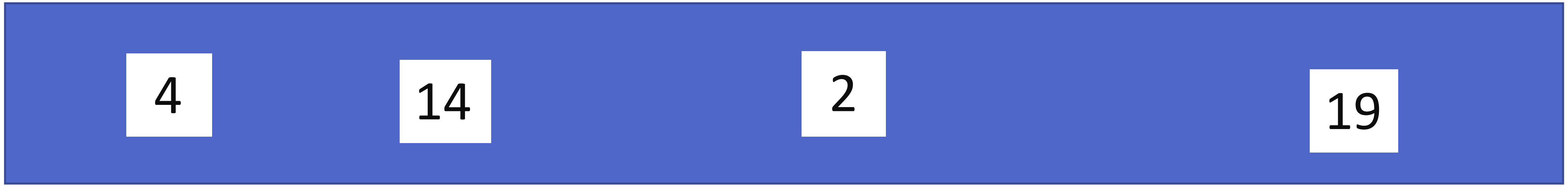


A second try:

Pick lots of random entries and pass if all in right order

Bad input type:

1 2 4 5 7 11 14 19 20 21 1 2 4 5 7 11 14 19 20 21



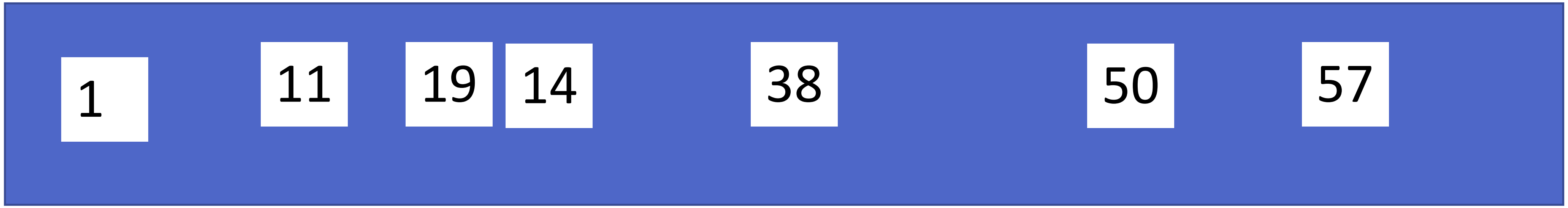
A second try:

Pick lots of random entries and pass if all in right order

How many?

Another bad input type:

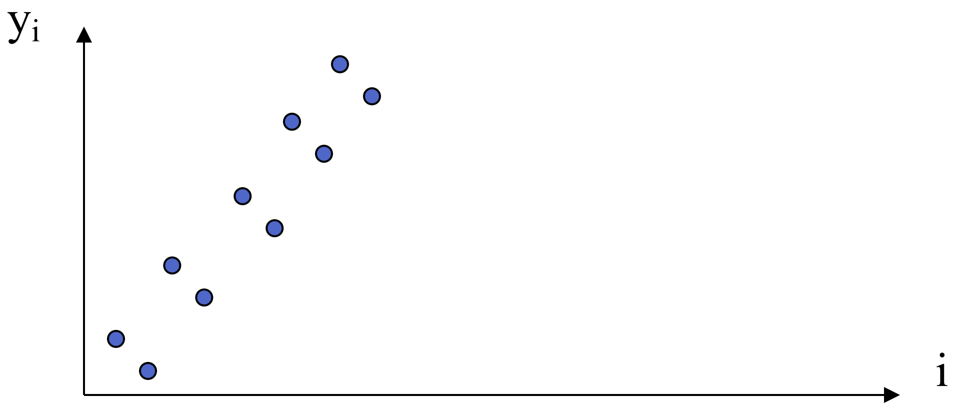
2 1 5 4 11 7 19 14 21 20 38 23 45 39 50 46 60 57 80 61



A second attempt:

- Proposed algorithm:
  - Pick random $i<j$ and test that $y_i \leq y_j$
- Bad input type:
  - $n/2$ groups of pairs of decreasing elements

    2, 1,4,3,6,5,…,2k, 2k-1,…
  - Largest monotone sequence is n/2
  - must pick i,j in same group to see problem
  - need $\Omega(n^{1/2})$ samples. (also $O(n^{1/2})$ is enough)



A third attempt

Before we start… a minor simplification:

- Assume list is distinct (i.e. $x_i \neq x_j$)

- Claim: this is not really easier
  - Why?

    Can "virtually" append i to each x_i

    $x_1, x_2, \ldots x_n \rightarrow (x_1, 1), (x_2, 2), \ldots, (x_n, n)$

    e.g., 1,1,2,6,6 $\rightarrow$ *(1,1),(1,2),(2,3),(6,4),(6,5)*

    Breaks ties without changing order

Well known trick –
often used in parallel algorithms!

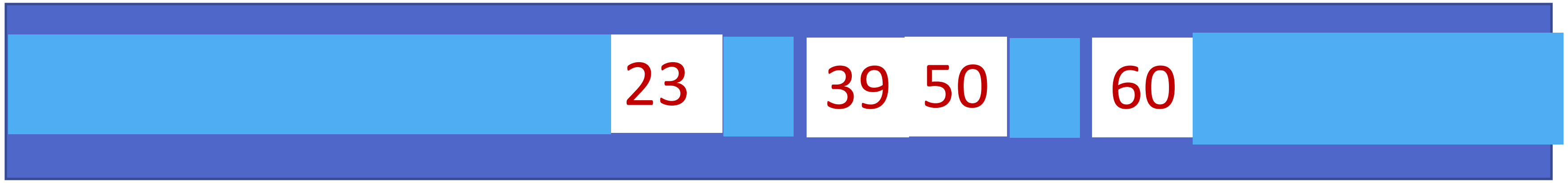
A super fast "random binary search" basic test

- Pick random i and look at value of y_i
- Do binary search for y_i
  - Does the binary search find y_i at location i? If not, FAIL
  - Does the binary search find any inconsistencies? If yes, FAIL

e.g., i = 9, y_i =45

Bad input type:

4 7 19 14 21 20 38 23 45 39 50 46 60 57 80 61



A test that works

- The test:

  Test $O(1/\varepsilon)$ times:
    - Pick random i
    - Look at value of y_i
    - Do binary search for y_i
      - Does the binary search find y_i at location i? If not, FAIL
      - Does the binary search find any inconsistencies? If yes, FAIL

  Pass if never failed

- Running time: $O(\varepsilon^{-1} \log n)$ time
- Why does this work?

Behavior of the test:

- Define index *i* to be good if binary search for y_i successful
- O(1/ε log n) time test (restated):
  - pick *O*(1/ε) *i*'s and pass if they are all good
- Correctness:
  - If list is sorted, then all i's good (uses distinctness) → test always passes
  - If list likely to pass test, then at least (1-ε)*n* i's are good.
    - Main observation: good elements form increasing sequence
      - Proof: if i<j both good need to show $y_i < y_j$
        - let k = least common ancestor of i,j
        - Search for i went left of k and search for j went right of k → $y_i < y_k < y_j$
    - Thus list is ε-close to monotone (delete < ε*n* bad elements)

Requirements for algorithm:

- Pass sorted lists
- Fail lists that are ε-far.
  - Equivalently: if list likely to pass test, can change (delete) at most ε fraction of list to make it sorted

Probability of success > ¾

- Can test in $O(1/\varepsilon \log n)$ time

(and can't do any better!)

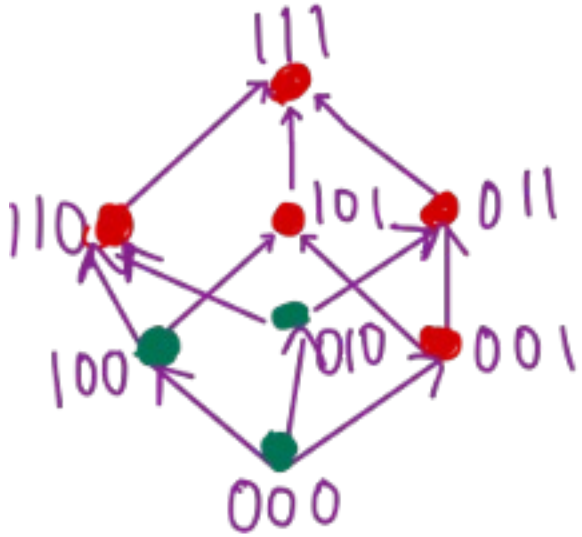
More complicated domain: The Boolean cube

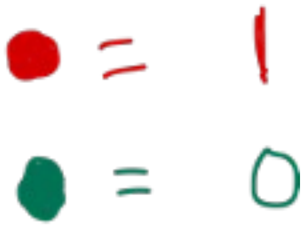
Partial order of Boolean cube:

$$x \prec y \quad \Leftrightarrow (\forall i \; x_i \leq y_i)$$

$f: \{0,1\}^n \rightarrow \{0,1\}$ is **monotone** if:

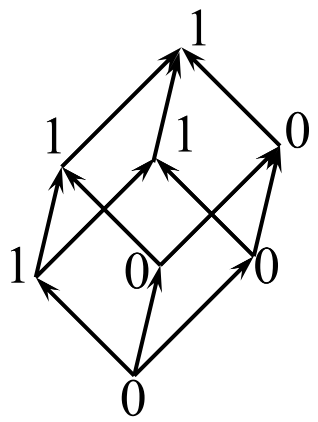
$$x \prec y \Rightarrow f(x) \leq f(y)$$





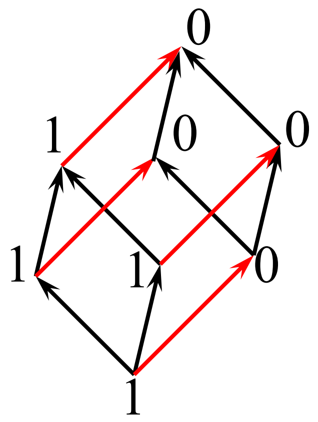
Monotonicity of functions on Boolean cube

- A function $f : \{0,1\}^n \to \{0,1\}$ is monotone
  if increasing a bit of x does not decrease $f(x)$.



monotone

- Violating edge:
  - Edge $x \rightarrow y$ is violated by f if $f(x) > f(y)$.



$\frac{1}{2}$-far from monotone

Time complexity of property tester:

- Today: $O(n/\varepsilon)$, logarithmic in the size of the input, 2^n
- Newer: $\Theta(\sqrt{n}/\varepsilon^2)$ for nonadaptive tests, $\Omega\left(n^{\frac{1}{3}}\right)$

Monotonicity Test

Idea: Show that functions that are far from monotone violate many edges.

EdgeTest (f, ε)

1. Pick $2n/\varepsilon$ edges (x, y) uniformly at random from the hypercube.
2. **Reject** if any (x, y) is violated (i.e. $f(x) > f(y)$). Otherwise, **accept**.

Analysis

- If f is monotone, EdgeTest always accepts.
- If f is ε-far from monotone, will show that $\geq \varepsilon/n$ fraction of edges (i.e., $\frac{\varepsilon}{n} \cdot 2^{n-1} n = \varepsilon 2^{n-1}$ edges) violated by f.
  - Let $V(f)$ denote the number of edges violated by f.

Contrapositive: If $V(f) < \varepsilon\, 2^{n-1}$,
f can be made monotone by changing $< \varepsilon\, 2^{n}$ values.

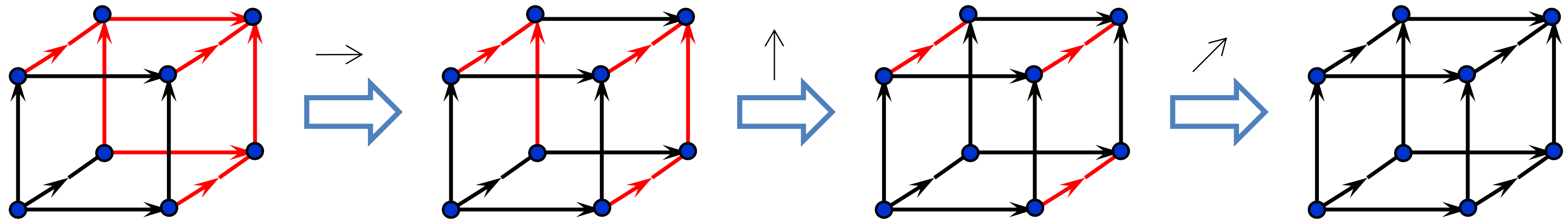
Repair Lemma:
f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

Repair Lemma: Proof Idea

Repair Lemma

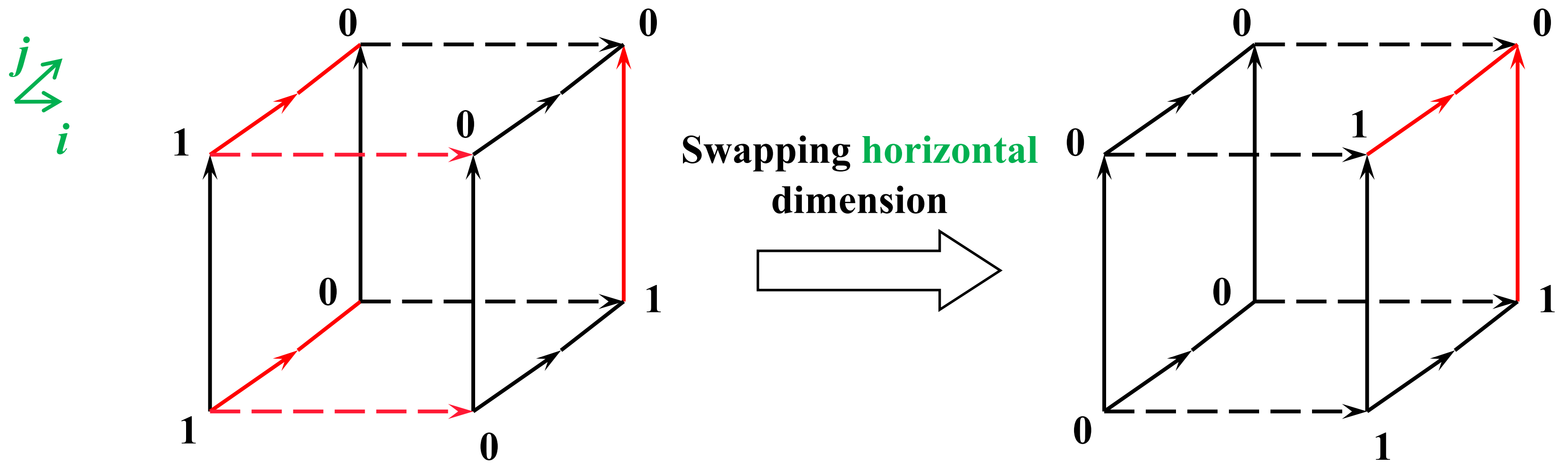
f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

Proof idea: Transform f into a monotone function by repairing edges in one dimension at a time.



Repairing Violated Edges in One Dimension

Swap violated edges $1 \to 0$ in one dimension to $0 \to 1$.



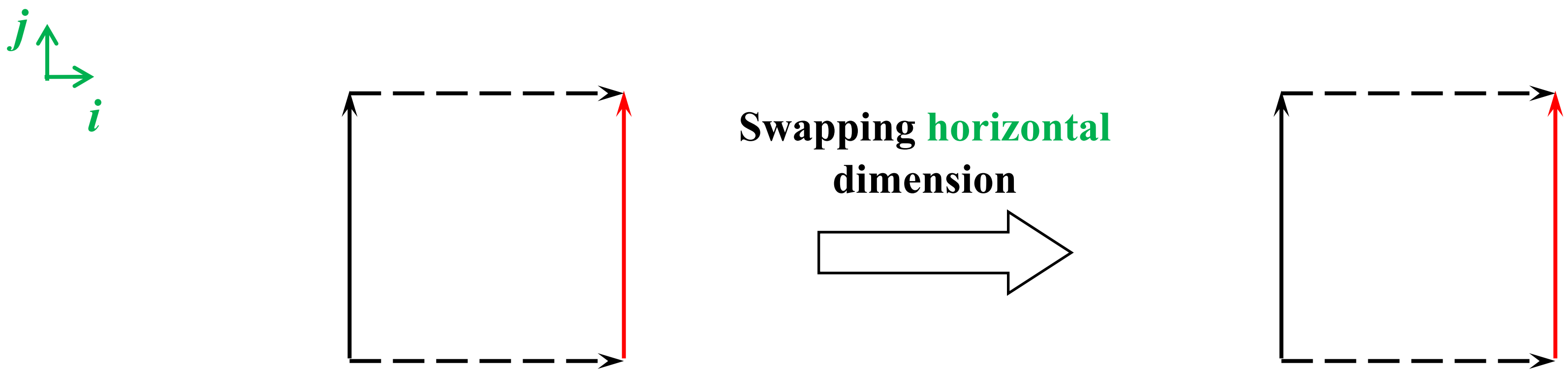
Let V_j = # of violated edges in dimension j

Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$

Enough to prove the claim for squares

Proof of The Claim for Squares

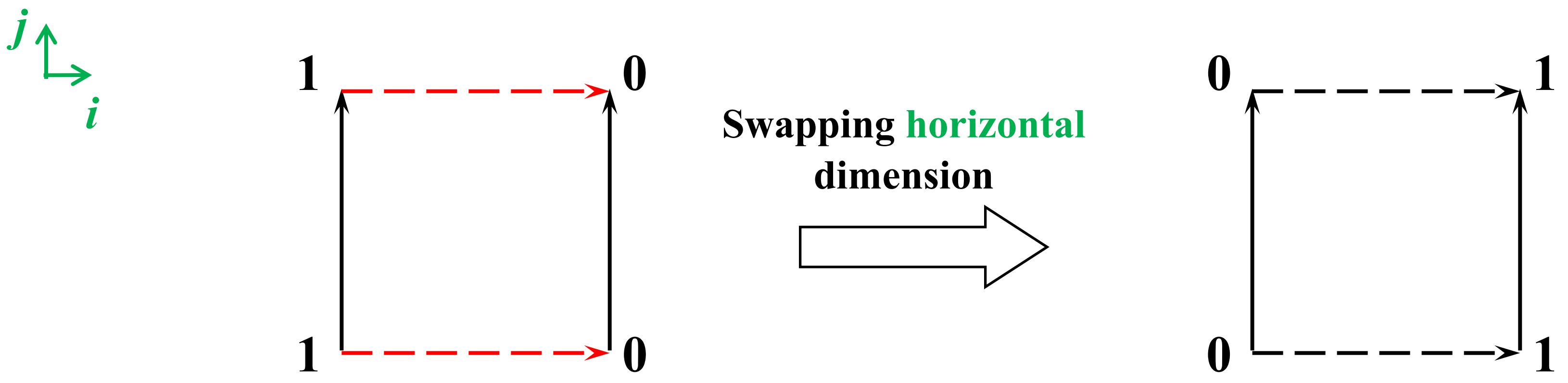
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- If no horizontal edges are violated, no action is taken.

Proof of The Claim for Squares

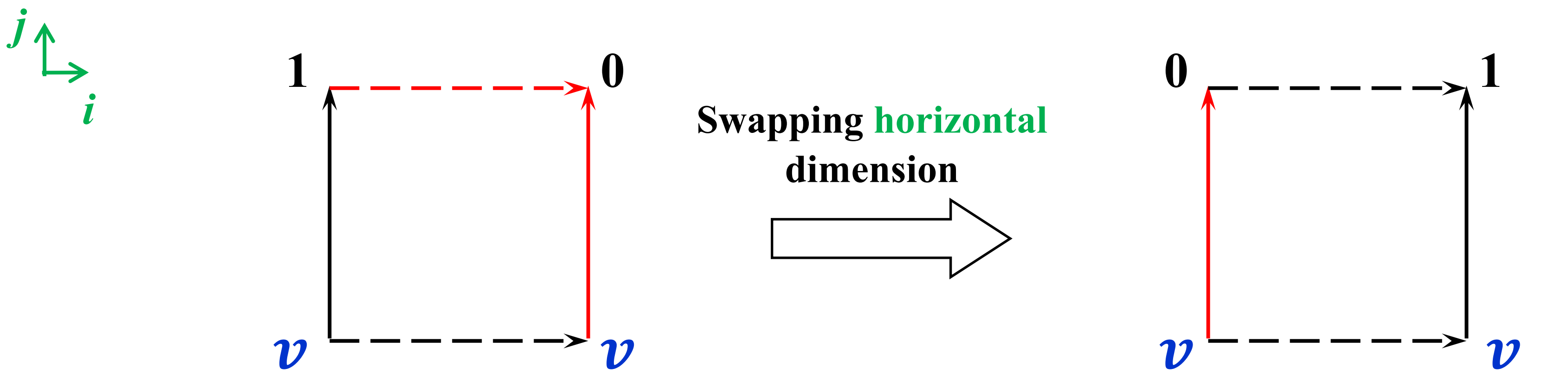
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- If both horizontal edges are violated, both are swapped, so the number of vertical violated edges does not change.

Proof of The Claim for Squares

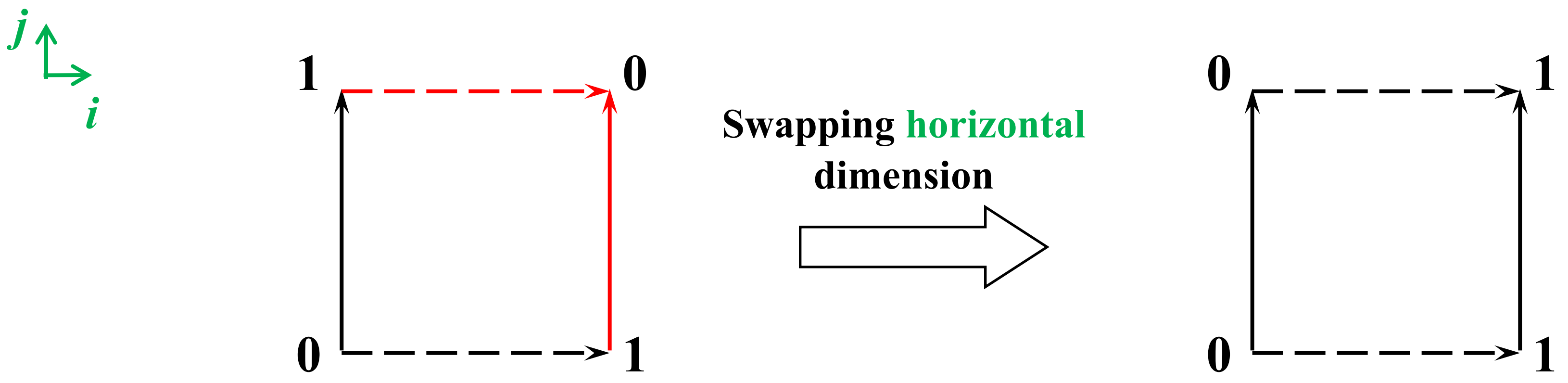
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- Suppose one (say, top) horizontal edge is violated.
- If both bottom vertices have the same label, the vertical edges get swapped.

Proof of The Claim for Squares

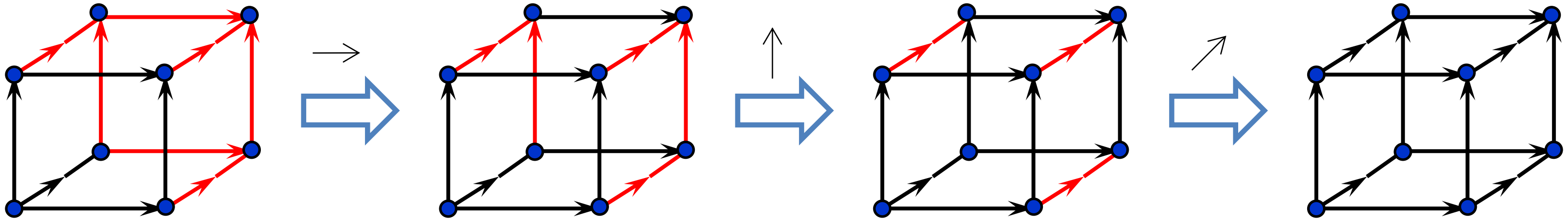
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$ ✔



- Suppose one (say, top) horizontal edge is violated.
- If both bottom vertices have the same label, the vertical edges get swapped.
- Otherwise, the bottom vertices are labeled 0→1, and the vertical violation is repaired.

Proof of The Claim for Squares

Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$ ✓



After we perform swaps in all dimensions:

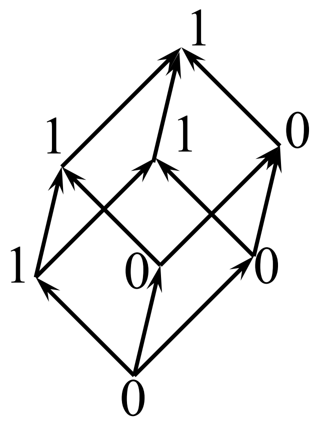
- f becomes monotone
- # of values changed:
  $2 \cdot V_1 + 2 \cdot$ (# violated edges in dim 2 after swapping dim 1)
  $+ 2 \cdot$ (# violated edges in dim 3 after swapping dim 1 and 2)
  $+ \ldots \leq 2 \cdot V_1 + 2 \cdot V_2 + \cdots 2 \cdot V_n = 2 \cdot V(f)$

Repair Lemma ✓

f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

- Can improve the bound by a factor of 2.

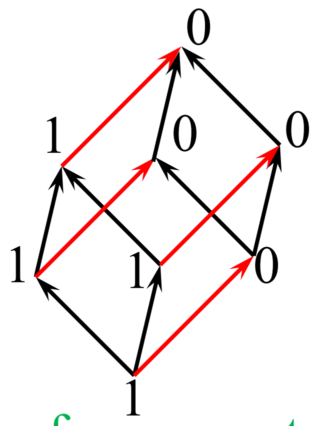
Testing if a Functions $f : \{0,1\}^n \rightarrow \{0,1\}$ is monotone



monotone

Monotone or

ε-far from monotone?



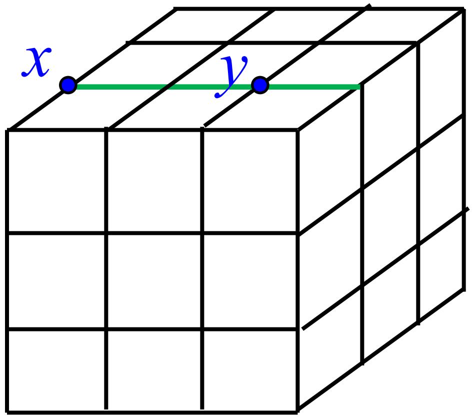
$\frac{1}{2}$-far from monotone

O(n/ε) time

(logarithmic in the size of the input)

Testing Properties of High-Dimensional Functions

In polylogarithmic time, we can test a large class of properties of functions $f: \{1, \ldots, n\}^d \to \mathbb{R}$, including:



- Lipschitz property
- Bounded-derivative properties
- Unateness