Data Structures Meet Circuits and Cryptography

Alexander Golovnev, Georgetown University

SACC 2021

Motivating Question

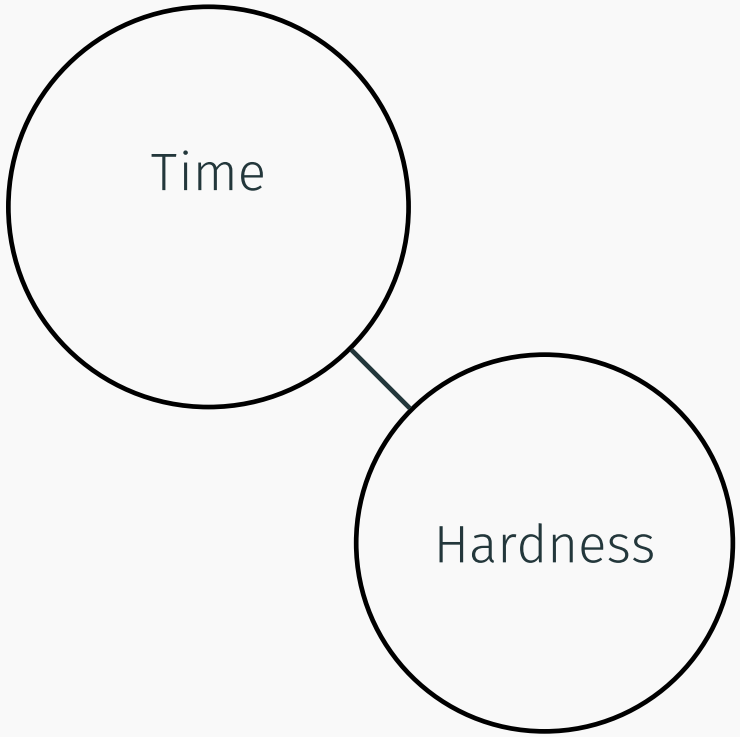
What resources are required to solve a given computational problem?

Motivating Question

What resources are required to solve a given computational problem?

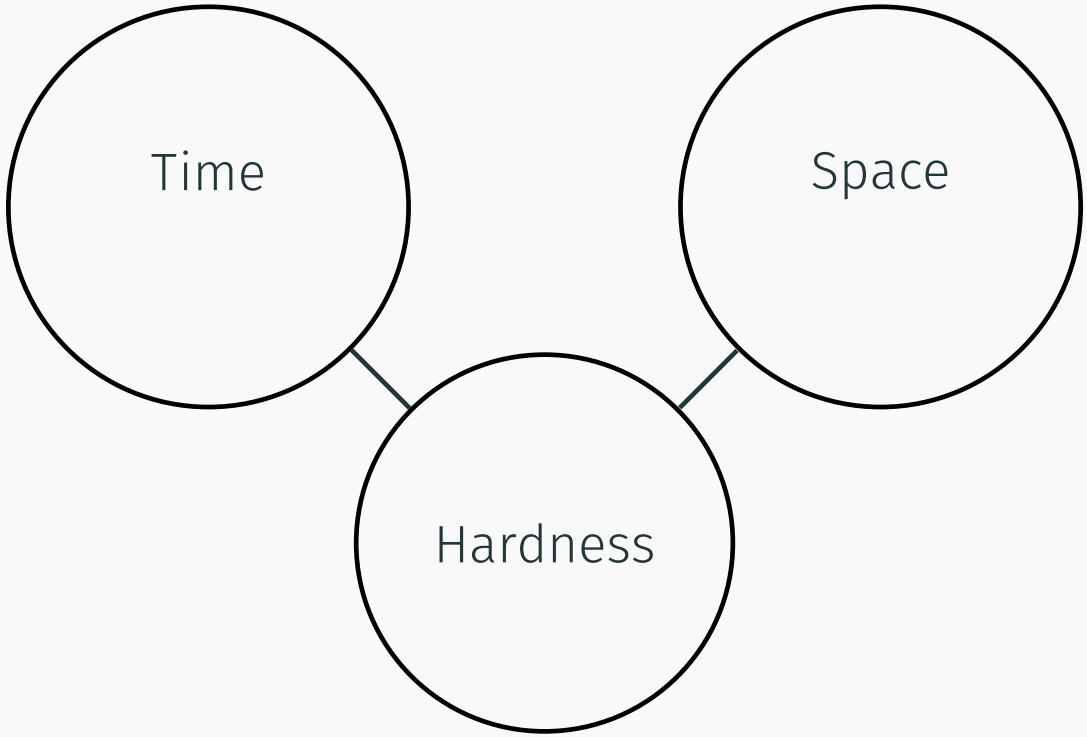
Motivating Question

What resources are required to solve a given computational problem?



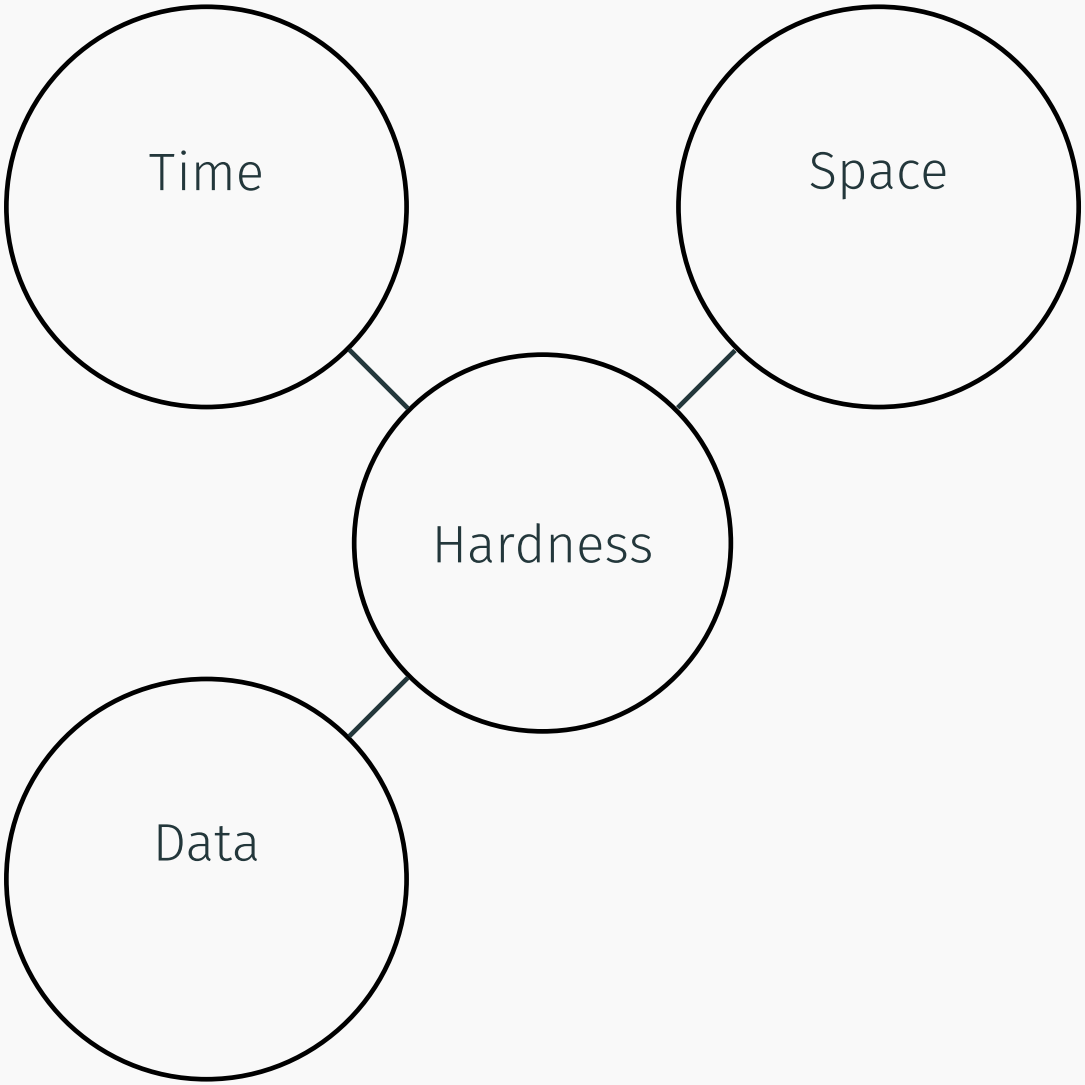
Motivating Question

What resources are required to solve a given computational problem?



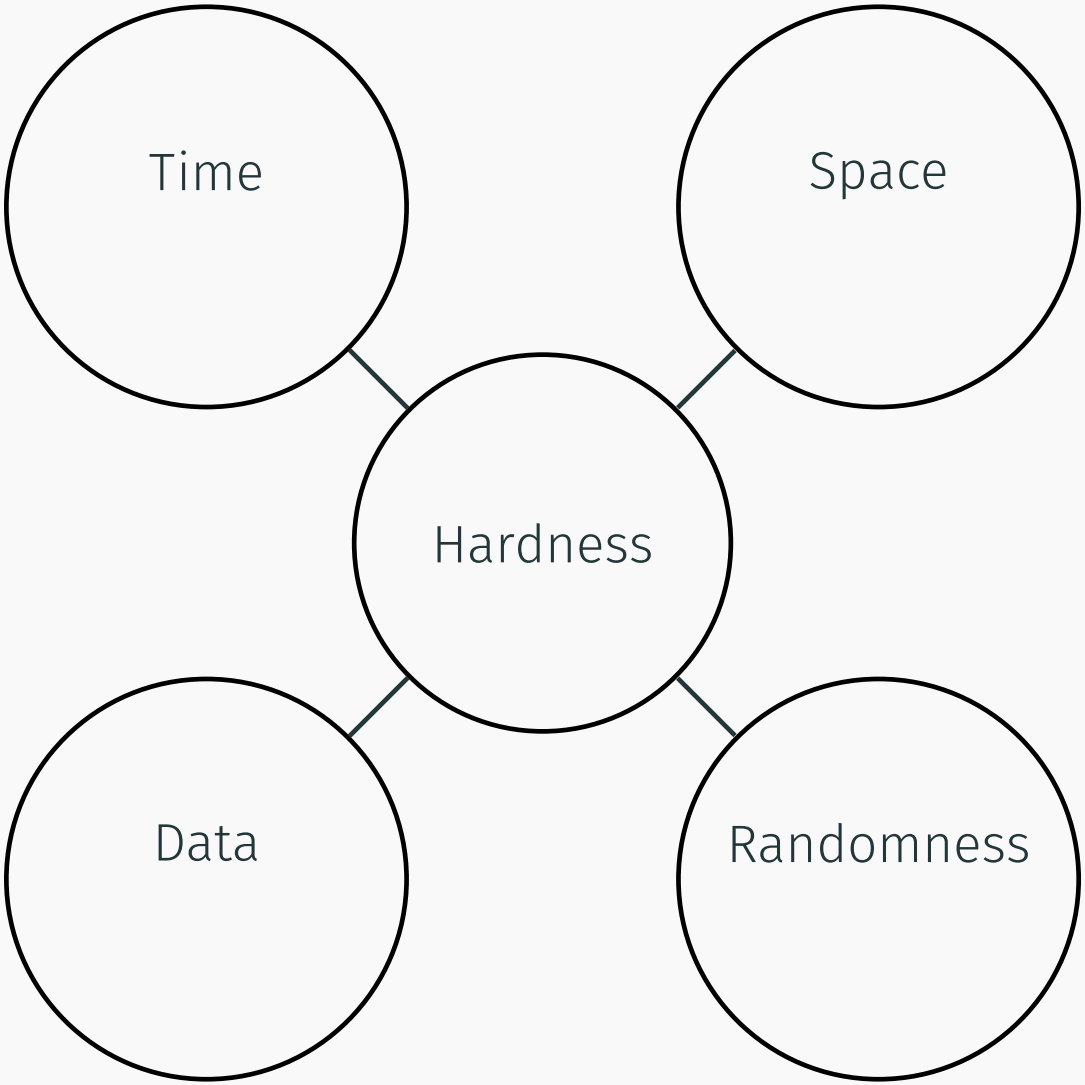
Motivating Question

What resources are required to solve a given computational problem?



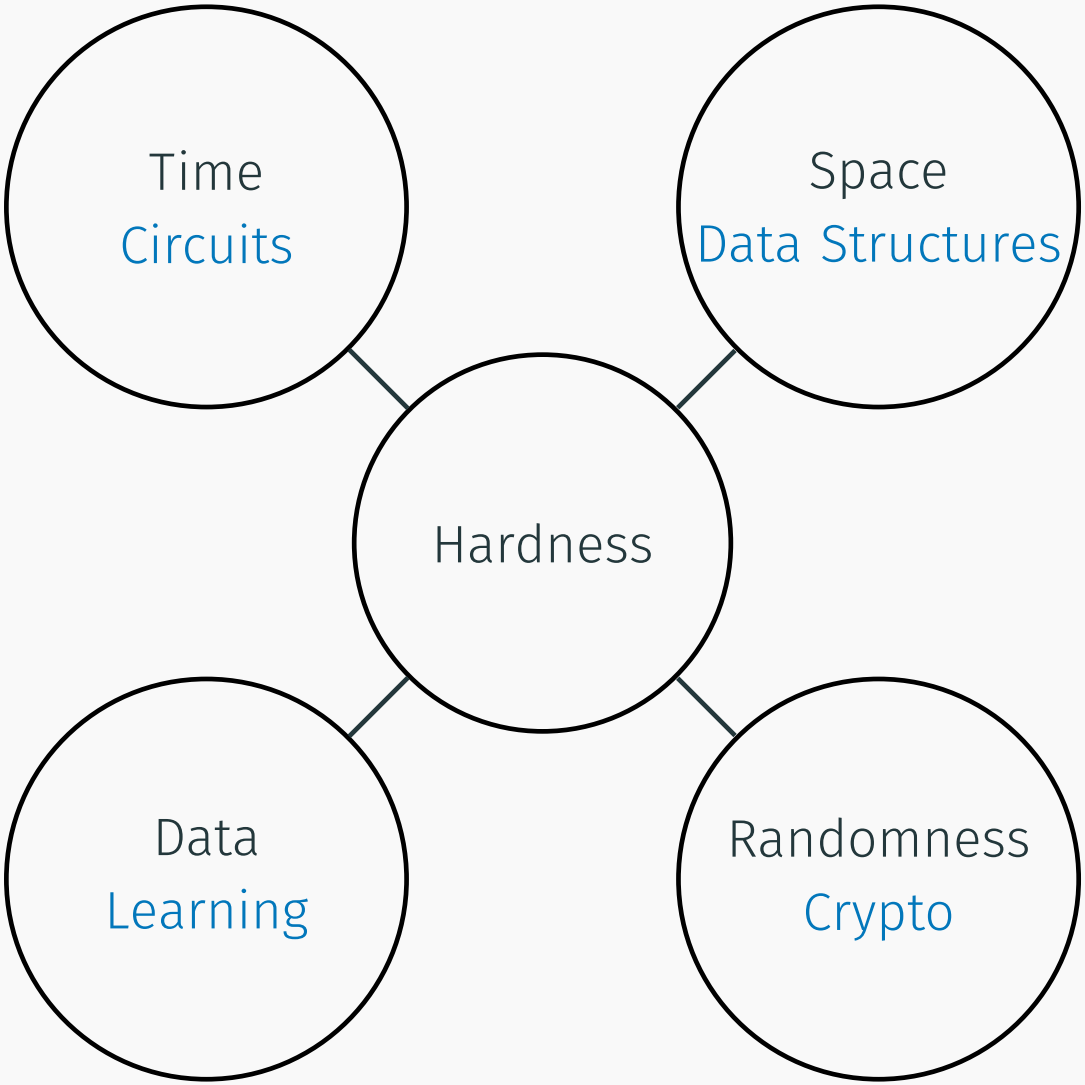
Motivating Question

What resources are required to solve a given computational problem?



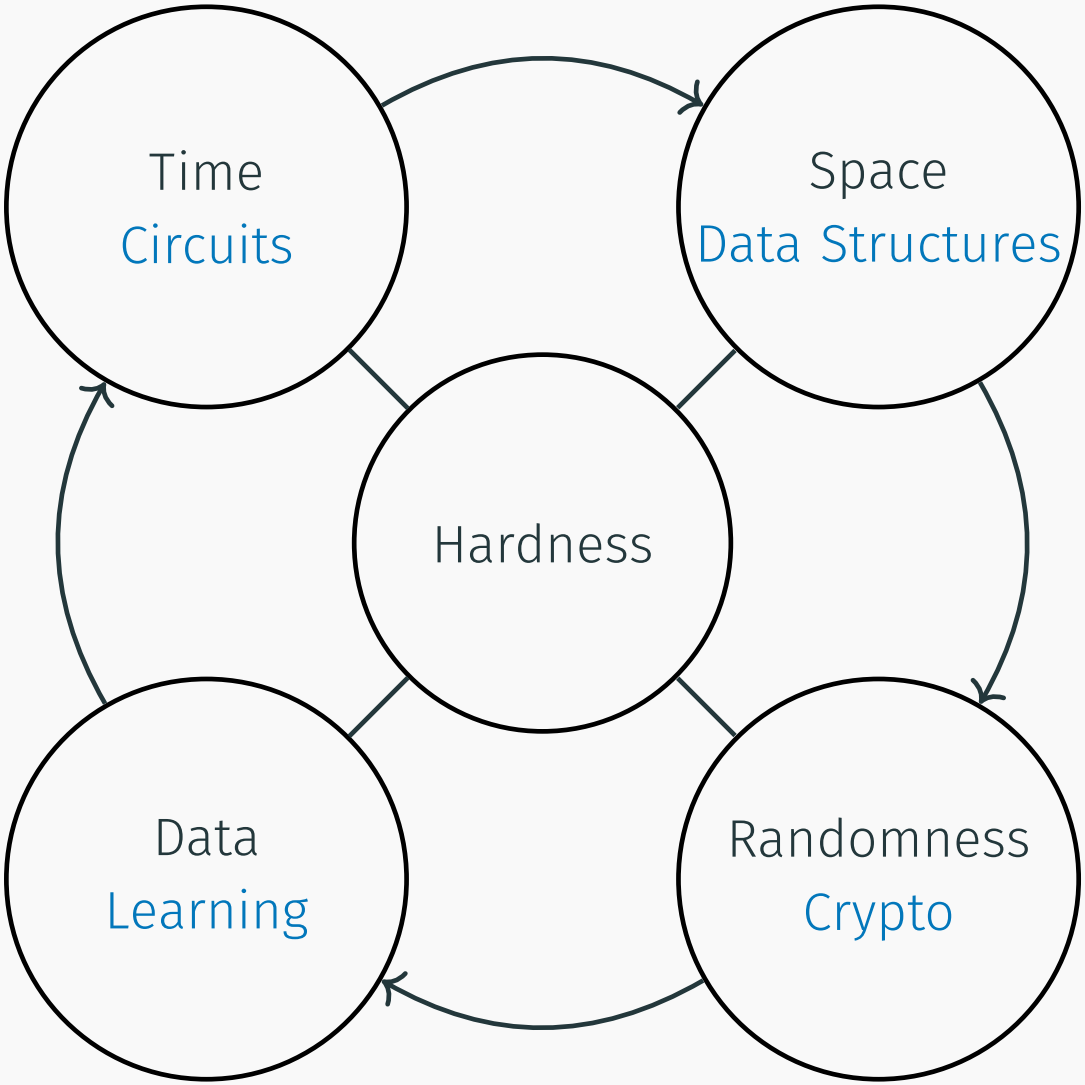
Outline

What resources are required to solve a given computational problem?



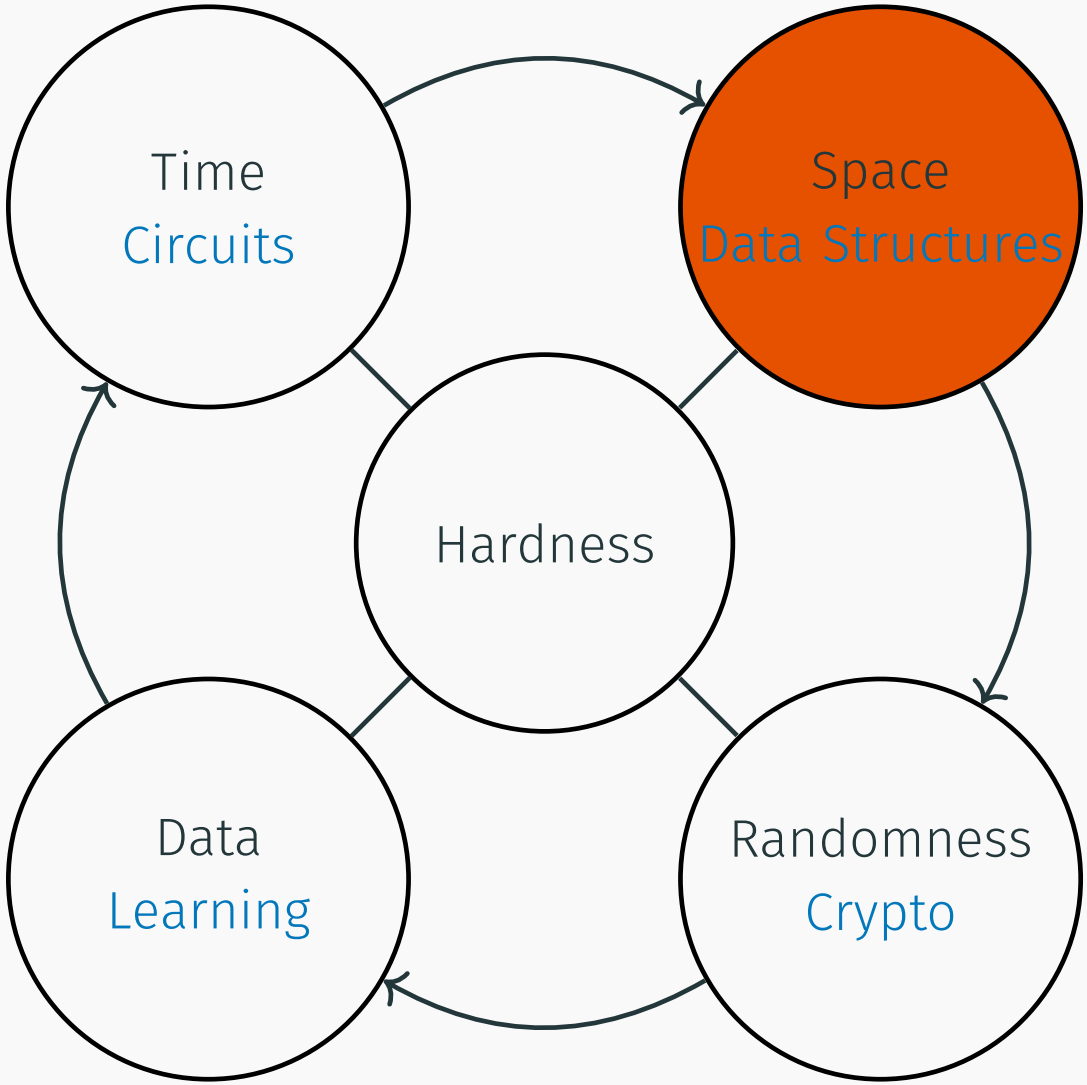
Outline

What resources are required to solve a given computational problem?



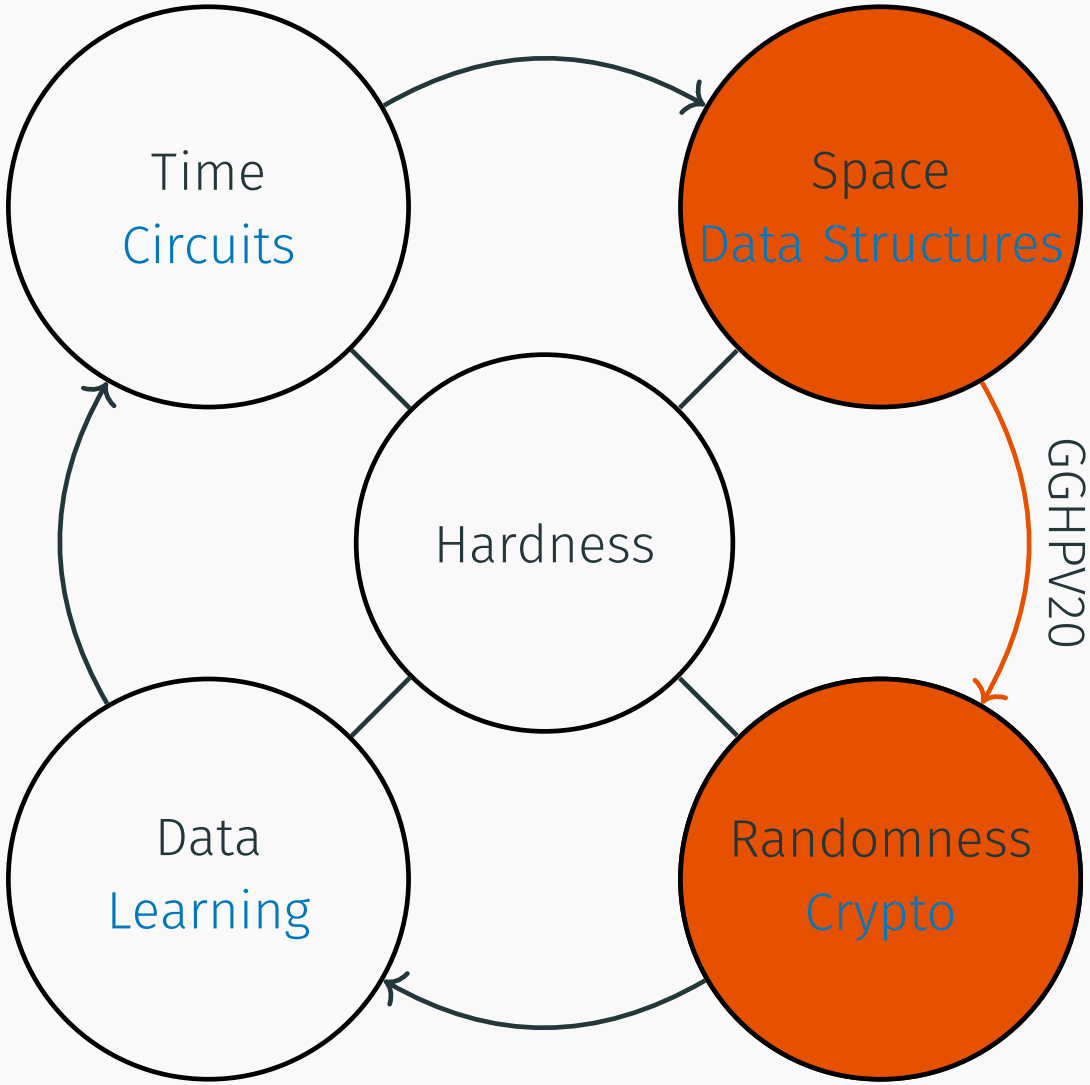
Outline

What resources are required to solve a given computational problem?



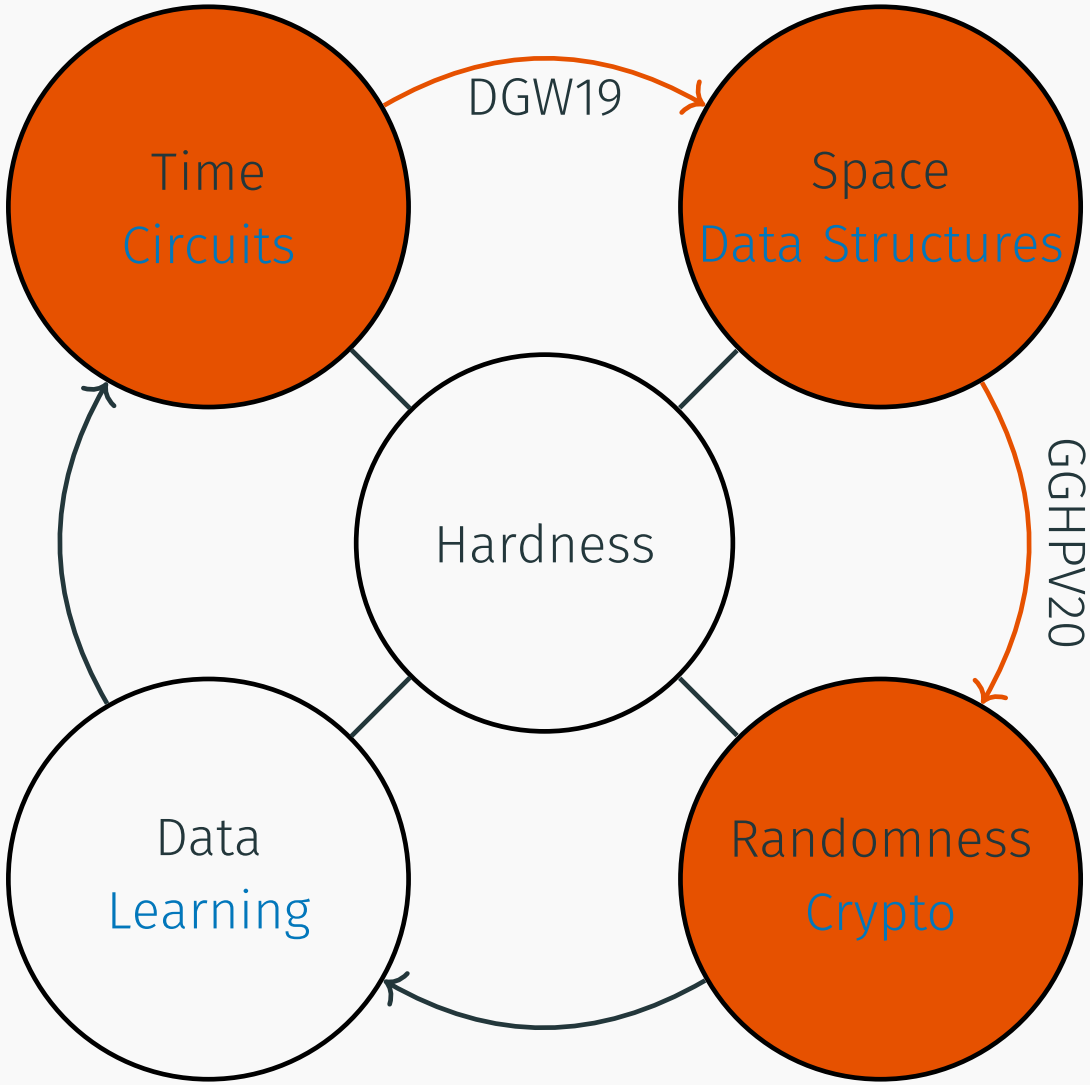
Outline

What resources are required to solve a given computational problem?

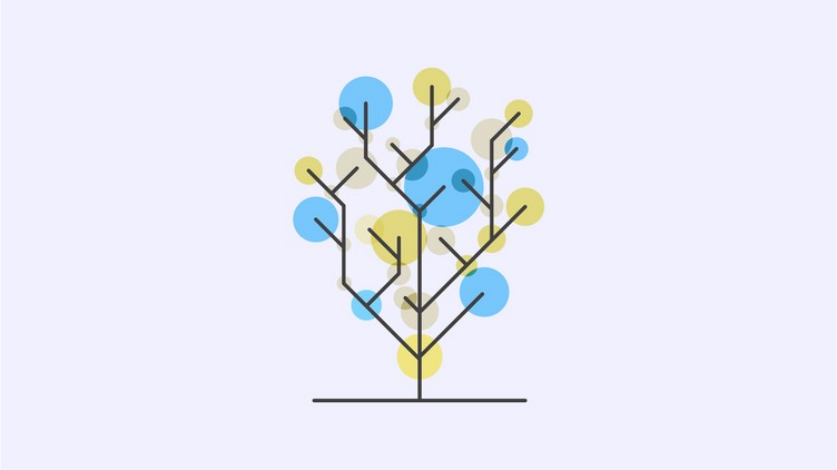


Outline

What resources are required to solve a given computational problem?

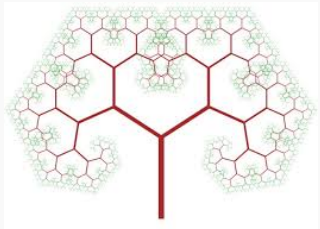


DATA STRUCTURES



Examples

Stack, Queue, List, Heap



Search Trees

```
hash(unsigned x) {
   x ^= x >> (w-m);
   return (a*x) >> (w-m);
}
```

Hash Tables

Static Data Structures. Examples

- Graph Distances: Preprocess a road network in order to efficiently compute distances between cities (Google Maps)

Static Data Structures. Examples

- Graph Distances: Preprocess a road network in order to efficiently compute distances between cities (Google Maps)
- Nearest Neighbors: Preprocess a set of points in order to efficiently find closest point to a query point (Netflix recommendations)

Static Data Structures. Examples

- Graph Distances: Preprocess a road network in order to efficiently compute distances between cities
  (Google Maps)
- Nearest Neighbors: Preprocess a set of points in order to efficiently find closest point to a query point
  (Netflix recommendations)
- Range Counting: Preprocess a set of points in order to efficiently compute the number of points in a given rectangle
  (Amazon market size estimation)

Static Data Structures

Preprocessing

Static Data Structures

Queries

Preprocessing

Static Data Structures

Queries

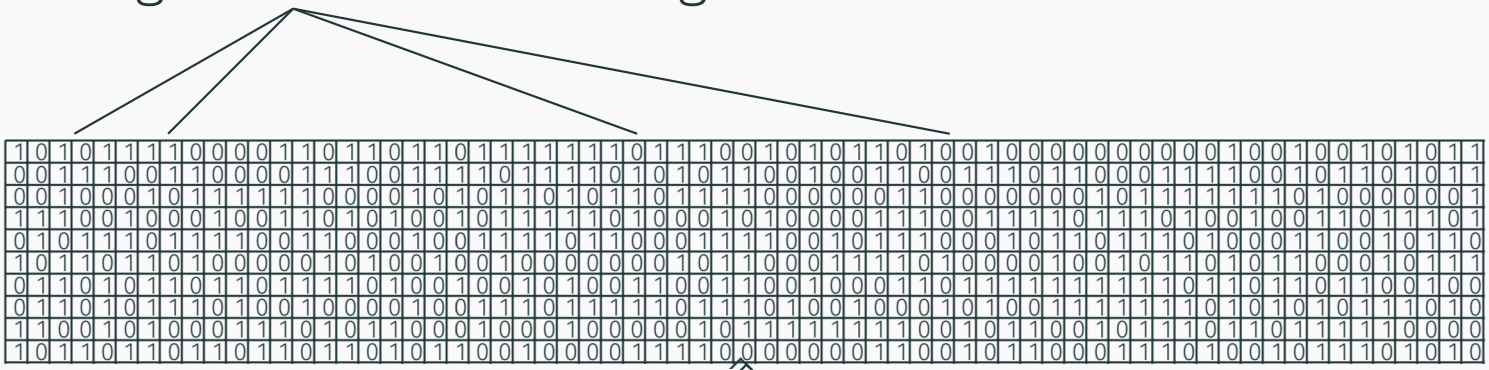
Washington — St. Petersburg

Preprocessing

Static Data Structures

Queries

Washington — St. Petersburg



Preprocessing

Static Data Structures

Queries

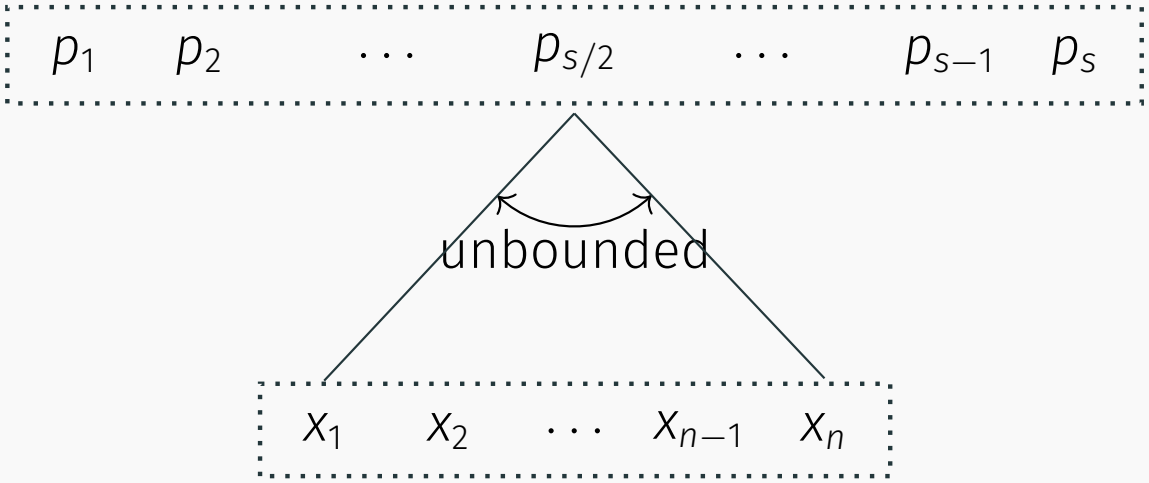
Washington — St. Petersburg

Athens — Vilnius

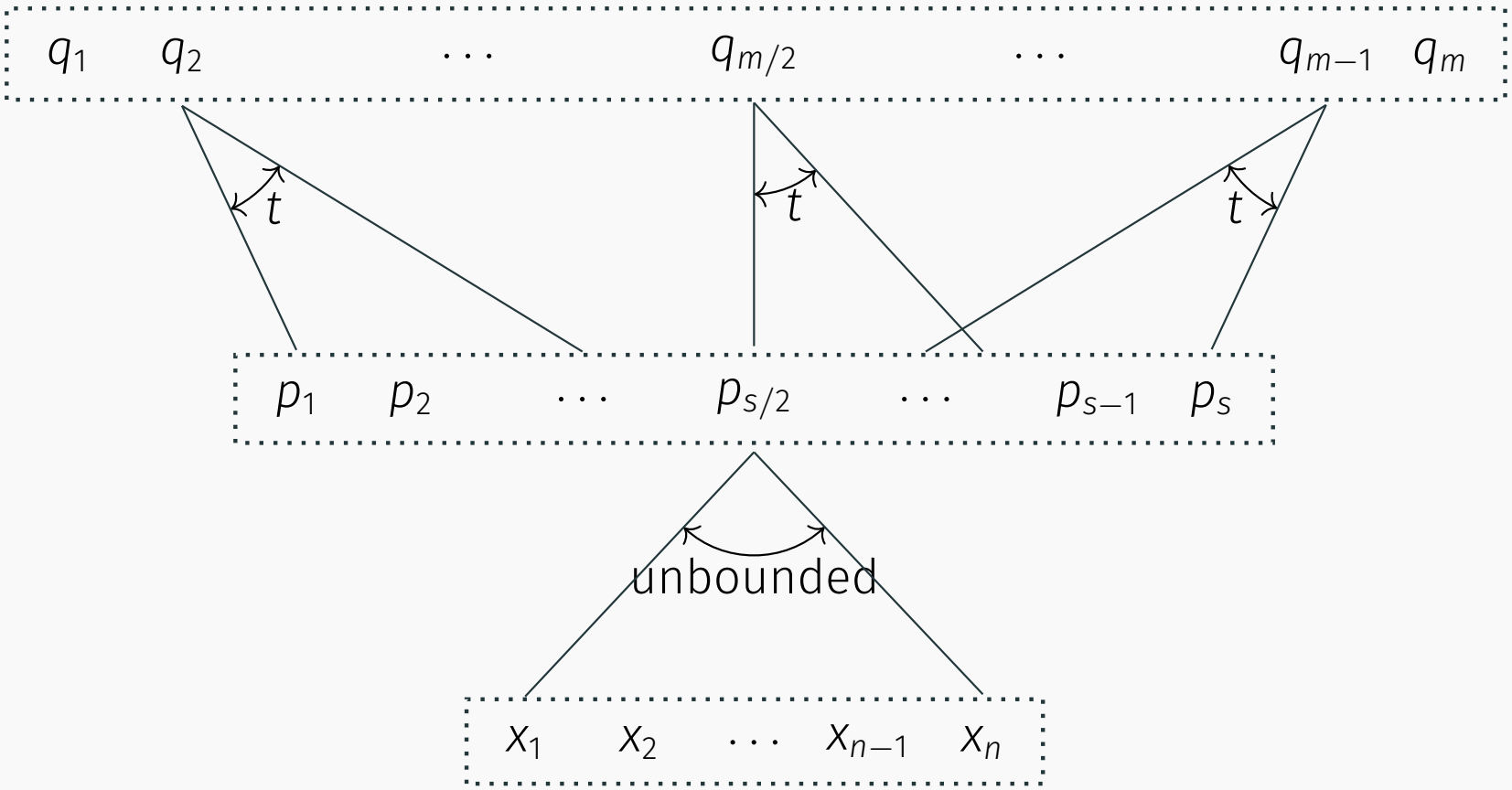
Preprocessing

Static Data Structures. Definition

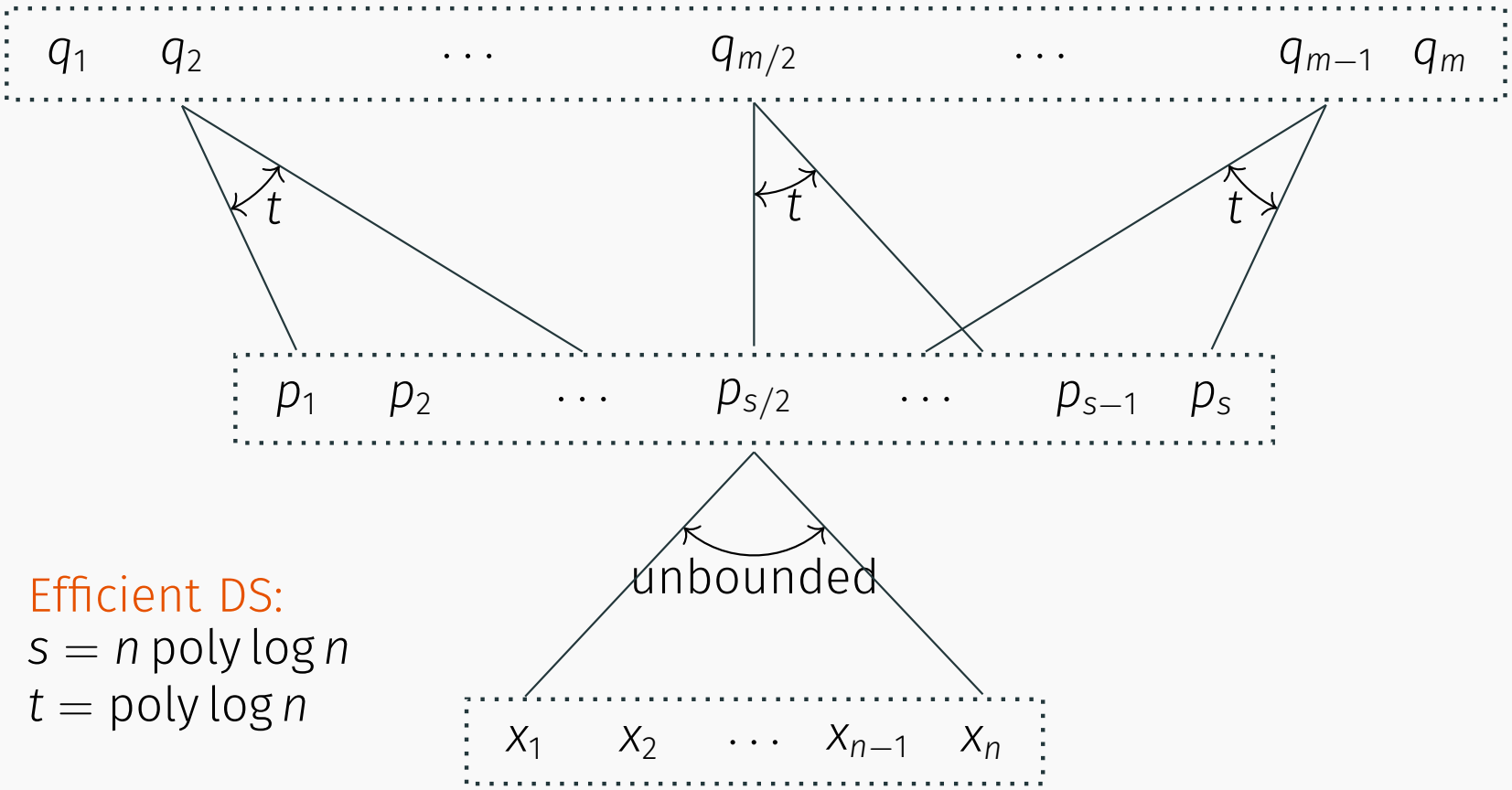
Static Data Structures. Definition



Static Data Structures. Definition



Static Data Structures. Definition



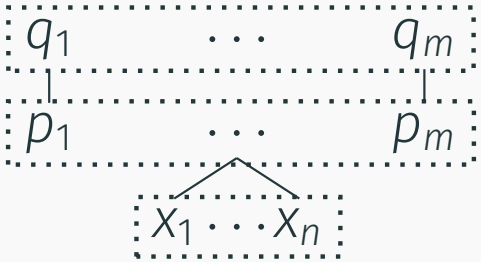
Efficient DS:
$s = n \operatorname{poly}\log n$
$t = \operatorname{poly}\log n$

DS Lower Bounds

- Two trivial solutions:

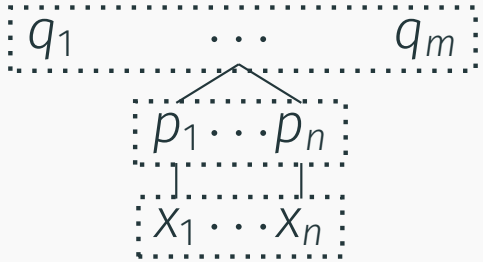
DS Lower Bounds

- Two trivial solutions:
  - $s = m, t = 1$



DS Lower Bounds

- Two trivial solutions:
  - $s = m, t = 1$
  - $s = n, t = n$



DS Lower Bounds

- Two trivial solutions:
  - $s = m, t = 1$
  - $s = n, t = n$
- There exist problems requiring $s \approx m$ or $t \approx n$

DS Lower Bounds

- Two trivial solutions:
  - $s = m, t = 1$
  - $s = n, t = n$
- There exist problems requiring $s \approx m$ or $t \approx n$
- Best known concrete lower bound [Sie89]:

$$t \geq \Omega\left(\frac{\log m}{\log(s/n)}\right)$$

DS Lower Bounds

- Two trivial solutions:
  - $s = m, t = 1$
  - $s = n, t = n$
- There exist problems requiring $s \approx m$ or $t \approx n$
- Best known concrete lower bound [Sie89]:

$$t \geq \Omega\left(\frac{\log m}{\log(s/n)}\right)$$

- $s = O(n) \implies t \geq \Omega(\log n)$

DS Lower Bounds

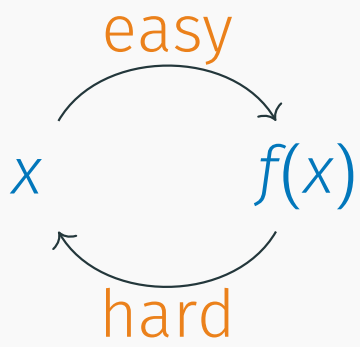
- Two trivial solutions:
  - $s = m, t = 1$
  - $s = n, t = n$
- There exist problems requiring $s \approx m$ or $t \approx n$
- Best known concrete lower bound [Sie89]:

$$t \geq \Omega\left(\frac{\log m}{\log(s/n)}\right)$$

- $s = O(n) \implies t \geq \Omega(\log n)$
- $s = n^{1+\varepsilon} \implies t \geq \Omega(1)$

CRYPTO WITH PREPROCESSING

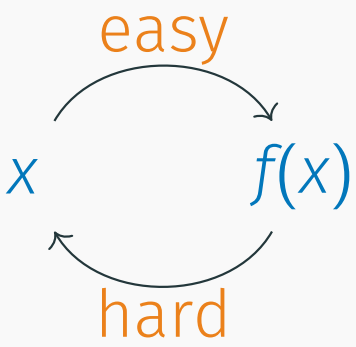
Cryptography with Preprocessing



Cryptography requires hard functions.

Secure (one-way) f: $\{0,1\}^n \to \{0,1\}^n$

Cryptography with Preprocessing



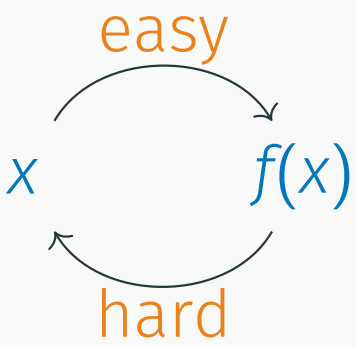
Cryptography requires hard functions.

Secure (one-way) f: $\{0,1\}^n \to \{0,1\}^n$

Can be broken by preprocessing attacks.

E.g., with space $S = 2^n$

Cryptography with Preprocessing



Cryptography requires hard functions.

Secure (one-way) $f\colon \{0,1\}^n \to \{0,1\}^n$

Can be broken by preprocessing attacks.

E.g., with space $S = 2^n$

Can we immunize a function against large preprocessing? Design $g\colon \{0,1\}^{5n} \to \{0,1\}^{5n}$ which is still secure?

Data Structures and Cryptography

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds (for problems with efficient query generation).

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds (for problems with efficient query generation).

Examples of such problems include

- 3SUM
- Nearest Neighbors
- Polygon containment
- ...

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds (for problems with efficient query generation).

Examples of such problems include

- 3SUM
- Nearest Neighbors
- Polygon containment
- ...

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

Foundation of computational geometry.

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

Foundation of computational geometry. Easy to solve in time $O(n^2)$

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

Foundation of computational geometry. Easy to solve in time $O(n^2)$

Strong 3SUM Conjecture

No $o(n^2)$-algorithm for 3SUM [GO95]

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

Foundation of computational geometry. Easy to solve in time $O(n^2)$

Strong 3SUM Conjecture

No $o(n^2)$-algorithm for 3SUM [GO95]

Theorem [BDP08,GP14,C18]

3SUM can be solved in time $\approx n^2/\log^2 n$

3SUM. Algorithmic version

3SUM

Given $a_1, \ldots, a_n$, check whether there exist $a_i + a_j = a_k$

Foundation of computational geometry. Easy to solve in time $O(n^2)$

Strong 3SUM Conjecture

No $o(n^2)$-algorithm for 3SUM [GO95]

Theorem [BDP08,GP14,C18]

3SUM can be solved in time $\approx n^2/\log^2 n$

Modern 3SUM Conjecture

No $n^{1.99}$-algorithm for 3SUM [GO95]

3SUM. Data Structure version

3SUM-Indexing

Preprocess $a_1, \ldots, a_n$ into S numbers, s.t. given b, can check $a_i + a_j = b$ in time T

3SUM. Data Structure version

3SUM-Indexing

Preprocess $a_1, \ldots, a_n$ into S numbers, s.t. given b, can check $a_i + a_j = b$ in time T

Easy to solve with $S = n^2$ and $T = \log n$

3SUM. Data Structure version

3SUM-Indexing

Preprocess $a_1, \ldots, a_n$ into S numbers, s.t. given b, can check $a_i + a_j = b$ in time T

Easy to solve with $S = n^2$ and $T = \log n$
Easy to solve with $S = n$ and $T = n$

3SUM. Data Structure version

3SUM-Indexing

Preprocess $a_1, \ldots, a_n$ into S numbers, s.t. given b, can check $a_i + a_j = b$ in time T

Easy to solve with $S = n^2$ and $T = \log n$
Easy to solve with $S = n$ and $T = n$

3SUM-Indexing Conjecture [GKLP17]

Any data structure with $S = n^{1.99}$ requires $T \geq \Omega(n)$

3SUM. Data Structure version

3SUM-Indexing

Preprocess $a_1, \ldots, a_n$ into S numbers, s.t. given b, can check $a_i + a_j = b$ in time T

Easy to solve with $S = n^2$ and $T = \log n$
Easy to solve with $S = n$ and $T = n$

3SUM-Indexing Conjecture [GKLP17]

Any data structure with $S = n^{1.99}$ requires $T \geq \Omega(n)$

Theorem [GGHPV20]

3SUM-Indexing can be solved with $S = n^{1.9}$ and $T = n^{0.3}$

Inverting a Function

- Let $f\colon [N] \to [N]$ be a function

Inverting a Function

- Let f: $[N] \to [N]$ be a function
- Preprocess f in space S, s.t. can invert f in time T

Inverting a Function

- Let $f\colon [N] \to [N]$ be a function
- Preprocess f in space S, s.t. can invert f in time T
- Invert f: given y find x s.t. $f(x) = y$

Inverting a Function

- Let $f\colon [N] \to [N]$ be a function
- Preprocess f in space S, s.t. can invert f in time T
- Invert f: given y find x s.t. $f(x) = y$
- [H80, FN00]: Invert a function with $S^3T = \widetilde{O}(N^3)$

Inverting a Function

- Let $f\colon [N] \to [N]$ be a function
- Preprocess f in space S, s.t. can invert f in time T
- Invert f: given y find x s.t. $f(x) = y$
- [H80, FN00]: Invert a function with $S^3T = \widetilde{O}(N^3)$
- E.g., $S = T = N^{3/4}$

Inverting a Function

- Let $f\colon [N] \to [N]$ be a function
- Preprocess f in space S, s.t. can invert f in time T
- Invert f: given y find x s.t. $f(x) = y$
- [H80, FN00]: Invert a function with $S^3T = \widetilde{O}(N^3)$
- E.g., $S = T = N^{3/4}$
- Special case: Invert a bijection with $S = T = \widetilde{O}(\sqrt{N})$

Inverting a Bijection

- Let f: $[N] \to [N]$ be a bijection

Inverting a Bijection

- Let $f\colon [N] \to [N]$ be a bijection
- [H80,FN00] invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$

Inverting a Bijection

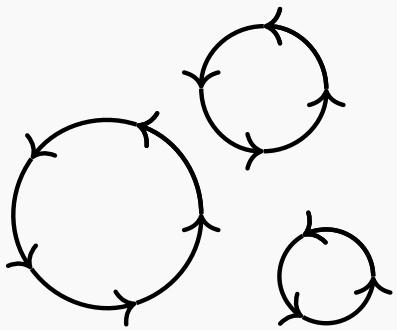
- Let f: $[N] \to [N]$ be a bijection
- [H80,FN00] invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$

Inverting a Bijection

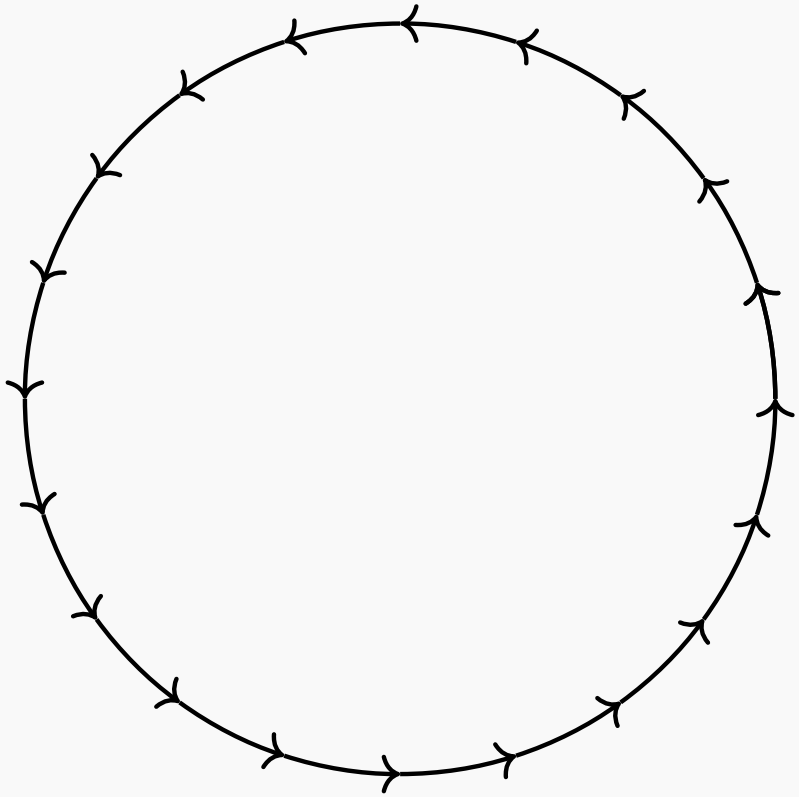
- Let $f\colon [N] \to [N]$ be a bijection
- [H80,FN00] invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1

Inverting a Bijection

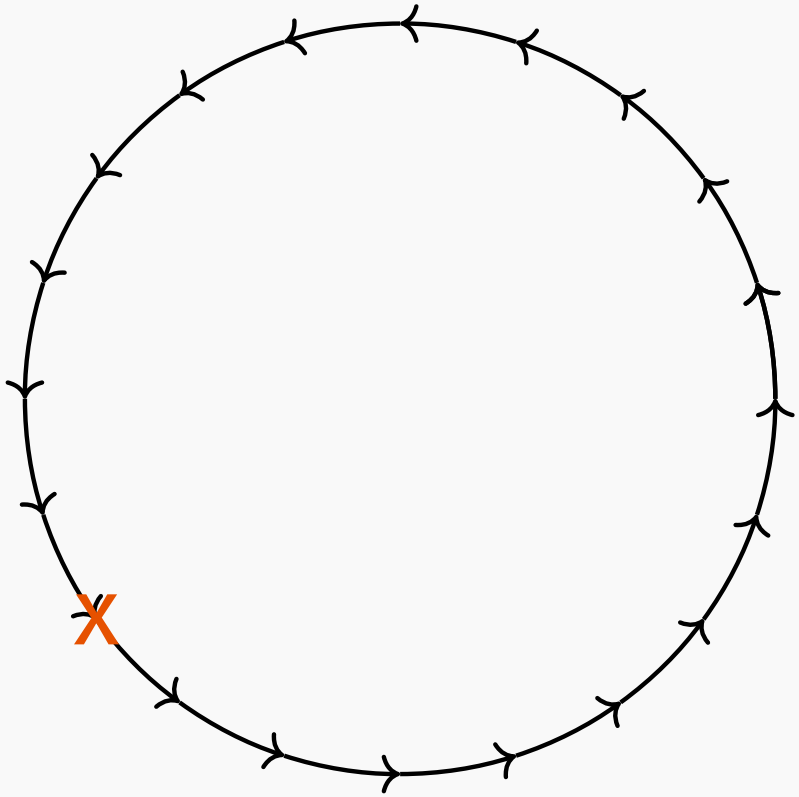
- Let $f\colon [N] \to [N]$ be a bijection
- [H80,FN00] invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1
- Thus, this graph is a union of cycles



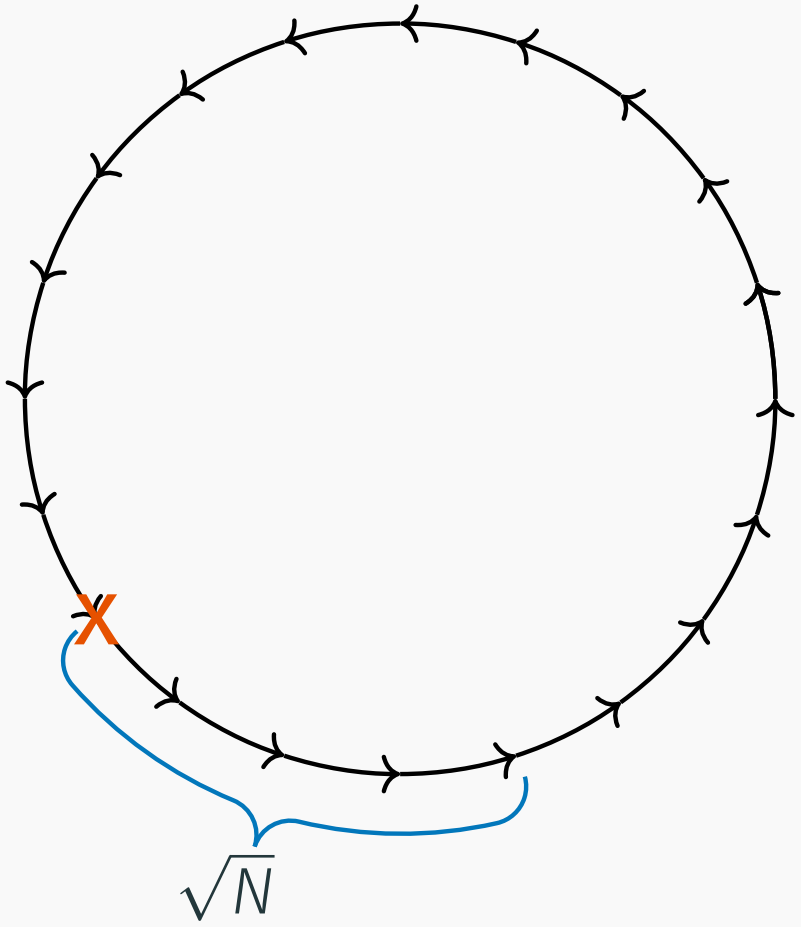
Inverting a Bijection



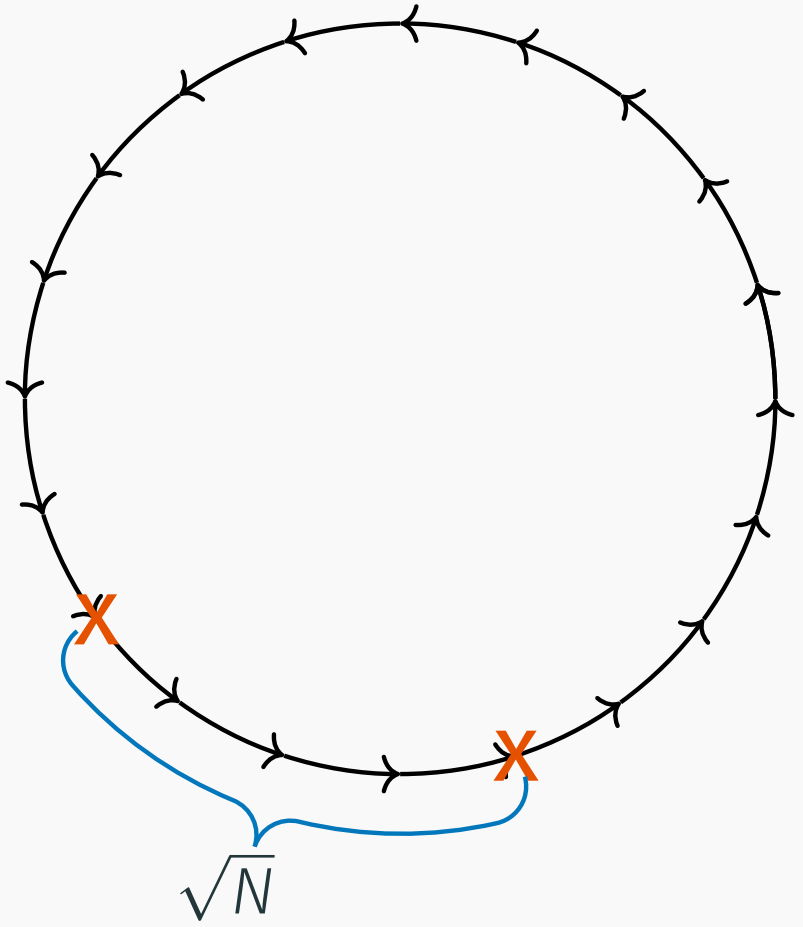
Inverting a Bijection



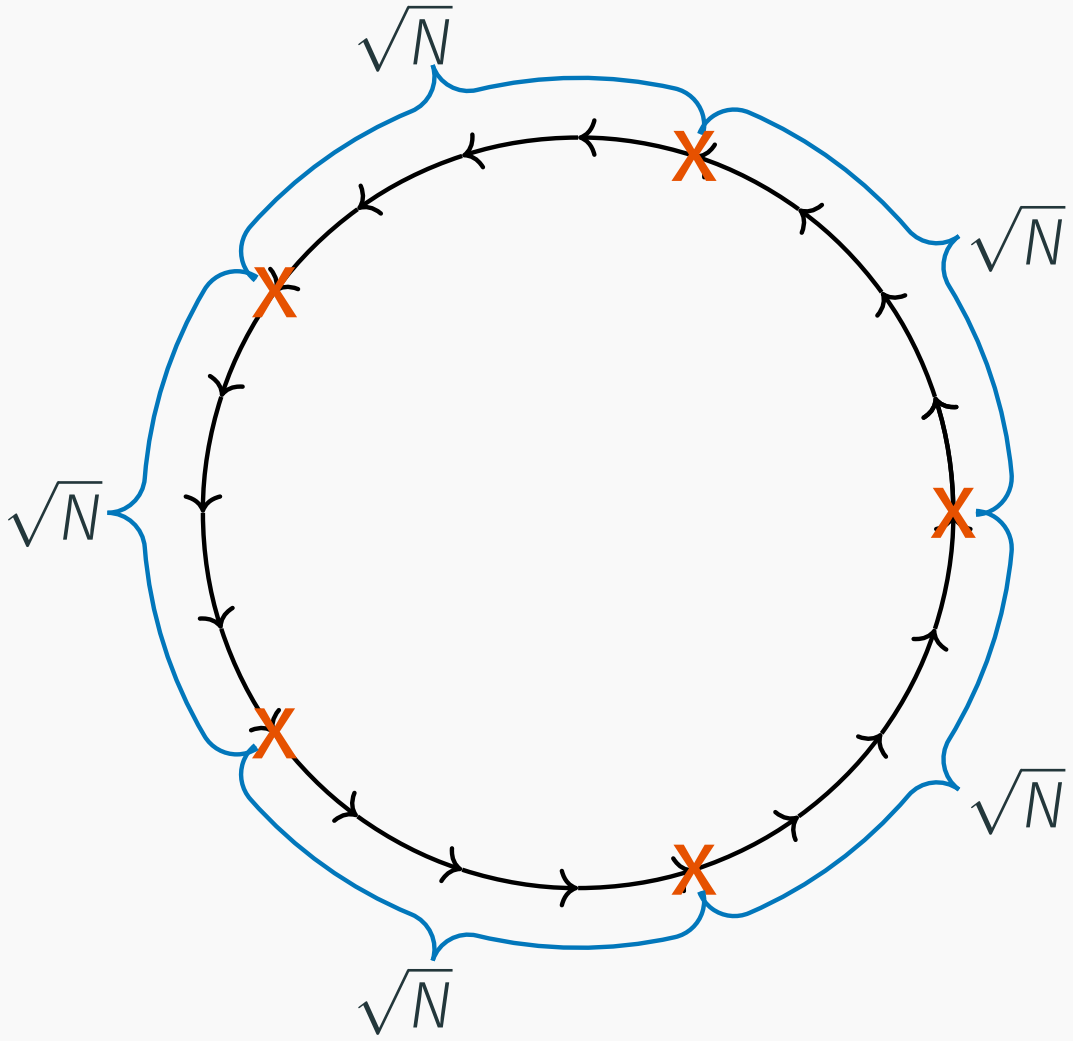
Inverting a Bijection



Inverting a Bijection

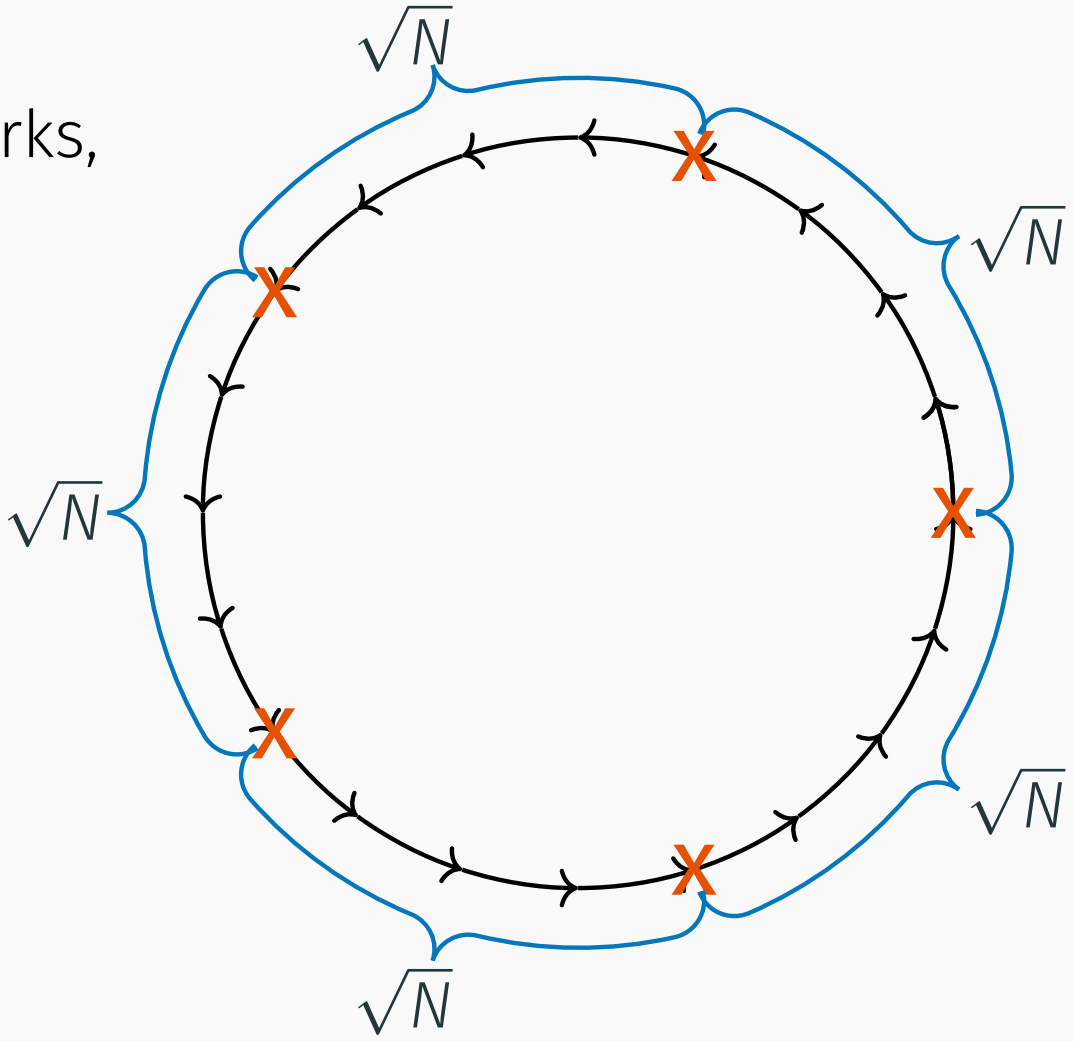


Inverting a Bijection



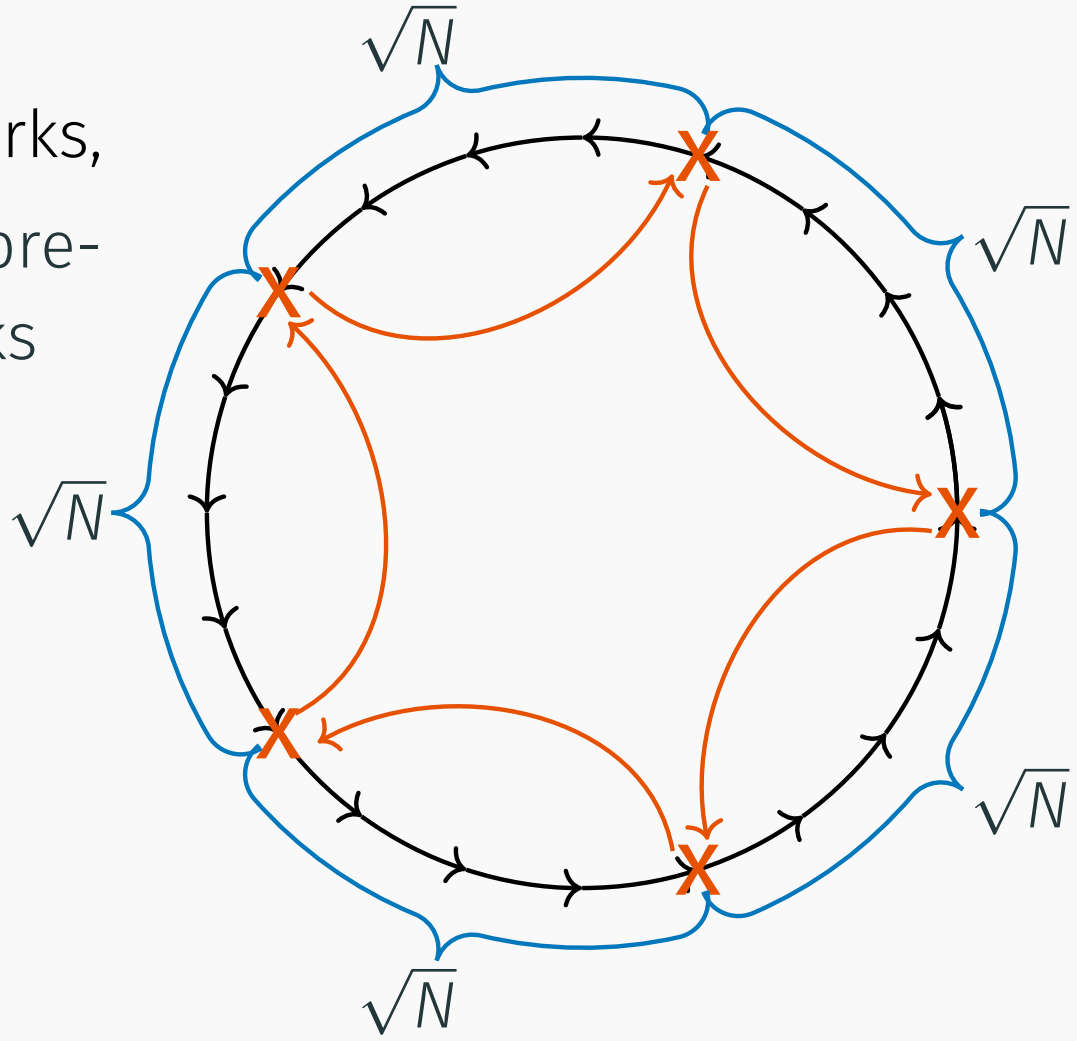
Inverting a Bijection

Store x landmarks,



Inverting a Bijection

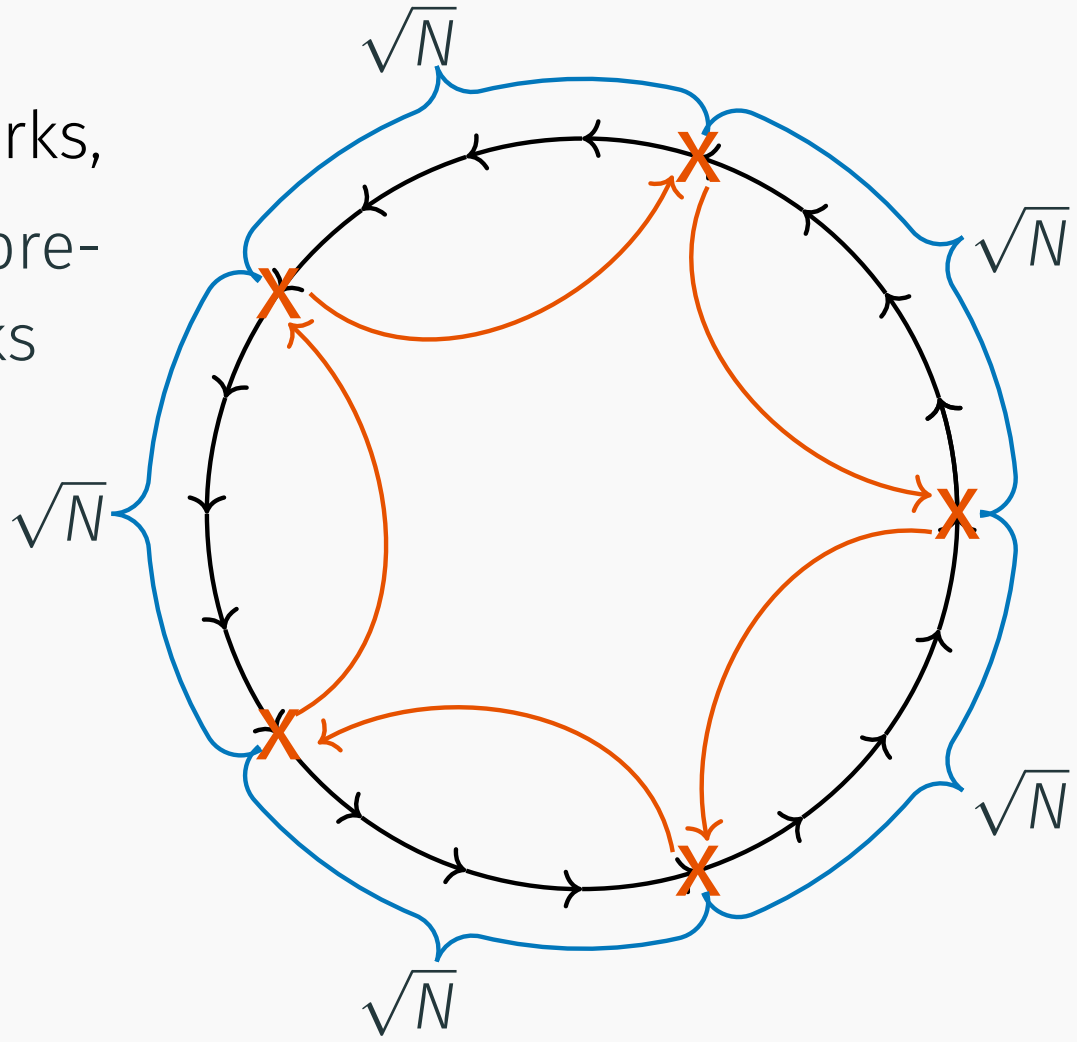
Store x landmarks, and links $\curvearrowright$ to previous landmarks



Inverting a Bijection

Store x landmarks, and links ↷ to previous landmarks

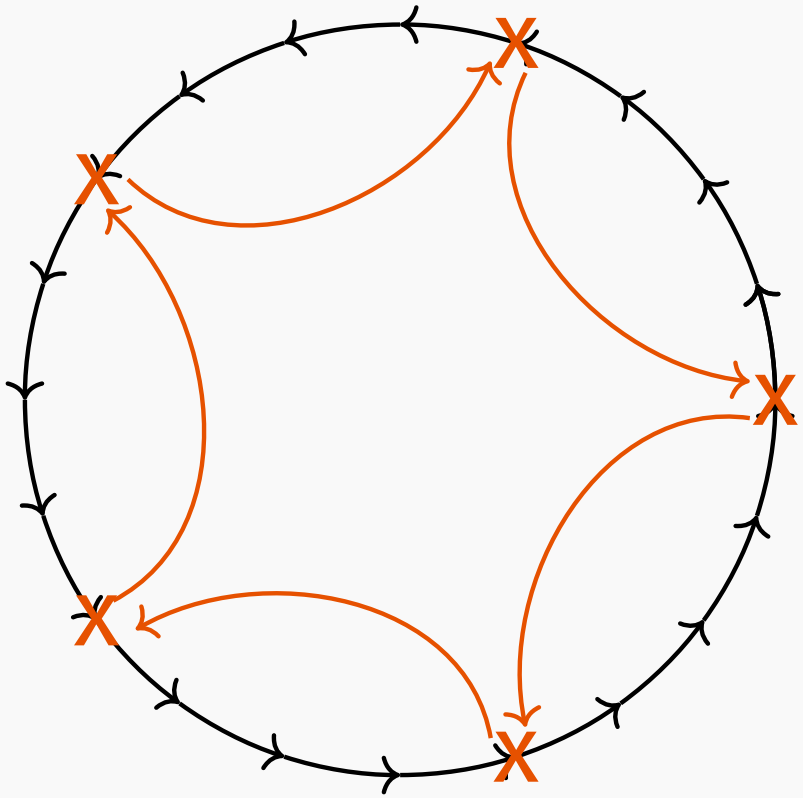
space $S \approx \sqrt{N}$



Inverting a Bijection

Store x landmarks, and links $\curvearrowright$ to previous landmarks

space $S \approx \sqrt{N}$

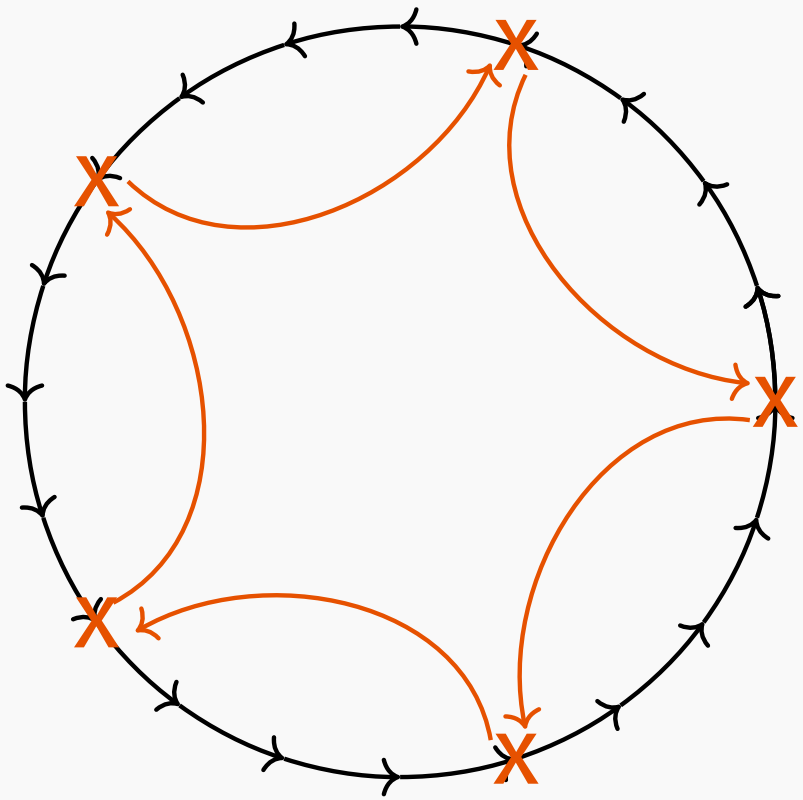


Inverting a Bijection

Store x landmarks, and links $\curvearrowright$ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:



Inverting a Bijection

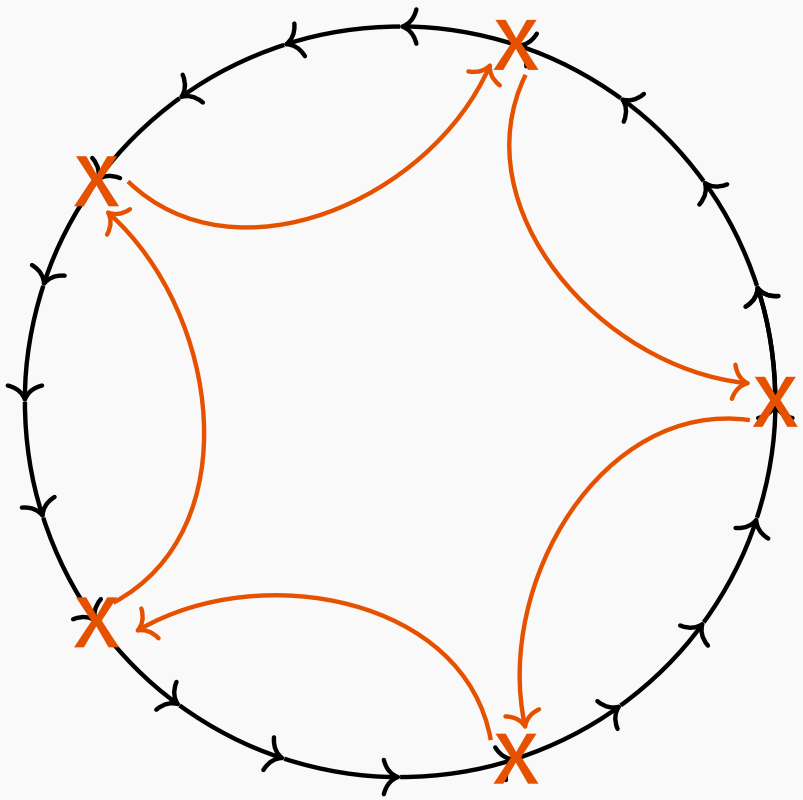
Store x landmarks,

and links ↷ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

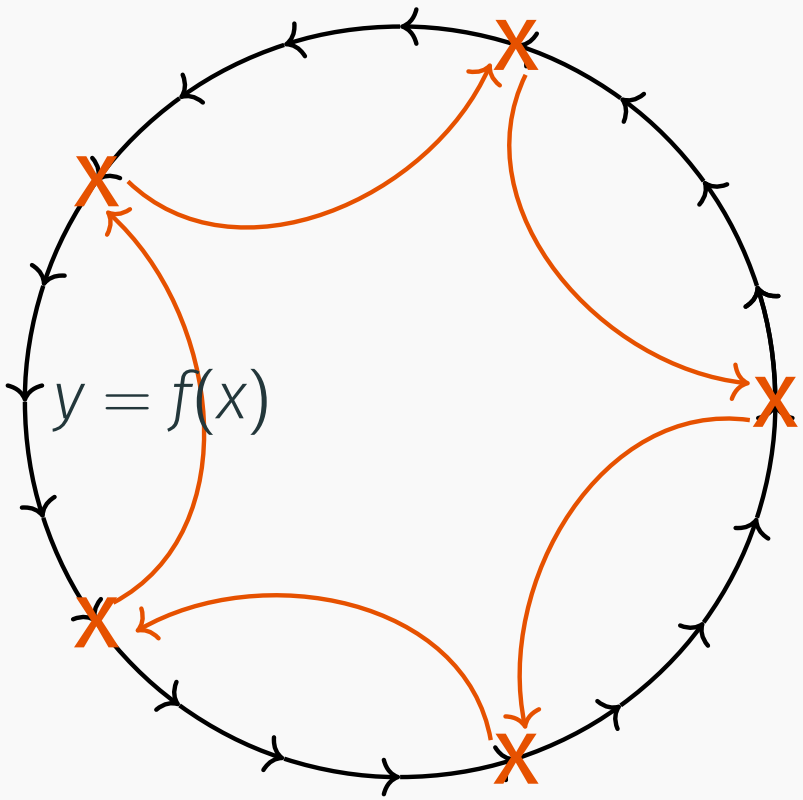
Store x landmarks,

and links $\curvearrowleft$ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

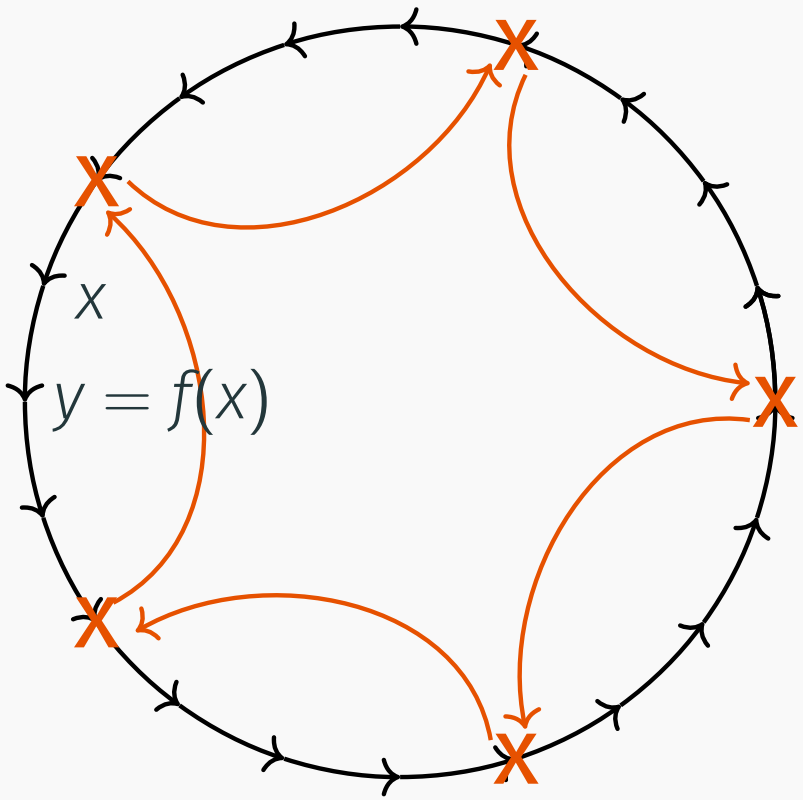
Store x landmarks,

and links $\curvearrowleft$ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

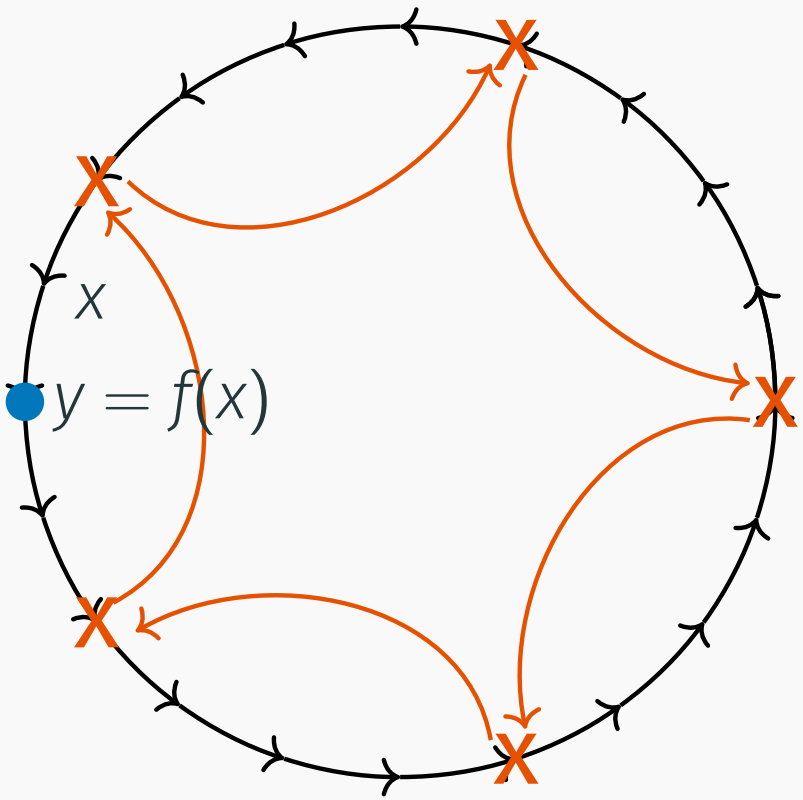
Store x landmarks,

and links $\curvearrowleft$ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

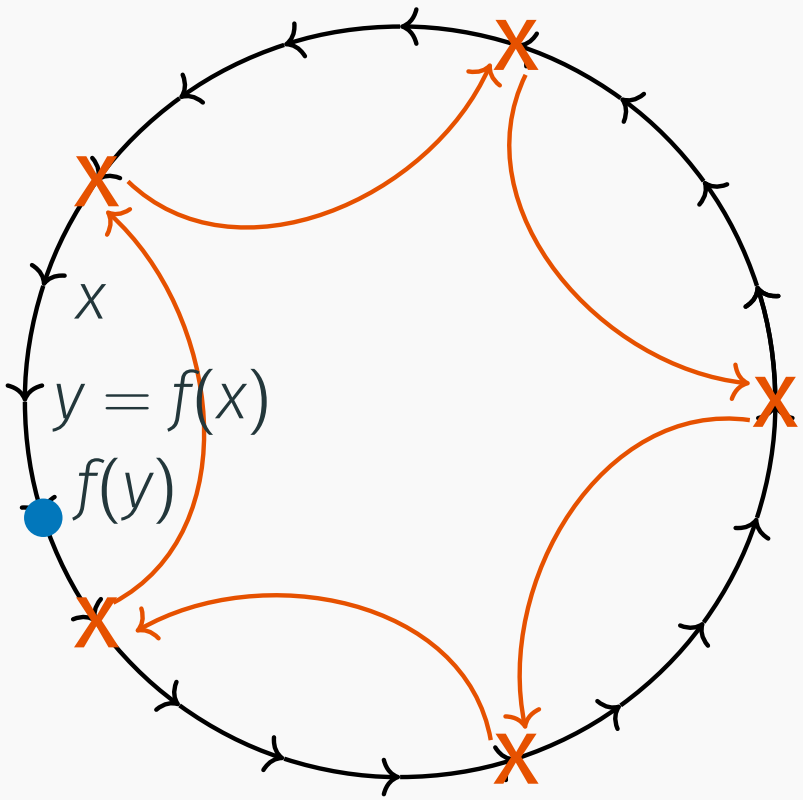
Store x landmarks,

and links ↷ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

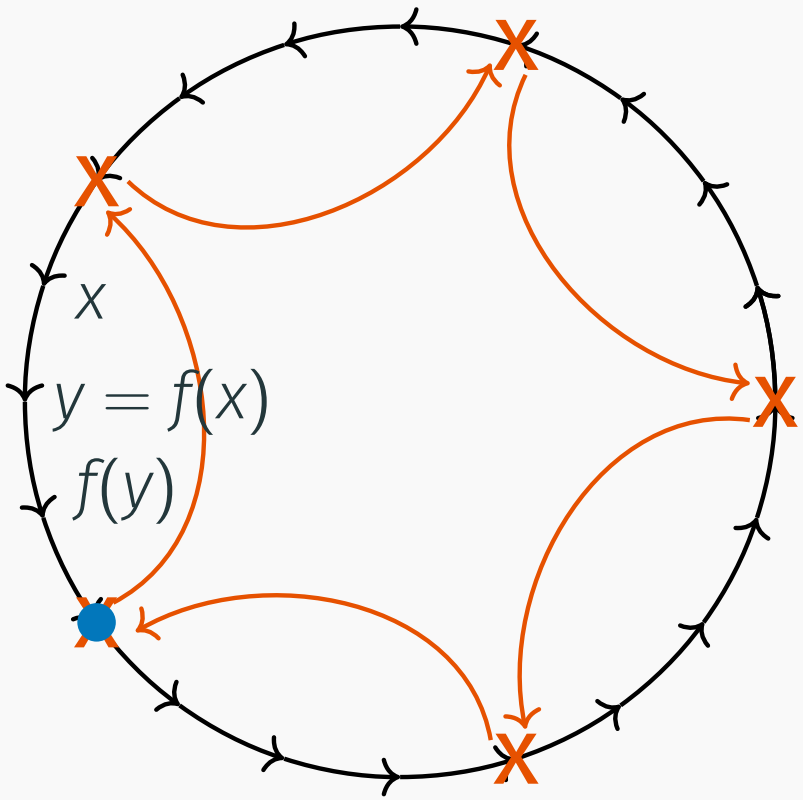
Store x landmarks,

and links ↷ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

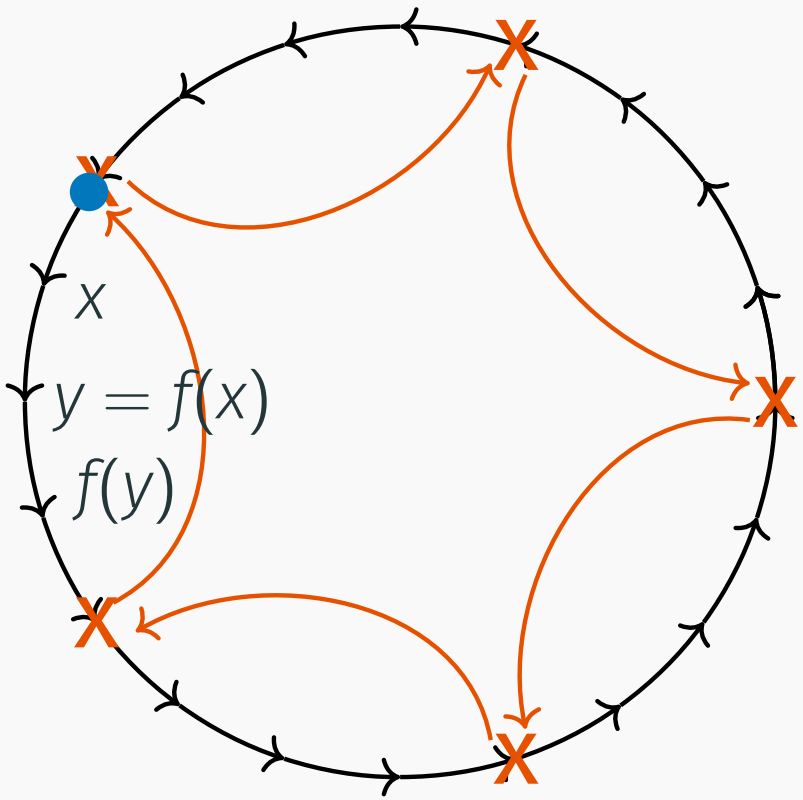
Store x landmarks,

and links ↷ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



Inverting a Bijection

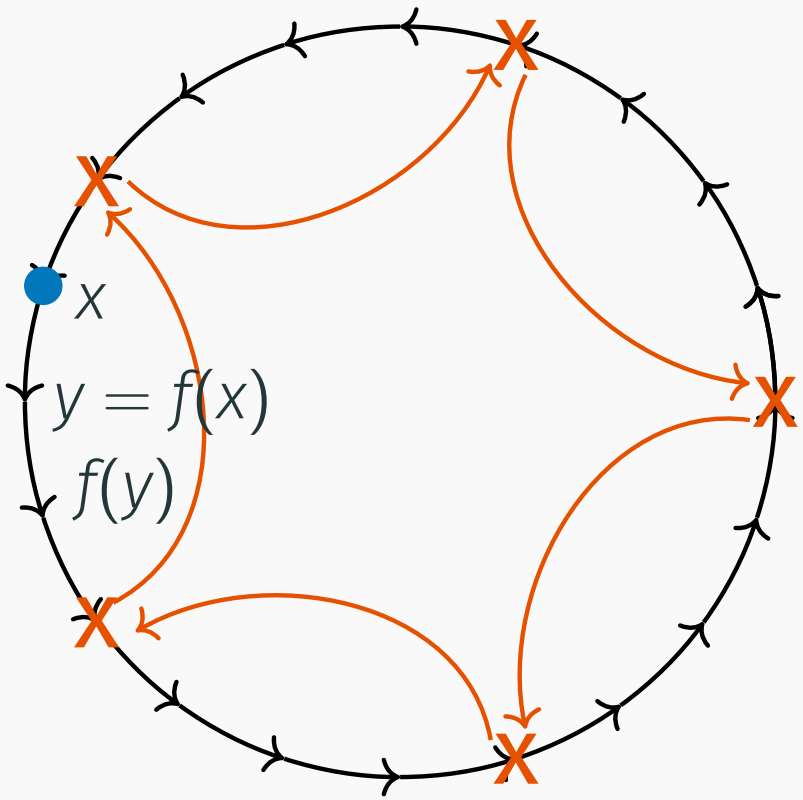
Store x landmarks,

and links $\curvearrowleft$ to previous landmarks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$:

Invert $y = f(x)$



DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$
  - f is easy to compute

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$
  - f is easy to compute
  - Hashing: the domain and range of f are of size n^2

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$
  - f is easy to compute
  - Hashing: the domain and range of f are of size n^2
  - Inverting f solves 3SUM-Indexing

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$
  - f is easy to compute
  - Hashing: the domain and range of f are of size n^2
  - Inverting f solves 3SUM-Indexing
- [Hel80, FN00] gives a way to invert any f which can be computed efficiently and has small domain and range

DS for 3SUM-Indexing

- Input $a_1, \ldots, a_n$
- Given b, check whether $b = a_i + a_j$ for some i, j
- Define $f(i, j) = a_i + a_j$ for $1 \leq i, j \leq n$
  - f is easy to compute
  - Hashing: the domain and range of f are of size n^2
  - Inverting f solves 3SUM-Indexing
- [Hel80, FN00] gives a way to invert any f which can be computed efficiently and has small domain and range
- Can solve 3SUM-Indexing in time T and space S for any $S^3T = n^6$. E.g., $S = n^{1.9}$ and $T = n^{0.3}$

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds for problems with efficient query generation.

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds for problems with efficient query generation.

- Secure $F\colon \{0,1\}^n \to \{0,1\}^n$

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds for problems with efficient query generation.

- Secure $F\colon \{0,1\}^n \to \{0,1\}^n$
- Define $F'\colon \{0,1\}^{n-1} \to \{0,1\}^{2n}$:

$$F'(x) = F(0,x) \| F(1,x)$$

Equivalence

Theorem (GGHPV20)

Cryptography with preprocessing is equivalent to data structure lower bounds for problems with efficient query generation.

- Secure $F\colon \{0,1\}^n \to \{0,1\}^n$
- Define $F'\colon \{0,1\}^{n-1} \to \{0,1\}^{2n}$:

$$F'(x) = F(0,x) \| F(1,x)$$

- Now $G\colon \{0,1\}^{2n-2} \to \{0,1\}^{2n}$:

$$G(x,y) = F'(x) + F'(y)$$

New DS Lower Bounds

Theorem (GGHPV20)

Any DS for 3SUM-Indexing must have $S \geq \Omega(N^{1+1/T})$

New DS Lower Bounds

Theorem (GGHPV20)

Any DS for 3SUM-Indexing must have $S \geq \Omega(N^{1+1/T})$

- For provable cryptographic security, we need stronger lower bounds

New DS Lower Bounds

Theorem (GGHPV20)

Any DS for 3SUM-Indexing must have $S \geq \Omega(N^{1+1/T})$

- For provable cryptographic security, we need stronger lower bounds
- But there is a barrier...

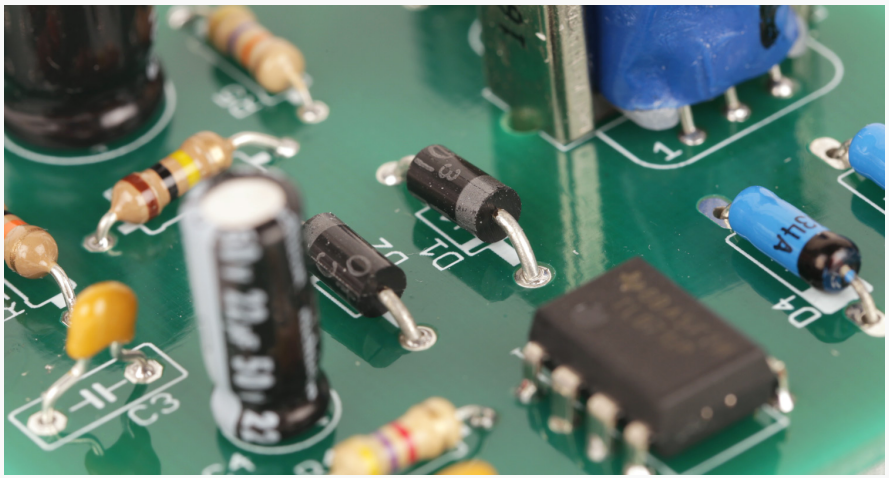
New DS Lower Bounds

Theorem (GGHPV20)

Any DS for 3SUM-Indexing must have $S \geq \Omega(N^{1+1/T})$

- For provable cryptographic security, we need stronger lower bounds
- But there is a barrier... in circuit complexity

CIRCUIT COMPLEXITY



Boolean Circuits

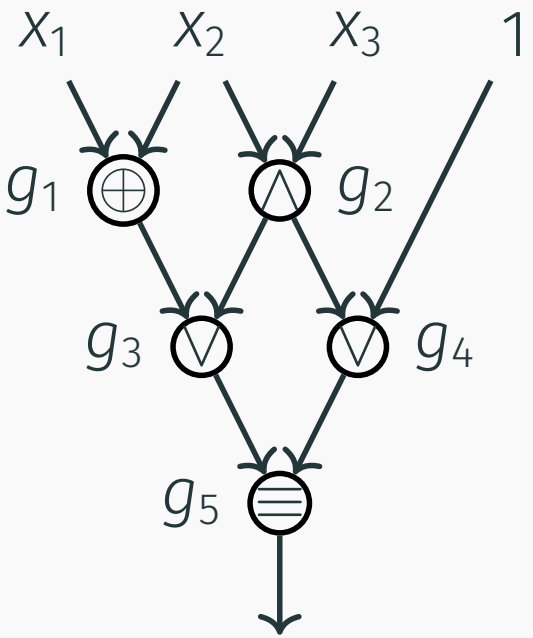
$$f\colon \{0,1\}^n \to \{0,1\}^n$$

$$\begin{aligned} g_1 &= x_1 \oplus x_2 \\ g_2 &= x_2 \wedge x_3 \\ g_3 &= g_1 \vee g_2 \\ g_4 &= g_2 \vee 1 \\ g_5 &= g_3 \equiv g_4 \end{aligned}$$

Boolean Circuits

$$f\colon \{0,1\}^n \to \{0,1\}^n$$

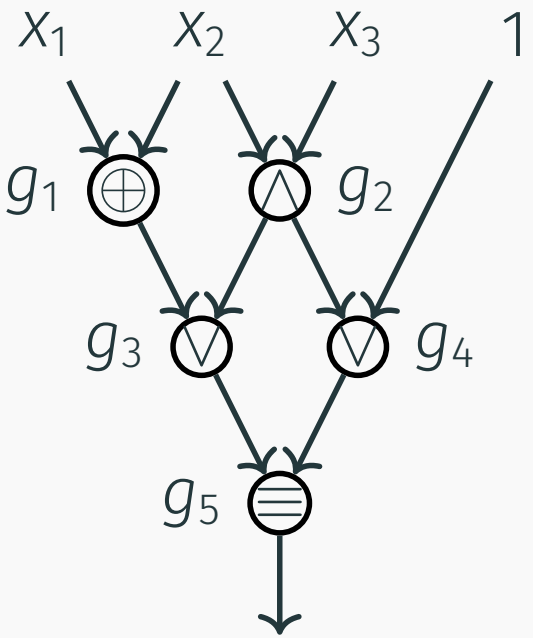
$$\begin{aligned} g_1 &= x_1 \oplus x_2 \\ g_2 &= x_2 \wedge x_3 \\ g_3 &= g_1 \vee g_2 \\ g_4 &= g_2 \vee 1 \\ g_5 &= g_3 \equiv g_4 \end{aligned}$$



Boolean Circuits

$$f\colon \{0,1\}^n \to \{0,1\}^n$$

$$\begin{aligned} g_1 &= x_1 \oplus x_2 \\ g_2 &= x_2 \wedge x_3 \\ g_3 &= g_1 \vee g_2 \\ g_4 &= g_2 \vee 1 \\ g_5 &= g_3 \equiv g_4 \end{aligned}$$



Inputs:

$x_1, \ldots, x_n, 0, 1$

Gates:

binary functions

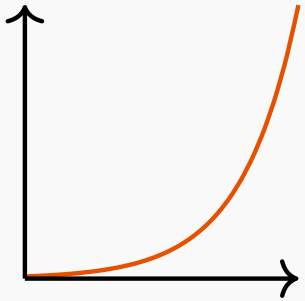
Fan-out:

unbounded

Depth:

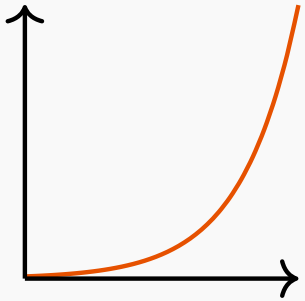
unbounded

Explicit Bounds



Most functions have exponential circuit complexity

Explicit Bounds

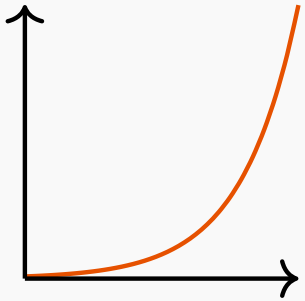


Most functions have exponential circuit complexity

$\mathbf{P \neq NP}$

We want to prove super-polynomial lower bounds

Explicit Bounds



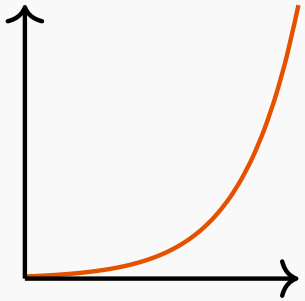
Most functions have exponential circuit complexity

$\mathbf{P \neq NP}$

We want to prove super-polynomial lower bounds

(for a function from **NP**)

Explicit Bounds

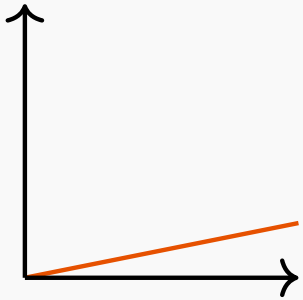


Most functions have exponential circuit complexity

$\mathbf{P \neq NP}$

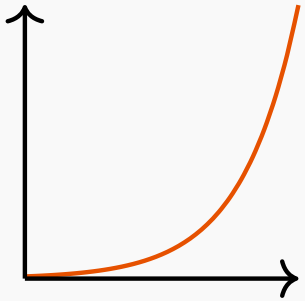
We want to prove super-polynomial lower bounds

(for a function from **NP**)



We can prove only $\approx 3n$ lower bounds

Explicit Bounds

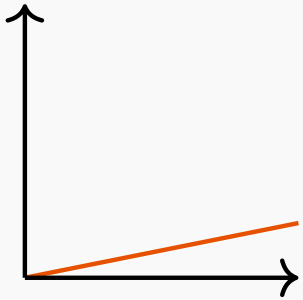


Most functions have exponential circuit complexity

$\mathbf{P} \neq \mathbf{NP}$

We want to prove super-polynomial lower bounds

(for a function from **NP**)



We can prove only $\approx 3n$ lower bounds

(even for a function from $\mathbf{E}^{\mathbf{NP}}$)

Super-linear Circuit Lower Bounds?

- Two n-bit integers can be multiplied by a circuit of size $O(n \log n)$ [SS71,F07,HH19]

Super-linear Circuit Lower Bounds?

- Two n-bit integers can be multiplied by a circuit of size $O(n \log n)$ [SS71,F07,HH19]
- Discrete Fourier Transform of a sequence of length n can be computed by a circuit of size $O(n \log n)$

Super-linear Circuit Lower Bounds?

- Two n-bit integers can be multiplied by a circuit of size $O(n \log n)$ [SS71,F07,HH19]
- Discrete Fourier Transform of a sequence of length n can be computed by a circuit of size $O(n \log n)$
- Shifts, Permutations

Super-linear Circuit Lower Bounds?

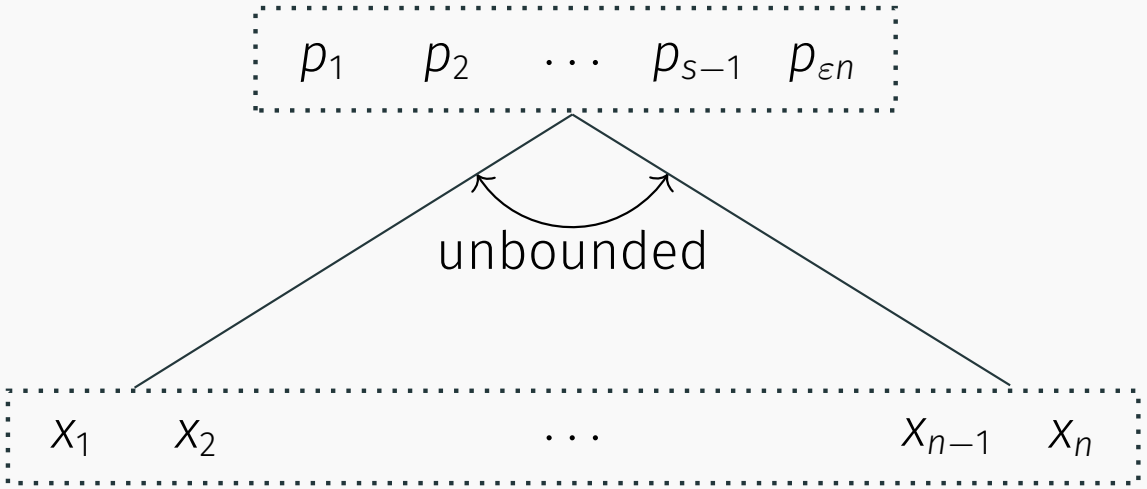
- Two n-bit integers can be multiplied by a circuit of size $O(n \log n)$ [SS71,F07,HH19]
- Discrete Fourier Transform of a sequence of length n can be computed by a circuit of size $O(n \log n)$
- Shifts, Permutations
- NP-hard problems

Super-linear Circuit Lower Bounds?

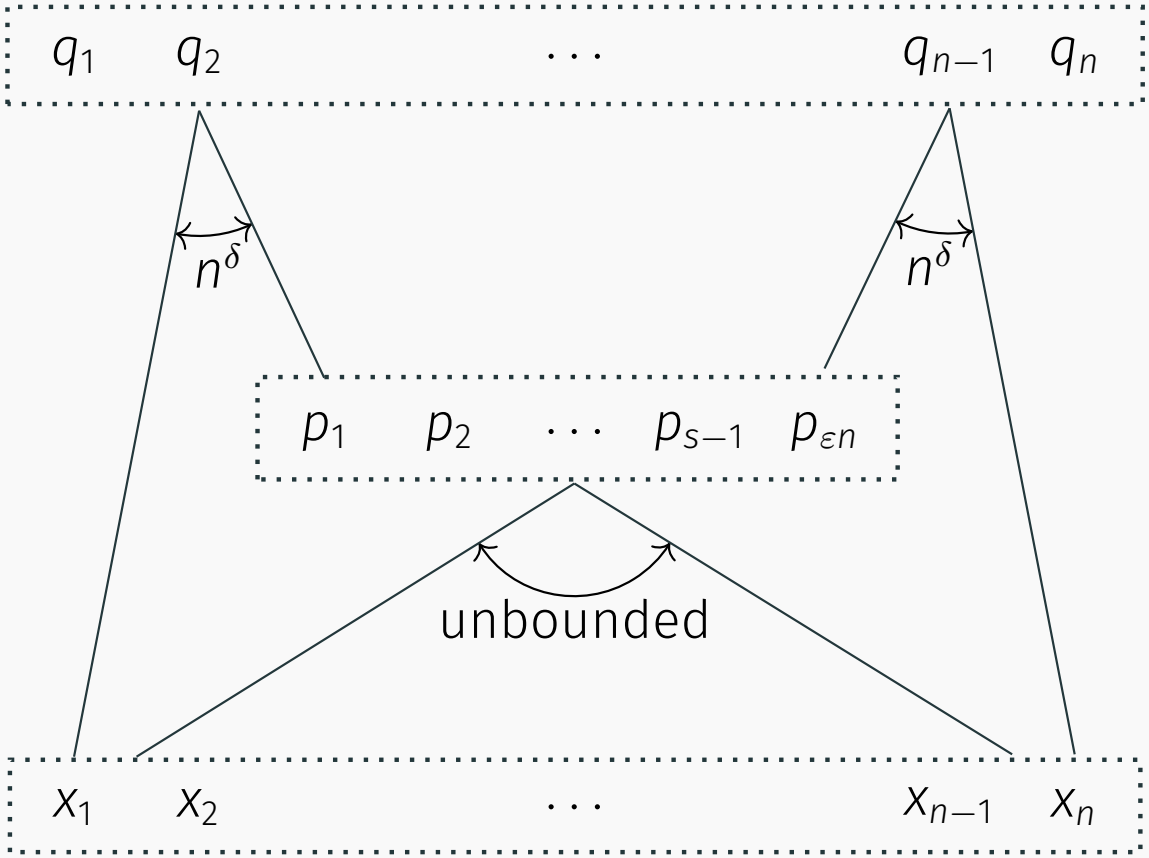
- Two n-bit integers can be multiplied by a circuit of size $O(n \log n)$ [SS71,F07,HH19]
- Discrete Fourier Transform of a sequence of length n can be computed by a circuit of size $O(n \log n)$
- Shifts, Permutations
- NP-hard problems
- ...

Linear-size Circuits [Val77]

Linear-size Circuits [Val77]

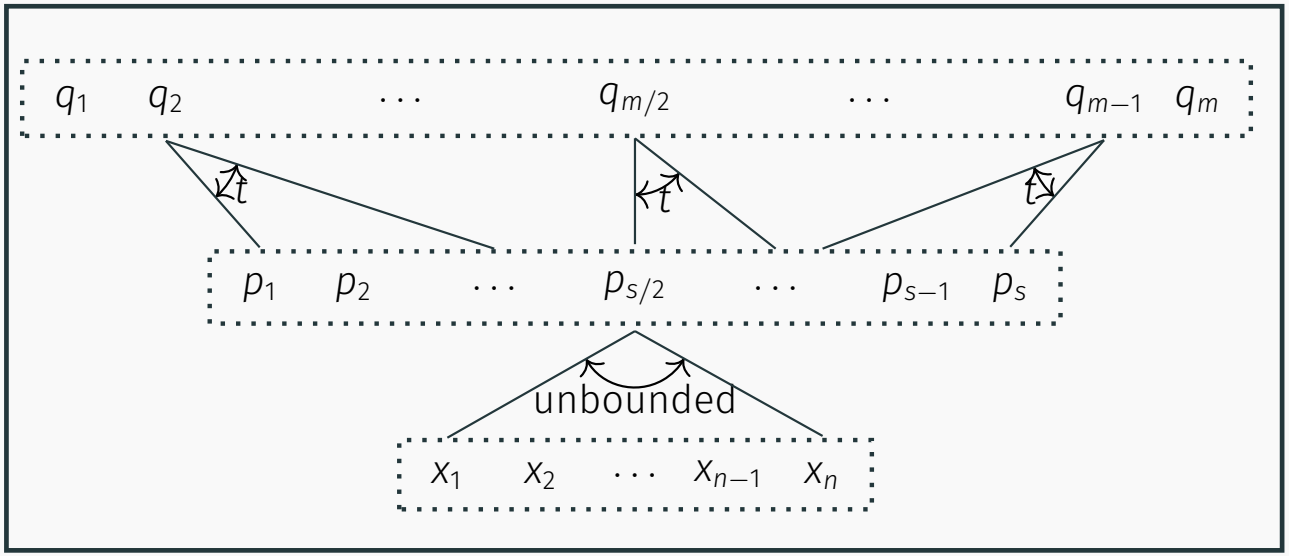


Linear-size Circuits [Val77]

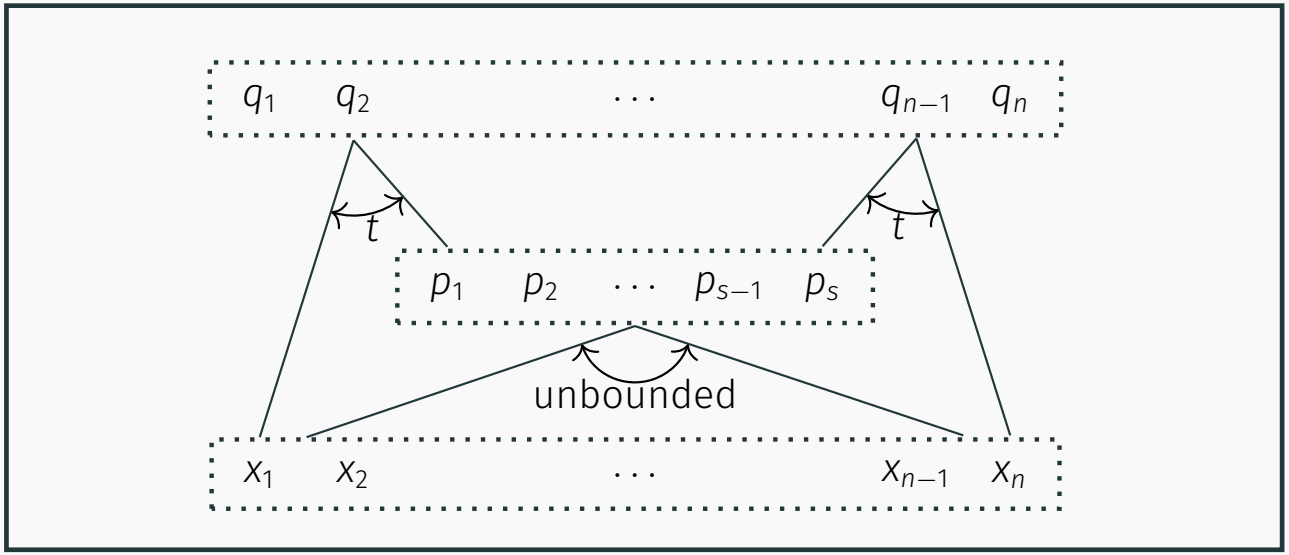


Comparison

Efficient Data Structures



Small Circuits



Equivalence

Theorem (DGW19)

Any improvement on the current lower bound for a linear data structure problem implies new lower bounds for circuits, and vice versa.

Equivalence

Theorem (DGW19)

Any improvement on the current lower bound for a linear data structure problem implies new lower bounds for circuits, and vice versa.

Linear problem is defined by a matrix $M \in \mathbb{F}^{m \times n}$. Data structure problem: given $x \in \mathbb{F}^n$ output $Mx \in \mathbb{F}^m$

Equivalence

Theorem (DGW19)

Any improvement on the current lower bound for a linear data structure problem implies new lower bounds for circuits, and vice versa.

Linear problem is defined by a matrix $M \in \mathbb{F}^{m \times n}$. Data structure problem: given $x \in \mathbb{F}^n$ output $Mx \in \mathbb{F}^m$

Examples: Polynomial evaluation, Matrix-vector multiplication, Range counting, Partial sums, ...

Comparison

Small circuit / Non-rigid

$$\overset{m \times n}{M} = \overset{m \times n}{A} + \overset{m \times n}{B}$$

A: t-sparse; B: $\mathrm{rk} \leq \varepsilon n$

Comparison

Small circuit / Non-rigid

$$\overset{m \times n}{M} = \overset{m \times n}{A} + \overset{m \times n}{B}$$

A: t-sparse; B: $\mathrm{rk} \leq \varepsilon n$

Efficient Data Structure

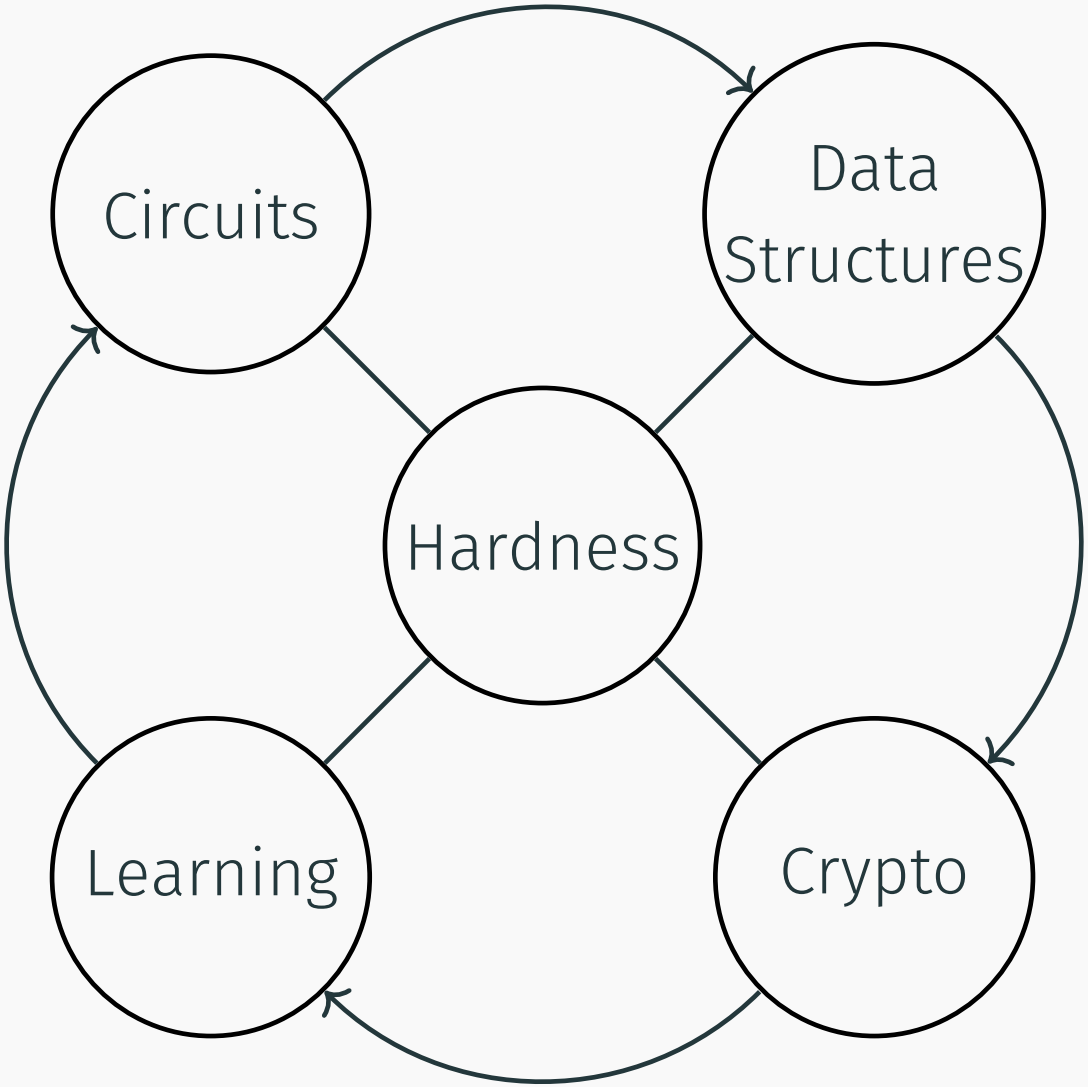
$$\overset{m \times n}{M} = \overset{m \times s}{A} \cdot \overset{s \times n}{B}$$

A: t-sparse; B: small

Equivalence

Theorem (DGW19)

Any improvement on the current lower bound for a linear data structure problem implies new lower bounds for circuits, and vice versa.



Thank you for your attention!