Fast and Simple Sorting Using Partial Information

Bernhard Haeupler[1,2], Richard Hladík[1,2], John Iacono[3], Václav Rozhoň[1], Robert E. Tarjan[4], Jakub Tětek[1]

[1] INSAIT, Sofia University "St. Kliment Ohridski"
[2] ETH Zürich
[3] Université libre de Bruxelles
[4] Princeton University

SODA 2025

Fast and Simple Sorting Using Partial Information

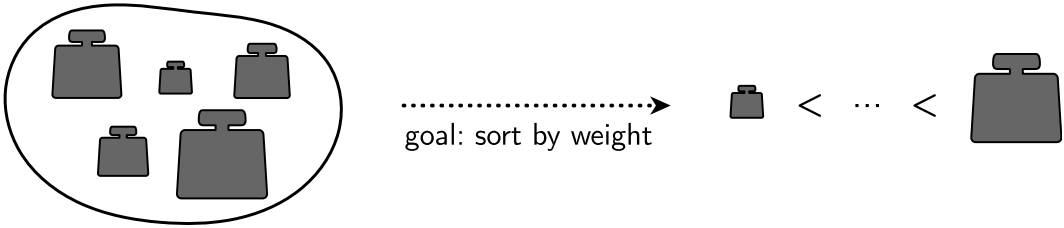
Bernhard Haeupler[1,2], Richard Hladík[1,2], John Iacono[3], Václav Rozhoň[1], Robert E. Tarjan[4], Jakub Tětek[1]

[1] INSAIT, Sofia University "St. Kliment Ohridski"
[2] ETH Zürich
[3] Université libre de Bruxelles
[4] Princeton University

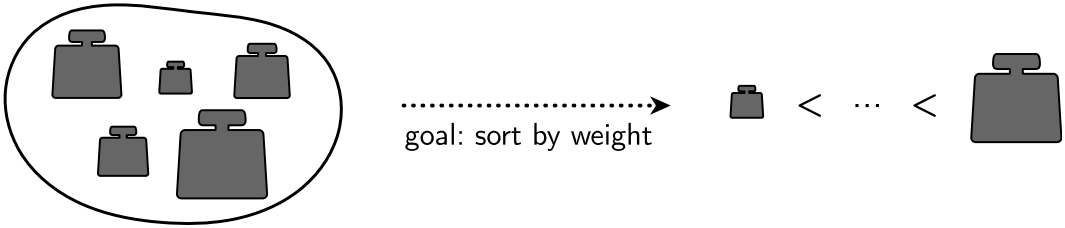
SODA 2025

This talk: resolving a sorting problem from 1976

Sorting Under Partial Information (SUPI)



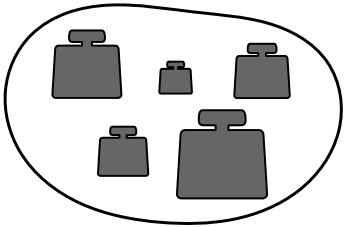
Sorting Under Partial Information (SUPI)



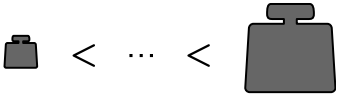
$x < y$

1. I can do my own comparisons

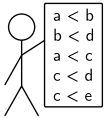
Sorting Under Partial Information (SUPI)



goal: sort by weight



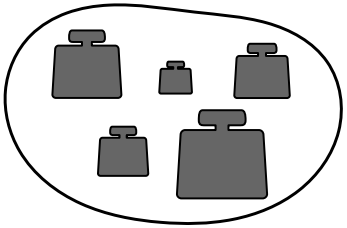
$x < y$



1. I can do my own comparisons

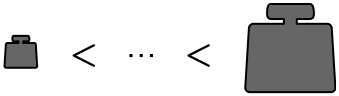
2. I know some comparisons for free

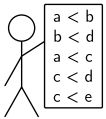
Sorting Under Partial Information (SUPI)



$x < y$

goal: sort by weight





1. I can do my own comparisons

2. I know some comparisons for free

Goal: Find the sorted order. Optimize for 1) #comparisons 2) time.

Outline

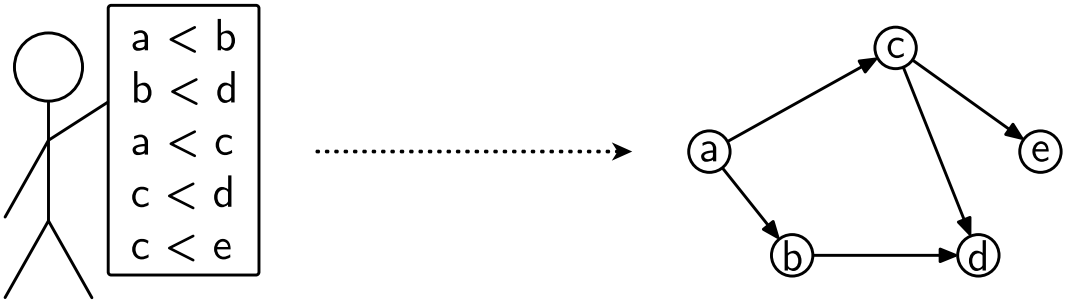
- What's known
  - problem properties
    - when is the problem interesting?
  - lower bound
  - related work

Outline

- What's known
  - problem properties
    - when is the problem interesting?
  - lower bound
  - related work
- What's new
  - our optimal algorithm
  - relationship with heaps
  - connections to other results

Part I: What's known

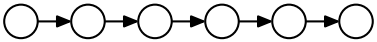
Graph rephrasing



Promise: $\forall uv \in E, \quad u < v$

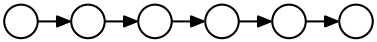
(= the comparison list isn't fake)

Lower Bound

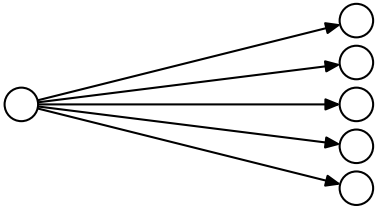


needs no comparisons

Lower Bound

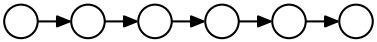


needs no comparisons

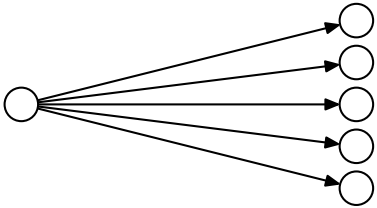


needs $\Theta(n \log n)$ comparisons

Lower Bound



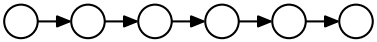
needs no comparisons



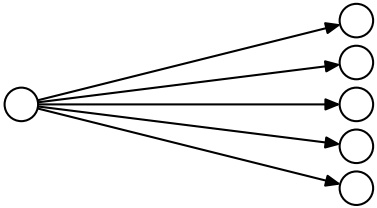
needs $\Theta(n \log n)$ comparisons

Some graphs are harder than others.

Lower Bound



needs no comparisons

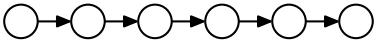


needs $\Theta(n \log n)$ comparisons

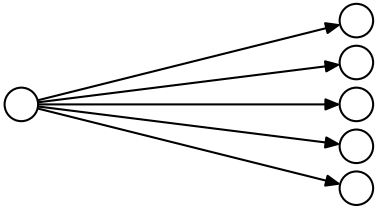
Some graphs are harder than others.

$T :=$ #orders consistent with G

Lower Bound



needs no comparisons



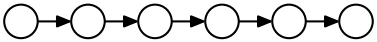
needs $\Theta(n \log n)$ comparisons

Some graphs are harder than others.

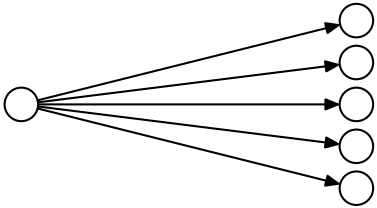
$T :=$ #orders consistent with G

Information-theoretic lower bound: $\log_2 T$ comparisons is needed.

Lower Bound



needs $\log_2(1)$ comparisons



needs $\log_2((n-1)!)$ comparisons

Some graphs are harder than others.

$T :=$ #orders consistent with G

Information-theoretic lower bound: $\log_2 T$ comparisons is needed.

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	poly(n)	ellipsoid method

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	poly(n)	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	poly(n)	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition
[VR24]	$\mathcal{O}(c \log T)$	$\mathcal{O}(n^{1+1/c})$	different model, $\mathcal{O}(n^{\omega})$ in our model

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	$\text{poly}(n)$	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition
[VR24]	$\mathcal{O}(c \log T)$	$\mathcal{O}(n^{1+1/c})$	different model, $\mathcal{O}(n^\omega)$ in our model
Our work	$\mathcal{O}(\log T)$	$\mathcal{O}(\log T + n + m)$	**optimal** w.r.t. time & comparisons

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	$\text{poly}(n)$	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition
[VR24]	$\mathcal{O}(c \log T)$	$\mathcal{O}(n^{1+1/c})$	different model, $\mathcal{O}(n^\omega)$ in our model
Our work	$\mathcal{O}(\log T)$	$\mathcal{O}(\log T + n + m)$	**optimal** w.r.t. time & comparisons

Previous approaches: find a good comparison / decompose into paths and merge cleverly

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	poly(n)	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition
[VR24]	$\mathcal{O}(c \log T)$	$\mathcal{O}(n^{1+1/c})$	different model, $\mathcal{O}(n^\omega)$ in our model
Our work	$\mathcal{O}(\log T)$	$\mathcal{O}(\log T + n + m)$	**optimal** w.r.t. time & comparisons

Previous approaches: find a good comparison / decompose into paths and merge cleverly
$\Rightarrow$ tradeoff between #comparisons and time

Related Work

Reference	comparisons	time	note
[Fre76]	$\log_2 T + 2n$	$\geq$ exponential	solves a more general problem
[KK92]	$\mathcal{O}(\log T)$	$\text{poly}(n)$	ellipsoid method
[Car+10]	$\mathcal{O}(\log T)$	$\mathcal{O}(n^{2.5})$	chain decomposition
[VR24]	$\mathcal{O}(c \log T)$	$\mathcal{O}(n^{1+1/c})$	different model, $\mathcal{O}(n^\omega)$ in our model
Our work	$\mathcal{O}(\log T)$	$\mathcal{O}(\log T + n + m)$	**optimal** w.r.t. time & comparisons

Previous approaches: find a good comparison / decompose into paths and merge cleverly
$\Rightarrow$ tradeoff between #comparisons and time

Our approach: ✨ just put it in a heap ✨

Part II: What's new

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes!

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes! ... sometimes!

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes! ... sometimes!
- **Idea:** adapt to the sequence of operations and be faster if the operations are "nice".

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes! ... sometimes!
- **Idea:** adapt to the sequence of operations and be faster if the operations are "nice".
- active field of research: splay trees, tango trees, pairing heaps, smooth heaps, ...

Beyond-worst-case heaps

Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes! ... sometimes!
- **Idea:** adapt to the sequence of operations and be faster if the operations are "nice".
- active field of research: splay trees, tango trees, pairing heaps, smooth heaps, ...

Definition: a *working-set heap* has Insert in $\mathcal{O}(1)$ and ExtractMin in $\mathcal{O}(\log(\textit{item age}))$

Beyond-worst-case heaps

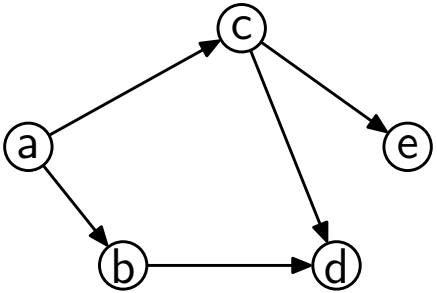
Motivation: Can comparison-based heaps / search trees break the $\mathcal{O}(\log n)$ barrier?

- Beyond-worst-case analysis: Yes! ... sometimes!
- **Idea:** adapt to the sequence of operations and be faster if the operations are "nice".
- active field of research: splay trees, tango trees, pairing heaps, smooth heaps, ...

Definition: a *working-set heap* has Insert in $\mathcal{O}(1)$ and ExtractMin in $\mathcal{O}(\log(\textit{item age}))$

- much faster than traditional heaps if the heap is large, but most ExtractMins are of "fresh" items

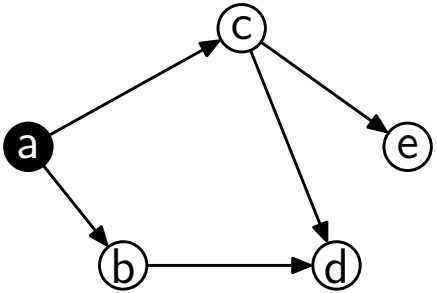
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

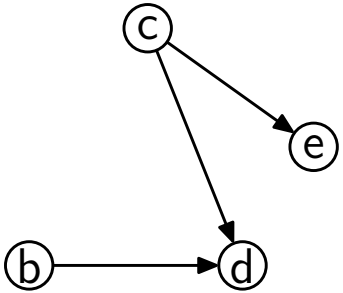
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

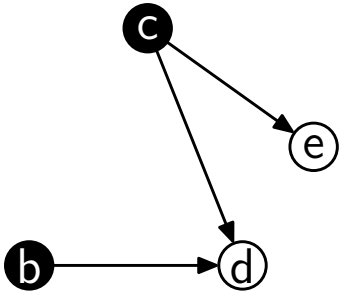
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

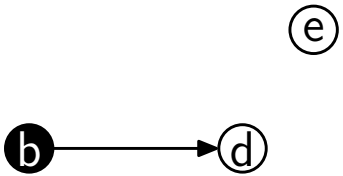
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

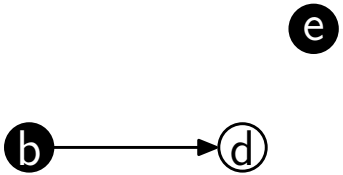
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

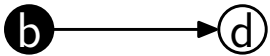
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

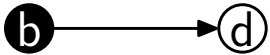
Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort



Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Main result: this algo is optimal w.r.t. time if we use a working-set heap.

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Our Algorithm: Topological Heapsort

Main result: this algo is optimal w.r.t. time if we use a working-set heap.
. . . and optimal w.r.t. #comparisons after a slight tweak.

Initialize an empty heap. Repeat:

1. Add to the heap all new candidates for the minimum. *(= sources in G)*
2. Find the minimum using the heap, delete it.

Connection 1: Dijkstra

Dijkstra (with a working-set heap) is optimal on every graph [Hae+24].

Connection 1: Dijkstra

Dijkstra (with a working-set heap) is optimal on every graph [Hae+24].

This inspired our SUPI result:

Connection 1: Dijkstra

Dijkstra (with a working-set heap) is optimal on every graph [Hae+24].

This inspired our SUPI result:

- Our algorithm $\approx$ "tweaked Dijkstra"

Connection 1: Dijkstra

Dijkstra (with a working-set heap) is optimal on every graph [Hae+24].

This inspired our SUPI result:

- Our algorithm $\approx$ "tweaked Dijkstra"
- Same heap property

Connection 1: Dijkstra

Dijkstra (with a working-set heap) is optimal on every graph [Hae+24].

This inspired our SUPI result:

- Our algorithm $\approx$ "tweaked Dijkstra"
- Same heap property
- Similar proof techniques for optimality

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

- Simpler optimality proof

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

- Simpler optimality proof
- New optimal algorithm for SUPI

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

- Simpler optimality proof
- New optimal algorithm for SUPI

Their algorithm nicely complements ours:

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

- Simpler optimality proof
- New optimal algorithm for SUPI

Their algorithm nicely complements ours:

- our algo: $\approx$ "smart selection sort", uses a beyond-worst-case heap

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

- Simpler optimality proof
- New optimal algorithm for SUPI

Their algorithm nicely complements ours:

- our algo: $\approx$ "smart selection sort", uses a beyond-worst-case heap
- their algo: $\approx$ "smart insertion sort", uses a beyond-worst-case search tree

Connection 2: Follow-up work [SOSA'25]

Follow-up work by Van der Hoog, Rotenberg, and Rutschmann [VRR24]:

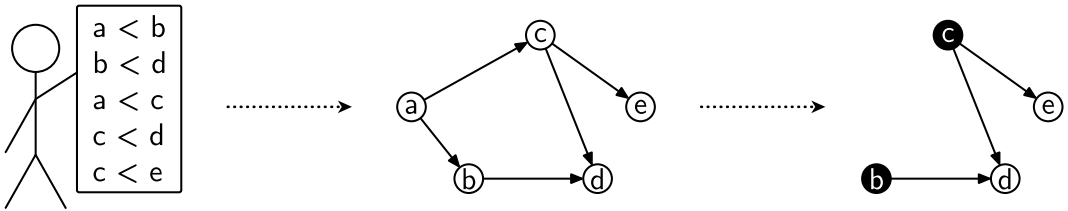
- Simpler optimality proof
- New optimal algorithm for SUPI

Their algorithm nicely complements ours:

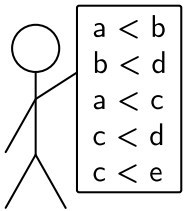
- our algo: $\approx$ "smart selection sort", uses a beyond-worst-case heap
- their algo: $\approx$ "smart insertion sort", uses a beyond-worst-case search tree

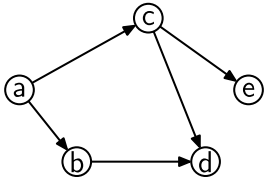
Come check it out tomorrow!

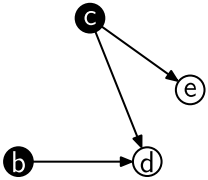
Conclusion



Conclusion







Thank you!

Bibliography I

[Fre76] Michael L Fredman. "How good is the information theory bound in sorting?" In: *Theoretical Computer Science* 1.4 (1976), pp. 355–361.

[KK92] Jeff Kahn and Jeong Han Kim. "Entropy and sorting". In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 178–187.

[Car+10] Jean Cardinal et al. "Sorting under partial information (without the ellipsoid algorithm)". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 359–368.

[Hae+24] Bernhard Haeupler et al. *Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps*. 2024. arXiv: 2311.11793 [cs.DS].

[VR24] Ivor van der Hoog and Daniel Rutschmann. *Tight Bounds for Sorting Under Partial Information*. 2024. arXiv: 2404.08468 [cs.DS].

[VRR24] Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. *Simpler Optimal Sorting from a Directed Acyclic Graph*. 2024. arXiv: 2407.21591 [cs.DS]. URL: https://arxiv.org/abs/2407.21591.

Backup Slides