# Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps

Bernhard Haeupler[1,2], Richard Hladík[1,2], Václav Rozhoň[2], Robert Tarjan[3], Jakub Tětek[2]

[1] ETH Zürich
[2] INSAIT, Sofia University "St. Kliment Ohridski"
[3] Princeton University

FOCS 2024

# Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps

Bernhard Haeupler[1,2], Richard Hladík[1,2], Václav Rozhoň[2], Robert Tarjan[3], Jakub Tětek[2]

[1] ETH Zürich
[2] INSAIT, Sofia University "St. Kliment Ohridski"
[3] Princeton University

FOCS 2024

# Intro

This talk: something new about Dijkstra's algorithm

# Intro

This talk: something new about Dijkstra's algorithm

Our result: Dijkstra + a nice heap is optimal on every graph

# Outline

- Setup
- Universal Optimality
- Nice Heaps
- Dijkstra + Nice Heaps
- Proof Intuition

# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$

# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task:** ???

# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task (usually):** compute distances from $s$

# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task (usually):** compute distances from $s$
  - but: fast algorithms exist: $\mathcal{O}(m\sqrt{\log n \cdot \log\log n})$ for undirected [Dua+23], many others for integer/bounded-ratio weights [FW93; FW94; Ram96; Ram97; Tho99; Hag00; Tho00a; Tho00b; Tho04]

# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task (usually):** compute distances from $s$
  - but: fast algorithms exist: $\mathcal{O}(m\sqrt{\log n \cdot \log\log n})$ for undirected [Dua+23], many others for integer/bounded-ratio weights [FW93; FW94; Ram96; Ram97; Tho99; Hag00; Tho00a; Tho00b; Tho04]
- **Task (this talk):** *order* vertices by distance from $s$

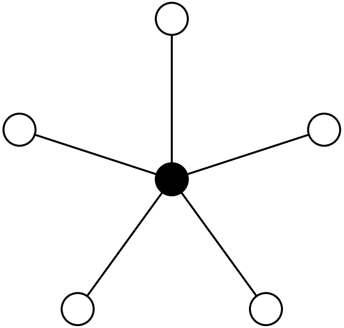
# Single-Source Shortest Paths (SSSP)

- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task (usually):** compute distances from $s$
  - but: fast algorithms exist: $\mathcal{O}(m\sqrt{\log n \cdot \log\log n})$ for undirected [Dua+23], many others for integer/bounded-ratio weights [FW93; FW94; Ram96; Ram97; Tho99; Hag00; Tho00a; Tho00b; Tho04]
- **Task (this talk):** *order* vertices by distance from $s$
- **Model:** ???

# Single-Source Shortest Paths (SSSP)

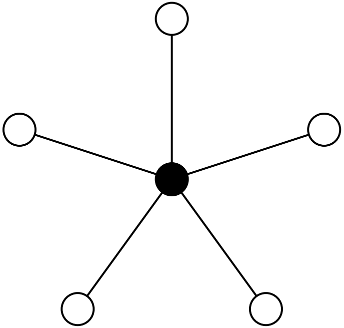
- **Input:** (un)directed $G$, edge weights $w$, source node $s$
- **Task (usually):** compute distances from $s$
  - but: fast algorithms exist: $\mathcal{O}(m\sqrt{\log n \cdot \log\log n})$ for undirected [Dua+23], many others for integer/bounded-ratio weights [FW93; FW94; Ram96; Ram97; Tho99; Hag00; Tho00a; Tho00b; Tho04]
- **Task (this talk):** *order* vertices by distance from $s$
- **Model:** positive real weights, only comparisons and additions

# Lower Bound

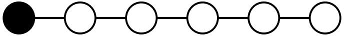


needs $\log(n!) = \Omega(n \log n)$ comparisons

# Lower Bound

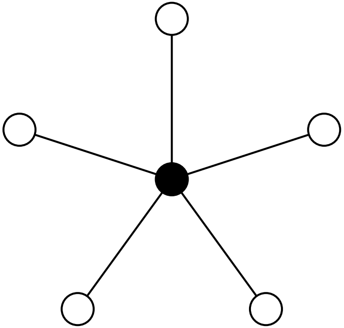


needs $\log(n!) = \Omega(n \log n)$ comparisons

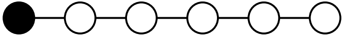


needs $\log(1) = 0$ comparisons

# Lower Bound



needs $\log(n!) = \Omega(n \log n)$ comparisons



needs $\log(1) = 0$ comparisons

$\implies$ Some graphs are harder than others.

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

$A$ is universally optimal if:

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

$A$ is universally optimal if:

$$\max_{w} \mathrm{Time}_A(G, w)$$

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

$A$ is universally optimal if:

$$\forall \text{ correct } A' \quad \max_w \mathrm{Time}_A(G, w) \leq \max_w \mathrm{Time}_{A'}(G, w)$$

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

$A$ is universally optimal if:

$$\forall G \; \forall \text{correct } A' \quad \max_{w} \mathrm{Time}_{A}(G, w) \leq \max_{w} \mathrm{Time}_{A'}(G, w)$$

# Universal Optimality

- notion from distributed computing [GKP98; HWZ21]
  - there, we have universally optimal algorithms for many problems (minimum spanning trees, minimum cut, approximate shortest paths) [HWZ21; GZ22; Roz+22; Zuz+22]
- on every underlying unweighted graph $G$, $A$ is worst-case optimal w.r.t. $w$

$A$ is universally optimal if:

$$\exists c \; \forall G \; \forall \text{correct } A' \qquad \max_{w} \mathrm{Time}_A(G, w) \leq c \cdot \max_{w} \mathrm{Time}_{A'}(G, w)$$

# Nice Heaps

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . insert a new item

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert ... insert a new item
- DeleteMin ... delete and return the minimum

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . insert a new item
- DeleteMin . . . delete and return the minimum
- Decrease . . . decrease item's value

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . $\mathcal{O}(1)$
- DeleteMin . . . $\mathcal{O}(\log n)$
- Decrease . . . $\mathcal{O}(1)$

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . $\mathcal{O}(1)$
- DeleteMin . . . $\mathcal{O}(\log n)$ & $\Omega(\log n)$ (implied by the sorting lower bound)
- Decrease . . . $\mathcal{O}(1)$

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . $\mathcal{O}(1)$
- DeleteMin . . . $\mathcal{O}(\log n)$ & $\Omega(\log n)$ (implied by the sorting lower bound)
- Decrease . . . $\mathcal{O}(1)$

**Idea:** Can we sidestep the lower bound when the operations have structure?

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert ... $\mathcal{O}(1)$
- DeleteMin ... $\mathcal{O}(\log n)$ & $\Omega(\log n)$ (implied by the sorting lower bound)
- Decrease ... $\mathcal{O}(1)$

**Idea:** Can we sidestep the lower bound when the operations have structure?

- Long line of research for heaps & other data structures [ST85; Fre+86; Iac00; Pet05; Elm06; Dem+07; EFI12; BHM13; EFI13; IÖ14; KS18; ST23]

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert . . . $\mathcal{O}(1)$
- DeleteMin . . . $\mathcal{O}(\log n)$ & $\Omega(\log n)$ (implied by the sorting lower bound)
- Decrease . . . $\mathcal{O}(1)$

**Idea:** Can we sidestep the lower bound when the operations have structure?

- Long line of research for heaps & other data structures [ST85; Fre+86; Iac00; Pet05; Elm06; Dem+07; EFI12; BHM13; EFI13; IÖ14; KS18; ST23]
  - Splay trees: dynamic optimality conjecture [ST85; Iac13]

# Nice Heaps

A *heap* is a data structure for storing a collection of items.

- Insert ... $\mathcal{O}(1)$
- DeleteMin ... $\mathcal{O}(\log n)$ & $\Omega(\log n)$ (implied by the sorting lower bound)
- Decrease ... $\mathcal{O}(1)$

**Idea:** Can we sidestep the lower bound when the operations have structure?

- Long line of research for heaps & other data structures [ST85; Fre+86; Iac00; Pet05; Elm06; Dem+07; EFI12; BHM13; EFI13; IÖ14; KS18; ST23]
  - Splay trees: dynamic optimality conjecture [ST85; Iac13]
  - No analogous conjecture for heaps

# Working-Set Bound

- **Idea:** be faster when there's temporal locality

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$ where $t = \#$operations since the item was inserted

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$ where $t = \#$operations since the item was inserted

$$\underbrace{\text{Ins}(999),\ \text{Ins}(999),\ \ldots,\ \text{Ins}(999)}_{k-1}$$

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$ where $t = \#$operations since the item was inserted

$$\underbrace{\mathrm{Ins}(999),\ \mathrm{Ins}(999),\ \ldots,\ \mathrm{Ins}(999)}_{k-1},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \ldots$$

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$
  where $t = \#$operations since the item was inserted

$$\underbrace{\mathrm{Ins}(999),\ \mathrm{Ins}(999),\ \ldots,\ \mathrm{Ins}(999)}_{k-1},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \mathrm{Ins}(1),\ \mathrm{Del},\quad \ldots$$

regular heap:

$\Theta(\log(\textit{heap size})) = \Theta(\log k)$

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$
  where $t = \#$operations since the item was inserted

$$\underbrace{\text{Ins}(999), \text{Ins}(999), \ldots, \text{Ins}(999)}_{k-1}, \quad \text{Ins}(1), \text{Del}, \quad \text{Ins}(1), \text{Del}, \quad \text{Ins}(1), \text{Del}, \quad \ldots$$

regular heap:
$\Theta(\log(\textit{heap size})) = \Theta(\log k)$

working-set heap:
$\mathcal{O}(\log(\textit{item age})) = \mathcal{O}(1)$

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$
  where $t = \#$operations since the item was inserted

$$\underbrace{\text{Ins}(999), \text{Ins}(999), \ldots, \text{Ins}(999)}_{k-1}, \quad \text{Ins}(1), \text{Del}, \quad \text{Ins}(1), \text{Del}, \quad \text{Ins}(1), \text{Del}, \quad \ldots$$

regular heap:
$\Theta(\log(\textit{heap size})) = \Theta(\log k)$

working-set heap:
$\mathcal{O}(\log(\textit{item age})) = \mathcal{O}(1)$

Our result: Dijkstra with **any** working-set heap is universally optimal.

# Working-Set Bound

- **Idea:** be faster when there's temporal locality
- **Working-set bound:** the cost of DeleteMin is $\mathcal{O}(\log t)$ where $t = \#$operations since the item was inserted

$$\underbrace{\text{Ins}(999),\ \text{Ins}(999),\ \ldots,\ \text{Ins}(999)}_{k-1},\quad \text{Ins}(1),\ \text{Del},\quad \text{Ins}(1),\ \text{Del},\quad \text{Ins}(1),\ \text{Del},\quad \ldots$$

regular heap:
$\Theta(\log(\textit{heap size})) = \Theta(\log k)$

working-set heap:
$\mathcal{O}(\log(\textit{item age})) = \mathcal{O}(1)$

- side result: we construct a working-set heap with $\mathcal{O}(1)$ Decrease

Our result: Dijkstra with **any** working-set heap is universally optimal.

# Dijkstra + Nice Heaps

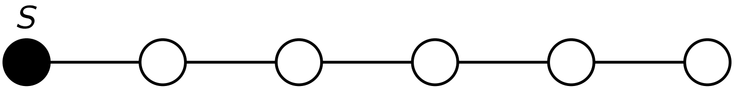
- Time complexity (with a fast heap): $\mathcal{O}(n + m + \textit{cost of all DeleteMins})$

# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$

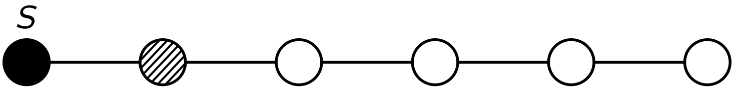
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



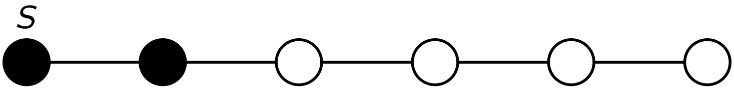
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



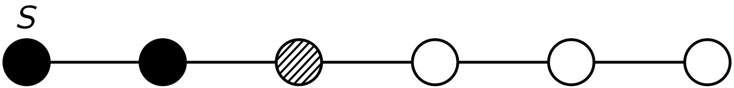
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



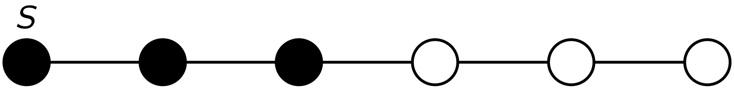
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



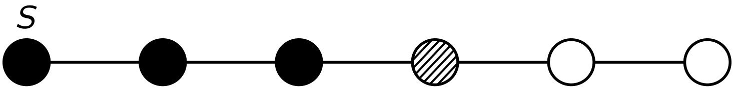
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



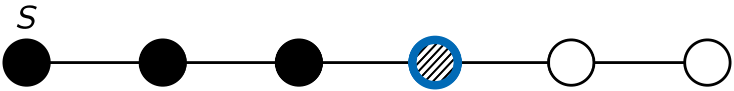
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



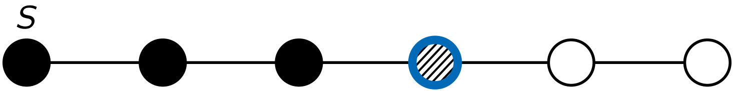
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



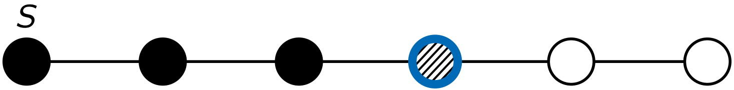
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$



# Dijkstra + Nice Heaps

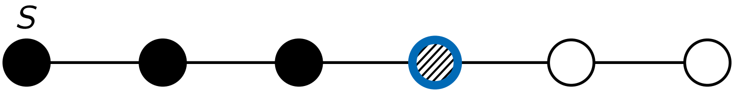
- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$




- working-set heap: $\mathcal{O}(\log(\textit{time since insertion})) = \mathcal{O}(1)$

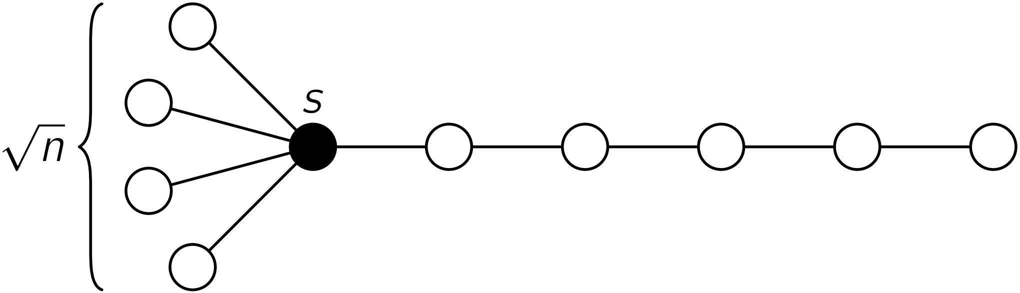
# Dijkstra + Nice Heaps

- Time complexity (with a fast heap): $\mathcal{O}(n + m + \underbrace{\textit{cost of all DeleteMins}}_{\text{goal: make this small}})$

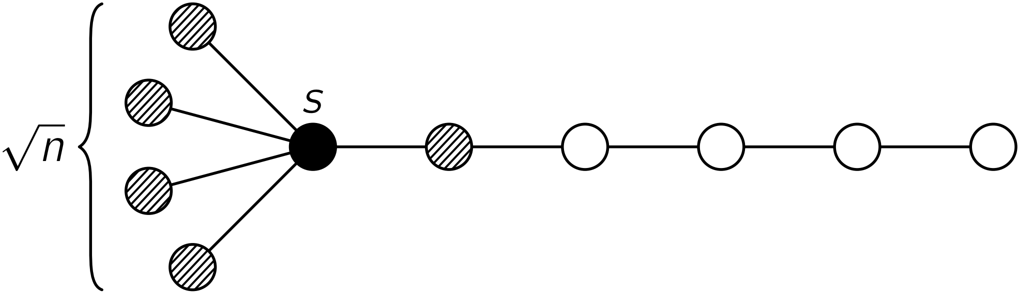



- working-set heap: $\mathcal{O}(\log(\textit{time since insertion})) = \mathcal{O}(1)$
- Fibonacci heap: $\mathcal{O}(\log(\textit{heap size})) = \mathcal{O}(1)$

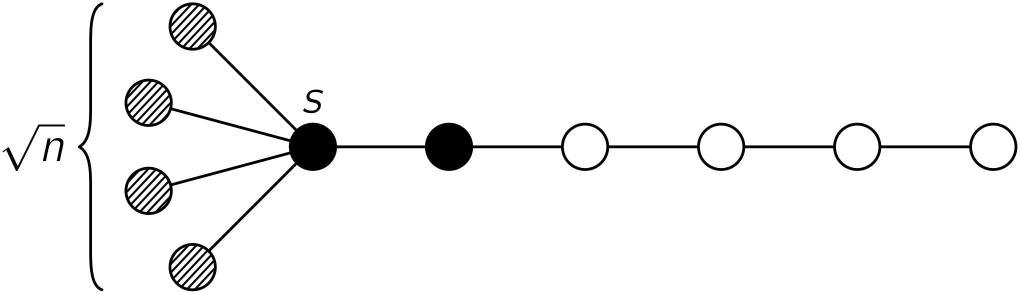
# Dijkstra + Nice Heaps



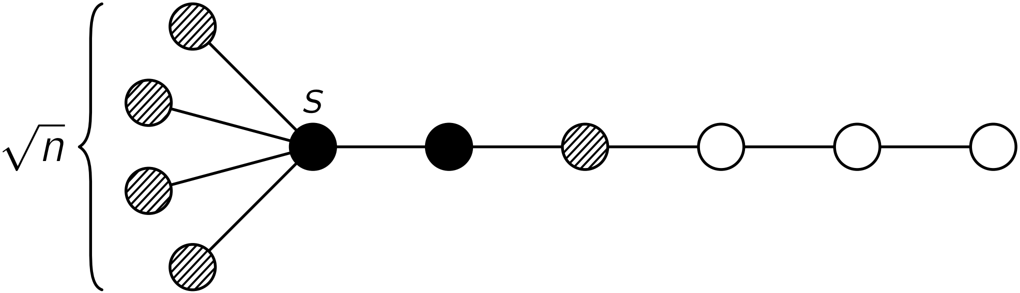
# Dijkstra + Nice Heaps



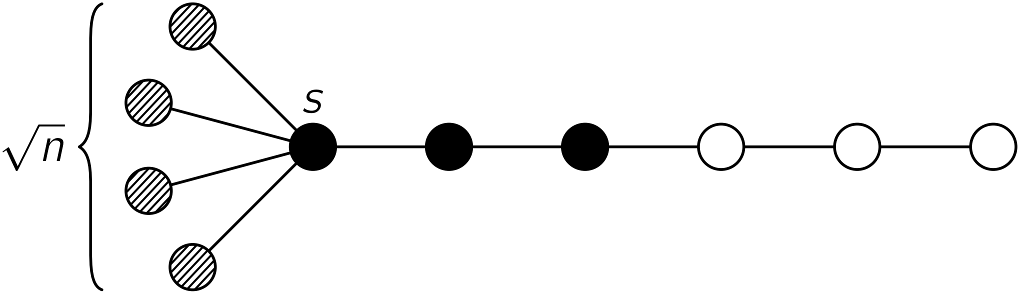
# Dijkstra + Nice Heaps



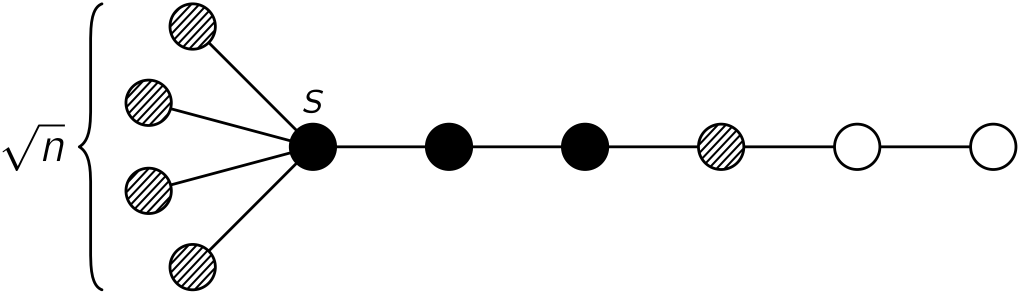
# Dijkstra + Nice Heaps



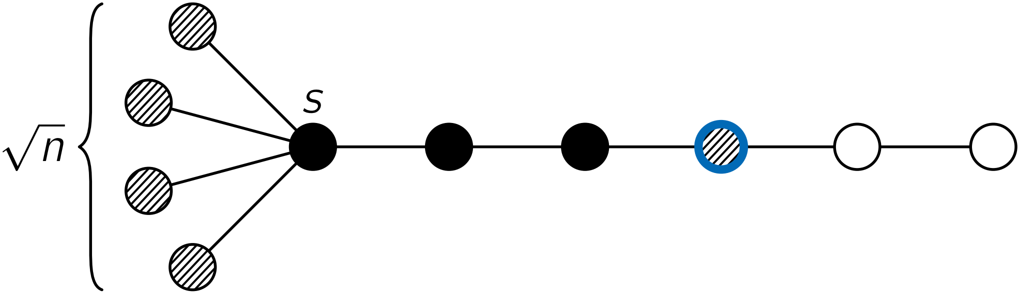
# Dijkstra + Nice Heaps



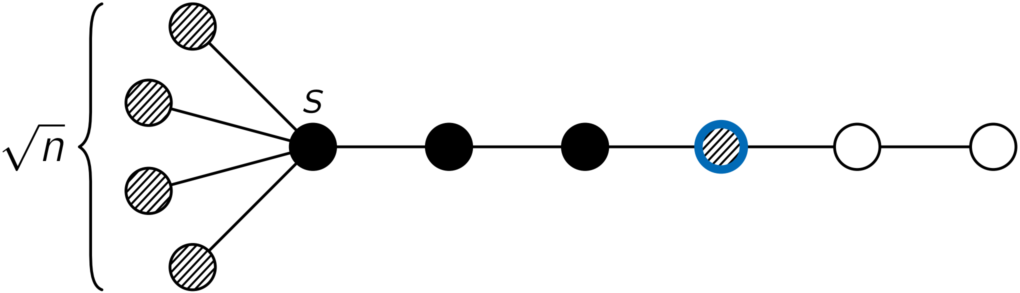
# Dijkstra + Nice Heaps



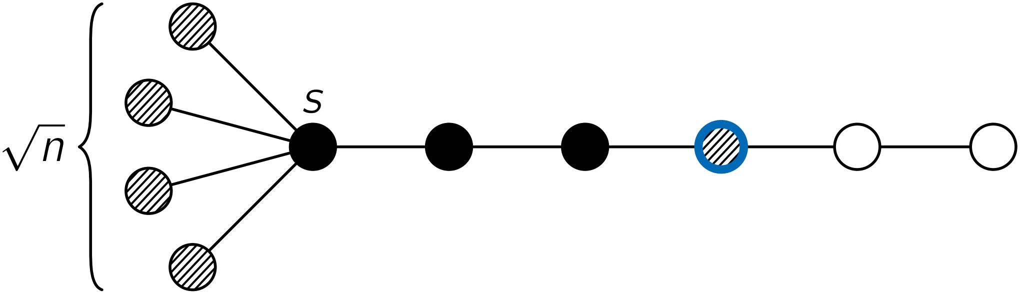
# Dijkstra + Nice Heaps



# Dijkstra + Nice Heaps

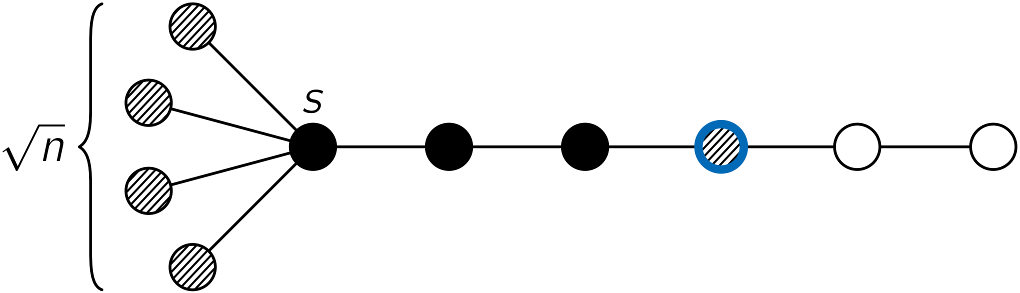


# Dijkstra + Nice Heaps



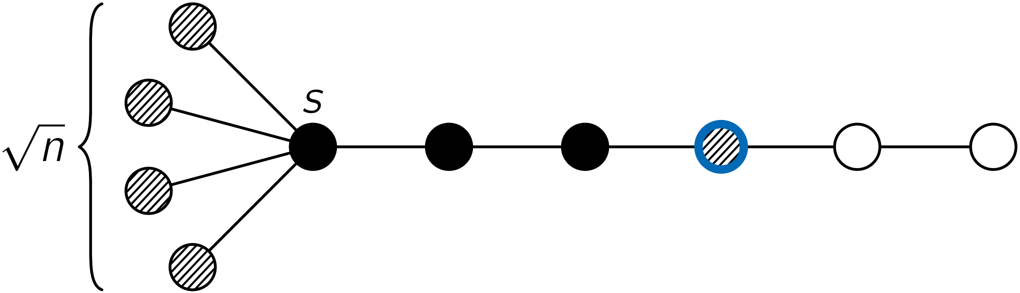

- Fibonacci heap: $\Theta(\log(\textit{heap size})) = \Theta(\log\sqrt{n}) = \Theta(\log n)$

# Dijkstra + Nice Heaps



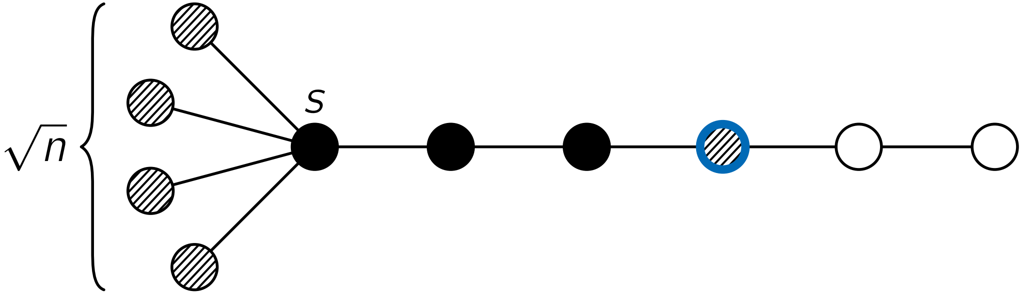

- Fibonacci heap: $\Theta(\log(\textit{heap size})) = \Theta(\log \sqrt{n}) = \Theta(\log n)$
- working-set heap: $\mathcal{O}(\log(\textit{time since insertion})) = \mathcal{O}(1)$

# Dijkstra + Nice Heaps



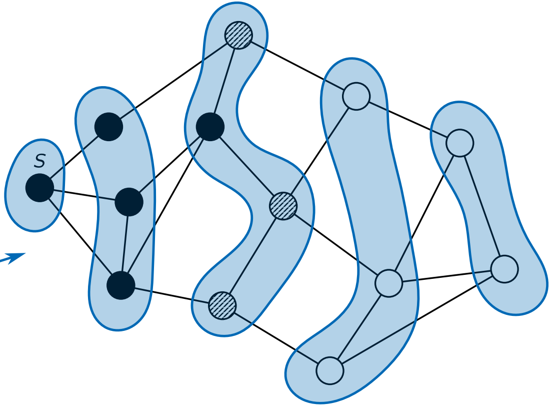

- Fibonacci heap: $\Theta(\log(\textit{heap size})) = \Theta(\log \sqrt{n}) = \Theta(\log n) \implies \Theta(n \log n)$ total
- working-set heap: $\mathcal{O}(\log(\textit{time since insertion})) = \mathcal{O}(1)$

# Dijkstra + Nice Heaps



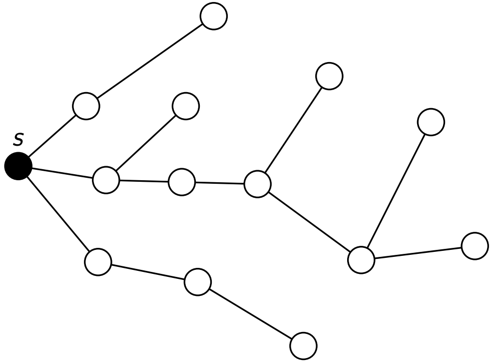

- Fibonacci heap: $\Theta(\log(\textit{heap size})) = \Theta(\log\sqrt{n}) = \Theta(\log n) \implies \Theta(n\log n)$ total
- working-set heap: $\mathcal{O}(\log(\textit{time since insertion})) = \mathcal{O}(1) \implies \mathcal{O}(n)$ total

# Proof Intuition



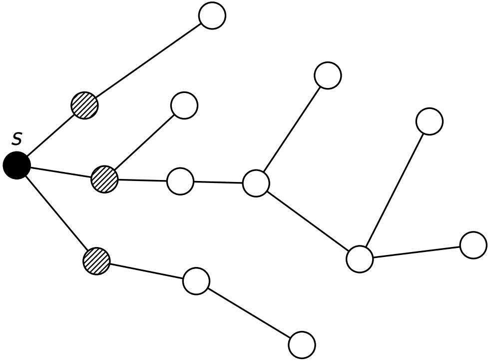

lower bound
matching the cost
of DeleteMins

# Proof Intuition



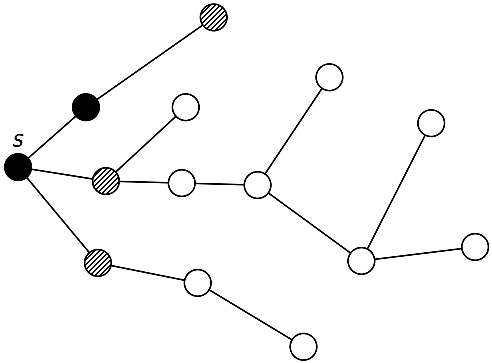

- Let Dijkstra run.

# Proof Intuition



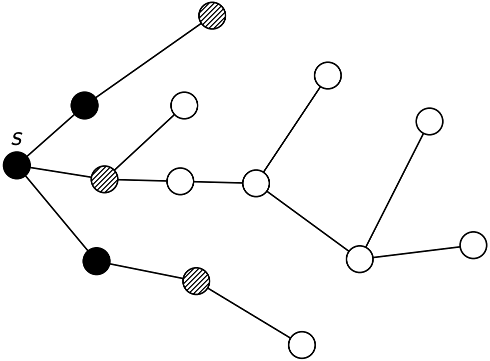

- Let Dijkstra run.

# Proof Intuition



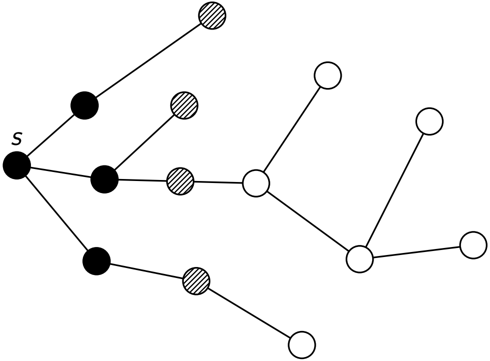

- Let Dijkstra run.

# Proof Intuition



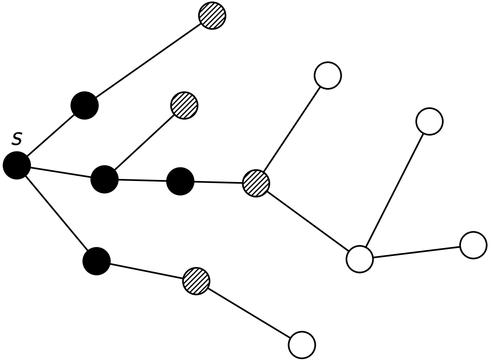

- Let Dijkstra run.

# Proof Intuition



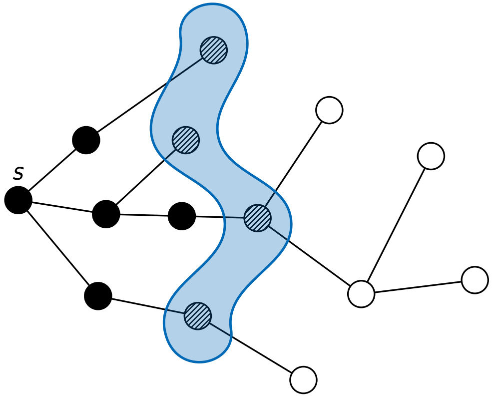

- Let Dijkstra run.

# Proof Intuition



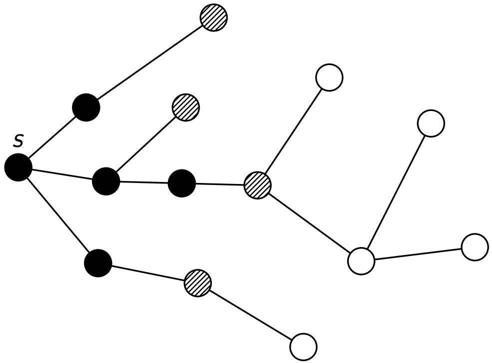

- Let Dijkstra run.

# Proof Intuition



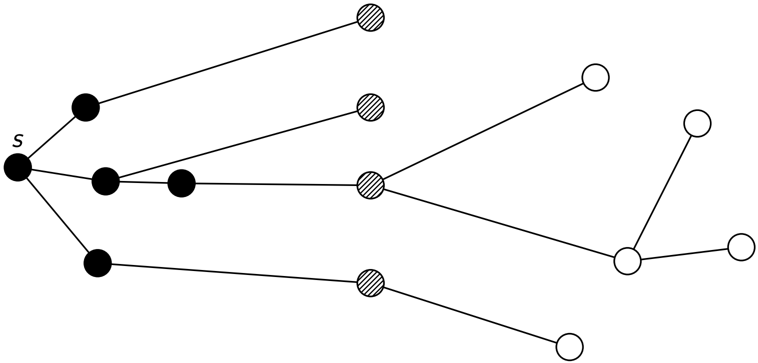

- Pause Dijkstra at any time. $B := |\textit{exploration boundary}|$.
- Claim: All $B!$ possible orderings of the exploration boundary are possible.
  ($\implies$ the algorithm needs to do $\Omega(B \log B)$ comparisons)

# Proof Intuition



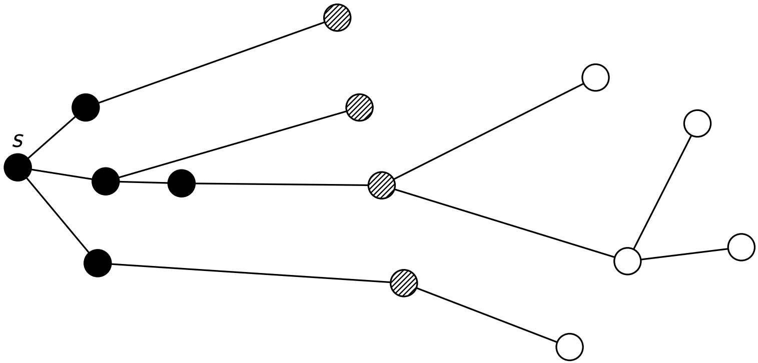

- Claim: All $B!$ possible orderings of the exploration boundary are possible.
- Proof: by picture.

# Proof Intuition



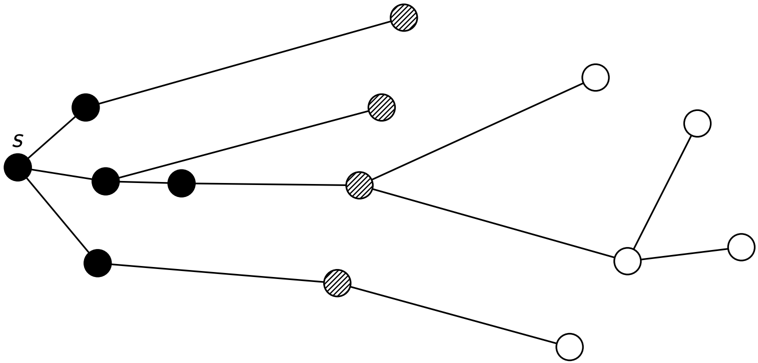

- Claim: All $B!$ possible orderings of the exploration boundary are possible.
- Proof: by picture.

# Proof Intuition



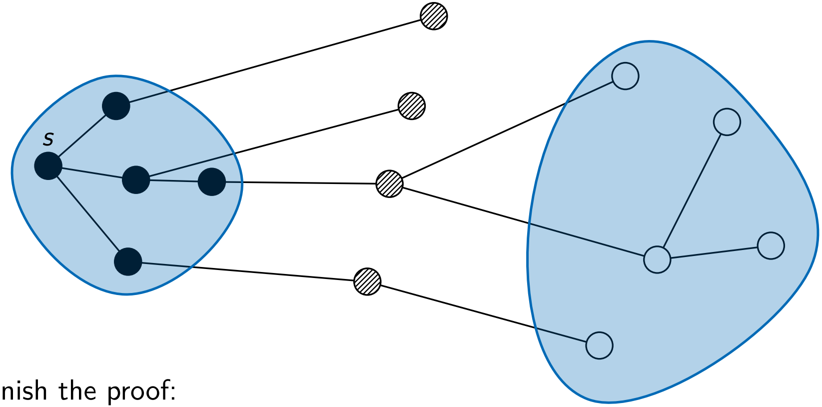

- Claim: All $B!$ possible orderings of the exploration boundary are possible.
- Proof: by picture.

# Proof Intuition




- Claim: All $B!$ possible orderings of the exploration boundary are possible.
- Proof: by picture.

# Proof Intuition




How to finish the proof:

1. Charge some DeleteMins to this exploration boundary.
2. Continue recursively on left and right parts.

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely:

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph.}}^{\text{universal optimality}}$$

- There are islands where universal(-like) optimality works nicely: distributed computing

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every}}^{\text{universal optimality}} \text{ graph.}$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24]

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24], sequential estimation [VV16; VV17; HLY21]

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24], sequential estimation [VV16; VV17; HLY21], bandit problems [LR85; CL16; CLQ17; Kir+21; Li+22]

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24], sequential estimation [VV16; VV17; HLY21], bandit problems [LR85; CL16; CLQ17; Kir+21; Li+22]
- Can we connect some of them?

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice heap}}^{\text{working-set bound}} \text{ is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24], sequential estimation [VV16; VV17; HLY21], bandit problems [LR85; CL16; CLQ17; Kir+21; Li+22]
- Can we connect some of them?
- Can we find more?

# Conclusion

$$\text{Dijkstra} + \overbrace{\text{nice}}^{\text{working-set bound}} \text{ heap is } \overbrace{\text{optimal on every graph}}^{\text{universal optimality}}.$$

- There are islands where universal(-like) optimality works nicely: distributed computing, data structures, sorting-based algorithms [DLM00; ABC17; Hae+24; HRR24], sequential estimation [VV16; VV17; HLY21], bandit problems [LR85; CL16; CLQ17; Kir+21; Li+22]
- Can we connect some of them?
- Can we find more?

PDF of slides & more

Thank you!

# Bibliography I


[LR85] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. "Self-adjusting binary search trees". In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 652–686.

[Fre+86] Michael L Fredman et al. "The pairing heap: A new form of self-adjusting heap". In: *Algorithmica* 1.1-4 (1986), pp. 111–129.

[FW93] Michael L. Fredman and Dan E. Willard. "Surpassing the information theoretic bound with fusion trees". In: *Journal of Computer and System Sciences* 47.3 (1993), pp. 424–436. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(93)90040-4. URL: https://www.sciencedirect.com/science/article/pii/0022000093900404.

[FW94] Michael L. Fredman and Dan E. Willard. "Trans-dichotomous algorithms for minimum spanning trees and shortest paths". In: *Journal of Computer and System Sciences* 48.3 (1994), pp. 533–551. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(05)80064-9. URL: https://www.sciencedirect.com/science/article/pii/S0022000005800649.

[Ram96] Rajeev Raman. "Priority Queues: Small, Monotone and Trans-Dichotomous". In: *Proceedings of the Fourth Annual European Symposium on Algorithms*. ESA '96. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 121–137. ISBN: 3540616802.

[Ram97] Rajeev Raman. "Recent Results on the Single-Source Shortest Paths Problem". In: *SIGACT News* 28.2 (June 1997), pp. 81–87. ISSN: 0163-5700. DOI: 10.1145/261342.261352. URL: https://doi.org/10.1145/261342.261352.

[GKP98] Juan A Garay, Shay Kutten, and David Peleg. "A sublinear time distributed algorithm for minimum-weight spanning trees". In: *SIAM Journal on Computing* 27.1 (1998), pp. 302–316.

# Bibliography II


[Tho99] Mikkel Thorup. "Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time". In: *J. ACM* 46.3 (May 1999), pp. 362–394. ISSN: 0004-5411. DOI: 10.1145/316542.316548. URL: https://doi.org/10.1145/316542.316548.

[DLM00] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. "Adaptive set intersections, unions, and differences". In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. 2000, pp. 743–752.

[Hag00] Torben Hagerup. "Improved Shortest Paths on the Word RAM". In: *Automata, Languages and Programming*. Ed. by Ugo Montanari, José D. P. Rolim, and Emo Welzl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 61–72. ISBN: 978-3-540-45022-1.

[Iac00] John Iacono. "Improved upper bounds for pairing heaps". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2000, pp. 32–45.

[Tho00a] Mikkel Thorup. "Floats, Integers, and Single Source Shortest Paths". In: *J. Algorithms* 35.2 (May 2000), pp. 189–201. ISSN: 0196-6774. DOI: 10.1006/jagm.2000.1080. URL: https://doi.org/10.1006/jagm.2000.1080.

[Tho00b] Mikkel Thorup. "On RAM Priority Queues". In: *SIAM Journal on Computing* 30.1 (2000), pp. 86–109. DOI: 10.1137/S0097539795288246. URL: https://doi.org/10.1137/S0097539795288246.

[Tho04] Mikkel Thorup. "Integer priority queues with decrease key in constant time and the single source shortest paths problem". In: *Journal of Computer and System Sciences* 69.3 (2004). Special Issue on STOC 2003, pp. 330–353. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2004.04.003. URL: https://www.sciencedirect.com/science/article/pii/S002200000400042X.

[Pet05] Seth Pettie. "Towards a final analysis of pairing heaps". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. IEEE. 2005, pp. 174–183.

# Bibliography III


[Elm06] Amr Elmasry. "A priority queue with the working-set property". In: *International Journal of Foundations of Computer Science* 17.06 (2006), pp. 1455–1465.

[Dem+07] Erik D Demaine et al. "Dynamic optimality—almost". In: *SIAM Journal on Computing* 37.1 (2007), pp. 240–251.

[EFI12] Amr Elmasry, Arash Farzan, and John Iacono. "A priority queue with the time-finger property". In: *Journal of Discrete Algorithms* 16 (2012), pp. 206–212.

[BHM13] Prosenjit Bose, John Howat, and Pat Morin. "A history of distribution-sensitive data structures". In: *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday* (2013), pp. 133–149.

[EFI13] Amr Elmasry, Arash Farzan, and John Iacono. "On the hierarchy of distribution-sensitive properties for data structures". In: *Acta informatica* 50.4 (2013), pp. 289–295.

[Iac13] John Iacono. "In pursuit of the dynamic optimality conjecture". In: *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. Springer, 2013, pp. 236–250.

[IÖ14] John Iacono and Özgür Özkan. "A tight lower bound for decrease-key in the pure heap model". In: *arXiv preprint arXiv:1407.6665* (2014).

[CL16] Lijie Chen and Jian Li. "Open problem: Best arm identification: Almost instance-wise optimality and the gap entropy conjecture". In: *Conference on Learning Theory*. PMLR. 2016, pp. 1643–1646.

[VV16] Gregory Valiant and Paul Valiant. "Instance optimal learning of discrete distributions". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 142–155.

# Bibliography IV


[ABC17] Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. "Instance-Optimal Geometric Algorithms". In: *J. ACM* 64.1 (Mar. 2017). ISSN: 0004-5411. DOI: 10.1145/3046673. URL: https://doi.org/10.1145/3046673.

[CLQ17] Lijie Chen, Jian Li, and Mingda Qiao. "Towards instance optimal bounds for best arm identification". In: *Conference on Learning Theory*. PMLR. 2017, pp. 535–592.

[VV17] Gregory Valiant and Paul Valiant. "An automatic inequality prover and instance optimal identity testing". In: *SIAM Journal on Computing* 46.1 (2017), pp. 429–455.

[KS18] László Kozma and Thatchaphol Saranurak. "Smooth heaps and a dual view of self-adjusting data structures". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 2018, pp. 801–814.

[HWZ21] Bernhard Haeupler, David Wajc, and Goran Zuzic. "Universally-optimal distributed algorithms for known topologies". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 1166–1179.

[HLY21] Ziyue Huang, Yuting Liang, and Ke Yi. "Instance-optimal mean estimation under differential privacy". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 25993–26004.

[Kir+21] Johannes Kirschner et al. "Asymptotically optimal information-directed sampling". In: *Conference on Learning Theory*. PMLR. 2021, pp. 2777–2821.

[GZ22] Mohsen Ghaffari and Goran Zuzic. "Universally-optimal distributed exact min-cut". In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. 2022, pp. 281–291.

[Li+22] Zhaoqi Li et al. "Instance-optimal pac algorithms for contextual bandits". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37590–37603.

# Bibliography V


[Roz+22] Václav Rozhoň et al. "Undirected (1+eps)-Shortest Paths via Minor-Aggregates: Near-Optimal Deterministic Parallel and Distributed Algorithms". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, pp. 478–487. ISBN: 9781450392648. DOI: 10.1145/3519935.3520074. URL: https://doi.org/10.1145/3519935.3520074.

[Zuz+22] Goran Zuzic et al. "Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based L1-Oblivious Routing". In: *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022.

[Dua+23] Ran Duan et al. "A Randomized Algorithm for Single-Source Shortest Path on Undirected Real-Weighted Graphs". In: *arXiv preprint arXiv:2307.04139* (2023).

[ST23] Corwin Sinnamon and Robert E Tarjan. "Efficiency of Self-Adjusting Heaps". In: *arXiv preprint arXiv:2307.02772* (2023).

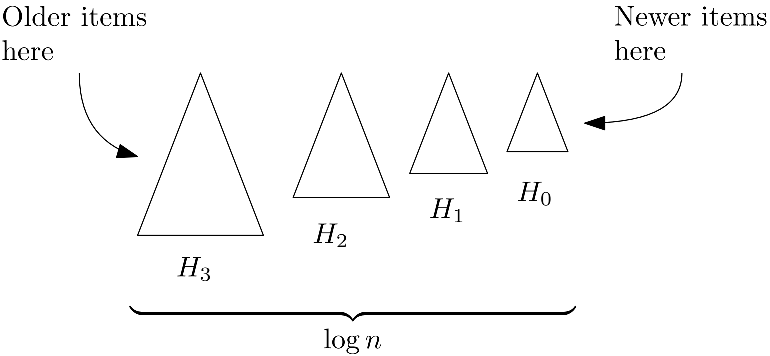
[Hae+24] Bernhard Haeupler et al. *Fast and Simple Sorting Using Partial Information*. 2024. arXiv: 2404.04552 [cs.DS].

[HRR24] Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. *Simpler Optimal Sorting from a Directed Acyclic Graph*. 2024. arXiv: 2407.21591 [cs.DS].

# ✨ Backup Slides ✨

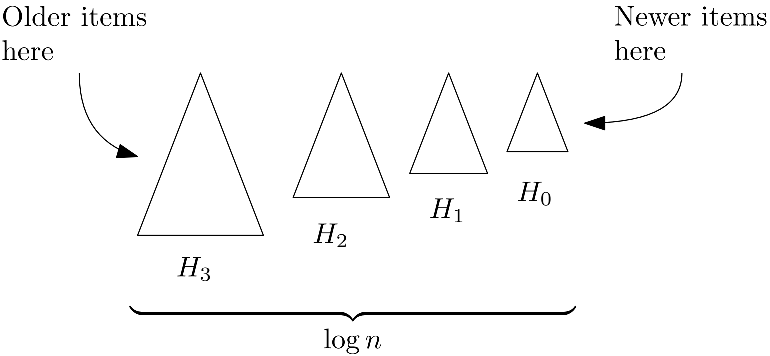
# Our heap

# Our heap




- $|H_r| \leq 2^r$

# Our heap




- $|H_r| \leq 2^r$
- guarantee: if $x \in H_r$, then $|W_x| = \Omega(2^r)$

# Two definitions of a working set

**Definition 1:** A heap has the *weak* working-set property if DeleteMin that deletes item $x$ has amortized cost $\mathcal{O}(\log t_x)$ where $t_x$ is the number of operations elapsed between inserting and deleting $x$.

**Definition 2:** A heap has the *medium* working-set property if DeleteMin that deletes item $x$ has amortized cost $\mathcal{O}(\log |W_x|)$ where $W_x$ is the maximum-cardinality set of items such that:

- all items in $W_x$ were inserted (non-strictly) after $x$,
- there was a moment in time where all items in $W_x$ were simultaneously in the heap.

**Claim:** Definitions 1 and 2 are equivalent.