Learning-Augmented Data Structures

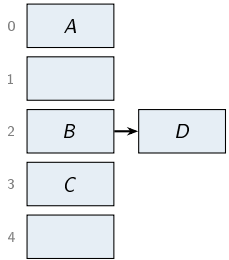
Alberto Vendramini

Scuola Ortogonale

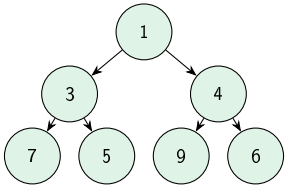
May 2026

Classical Data Structures

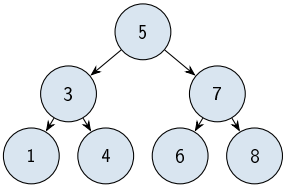
Hash Table



Min/Max Heap

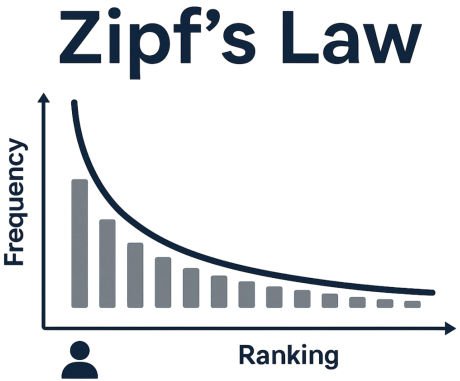


Binary Search Tree



and many more...

Zipfian Distribution



Zipfian Distribution

Frequency and rank are **inversely proportional**:

$$\text{freq}(k) \propto \frac{1}{k^s}, \quad s \approx 1$$

The k-th most popular element is accessed k times less often than the most popular.

Key Idea – Core Principle

Learning-Augmented Paradigm

Enhance algorithms and data structures with **predictions** about future inputs, access patterns, or distributions.

Predictions can come from **ML models** or **heuristics**.

Key Idea – Robustness

Robustness

If predictions are **wrong**, the algorithm *does not collapse*.

Performance degrades gracefully and stays close to the **classic baseline**.

Key Idea – Consistency

Consistency

If predictions are **correct**, obtain a clear, *measurable advantage* over the classic version.

Summary

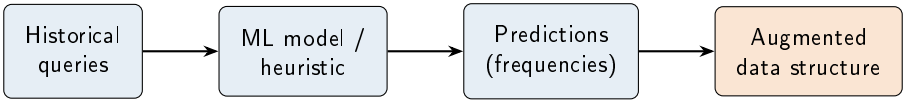
Consistency: good predictions $\Rightarrow$ better than classical cost.

Robustness: bad predictions $\Rightarrow$ fallback to the classical baseline.

From Algorithms to Data Structures

Francesco Visonà

Key Idea – The Common Pattern



What We Will See

Two case studies of learning-augmented data structures:

1. **Treaps** – a randomized BST whose priorities are replaced by predicted access frequencies.

What We Will See

Two case studies of learning-augmented data structures:

1. **Treaps** – a randomized BST whose priorities are replaced by predicted access frequencies.
2. **Skip Lists** – a layered linked structure where hot elements are promoted to higher levels.

What We Will See

Two case studies of learning-augmented data structures:

1. **Treaps** – a randomized BST whose priorities are replaced by predicted access frequencies.
2. **Skip Lists** – a layered linked structure where hot elements are promoted to higher levels.

For each structure we will examine:

- how predictions are **incorporated**,
- the **complexity gains** when predictions are accurate, and
- the **robustness guarantees** when predictions are wrong.

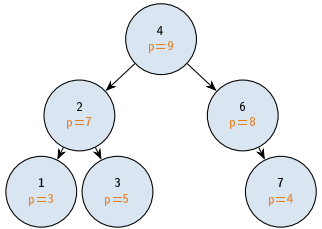
Treaps

What is a Treap?

A **treap** (= **tree** + **heap**) is a randomized binary search tree introduced by Seidel & Aragon (1996).

Each node stores two values:

Example treap (key / priority):



BST on keys + max-heap on priorities.

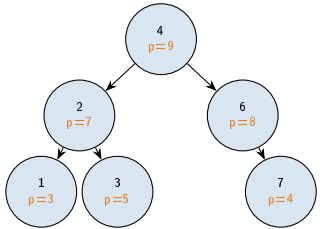
What is a Treap?

A **treap** (= **tree** + **heap**) is a randomized binary search tree introduced by Seidel & Aragon (1996).

Each node stores two values:

- a **key** – used to maintain the BST ordering,

Example treap (key / priority):



BST on keys + max-heap on priorities.

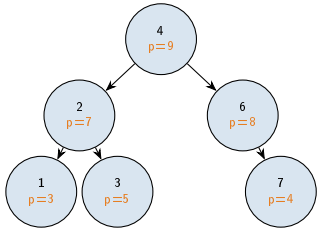
What is a Treap?

A **treap** (= **tree** + **heap**) is a randomized binary search tree introduced by Seidel & Aragon (1996).

Each node stores two values:

- a **key** – used to maintain the BST ordering,
- a **priority** – assigned *randomly*, used to keep the tree balanced via a heap structure.
  $\Rightarrow$ All elements treated **equally** regardless of access frequency.

Example treap (key / priority):



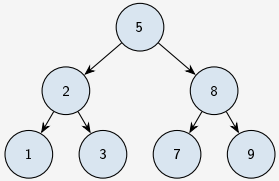
BST on keys + max-heap on priorities.

BST Invariant

BST Invariant (on *keys*): for every node x in the tree:

$$\forall y \in \mathsf{left}(x) : \ \mathrm{key}(y) < \mathrm{key}(x) \qquad \forall z \in \mathsf{right}(x) : \ \mathrm{key}(z) > \mathrm{key}(x)$$

Example

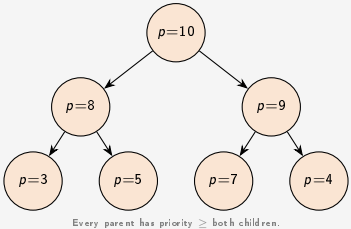


Max-Heap Invariant

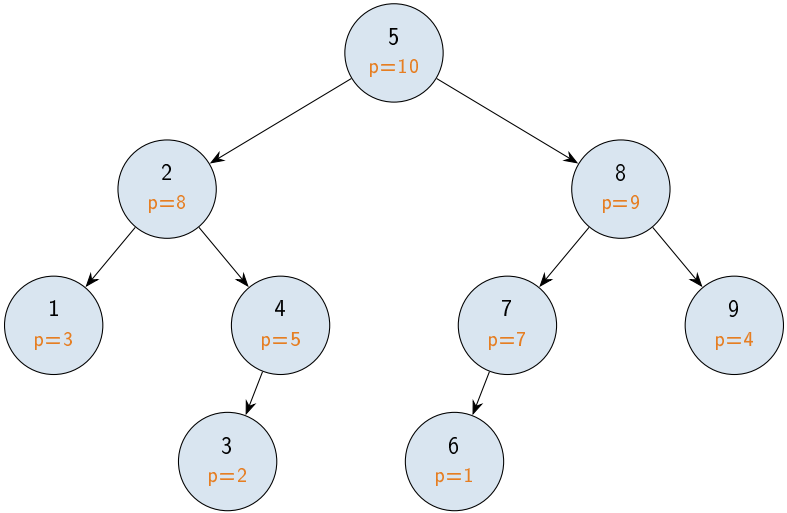
Max-Heap Invariant (on *priorities*): for every node x with children y_1, y_2:

$$\text{priority}(x) \geq \text{priority}(y_1) \qquad \text{and} \qquad \text{priority}(x) \geq \text{priority}(y_2)$$

Example



Treap – Example



Treap – The Idea

Assume access to a **noisy oracle** that predicts **frequency ranks** of elements.

Replace random priorities with **predicted frequency ranks**:

Higher frequency $\longrightarrow$ *higher priority* $\longrightarrow$ *closer to the root.*

- Frequently queried elements are found faster (**shorter search paths**).
- The tree is **shaped by the access distribution**, not by randomness.

Treap – Key Assumption

Assumption (Lin et al. 2022)

The **key values** are in **random order** with respect to predicted ranks.

Formally: there is **no adversarial correlation** between a key's BST position and its predicted access frequency.

Treap – Search

Access (Search)

Standard **BST search**: compare key at each node and go left or right.

- Classical treap: expected depth $\mathcal{O}(\log n)$.
- Learned treap: element with predicted rank i has expected depth $\mathcal{O}(\log i)$.
- **Hot elements** (low rank i) are close to the root $\Rightarrow$ found much faster.

Treap – Theorem 3.2 (Lin, Luo, Woodruff 2022)

Theorem

The expected depth of e_i in a learned treap is $\mathcal{O}(\log i)$ ***with high probability***.

Treap – Theorem 3.3 (Lin, Luo, Woodruff 2022)

Theorem

With constant probability, for every i, element e_i has depth $\mathcal{O}(\log i)$.

In other words, the ***entire tree*** *is well-balanced.*

Treap – Insert & Delete

Insertion

Insert as a **BST leaf**, then **rotate up** to restore the heap invariant.

Expected time: $\mathcal{O}(\log n)$.

Deletion

Rotate the node down until it is a leaf, then **remove** it.

Expected time: $\mathcal{O}(\log n)$.

Treap – Theorem: Insert, Delete

Theorem (Insert & Delete – Lin, Luo, Woodruff 2022)

*Both **Insert** and **Delete** run in $\mathcal{O}(\log n)$ expected time, matching the classical treap.*

Treaps – Robustness: Performance Guarantees

Robustness depends on the oracle error model.

- **Bounded noise** ($\hat{r}_i \leq \varepsilon r_i + \delta$ for constants $\varepsilon, \delta \geq 1$):
  Performance is *near-perfect*: $\mathbb{E}[\text{depth}(x_i)] = \mathcal{O}(\log i) + O(1)$,
  i.e. within an **additive constant** of the perfect-oracle treap.

Treaps – Robustness: Performance Guarantees

Robustness depends on the oracle error model.

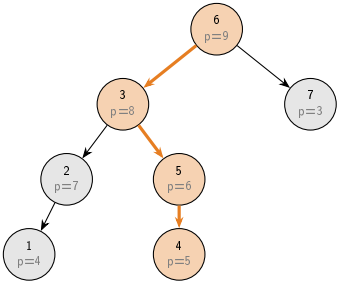
- **Bounded noise** ($\hat{r}_i \leq \varepsilon r_i + \delta$ for constants $\varepsilon, \delta \geq 1$):
  Performance is *near-perfect*: $\mathbb{E}[\text{depth}(x_i)] = \mathcal{O}(\log i) + O(1)$,
  i.e. within an **additive constant** of the perfect-oracle treap.
- **Inaccurate but non-adversarial oracle**:
  No worse than a *random treap*: $\mathbb{E}[\text{depth}(x_i)] \leq \mathcal{O}(\log n) + O(1)$,
  i.e. the **classical baseline** is preserved up to a small additive constant.

Treaps – Lifting the Assumption

- **Problem:** If key order and predicted ranks are **adversarially correlated**, the learned treap may become highly unbalanced $\Rightarrow$ depth $\mathcal{O}(n)$.
- **Fix:** assign each element i a **random surrogate key** s_i, build the learned treap on the surrogate keys.
- **Result:** the $\mathcal{O}(\log i)$ expected-depth guarantee is **recovered**.

Treaps – Classical vs. Learning-Augmented

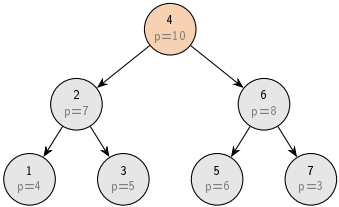
Classical Treap



Query 4: **4 steps** (depth 3).

Random priorities ⇒ hot item can end up deep.

Learned Treap



Query 4: **1 step** (root!).

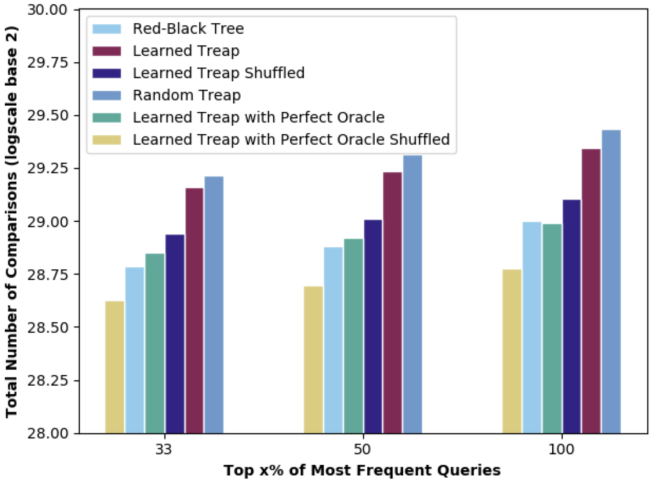
Predicted priorities ⇒ hot item at root.

Treaps – Complexity Comparison

Operation	Classical Treap	Learned Treap	Wrong Predictions*
Search / Access	$\mathcal{O}(\log n)$	$\mathcal{O}(\log i)$ for rank i	$\mathcal{O}(\log n)$
Insert	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Delete	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Priority update	N/A	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Space	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

*Under the robustness assumptions / randomized safeguard described above.

Treaps – Empirical Results



Skip Lists

What is a Skip List?

An **ordered data structure** supporting search, insert, and delete in $\mathcal{O}(\log n)$ on average.

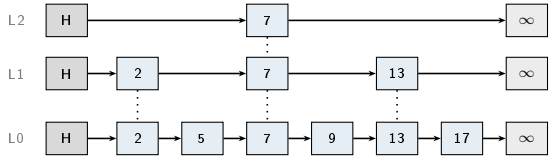
Core Idea

Build multiple **layers** on top of a sorted linked list:

Level 0: all n elements
Level k: $\approx n/2^k$ elements.

- To **search**: start at the top level, jump right, drop down when overshooting.
- Each level roughly halves the work $\Rightarrow$ total $\mathcal{O}(\log n)$ steps.

Example skip list ($n = 6$):



Skip List – Node Heights

Randomized Height Assignment

Each node's height is drawn i.i.d. from a **geometric distribution**:

$$P(\text{height} \geq k) = \frac{1}{2^k}$$

Skip List – Node Heights

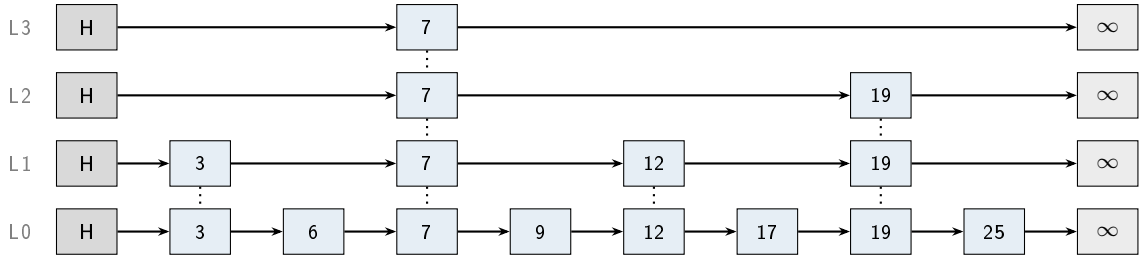
Randomized Height Assignment

Each node's height is drawn i.i.d. from a **geometric distribution**:

$$P(\text{height} \geq k) = \frac{1}{2^k}$$

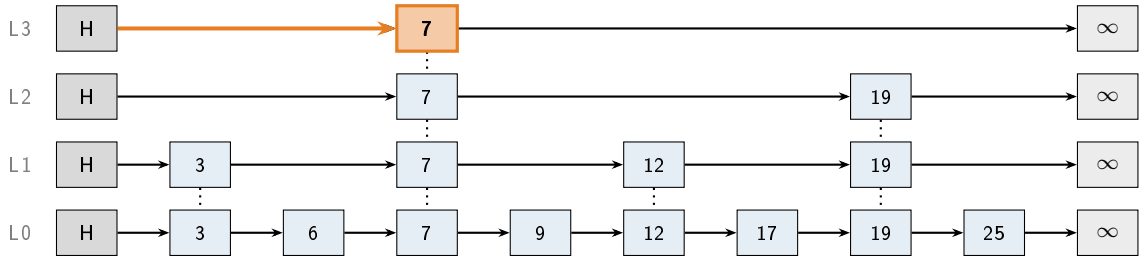
- Expected maximum height: $\mathcal{O}(\log n)$.
- Expected total space: $\mathcal{O}(n)$.
- All elements treated **equally** regardless of access frequency.

Skip List – Example (Search for **17**)



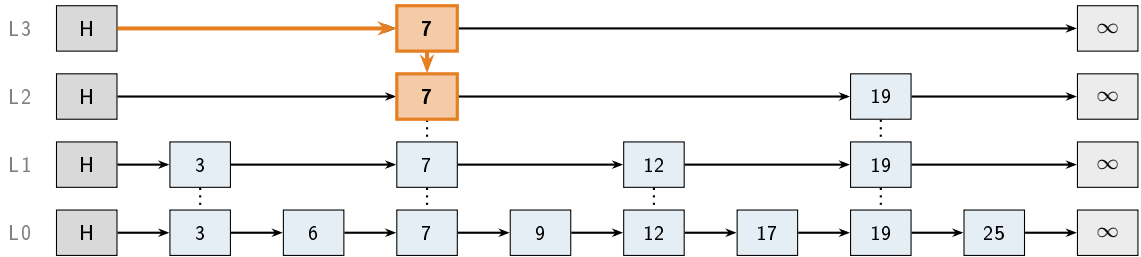
Start at L3 (top-left head) and search for **17**.

Skip List – Example (Search for **17**)



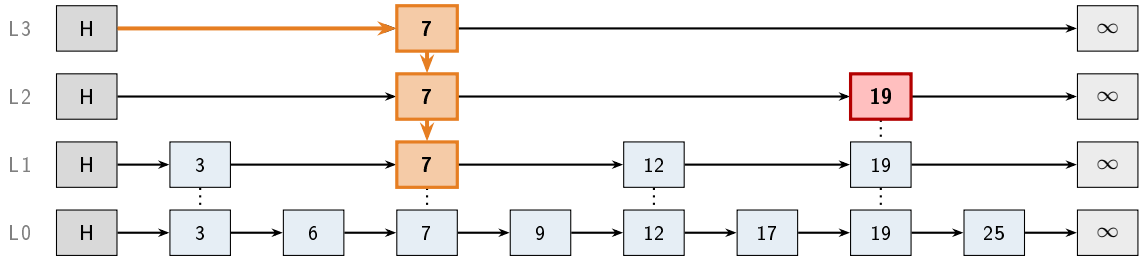
$7 < 17 \Rightarrow$ move **right** $\rightarrow$ at L3.

Skip List – Example (Search for **17**)



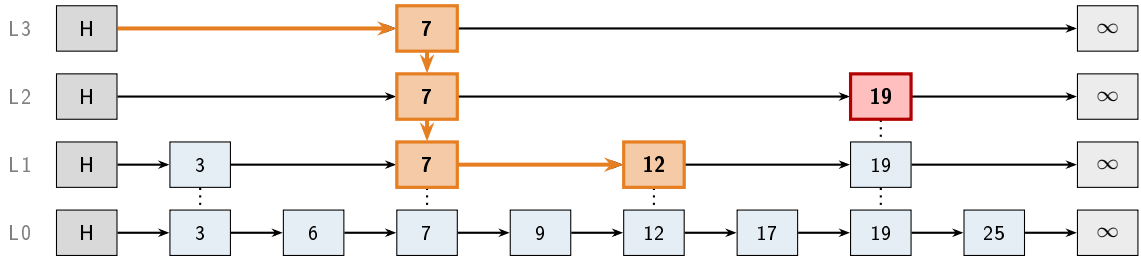
Next at L3 is $\infty > 17 \Rightarrow$ **drop down** $\downarrow$ to L2.

Skip List – Example (Search for **17**)



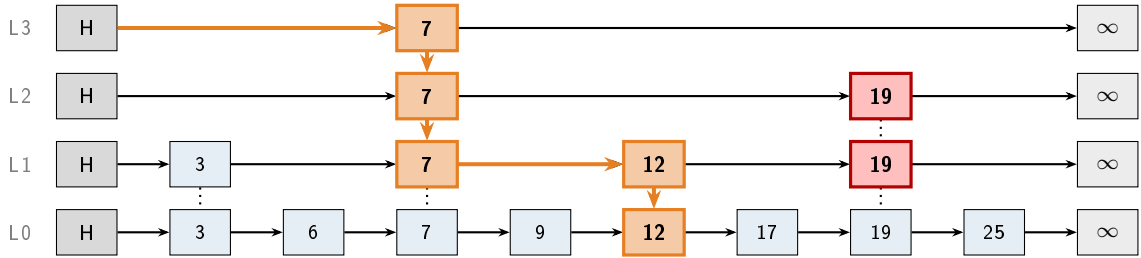
19 $>$ 17 $\Rightarrow$ **drop down** $\downarrow$ to L1.

Skip List – Example (Search for **17**)



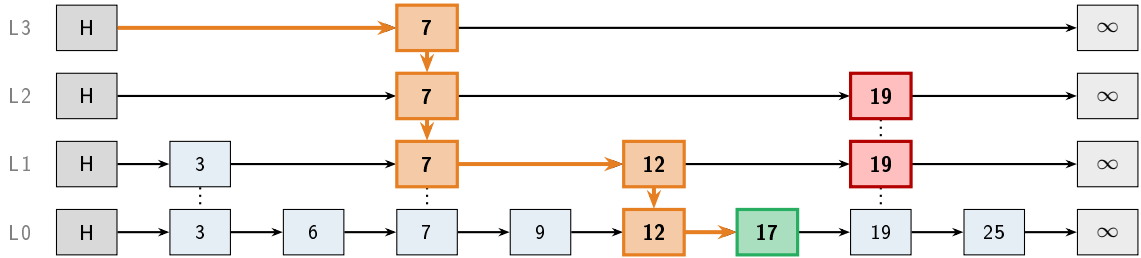
$12 < 17 \Rightarrow$ move **right** $\rightarrow$ at L1.

Skip List – Example (Search for **17**)



19 $>$ 17 $\Rightarrow$ **drop down** $\downarrow$ to L0.

Skip List – Example (Search for **17**)



$17 = 17 \Rightarrow$ **Found!**

Learning-Augmented Skip List – The Idea

Learned Skip List (Fu et al., ICLR 2025)

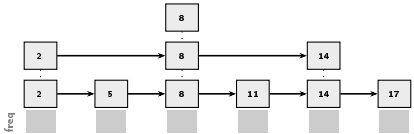
Use a **predicted access distribution** $\hat{p}_i$ to set node levels:

- **Hot items** ($\hat{p}_i \geq 2^{\ell-1}/n$): promoted to level ℓ **deterministically**.
- **Cold items**: if present at level $\ell-1$, promoted with probability $\frac{1}{2}$ (classical rule).

Level 0 contains all items, highways are shaped by the access distribution.

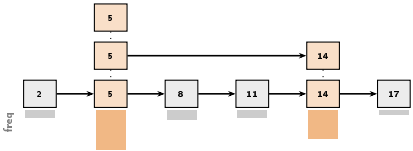
Skip Lists – Classical vs. Learning-Augmented

Classical Skip List



Uniform access $\Rightarrow$ random heights.

Learned Skip List



Skewed access $\Rightarrow$ hot items promoted higher.

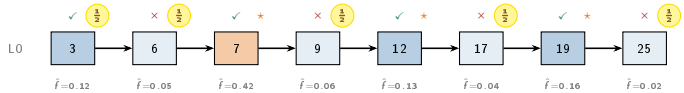
Learned Skip List – Layer Construction

Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).

★ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

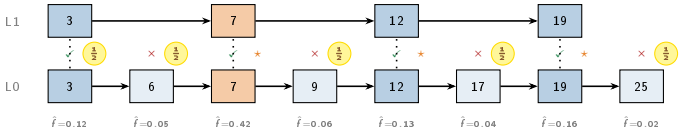
Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



⋆ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

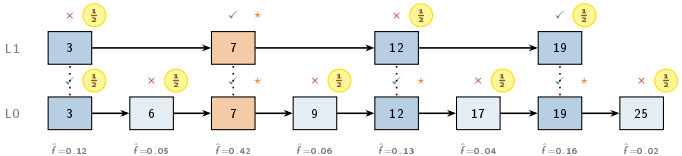
Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



⋆ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

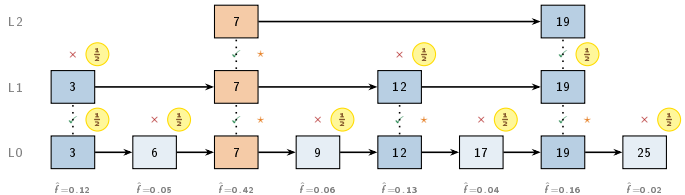
Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



⋆ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

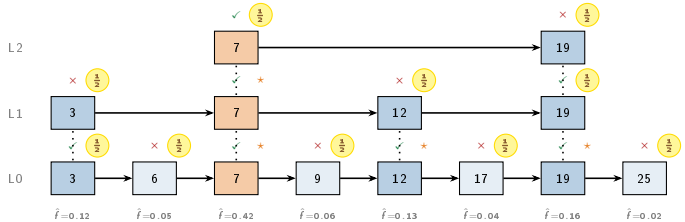
Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



★ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

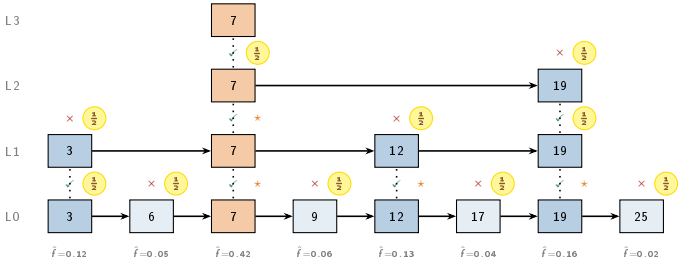
Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



★ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Learned Skip List – Layer Construction

Rule: at level ℓ, item i is included **deterministically** if $\hat{p}_i \geq 2^{\ell-1}/n$; otherwise it is promoted with probability $\frac{1}{2}$ (classical geometric rule).



⋆ deterministic (hot/warm: $\hat{p}_i \geq 2^{\ell-1}/n$) $\frac{1}{2}$ coin flip ($\hat{p}_i < 2^{\ell-1}/n$, promoted w.p. $\frac{1}{2}$)

Skip Lists – Height Lemma

Lemma (Height Bound)

With probability ≥ 0.99*, the skip list has at most* $10 + \log n$ *levels.*

Skip Lists – Main Theorems (Fu et al., ICLR 2025)

The Height Lemma is the key technical tool that drives the following two results.

Theorem (Per-Item Search Bound)

For each item i, the expected search cost satisfies

$$\mathbb{E}[T_i] = \mathcal{O}\left(\min\left(\log \frac{1}{p_i}, \log n\right)\right).$$

Skip Lists – Main Theorems (Fu et al., ICLR 2025)

The Height Lemma is the key technical tool that drives the following two results.

Theorem (Per-Item Search Bound)

For each item i, the expected search cost satisfies

$$\mathbb{E}[T_i] = \mathcal{O}\left(\min\left(\log \frac{1}{p_i}, \log n\right)\right).$$

Theorem (Expected Total Search Cost)

Let f_i be the true access frequency of item i. The expected total search time satisfies

$$\mathbb{E}[T] = \mathcal{O}\left(\sum_i f_i \cdot \min\left(\log \frac{1}{p_i}, \log n\right)\right).$$

Skip Lists – Complexity Comparison

Operation	Classical Skip List	Learned Skip List	Wrong Predictions
Search	$\mathcal{O}(\log n)$	$\mathcal{O}(\min(H(p), \log n))$	$\mathcal{O}(\log n)$
Insert	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Delete	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Space	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Construction	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

- $H(p) = \sum_i p_i \log \frac{1}{p_i}$ is the entropy of the query distribution.

Beyond Trees and Lists: KD-Trees

Classical KD-Tree

Splits hyperspace at the **median** of each dimension.

Agnostic to where queries arrive $\Rightarrow$ equal depth for hot and cold regions.

Search: $\mathcal{O}(\log n)$ on average.

Learned KD-Tree (Fu et al., ICLR 2025)

Splits chosen to **balance query probability mass**, not point count.

Finer partitions in high-density query regions; coarser elsewhere.

Search: $\mathcal{O}(\min(H(p), \log n))$.

Due to time constraints, we will not go into further detail on KD-Trees today.

References

Papers discussed in this talk

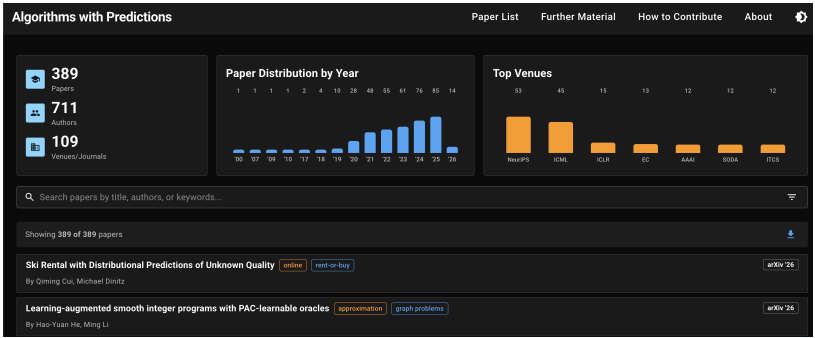
- **Learned BSTs / Treaps**
  Lin, Luo, Woodruff – *Learning Augmented Binary Search Trees*. ICML 2022.
- **Learned Skip Lists & KD-Trees**
  Fu, Nguyen, Seo, Zesch, Zhou – *Learning-Augmented Search Data Structures*. ICLR 2025.
- **Learned Priority Queues**
  Benomar, Coester – *Learning-Augmented Priority Queues*. NeurIPS 2024.
- **Binary Search with Predictions**
  Dinitz, Im, Lavastida, Moseley, Niaparast, Vassilvitskii – *Binary Search with Distributional Predictions*. NeurIPS 2024.
- **Learned Range Minimum Queries**
  Ferragina, Lari – *FL-RMQ: A Learned Approach to Range Minimum Queries*. CPM 2025.

Further Reading

Algorithms with Predictions – Community Resource

A curated, continuously updated collection of papers on learning-augmented algorithms and data structures:

`algorithms-with-predictions.github.io`



Summary

The Big Picture

Real-world query distributions are **skewed** (Zipfian). Classical data structures ignore this.

Learning-augmented structures exploit predictions to speed up hot queries while remaining **robust** when predictions fail.

Summary

The Big Picture

Real-world query distributions are **skewed** (Zipfian). Classical data structures ignore this.

Learning-augmented structures exploit predictions to speed up hot queries while remaining **robust** when predictions fail.

Treaps

Replace random priorities with **predicted ranks**.

Hot element of rank i: depth $\mathcal{O}(\log i)$ instead of $\mathcal{O}(\log n)$.

Summary

The Big Picture

Real-world query distributions are **skewed** (Zipfian). Classical data structures ignore this.

Learning-augmented structures exploit predictions to speed up hot queries while remaining **robust** when predictions fail.

Treaps

Replace random priorities with **predicted ranks**.

Hot element of rank i: depth $\mathcal{O}(\log i)$ instead of $\mathcal{O}(\log n)$.

Skip Lists

Promote **hot elements** to higher levels.

Search cost $\mathcal{O}(H(p))$ instead of $\mathcal{O}(\log n)$.

Summary

The Big Picture

Real-world query distributions are **skewed** (Zipfian). Classical data structures ignore this.

Learning-augmented structures exploit predictions to speed up hot queries while remaining **robust** when predictions fail.

Treaps

Replace random priorities with **predicted ranks**.

Hot element of rank i: depth $\mathcal{O}(\log i)$ instead of $\mathcal{O}(\log n)$.

Skip Lists

Promote **hot elements** to higher levels.

Search cost $\mathcal{O}(H(p))$ instead of $\mathcal{O}(\log n)$.

KD-Trees

Split by **query mass**, not by point count.

Search cost $\mathcal{O}(H(p))$ in hot regions.

Thank you!

Questions?

Appendix: Complete Proofs

Treap – Theorem 3.2: Proof (1/2)

Theorem

The expected depth of e_i in a learned treap is $\mathcal{O}(\log i)$ ***with high probability****.*

Proof idea. Searching for e_i is equivalent to inserting a random element $x \in [i]$ into a sorted array of $[i] \setminus \{x\}$ via **randomized binary search** (QuickSort-style analysis).

Let X_k = size of the sub-array at iteration k.

With probability $\frac{1}{2}$ the pivot lands in $\left[\frac{1}{4}X_k, \frac{3}{4}X_k\right]$, forcing $X_{k+1} \leq \frac{3}{4}X_k$; otherwise $X_{k+1} \leq X_k$. Hence:

$$\mathbb{E}[X_k] \;\leq\; \tfrac{1}{2}\,X_{k-1} + \tfrac{3}{8}\,X_{k-1} \;=\; \tfrac{7}{8}\,X_{k-1} \;\leq\; \left(\tfrac{7}{8}\right)^k X_0 \;\leq\; \left(\tfrac{7}{8}\right)^k i.$$

Treap – Theorem 3.2: Proof (2/2)

By **Markov's inequality** ($\Pr\{X_k \geq 1\} \leq \mathbb{E}[X_k]$):

$$\Pr\{\operatorname{depth}(e_i) > k\} = \Pr\{X_k \geq 1\} \leq \left(\frac{7}{8}\right)^k i.$$

Set $k = c \log_{8/7} i$ for a constant $c \geq 2$:

$$\Pr\{\operatorname{depth}(e_i) > k\} \leq \frac{i}{i^c} = \frac{1}{i^{c-1}} \leq \frac{1}{i}.$$

Therefore $\operatorname{depth}(e_i) = \mathcal{O}(\log_{8/7} i) = \mathcal{O}(\log i)$ with high probability. □

Skip Lists – Height Lemma: Proof (1/2)

Theorem (Height Bound)

With probability ≥ 0.99*, the skip list has at most* $10 + \log n$ *levels.*

Setup. Let $n_\ell = \#$ elements deterministically promoted to *exactly* level ℓ (i.e. $\hat{p}_i \in \left[\frac{2^{\ell-1}}{n}, \frac{2^\ell}{n}\right)$). Beyond level ℓ, each element advances with independent coin flips ($p = \frac{1}{2}$).

Step 1: per-element probability. An element at deterministic level ℓ must win $10 + \log n - \ell$ consecutive flips to reach level $10 + \log n$:

$$\Pr[\text{element reaches level } 10+\log n] \;\leq\; \left(\tfrac{1}{2}\right)^{10+\log n-\ell} = \frac{2^\ell}{2^{10}\, n} = \frac{2^\ell}{1024\, n}.$$

Step 2: union bound over all levels. Since $\hat{p}_i \leq 1$, deterministic promotion stops at $\ell \leq 2 + \log n$.

$$\Pr[\exists \text{ element at level } 10+\log n] \;\leq\; \sum_{\ell=0}^{2+\log n} n_\ell \cdot \frac{2^\ell}{1024\, n} = \frac{1}{1024\, n} \sum_{\ell=0}^{2+\log n} n_\ell\, 2^\ell.$$

Skip Lists – Height Lemma: Proof (2/2)

Step 3: bound the sum. For each element at level ℓ: $\hat{p}_i \geq \frac{2^{\ell-1}}{n}$, so $2^\ell \leq 2n\hat{p}_i$.

$$\sum_{\ell=0}^{2+\log n} n_\ell \, 2^\ell = \sum_i 2^{\ell(i)} \leq \sum_i 2n\hat{p}_i = 2n \underbrace{\sum_i \hat{p}_i}_{=1} = 2n.$$

Conclusion. Plugging back into the union bound:

$$\Pr[\exists \text{ element at level } 10+\log n] \;\leq\; \frac{1}{1024\, n} \cdot 2n = \frac{1}{512} < 0.01.$$

Hence the number of levels is $\leq 10 + \log n$ with probability ≥ 0.99. □