Competitive Data-Structure Dynamization

— SODA 2021 —

(a 25-minute talk summarizing the conference paper)

Claire Mathieu
CNRS, Paris

Rajmohan Rajaraman
Northeastern University

Neal E. Young
University of California Riverside
Northeastern University

Arman Yousefi
Google

— research funded by NSF and Google

Welcome to the 25-minute talk for the SODA 2021 paper, Competitive Data-Structure Dynamization, by Claire Mathieu, Rajmohan Rajaraman, Neal Young, and Arman Yousefi. I'm Neal Young.

Competitive Data-Structure Dynamization

— SODA 2021 —

(a 25-minute talk summarizing the conference paper)


Claire Mathieu
CNRS, Paris

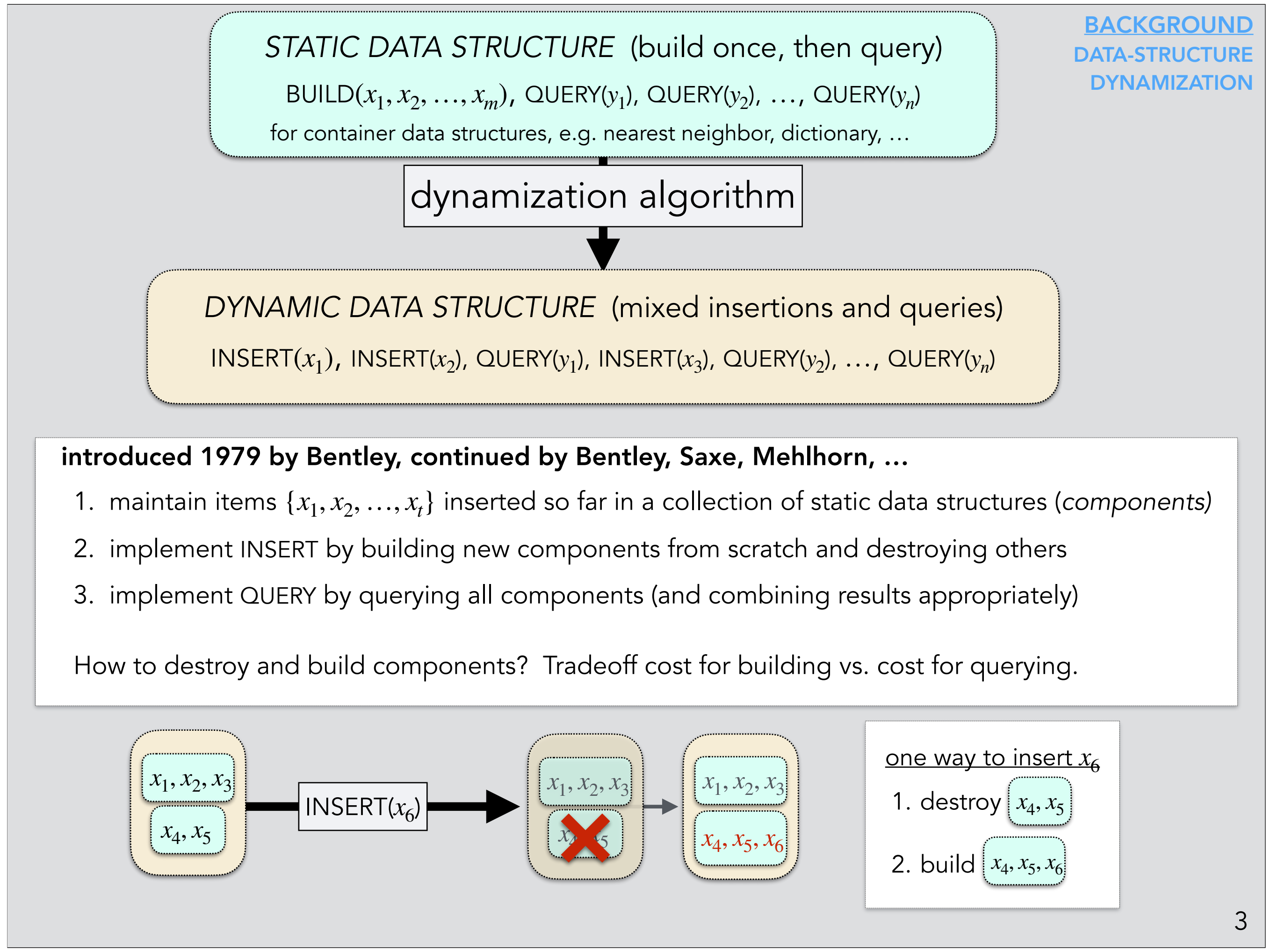

Rajmohan Rajaraman
Northeastern University


Neal E. Young
University of California Riverside
Northeastern University


Arman Yousefi
Google

— research funded by NSF and Google

The paper studies compaction policies for LSM (log-structured merge) systems through the lens of competitive analysis. The talk has two goals: to convey the mathematical flavor of the problem and to give at least some sense for the practical context in which it arises. We'll start with two slides reviewing some background, formally define the two problems that we study, state our main results, give a brief taste of how we prove those results, and conclude with a brief addendum with one more piece of context.



Data-structure dynamization was introduced around 1979 by Jon Bentley and others,
as a generic approach for making static data structures dynamic.
A static data structure is built once to hold a fixed set of items, and then queried any number of times.
In contrast, a dynamic data structure supports mixed insertions and queries.
In some settings, static data structures are easier to design.
Bentley observed that any static data structure can be used to build a dynamic variant as follows.

Maintain the items inserted so far in a collection of immutable static data structures, called components.
Implement each insertion by building new components (at least one of which includes the inserted item), and possibly destroying others.
(This is illustrated in the example at the bottom of the slide.)
Implement each query by querying all current components and combining the results appropriately for the underlying data type.

The key design question is how to manage the components, specifically, how to keep the number of components from growing too large without spending too much time building new components. This is the problem that a dynamization algorithm must solve.

STATIC DATA STRUCTURE (build once, then query)

BUILD($x_1, x_2, \ldots, x_m$), QUERY(y_1), QUERY(y_2), …, QUERY(y_n)

for container data structures, e.g. nearest neighbor, dictionary, …

dynamization algorithm

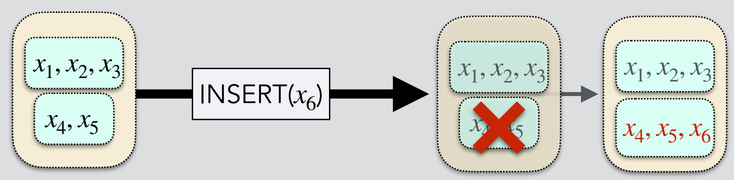
DYNAMIC DATA STRUCTURE (mixed insertions and queries)

INSERT(x_1), INSERT(x_2), QUERY(y_1), INSERT(x_3), QUERY(y_2), …, QUERY(y_n)

introduced 1979 by Bentley, continued by Bentley, Saxe, Mehlhorn, …

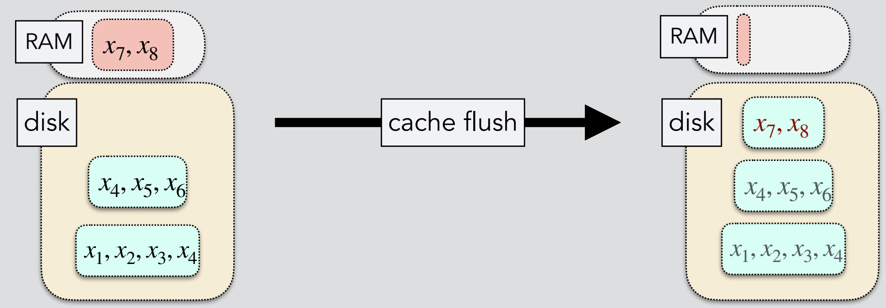
1. maintain items $\{x_1, x_2, \ldots, x_t\}$ inserted so far in a collection of static data structures (*components)*
2. implement INSERT by building new components from scratch and destroying others
3. implement QUERY by querying all components (and combining results appropriately)

How to destroy and build components? Tradeoff cost for building vs. cost for querying.



one way to insert x_6

1. destroy x_4, x_5
2. build x_4, x_5, x_6



LSM = "log-structured merge" [O'Neil et al 1996, and others] for external-memory dictionaries

1. INSERTs are cached in RAM
2. periodically flush RAM cache to disk in a single batch
3. maintain on-disk items in *immutable* sorted files (called *components*)
4. each QUERY checks the cache, then (if necessary) all on-disk components
5. the components (on disk) are managed using a data-structure-dynamization algorithm

note:

a. each INSERT is to RAM, requires no disk access

b. component builds use high-throughput sequential disk access, not random access

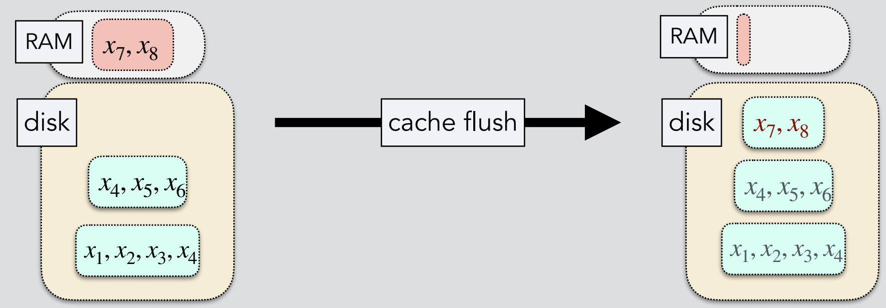
3. checking one given component (during a query) takes just one disk access (using in-RAM index)

4. academic work assumed uniform batch sizes and uniform INSERT/QUERY rates,
but these assumptions don't hold in industrial systems, e.g. Google Bigtable

Dynamization is also employed the context of external-memory dictionary data structures. External-memory data structures are used when the data to be stored is much larger than can fit in RAM (fast memory), so that most of the data must be held on disk. Around 1996 O'Neil et al proposed the so-called "log-structured merge" approach. Each inserted item is simply cached in RAM, without accessing the disk at all. Every once in a while, the items cached in RAM are flushed in a batch write to the disk, as a single immutable file. These on-disk files, called components, are managed using a dynamization algorithm. That is, the dynamization algorithm (in this context called a "compaction" or "merge" policy) periodically destroys some components and builds new ones from scratch. Crucially, components are only built and destroyed, never altered, and builds use sequential, not random, disk access.

Each query is implemented by checking the cache, and if the desired item is not found, querying each on-disk component. Note that (for reasons explained in the final slide of this talk) checking a given component for a given item requires just one random disk access.

For insert-heavy workloads (or when queries exhibit enough locality of reference to make them amenable to caching), LSM systems substantially outperform classical data structures such as B-trees. LSM systems are used by many big-data companies, such as Google, for data-storage backends. Most academic work on LSM systems has assumed batch sizes (as if the cache was flushed only when full) and uniform insert/query rates. But these assumptions don't hold in production systems.



LSM = "log-structured merge" [O'Neil et al 1996, and others] for external-memory dictionaries

1. INSERTs are cached in RAM
2. periodically flush RAM cache to disk in a single batch
3. maintain on-disk items in *immutable* sorted files (called *components*)
4. each QUERY checks the cache, then (if necessary) all on-disk components
5. the components (on disk) are managed using a data-structure-dynamization algorithm

note:

a. each INSERT is to RAM, requires no disk access

b. component builds use high-throughput sequential disk access, not random access

3. checking one given component (during a query) takes just one disk access (using in-RAM index)

4. academic work assumed uniform batch sizes and uniform INSERT/QUERY rates,
   but these assumptions don't hold in industrial systems, e.g. Google Bigtable

Next we formally define two optimization problems that model the task that a dynamization algorithm must perform. Each problem models a particular tradeoff between query cost and build cost. We study these problems through the lens of competitive analysis.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) \;+\; \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S}\mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost

We call the first problem min-sum dynamization. The input is a sequence of n batches, given one at a time. In response to each batch, the algorithm must produce a set cover such that the union of the sets in the cover is the set of all items inserted so far. At each time t, the algorithm incurs two costs: a query cost equal to the size of the current cover, and a build cost, which is best understood as follows: the current cover is obtained from the previous cover by adding some new sets and destroying others. For each new set, the algorithm pays a build cost equal to the weight of the items in the set. The goal is to minimize the sum of all the build costs and query costs.

Here's an example….

[BUILD IN]

At time 1, the input batch is a set containing two elements A and B.

[BUILD IN]

THe algorithm might respond with a cover containing two sets, one containing A, and the other containing B.

If it does this, the query cost will be 2, because there are two sets in the cover, and the build cost will be the weight of A plus the weight of B. Now you might be thinking that it would have been better to use just one component containing both A and B, incurring query cost 1 and the same build cost, and you would be right.

[BUILD IN]

At time 2, let's say the input batch contains just a single new item C,

[BUILD IN]

and the algorithm responds by destroying the component containing A, and building a new component containing A and C.

[BUILD IN]

The query cost is 2, and the build cost is the wt of A plus the wt of C, because elements A and C are the ones in the new component.

[BUILD IN]

At time 3, let's say the input batch contains two new items D and E.

[BUILD IN]

The algorithm responds by, say, adding the batch as a single new component.
If it does this, the query cost will be 3, and the build cost will be wt(d) + wt(e).

[BUILD IN]

At time 4, let's say the input batch contains just item {F}, and the algorithm
responds with a cover containing just one set, containing all items.
This incurs query cost 1, and build cost equal to the sum of the weights of all of the items.

[BUILD IN]

This gives total cost as shown at the bottom of the slide, and of course the goal is to minimize
this total cost.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1				

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$			

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$		

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2				

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$			

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$		

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3				

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$			

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$	$\{b\}, \{a, c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$	$\{b\}, \{a, c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$
4	$\{f\}$	$\{a, b, c, d, e, f\}$	1	$\mathrm{wt}(a) + \mathrm{wt}(b) + \mathrm{wt}(c) + \mathrm{wt}(d) + \mathrm{wt}(e) + \mathrm{wt}(f)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$	$\{b\}, \{a, c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$
4	$\{f\}$	$\{a, b, c, d, e, f\}$	1	$\mathrm{wt}(a) + \mathrm{wt}(b) + \mathrm{wt}(c) + \mathrm{wt}(d) + \mathrm{wt}(e) + \mathrm{wt}(f)$
		total cost:	8 +	$3\,\mathrm{wt}(a)+2\mathrm{wt}(b)+2\mathrm{wt}(c)+2\mathrm{wt}(d)+2\mathrm{wt}(e)+\mathrm{wt}(f)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^t I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n^2)$
pays build cost $\Theta(n)$

TRIVIAL ALGORITHM 1: *minimize build cost*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{a, b\}, \{c\}$	2	$\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b\}, \{c\}, \{d, e\}$	3	$\mathrm{wt}(\mathrm{d}) + \mathrm{wt}(e)$
4	$\{f\}$	$\{a, b\}, \{c\}, \{d, e\}, \{f\}$	4	$\mathrm{wt}(f)$

total cost: $10 + \mathrm{wt}(a) + \mathrm{wt}(b) + \mathrm{wt}(c) + \mathrm{wt}(d) + \mathrm{wt}(e) + \mathrm{wt}(f)$

One trivial algorithm, which minimizes the build cost, is to respond to each batch by inserting the batch as a new set. Then each item is involved in just one build, so the build cost is as small as possible. But the query cost at time t is t, so the total query cost is quadratic in n. For uniform inputs, the total cost is quadratic in n.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n^2)$
pays build cost $\Theta(n)$

TRIVIAL ALGORITHM 1: *minimize build cost*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{a, b\}, \{c\}$	2	$\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b\}, \{c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$
4	$\{f\}$	$\{a, b\}, \{c\}, \{d, e\}, \{f\}$	4	$\mathrm{wt}(f)$

total cost: $10 + \mathrm{wt}(a) + \mathrm{wt}(b) + \mathrm{wt}(c) + \mathrm{wt}(d) + \mathrm{wt}(e) + \mathrm{wt}(f)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n)$
pays build cost $\Theta(n^2)$

TRIVIAL ALGORITHM 2: *minimize query cost*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)$
2	$\{c\}$	$\{a, b, c\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b, c, d, e\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)$
4	$\{f\}$	$\{a, b, c, d, e, f\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)+\mathrm{wt}(f)$

total cost: 4 + 4wt(a)+4wt(b)+3wt(c)+2wt(d)+2wt(e)+wt(f)

Another trivial algorithm, which minimizes the query cost, is to respond to each batch with a cover that contains just one set, containing all the items inserted so far. Then at each time the query cost is 1, so the total query cost is n. But the build cost is large. For uniform inputs, the build cost is quadratic in n, so the total cost is quadratic in n.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n)$
pays build cost $\Theta(n^2)$

TRIVIAL ALGORITHM 2: *minimize query cost*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)$
2	$\{c\}$	$\{a, b, c\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b, c, d, e\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)$
4	$\{f\}$	$\{a, b, c, d, e, f\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)+\mathrm{wt}(f)$

total cost: 4 + 4wt(a)+4wt(b)+3wt(c)+2wt(d)+2wt(e)+wt(f)

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

BINARY TRANSFORM [Bentley, 1979]

time	input batch	cover
1	I_1	I_1
2	I_2	$I_1 \cup I_2$
3	I_3	$I_1 \cup I_2$, I_3
4	I_4	$I_1 \cup I_2 \cup I_3 \cup I_4$
5	I_5	$I_1 \cup I_2 \cup I_3 \cup I_4$, I_5
6	I_6	$I_1 \cup I_2 \cup I_3 \cup I_4$, $I_5 \cup I_6$
7	I_7	$I_1 \cup I_2 \cup I_3 \cup I_4$, $I_5 \cup I_6$, I_7
8	I_8	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$
9	I_9	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$, I_9
10	I_{10}	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$, $I_9 \cup I_{10}$
⋮	⋮	⋮

At each time t, there is one set for each 1 in the binary representation of t. Each step emulates an increment in binary.

⟶ on uniform input:
pays build cost $\Theta(n \log n)$
pays query cost $\Theta(n \log n)$

total cost $\Theta(n \log n)$

optimal for uniform input

The Binary Transform is a dynamization algorithm, designed by Bentley in 1979 for uniform inputs. On uniform inputs, it incurs cost order n log n, which is optimal for uniform inputs. It does this by maintaining at most log n sets at all times, and ensuring that each item is involved in at most log n builds. The basic idea is that, at each time t, the cover has a set for each 1 in the binary representation of t, and each insertion mimics a binary increment. This is the same idea underlying the well-known binomial-heap data structure.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^t I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

BINARY TRANSFORM **[Bentley, 1979]**

time	input batch	cover
1	I_1	I_1
2	I_2	$I_1 \cup I_2$
3	I_3	$I_1 \cup I_2$, I_3
4	I_4	$I_1 \cup I_2 \cup I_3 \cup I_4$
5	I_5	$I_1 \cup I_2 \cup I_3 \cup I_4$, I_5
6	I_6	$I_1 \cup I_2 \cup I_3 \cup I_4$, $I_5 \cup I_6$
7	I_7	$I_1 \cup I_2 \cup I_3 \cup I_4$, $I_5 \cup I_6$, I_7
8	I_8	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$
9	I_9	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$, I_9
10	I_{10}	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$, $I_9 \cup I_{10}$
⋮	⋮	⋮

At each time t, there is one set for each 1 in the binary representation of t. Each step emulates an increment in binary.

⟶ on uniform input:

pays build cost $\Theta(n \log n)$

pays query cost $\Theta(n \log n)$

total cost $\Theta(n \log n)$

optimal for uniform input

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

same as before

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S)$

K-BINOMIAL TRANSFORM **[Bentley & Saxe, 1980]**

At each time t:

1. Let $i_1, \ldots, i_k$ be the k integers such that $0 \le i_1 < i_2 < i_3 < \cdots < i_k$ and $\sum_{j=1}^{k} \binom{i_j}{j} = t$.
2. Use the cover consisting of k sets, where
   - the first set contains the first $\binom{i_k}{k}$ batches,
   - the second set contains the next $\binom{i_{k-1}}{k-1}$ batches,
   - and so on.

On uniform input,
pays build cost $\Theta(kn^{1+1/k})$.
Optimal for uniform input.

We call the second problem that we study k-component dynamization. The input and output are the same as for min-sum dynamization, except that each cover is constrained to have size at most k, so that no query incurs cost more than k. The objective is to minimize the total build cost.

The k-binomial transform, a dynamization policy designed by Bentley and Saxe in 1980 for uniform inputs, meets the query-cost constraint, and guarantees that no item is involved in more than O(k n^{1/k}) builds. In this way, for uniform inputs, it incurs build cost theta(k n^{1+1/k}), which is optimal for uniform inputs.

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

same as before

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S)$

***K-BINOMIAL TRANSFORM* [Bentley & Saxe, 1980]**

At each time t:

1. Let $i_1, \ldots, i_k$ be the k integers such that $0 \le i_1 < i_2 < i_3 < \cdots < i_k$ and $\sum_{j=1}^{k} \binom{i_j}{j} = t$.
2. Use the cover consisting of k sets, where
   - the first set contains the first $\binom{i_k}{k}$ batches,
   - the second set contains the next $\binom{i_{k-1}}{k-1}$ batches,
   - and so on.

On uniform input,
pays build cost $\Theta(kn^{1+1/k})$.
Optimal for uniform input.

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

Prior results are for uniform input, but in LSM systems inputs are *online* and *non-uniform*.

Non-uniform inputs can be *easier* (less costly) than uniform inputs.

COMPETITIVE ANALYSIS

DEFN: An algorithm is *online* if its cover at each time t is independent of $I_{t+1}, I_{t+2}, \ldots, I_n$.

DEFN: An algorithm is *c-competitive* if, for every input, its solution costs at most c times the optimum for that input. The *competitive ratio* of the algorithm is the minimum such c.

standard

QUESTION: What competitive ratios can online algorithms achieve?

As previously mentioned, most previous academic work on dynamization algorithms (and compaction policies in LSM systems) has assumed uniform inputs.. that is, uniform batch sizes (as if the cache is flushed only when full), and uniform insert/query rates. But these assumptions don't hold for production systems. Non-uniform inputs can be _easier_ (that is, less costly),
Compaction policies in current LSM systems such as Bigtable do adapt to non-uniformity, but in a somewhat adhoc way. Our goal is to explicitly design policies through the lens of competitive analysis, so that the policies adapt in a provably robust way. From a theoretical point of view, our goal is to design optimally competitive online algorithms.

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

Prior results are for uniform input, but in LSM systems inputs are *online* and *non-uniform*.

Non-uniform inputs can be *easier* (less costly) than uniform inputs.

COMPETITIVE ANALYSIS

DEFN: An algorithm is *online* if its cover at each time t is independent of $I_{t+1}, I_{t+2}, \ldots, I_n$.

DEFN: An algorithm is c-*competitive* if, for every input, its solution costs at most c times the optimum for that input. The *competitive ratio* of the algorithm is the minimum such c.

standard

QUESTION: What competitive ratios can online algorithms achieve?

Next we state our results and try to give a taste of the underlying mathematics.

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.

For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.
- **THMS 3.1—3.4.** *For k-Component Dynamization, the optimal competitive ratio for deterministic online algorithms is k.*
- **EXTENSIONS**: the results on *k*-Component Dynamization extend to allow *lazy deletions, updates, item expiration* as they occur in big-data storage systems (see the paper).

….

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.

For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.
- **THMS 3.1—3.4.** *For k-Component Dynamization, the optimal competitive ratio for deterministic online algorithms is k.*
- **EXTENSIONS**: the results on *k*-Component Dynamization extend to allow *lazy deletions, updates, item expiration* as they occur in big-data storage systems (see the paper).

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.

For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.
  OPEN: *constant* competitive ratio?
- **THMS 3.1—3.4.** *For k-Component Dynamization, the optimal competitive ratio for deterministic online algorithms is k.*
  OPEN: randomized algorithms?
- **EXTENSIONS**: the results on *k*-Component Dynamization extend to allow *lazy deletions, updates, item expiration* as they occur in big-data storage systems (see the paper).
  OPEN: same extensions for Min-Sum Dynamization?

The paper suggests many more open problems.

…

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.

For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.
  OPEN: *constant* competitive ratio?
- **THMS 3.1—3.4.** *For k-Component Dynamization, the optimal competitive ratio for deterministic online algorithms is k.*
  OPEN: randomized algorithms?
- **EXTENSIONS**: the results on *k*-Component Dynamization extend to allow *lazy deletions, updates, item expiration* as they occur in big-data storage systems (see the paper).
  OPEN: same extensions for Min-Sum Dynamization?

The paper suggests many more open problems.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

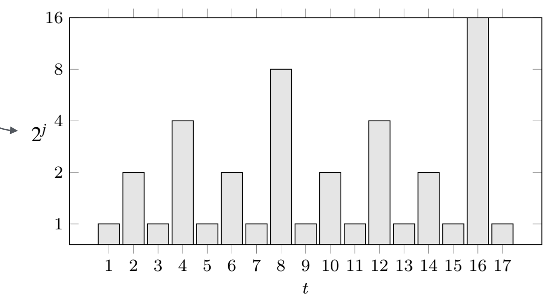
MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

at each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add current batch I_t to the current cover as a single new set
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \leq 2^j$ into one new set

Roughly, every 2^j time steps it merges together all sets of weight 2^j or less.



… note that a set of a given weight W will last at most about 2W time units before being merged with other sets… this ensures that the set's contribution to the query cost is bounded by twice its contribution to the build cost.
… on uniform inputs, this algorithm gives the same (optimal) solution as the binary transform.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

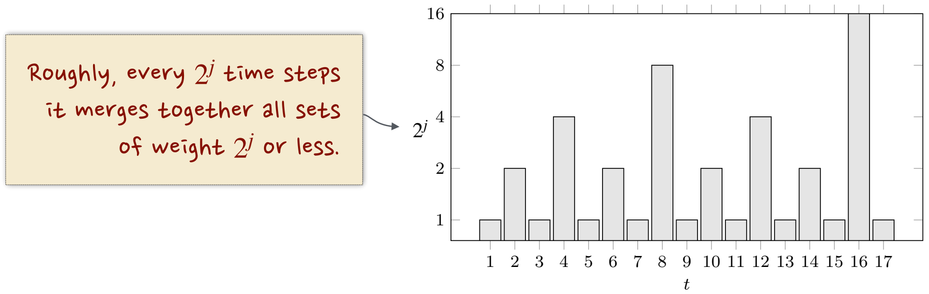
OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

at each time $t \leftarrow 1,2,\ldots,n$ do:

1. add current batch I_t to the current cover as a single new set
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into one new set



for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\le 2^j$ into a single set

EXAMPLE EXECUTION:

$$\underbrace{2^9 \;\cdots\; 2^9}_{2^6 \text{ times}} \quad \underbrace{2^{10} \;\cdots\; 2^{10}}_{2^5} \quad \underbrace{2^{11} \;\cdots\; 2^{11}}_{2^4} \quad \underbrace{2^{12} \;\cdots\; 2^{12}}_{2^3} \quad \underbrace{2^{13} \;\cdots\; 2^{13}}_{2^3} \quad 2^{14} \quad 2^{14} \quad 2^{14} \quad 2^{14}$$

INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

[describe example]
… note that this input is particularly simple in that merges before the last non-empty insertion. in the general case, of course, merges and insertions will be intermixed.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \leq 2^j$ into a single set

EXAMPLE EXECUTION:

$$\underbrace{2^9 \quad \cdots \quad 2^9}_{2^6 \text{ times}} \quad \underbrace{2^{10} \quad \cdots \quad 2^{10}}_{2^5} \quad \underbrace{2^{11} \quad \cdots \quad 2^{11}}_{2^4} \quad \underbrace{2^{12} \quad \cdots \quad 2^{12}}_{2^3} \quad \underbrace{2^{13} \quad \cdots \quad 2^{13}}_{2^3} \quad 2^{14} \quad 2^{14} \quad 2^{14} \quad 2^{14}$$

INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

EXAMPLE EXECUTION:

the cover at time
$t = 2^9 - 1$

$$\underbrace{2^9 \ \cdots \ 2^9}_{2^6 \text{ times}} \ \underbrace{2^{10} \ \cdots \ 2^{10}}_{2^5} \ \underbrace{2^{11} \ \cdots \ 2^{11}}_{2^4} \ \underbrace{2^{12} \ \cdots \ 2^{12}}_{2^3} \ \underbrace{2^{13} \ \cdots \ 2^{13}}_{2^3} \ 2^{14} \ 2^{14} \ 2^{14} \ 2^{14}$$

INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

the lightest batch has weight 2^9, so the algorithm does no merges until time t=2^9. before that, it just creates one set for each batch. so, at time 2^9-1, the cover consists of one set for each inserted batch, as shown.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

EXAMPLE EXECUTION:

the cover at time
$t = 2^9 - 1$

$$\underbrace{2^9 \ \cdots \ 2^9}_{2^6 \text{ times}} \ \underbrace{2^{10} \ \cdots \ 2^{10}}_{2^5} \ \underbrace{2^{11} \ \cdots \ 2^{11}}_{2^4} \ \underbrace{2^{12} \ \cdots \ 2^{12}}_{2^3} \ \underbrace{2^{13} \ \cdots \ 2^{13}}_{2^3} \ 2^{14} \ 2^{14} \ 2^{14} \ 2^{14}$$

INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

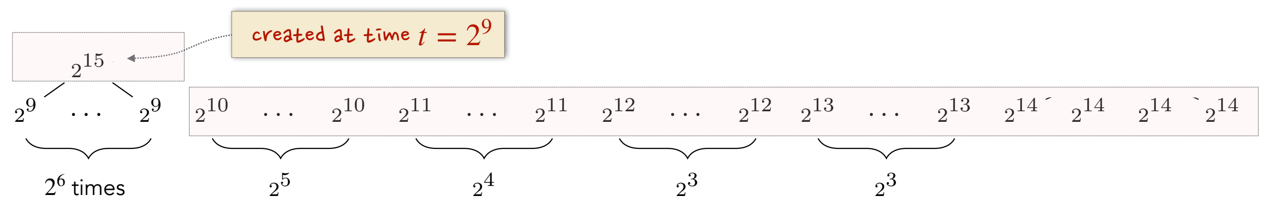
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

at time 2^9…

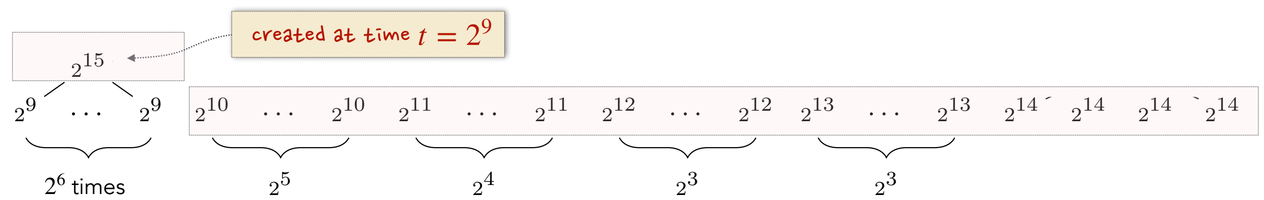
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

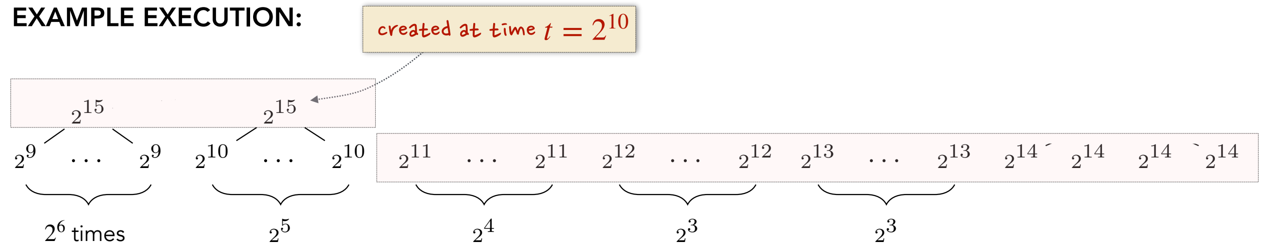
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

at time 2^10..

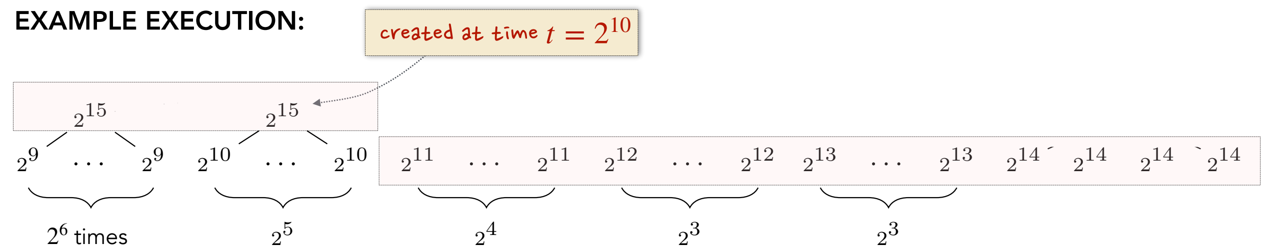
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

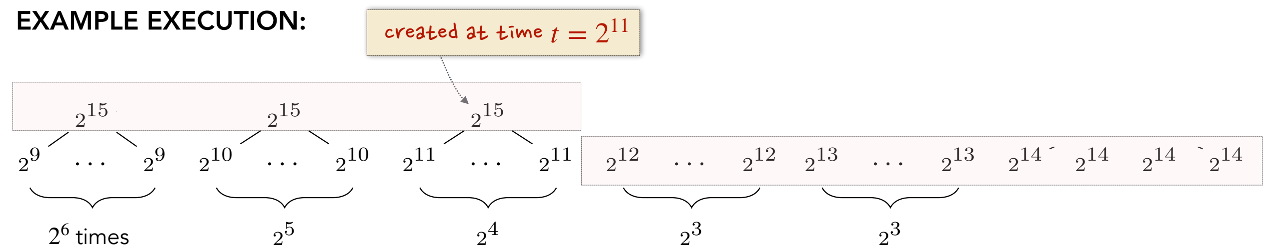
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

at time 2^11

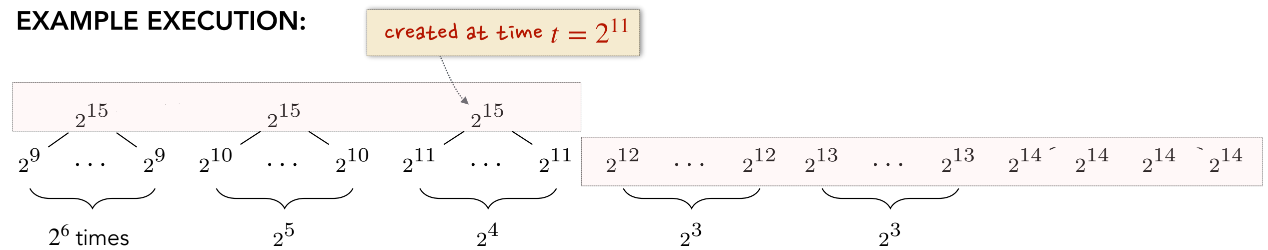
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

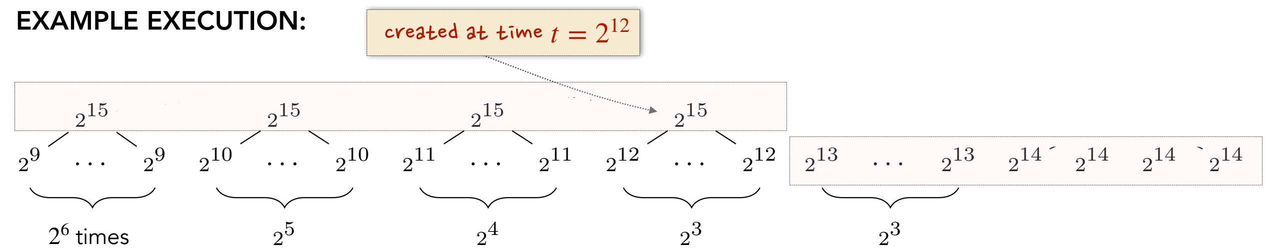
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

at time 2^12...

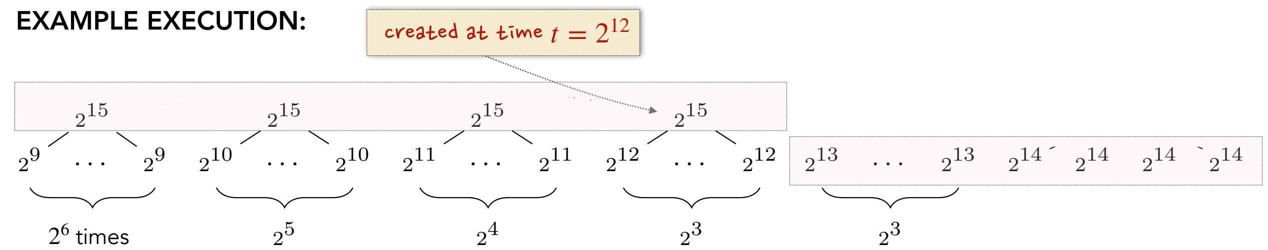
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

for Problem 1: MIN-SUM DYNAMIZATION

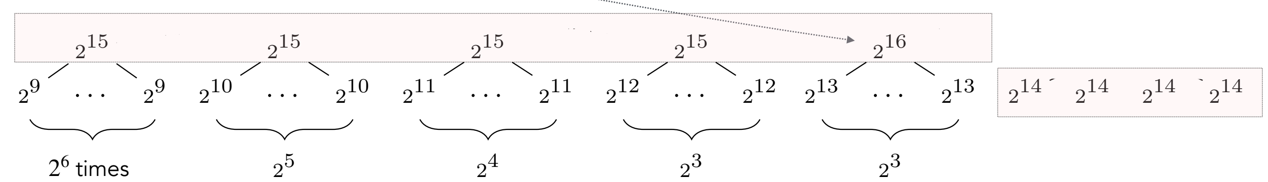
THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:

created at time $t = 2^{13}$



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

at time 2^13,

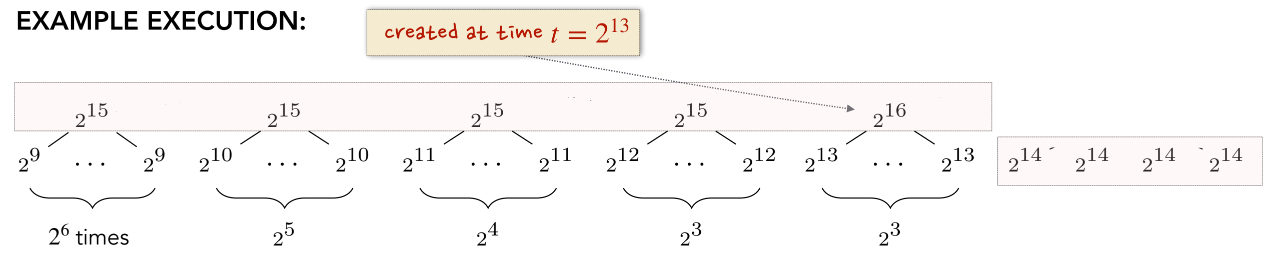
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10},2^{11},2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

for Problem 1: MIN-SUM DYNAMIZATION

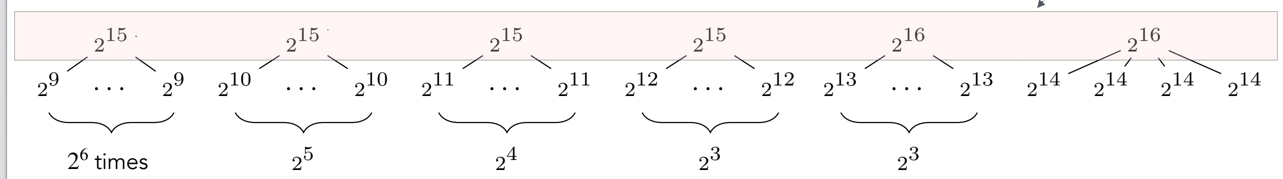
THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

the cover at time $t = 2^{14}$

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

at time 2^14.. at this point the cover consists of the nodes of weight 2^15 and 2^16, highlighted in pink in the slide.

for Problem 1: MIN-SUM DYNAMIZATION

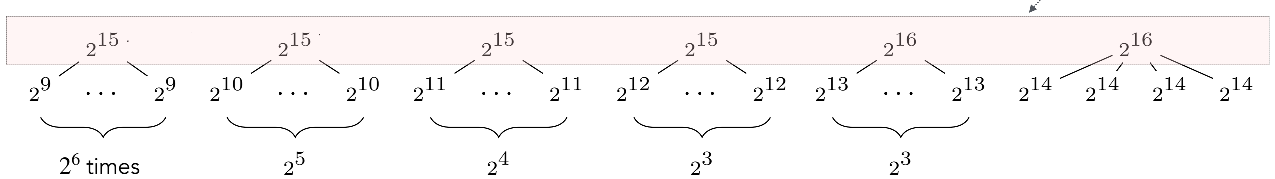
THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

the cover at time $t = 2^{14}$

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

for Problem 1: MIN-SUM DYNAMIZATION

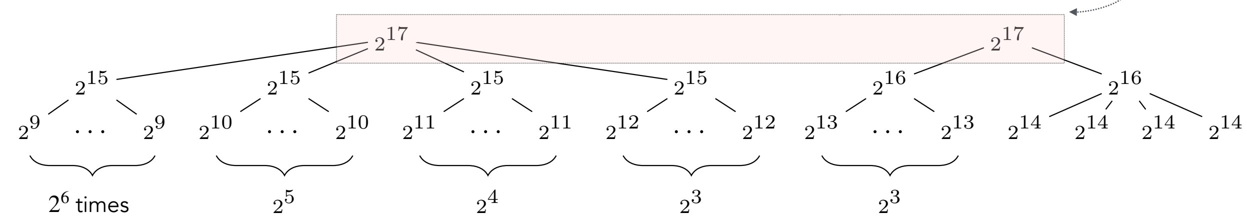
THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

the cover at time $t = 2^{16}$

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

at times 2^15 and then 2^16 more merges occur, leaving two sets each of weight 2^17.

for Problem 1: MIN-SUM DYNAMIZATION

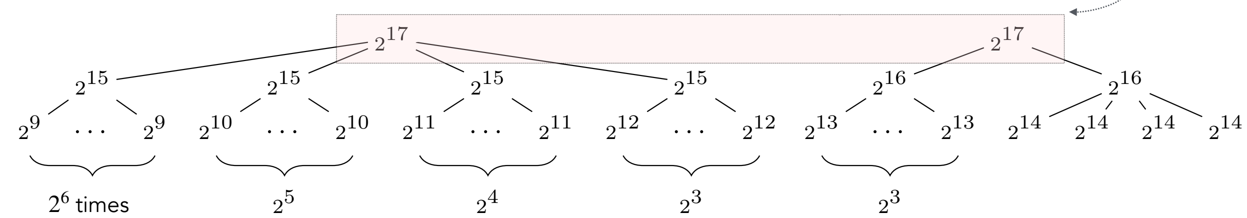
THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

the cover at time $t = 2^{16}$

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

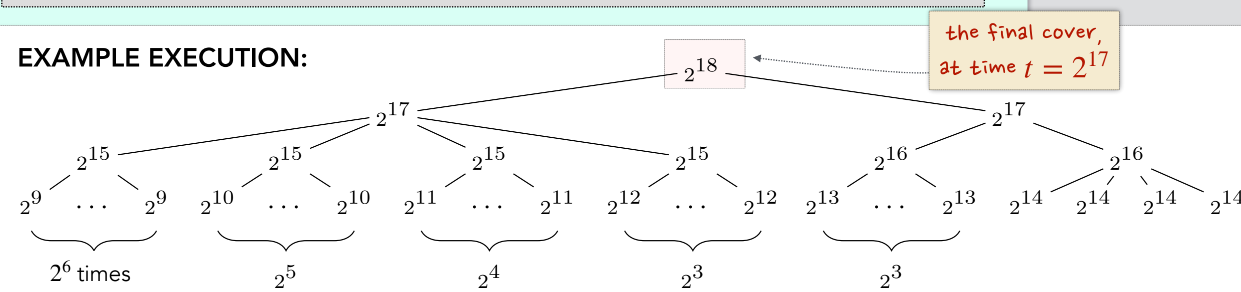
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

Finally at time $t = 2^{17}$, the two remaining sets, of weight 2^{17} merge into one set of weight 2^{18}.

finally at time 2^17 a single set remains, of total weight 2^18.
the total build cost is the sum, over the leaves of the merge tree (as shown in the slide), of the weight of the leaf times the depth of the leaf. in this case all leaves are at depth 4, so the total build cost is 2^18 times 4. one can show that the query cost is about the same.

a cheaper solution would have been to merge all batches in to one set at time 2^14.

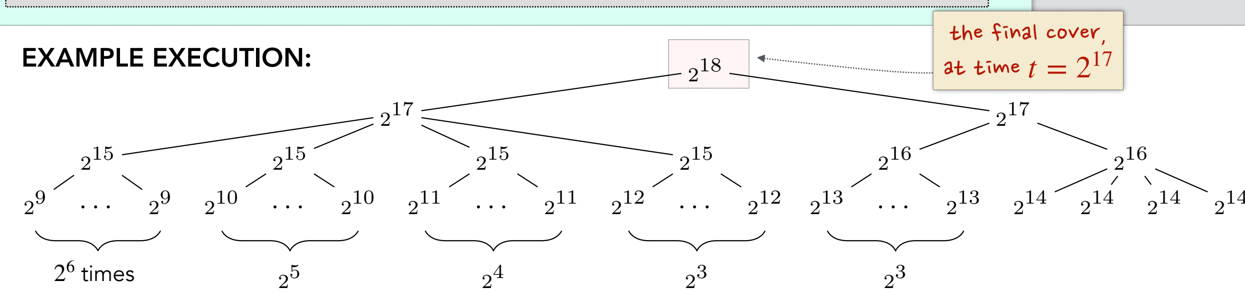
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

EXAMPLE EXECUTION:



INPUT: Insert batches above, in left-to-right order, then repeatedly insert empty batches.

At time $t = 2^9$, the 2^6 leaves of weight 2^9 merge into one set of weight 2^{15}.

Likewise at each time $t \in \{2^{10}, 2^{11}, 2^{12}\}$, the leaves of weight t merge into one set of weight 2^{15}.

At times $t \in \{2^{13}, 2^{14}\}$, the leaves of weight t merge into one set of weight 2^{16}, and so on.

Finally at time $t = 2^{17}$, the two remaining sets, of weight 2^{17} merge into one set of weight 2^{18}.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:

(generalizes example from previous slide)

1. define the desired merge tree greedily, in reverse (breadth-first from the root): start by creating the root, with weight 2^N (an arbitrarily large power of 2)
2. repeat: choose the next node to split, and the next "unused" threshold 2^j, then split the leaf into children all of weight 2^j
3. show that the number of nodes at depth d is at most $\left.4^{4^{\cdot^{\cdot^{4}}}}\right\}d$ (tower of height d)

we show this algorithm has competitive ratio theta(log*n). first we show that the algorithm's ratio is at least log^* n. this ratio is achieved on a family of inputs that generalize the example just shown. the general method for generating the input is as follows. first we define the desired merge tree, by starting at the root and working down the tree BFS order. at each step, to define the children of a given node, we "split" the node into equal-weight children, where the weight is the "next available" power of 2. a quick example will give the idea.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:

(generalizes example from previous slide)

1. define the desired merge tree greedily, in reverse (breadth-first from the root): start by creating the root, with weight 2^N (an arbitrarily large power of 2)
2. repeat: choose the next node to split, and the next "unused" threshold 2^j, then split the leaf into children all of weight 2^j
3. show that the number of nodes at depth d is at most $\left.4^{4^{\cdot^{\cdot^{4}}}}\right\}d$ (tower of height d)

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:

2^{18}

we start with the root, giving it weight equal to some large power of 2.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:

2^{18}

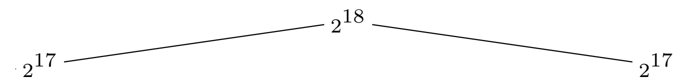
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



we split the root into children each having weight the next smaller power of two.

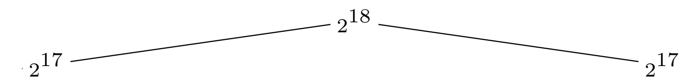
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



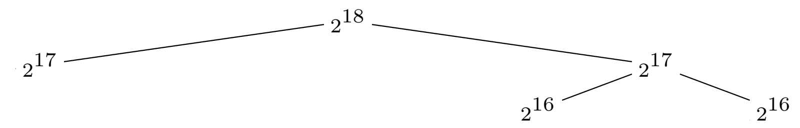
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



then we split the rightmost child into children having weight 2^16, the next smaller power of 2.

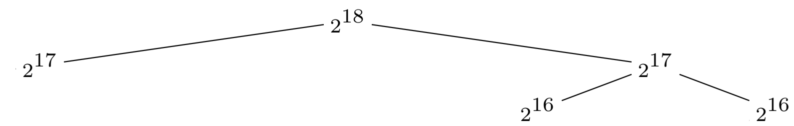
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



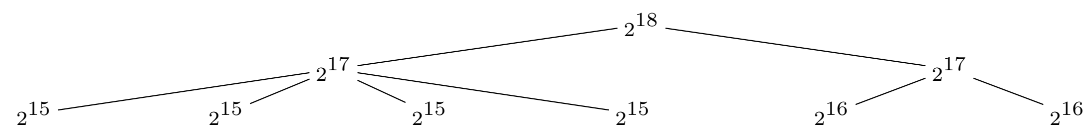
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$**:**



proceeding in breadth-first order, we split the next node of weight 2^17 into children of weight 2^15. we use 2^15 because it is the next "unused" power of two. because the children have weight 2^15, and their total weight must equal the parent's weight, there must be four children.

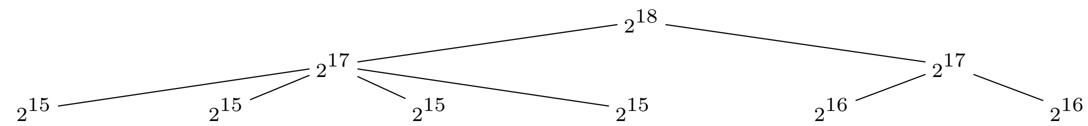
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



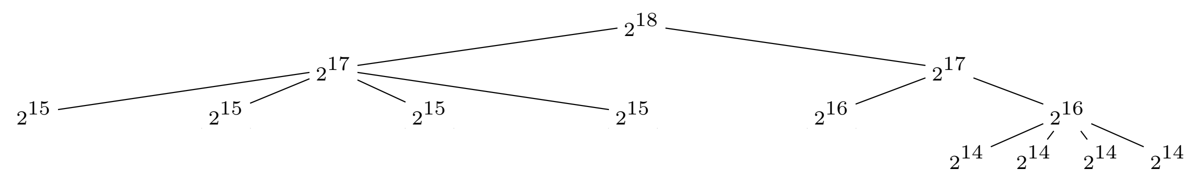
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



we continue in this way, node by node, as shown.

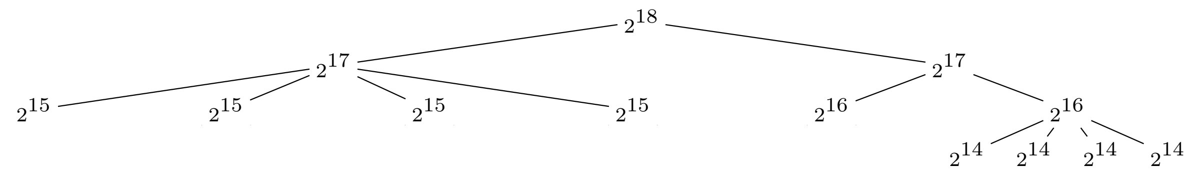
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



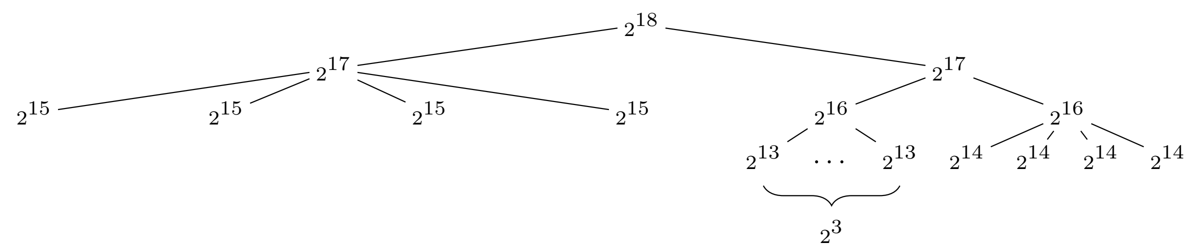
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$**:**



[no audio]

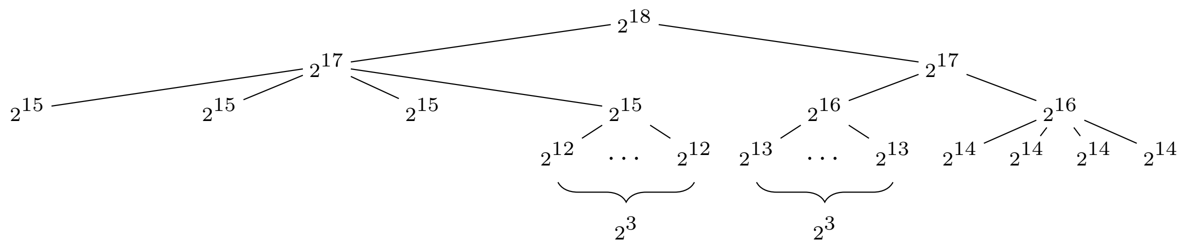
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:



[no audio]

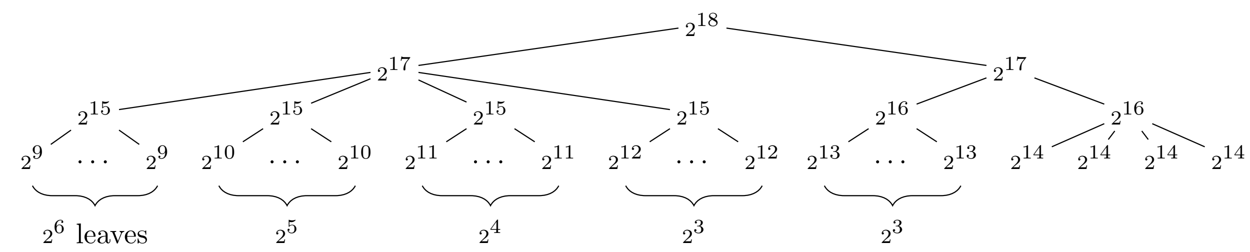
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$**:**



we stop when the leaf weights are sufficiently small. the leaves give us the input. running the algorithm on this input will create this merge tree, as desired. the build cost will be the tree depth times the total leaf weight. to complete the proof we show that the tree depth is log^* n, and that there is an optimal solution whose cost is proportional to the leaf weight.

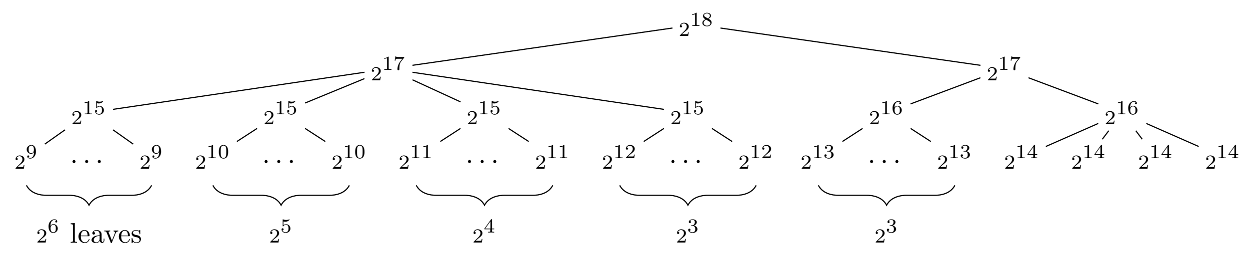
for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1, 2, \ldots, n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \le 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$**:**



for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt(S) $\leq 2^j$ into a single set

PROOF OUTLINE FOR LOWER BOUND $\Omega(\log^* n)$:

(generalizes example from previous slide)

1. build the desired merge tree greedily, in reverse (breadth-first from the root): start by creating the root, with weight 2^N (an arbitrarily large power of 2)
2. repeat: choose the next node to split, and the next "unused" threshold 2^j, then split the leaf into children all of weight 2^j
3. show that the number of nodes at depth d is at most $\left.4^{4^{\cdot^{\cdot^{\cdot^{4}}}}}\right\}d$ (tower of height d)

[no audio]

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio $\Theta(\log^* n)$.*

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into a single set

PROOF OUTLINE FOR UPPER BOUND $O(\log^* n)$:

1. charge algorithm's query cost to its build cost
2. charge the cost of building each set, via the items in it, to the OPT sets containing those items
3. show that each OPT set is charged $O(\log^* n)$ times its contribution to the OPT cost

Step 3 is the hard part. The intuition is that, in the "merge tree" formed by tracking how the algorithm merges the elements in the OPT set, each important merge must be associated with a unique threshold 2^j.
This forces the merge tree to have weighted average leaf depth $O(\log^* n)$.

Next we show that the competitive ratio is at most log* n on any input. We do this in three steps. First we observe that the algorithms' query cost is proportional to its build cost, so it suffices to bound the build cost. To do that, for each set that the algorithm builds, we charge the cost of building the set, via the set's items, to the OPT sets containing those items at that time. finally, we show that each set in the optimal solution is charged at most log* n times its contribution to the optimal cost. This is the hard part. Very roughly, we consider each OPT set. We show that the merge tree that the algorithm induces on the elements in the OPT set has weighted average depth log* n. Intuitively, the reason for this is that the nodes (merges) in the tree must have distinct powers 2^j associated with them. As in the lower-bound example, this forces the node degrees to increase exponentially level by level as we descend from the root in the merge tree, which allows us to show that the depth cannot be too large.

for Problem 1: MIN-SUM DYNAMIZATION

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

for each time $t \leftarrow 1,2,\ldots,n$ do:

1. add I_t to the current cover
2. let j be the largest integer such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that wt$(S) \leq 2^j$ into a single set

PROOF OUTLINE FOR UPPER BOUND $O(\log^* n)$:

1. charge algorithm's query cost to its build cost
2. charge the cost of building each set, via the items in it, to the OPT sets containing those items
3. show that each OPT set is charged $O(\log^* n)$ times its contribution to the OPT cost

Step 3 is the hard part. The intuition is that, in the "merge tree" formed by tracking how the algorithm merges the elements in the OPT set, each important merge must be associated with a unique threshold 2^j.
This forces the merge tree to have weighted average leaf depth $O(\log^* n)$.

next we consider problem 2 (k-component dynamization). we start with the lower bound of k for dete4rministic algorithms, then discuss some algorithms that achieve it.

PROBLEM 2
LOWER BOUND

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^t I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we sketch a proof for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\}, \{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\}, \{\varepsilon, 0\}$	ε	0	ε
4	0	$\{1\}, \{\varepsilon, 0, 0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\}, \{\varepsilon, 0, 0, \ldots, 0\}$	ε	0	ε
m	0	$\{1, \varepsilon, 0, 0, \ldots, 0, 0\}$	$1+\varepsilon$	0	ε
			total: $2 + (m\text{-}1)\varepsilon$	$\min(2+\varepsilon,$	$1+(m\text{-}1)\varepsilon)$

Alg chooses $m \approx 1/\varepsilon$, so $\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{2+1}{2} = 3/2$

If there were no "setup cost" of 1 at time 1, ratio would be $\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{1+1}{1} = 2$

rent or buy.

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \leq k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we sketch a proof for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\}, \{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\}, \{\varepsilon, 0\}$	ε	0	ε
4	0	$\{1\}, \{\varepsilon, 0, 0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\}, \{\varepsilon, 0, 0, \ldots, 0\}$	ε	0	ε
m	0	$\{1, \varepsilon, 0, 0, \ldots, 0, 0\}$	$1+\varepsilon$	0	ε
			total: $2+(m\text{-}1)\varepsilon$	$\min(2+\varepsilon,$	$1+(m\text{-}1)\varepsilon)$

Alg chooses $m \approx 1/\varepsilon$, so $\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{2+1}{2} = 3/2$

If there were no "setup cost" of 1 at time 1, ratio would be $\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{1+1}{1} = 2$

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we sketch a proof for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\}, \{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\}, \{\varepsilon, 0\}$	ε	0	ε
4	0	$\{1\}, \{\varepsilon, 0, 0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\}, \{\varepsilon, 0, 0, \ldots, 0\}$	ε	0	ε
m	0	$\{1, \varepsilon, 0, 0, \ldots, 0, 0\}$	$1+\varepsilon$	0	ε
m+1	$\sqrt{\epsilon}$	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}\}$	$\sqrt{\epsilon}$	$1+\sqrt{\epsilon}+\epsilon$	$\sqrt{\epsilon}+\epsilon$
m+2	0	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}, 0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
⋮	⋮	⋮	⋮	⋮	⋮
	0	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}, 0, \ldots, 0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
	0	$\{1, \varepsilon, 0, \ldots, 0, \sqrt{\epsilon}, 0, \ldots, 0\}$	$1+\sqrt{\epsilon}+\epsilon$	0	$\sqrt{\epsilon}+\epsilon$

⋮

For this second round the ratio is $2 - O(\sqrt{\epsilon})$. Repeating drives the total ratio arbitrarily near 2.

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^t I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we sketch a proof for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\},\{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\},\{\varepsilon,0\}$	ε	0	ε
4	0	$\{1\},\{\varepsilon,0,0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\},\{\varepsilon,0,0,\ldots,0\}$	ε	0	ε
m	0	$\{1,\varepsilon,0,0,\ldots,0,0\}$	$1+\varepsilon$	0	ε
m+1	$\sqrt{\epsilon}$	$\{1,\varepsilon,0,\ \ldots,0\},\{\sqrt{\epsilon}\}$	$\sqrt{\epsilon}$	$1+\sqrt{\epsilon}+\epsilon$	$\sqrt{\epsilon}+\epsilon$
m+2	0	$\{1,\varepsilon,0,\ \ldots,0\},\{\sqrt{\epsilon},0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
⋮	⋮	⋮	⋮	⋮	⋮
	0	$\{1,\varepsilon,0,\ \ldots,0\},\{\sqrt{\epsilon},0,\ldots,0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
	0	$\{1,\varepsilon,0,\ \ldots,0,\sqrt{\epsilon},0,\ldots,0\}$	$1+\sqrt{\epsilon}+\epsilon$	0	$\sqrt{\epsilon}+\epsilon$

⋮

For this second round the ratio is $2-O(\sqrt{\epsilon})$. Repeating drives the total ratio arbitrarily near 2.

next we discuss a k-competitive algorithm for k-component dynamization.

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

we associate a "credit" with each set in the current cover

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

The paper also gives a second "recursive rent-or buy" algorithm with a very different analysis.

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

we associate a "credit" with each set in the current cover

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

The paper also gives a second "recursive rent-or buy" algorithm with a very different analysis.

PROBLEM 2
ALGORITHM 1

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

PROOF OUTLINE:

1. let δ_t be the decrease in credit in iteration t
2. total credit given to sets is $k\sum_t \delta_t$
3. sets S_t contribute at most $k\sum_t \delta_t$ to algorithm's cost (as credit$[S_t] \geq$ wt(S_t) when merged)
4. remaining sets contribute at most $k\sum_t$ wt(I_t) to algorithm's cost (as items decrease in "rank")
5. so algorithm's cost is at most $k\sum_t$ wt$(I_t) + \delta_t$
6. charge credit to OPT (via implicit LP-dual soln) to show OPT cost is at least $\sum_t$ wt$(I_t) + \delta_t$

The proof that the algorithm is k-competitive can be viewed as showing that the algorithm is a primal-dual algorithm, that is, that in addition to generating a solution for the given problem, it implicitly generates a solution to the dual of the linear-program relaxation of the problem. We show that the algorithm's cost is at most k times the cost of the dual solution, which is a lower bound on the optimal cost. In particular, if we let delta-t be the increase in credit in iteration t, these delta-t's somehow define a dual solution, the cost of which is sum_t wt(..) + delta_t. It is not hard to bound the algorithm's cost by k times this amount. See the paper for more details.

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

PROOF OUTLINE:

1. let δ_t be the decrease in credit in iteration t
2. total credit given to sets is $k \sum_t \delta_t$
3. sets S_t contribute at most $k \sum_t \delta_t$ to algorithm's cost (as credit$[S_t] \geq$ wt(S_t) when merged)
4. remaining sets contribute at most $k \sum_t$ wt(I_t) to algorithm's cost (as items decrease in "rank")
5. so algorithm's cost is at most $k \sum_t$ wt$(I_t) + \delta_t$
6. charge credit to OPT (via implicit LP-dual soln) to show OPT cost is at least $\sum_t$ wt$(I_t) + \delta_t$

the remaining slide is for those who are interested in better understanding how LSM systems relate to classical structures such as b-trees. we start with some observations about how Moore's law has qualitatively changed how we should think about b-trees over recent decades.

Since 2000 or so, B-trees are not optimal for many big-data workloads…

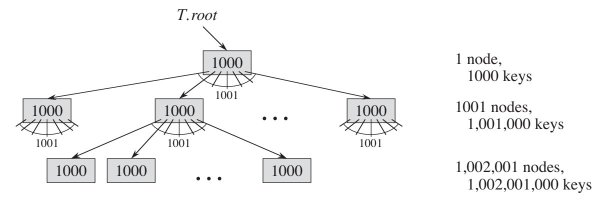
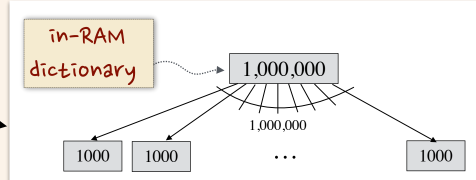


Figure 18.3 A B-tree of height 2 containing over one billion keys. Shown inside each node x is $x.n$, the number of keys in x. Each internal node and leaf contains 1000 keys. This B-tree has 1001 nodes at depth 1 and over one million leaves at depth 2.

As suggested in Figure 18.3 from *Introduction to Algorithms* by CLRS



Contract the non-leaf nodes into a single mega-root, *held in RAM* and implemented as a RAM-based ordered dictionary.

But doing it THIS way achieves about 1 disk access per read.

Observation 1. *For on-disk storage in 2020 and beyond, B-tree node degree should be over 1000.*

Why?

a. node degree ≈ number of keys that can be fetched disk in the twice the disk-access time

b. disks in 2020: access time ≈ *milliseconds*; throughput ≈ *gigabytes per second*
⟶ can fetch *megabytes* from disk in twice the disk-access time

c. (assuming 1K keys, say) we can fetch *thousands* of keys in twice the disk access time

tldr: ideal degree grows rapidly, following Moore's law, over the years.

Observation 2. *→ The non-leaf nodes will make up less than 0.1% of the total bytes used by the B-tree.*

Observation 3. *Database servers are typically configured so that RAM size is 1–3% of disk size [31, p. 227] !!!*

Observation 4. *→ Can easily hold all non-leaf nodes in (10% of) RAM. Then each read/write requires about 1 disk access.*

Observation 5. *Contract all non-leaf nodes into one single mega-root, store in (RAM-based) ordered dictionary.*

Since 2000 or so, B-trees are not optimal for many big-data workloads…

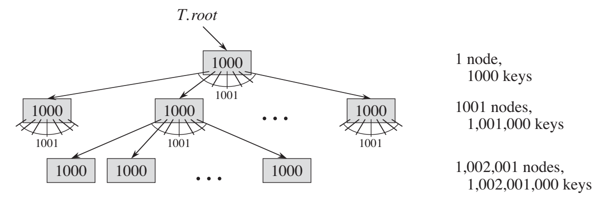
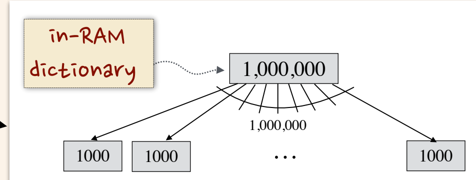


Figure 18.3 A B-tree of height 2 containing over one billion keys. Shown inside each node x is $x.n$, the number of keys in x. Each internal node and leaf contains 1000 keys. This B-tree has 1001 nodes at depth 1 and over one million leaves at depth 2.

As suggested in Figure 18.3 from *Introduction to Algorithms* by CLRS



Contract the non-leaf nodes into a single mega-root, *held in RAM* and implemented as a RAM-based ordered dictionary.

But doing it THIS way achieves about 1 disk access per read.

Observation 1. *For on-disk storage in 2020 and beyond, B-tree node degree should be over 1000.*

Why?

a. node degree ≈ number of keys that can be fetched disk in the twice the disk-access time
b. disks in 2020: access time ≈ *milliseconds*; throughput ≈ *gigabytes per second*
⟶ can fetch *megabytes* from disk in twice the disk-access time
c. (assuming 1K keys, say) we can fetch *thousands* of keys in twice the disk access time

tldr: ideal degree grows rapidly, following Moore's law, over the years.

Observation 2. *→ The non-leaf nodes will make up less than 0.1% of the total bytes used by the B-tree.*

Observation 3. *Database servers are typically configured so that RAM size is 1–3% of disk size [31, p. 227] !!!*

Observation 4. *→ Can easily hold all non-leaf nodes in (10% of) RAM. Then each read/write requires about 1 disk access.*

Observation 5. *Contract all non-leaf nodes into one single mega-root, store in (RAM-based) ordered dictionary.*

Competitive Data-Structure Dynamization

— SODA 2021 —

(a 25-minute talk summarizing the conference paper)

Claire Mathieu
CNRS, Paris

Rajmohan Rajaraman
Northeastern University

Neal E. Young
University of California Riverside
Northeastern University

Arman Yousefi
Google

— research funded by NSF and Google

This is the end of the talk. Thank you for your attention.

Competitive Data-Structure Dynamization

— SODA 2021 —

(a 25-minute talk summarizing the conference paper)

Claire Mathieu
CNRS, Paris

Rajmohan Rajaraman
Northeastern University

Neal E. Young
University of California Riverside
Northeastern University

Arman Yousefi
Google

— research funded by NSF and Google