

Lazy Search Trees

Bryce Sandlund – University of Waterloo

Sebastian Wild – University of Liverpool

Problem Description

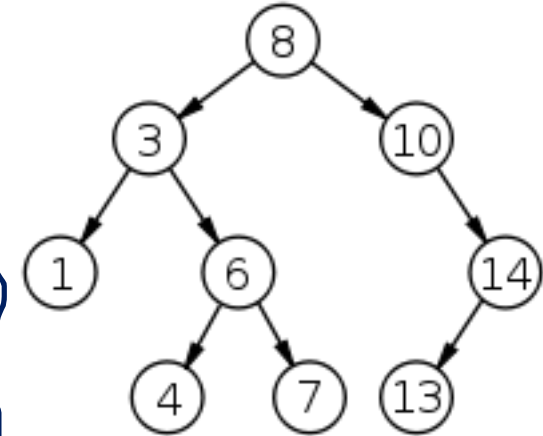
Support the operations of a binary search tree, including:

Queries:

- Rank(k)
- Select(r)
- Membership(k)
- Predecessor(k)
- Successor(k)
- Minimum()
- Maximum()

Updates:

- Insert(k)
- Delete(ptr)
- ChangeKey(ptr, k')
- Split(r)
- Merge(T_1 , T_2)



Solvable in $O(1)$ time per operation using hashing.

The standard dictionary problem need not maintain order.

We call this the **sorted dictionary problem**.

Existing Sorted Dictionaries

- AVL trees
 - Red-black trees
 - Splay trees
- $O(\log n)$ time per operation, all balanced binary search trees

Supports stronger theorems (complexities are amortized):

Theorem Name	Access Time of Element i	Explanation
Static Optimality	$O\left(\log\left(\frac{m}{q(i)}\right)\right)$	Element i is accessed $q(i)$ times in an access sequence of length m .
Working Set	$O(\log(t(i) + 1))$	$t(i)$ is the number of items accessed since i was last accessed.
Dynamic Finger	$O(\log(j - i + 1))$	j was the last item accessed.

Dynamic Optimality Conjecture

Conjectures that **splay trees** are $O(1)$ -competitive with any other binary search tree on **any** sufficiently long access sequence.

Has received vast attention:

Paper	Result	Citation
Self-adjusting binary search trees	Splay tree paper	Sleator & Tarjan, J. ACM '85
Lower bounds for accessing binary search trees with rotations	Gives lower bounds on required complexity of binary search tree access sequences.	Wilbur, SICOMP '89
On the dynamic finger conjecture for splay trees (parts I and II)	Proves the dynamic finger property for splay trees.	Cole et al., SICOMP '00 (Part I) Cole, SICOMP '00 (Part II)
Alternatives to splay trees with $O(\log n)$ worst-case access times	Generalizes the working set and dynamic finger properties to the unified bound .	Iacono, SODA '01
Dynamic optimality-almost	Gives an $O(\log \log n)$ -competitive BST.	Demaine et al., SICOMP '07
The geometry of binary search trees	Gives a geometric view of BST access and proposes a conjectured-optimal “greedy BST”.	Demaine et al., SODA '09

Dynamic Optimality Conjecture

Conjectures that **splay trees** are $O(1)$ -competitive with any other binary search tree on **any** sufficiently long access sequence.

Has received vast attention:

Paper	Citation
A unified access bound on comparison-based dynamic dictionaries	Bădoiu et al., Theoretical Computer Science '07
Pattern-avoiding access in binary search trees	Chalermsook et al., FOCS '15
Weighted dynamic finger in binary search trees	Iacono & Langerman, SODA '16
A new path from splay to dynamic optimality	Levy & Tarjan, SODA '19
Smooth heaps and a dual view of self-adjusting data structures	Kozma & Saranurak, SICOMP '19
Competitive online search trees on trees	Bose et al., SODA '20

Dynamic Optimality Conjecture

Conjectures that **splay trees** are $O(1)$ -competitive with any other binary search tree on **any** sufficiently long access sequence.

Key Questions:

1. Why consider **access** instead of **insert** and **query**?
2. Why only consider **binary search trees**?

Motivation

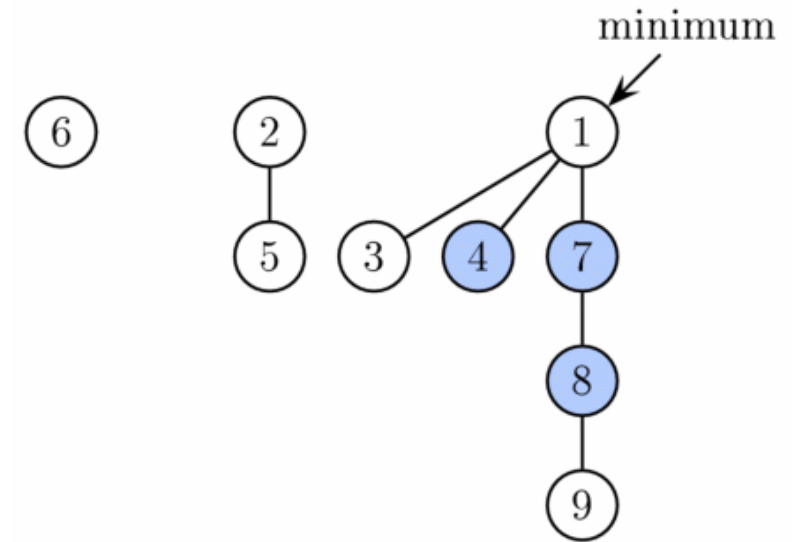
Fibonacci heaps and its derivatives offer **improved runtimes** if we restrict the operation set.

Queries:

- ~~Rank(k)~~
- ~~Select(r)~~
- ~~Membership(k)~~
- ~~Predecessor(k)~~
- ~~Successor(k)~~
- Minimum(): $O(\log n)$
- Maximum(): $O(\log n)$

Updates:

- Insert(k): $O(1)$
- Delete(ptr): $O(\log n)$
- ChangeKey(ptr, k'): $O(1)$
- ~~Split(r)~~
- Merge(T_1, T_2): $O(1)$



Motivation

Can **efficient priority queues** be generalized to support queries for any rank?

Such a data structure could provide a **sorted dictionary** with superior insertion times to a binary search tree.

Intuition

Avoid **sorting** on insert.

Progressively **sort** as queries are answered.

Related Work – Deferred Data Structures

Paper	Result	Citation
Deferred data structuring	Shows q queries on a static set of n unsorted elements can be answered in $O(n \log q + q \log n)$ time and gives a matching lower bound.	Karp, Motwani, & Raghavan, SICOMP '88
Dynamic deferred data structuring	Shows q' queries, insertions, and deletions can be answered on an initial set of n_0 unsorted elements in $O(n_0 \log q' + q' \log n_0)$.	Ching, Melhorn, & Smid, Info. Proc. Letters '90

Related Work – Online Dynamic Multiple Selection

Given an unsorted array A , select elements of ranks r_1, r_2, \dots, r_q .

A	3	7	2	5	11	16	2	11	6	0	2	9	10	14	2	8
-----	---	---	---	---	----	----	---	----	---	---	---	---	----	----	---	---

Related Work – Online Dynamic Multiple Selection

Given an unsorted array A , select elements of ranks r_1, r_2, \dots, r_q .

$$r_1 = 3$$

$$r_2 = 7$$

$$r_3 = 13$$

A	0	2	2	5	3	2	6	7	6	10	8	9	11	14	16	11
-----	---	---	---	---	---	---	---	---	---	----	---	---	----	----	----	----

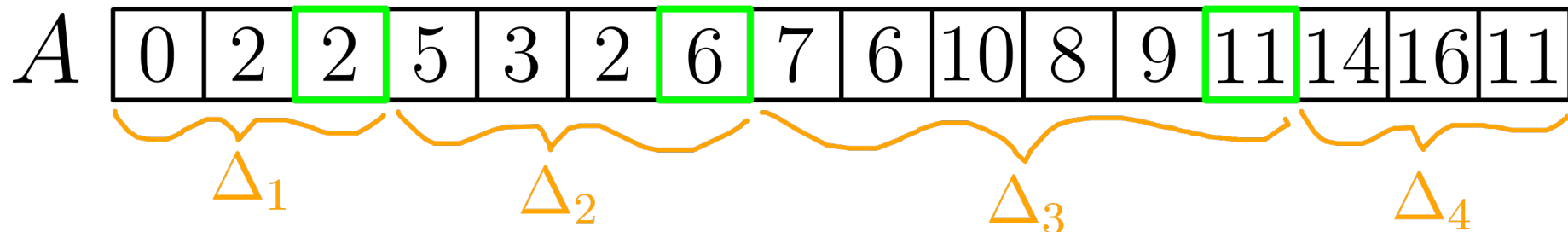
Related Work – Online Dynamic Multiple Selection

Given an unsorted array A , select elements of ranks r_1, r_2, \dots, r_q .

$$r_1 = 3$$

$$r_2 = 7$$

$$r_3 = 13$$



- Let Δ_i be the set of elements between r_{i-1} and r_i .
- Define $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- The complexity of multiple selection is $\Theta(B + n)$.

Related Work – Online Dynamic Multiple Selection

Paper	Result	Citation
Theory and implementation of online multiselection algorithms	Multiple selection where the ranks r_1, r_2, \dots, r_q are given online and in any order. $O(B + n)$ time.	Barbay et al., ESA '13
Dynamic online multiselection in internal and external memory	Online dynamic multiple selection in $O(B + n + q' \log n)$ time, where it is assumed each insertion is preceded by a search for the inserted element and again q' is the number of search, insert, and delete operations.	Barbay et al., Journal of Discrete Algorithms, '16

Shortfalls of Related Work

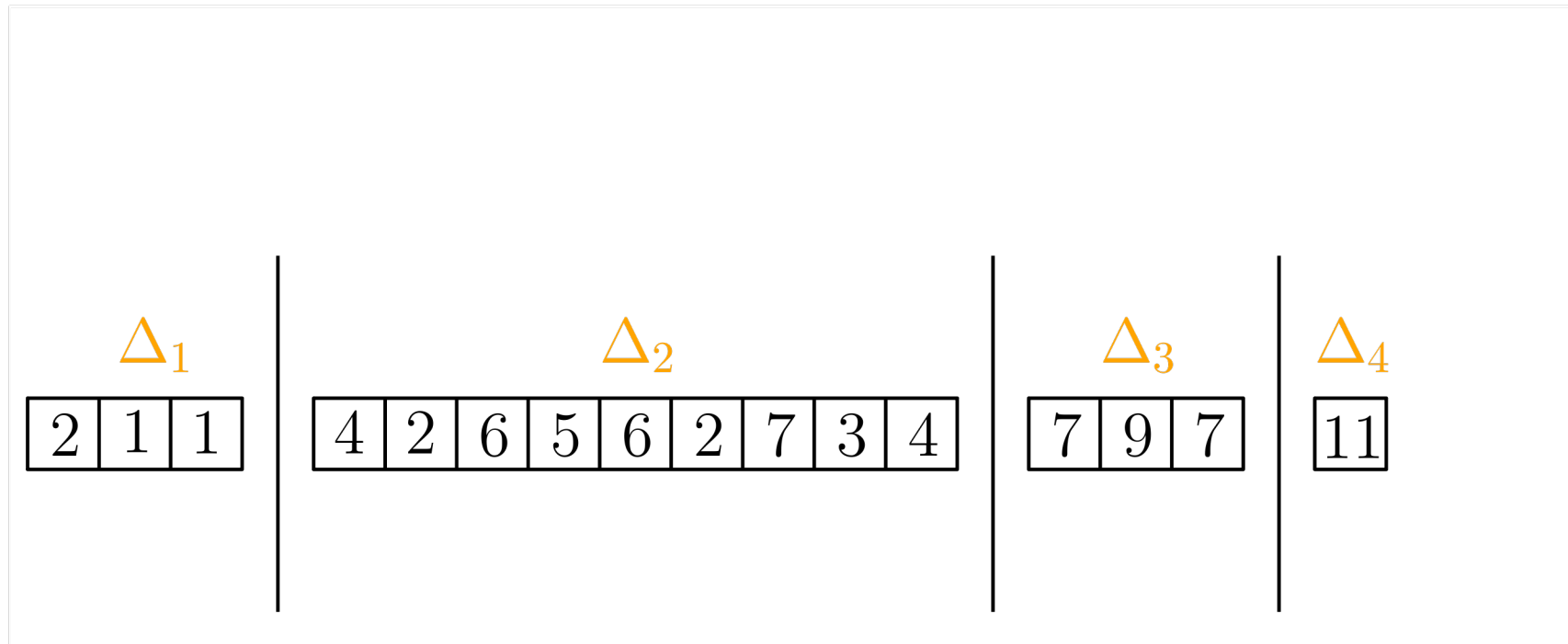
1: Weak model; query ranks are not considered.

2: Weak model; insert and query operations are coupled.

	Paper	Result	Citation	
Deferred Data Structures	Deferred data structuring	Shows q queries on a static set of n unsorted elements can be answered in $O(n \log q + q \log n)$ time and gives a matching lower bound.	Karp, Motwani, & Raghavan, SICOMP '88	1
	Dynamic deferred data structuring	Shows q' queries, insertions, and deletions can be answered on an initial set of n_0 unsorted elements in $O(n_0 \log q' + q' \log n_0)$.	Ching, Melhorn, & Smid, Info. Proc. Letters '90	2 1
Multiple Selection	Theory and implementation of online multiselection algorithms	Multiple selection where the ranks r_1, r_2, \dots, r_q are given online and in any order. $O(B + n)$ time.	Barbay et al., ESA '13	
	Dynamic online multiselection in internal and external memory	Online dynamic multiple selection in $O(B + n + q' \log n)$ time, where it is assumed each insertion is preceded by a search for the inserted element and again q' is the number of search, insert, and delete operations.	Barbay et al., Journal of Discrete Algorithms, '16	2

Our Model

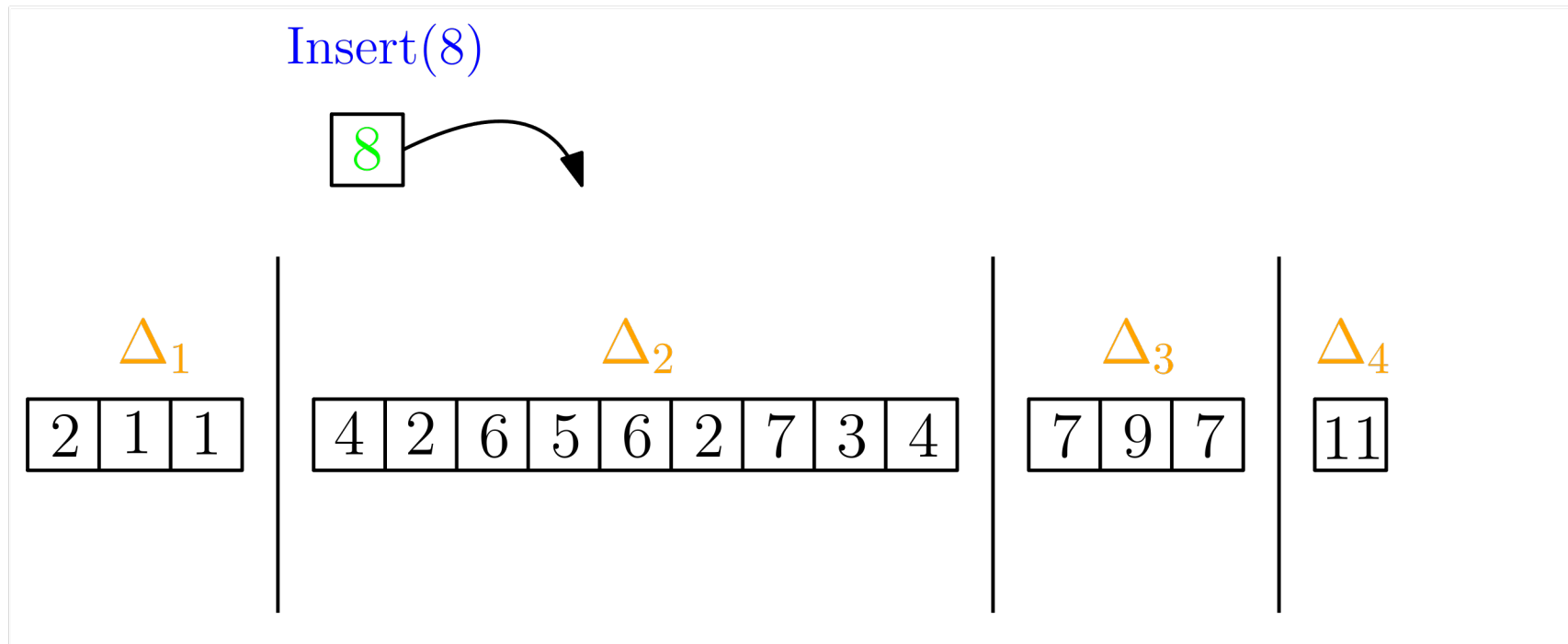
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



Inserted elements are placed into a gap that respects the partition.

Our Model

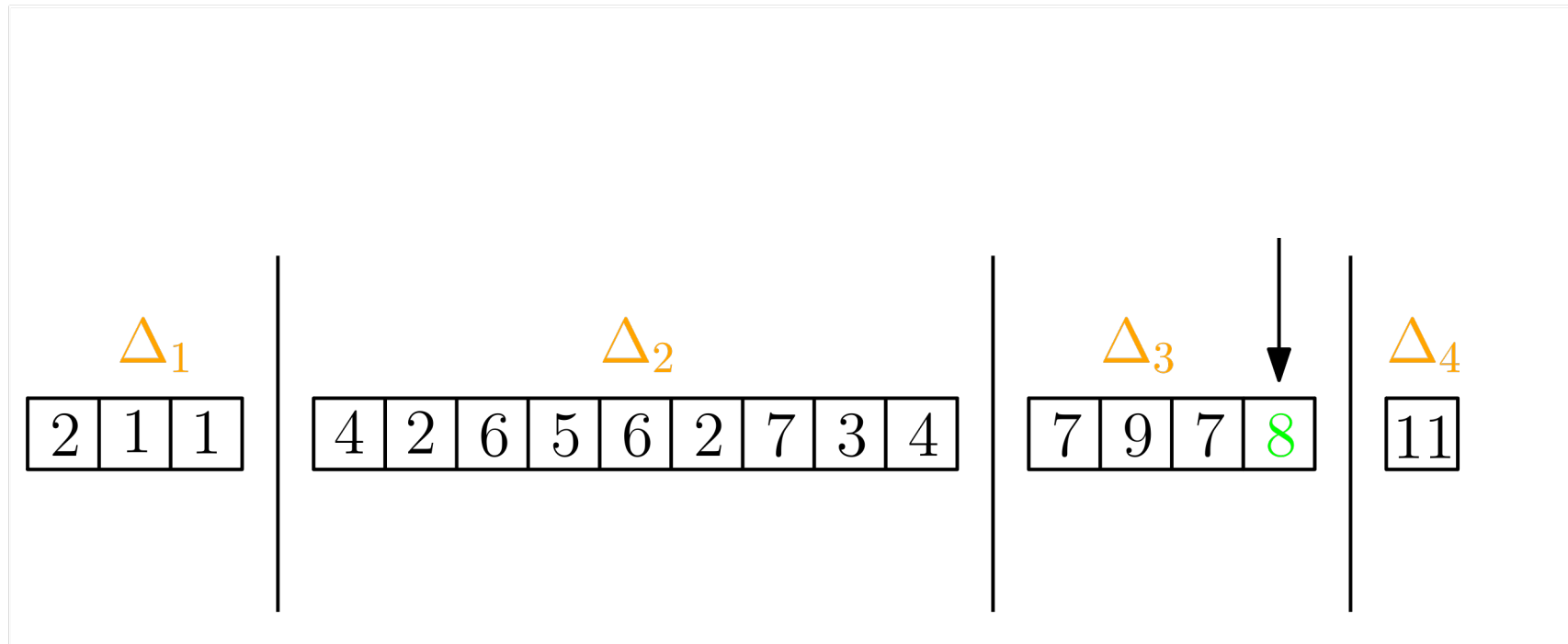
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



Inserted elements are placed into a gap that respects the partition.

Our Model

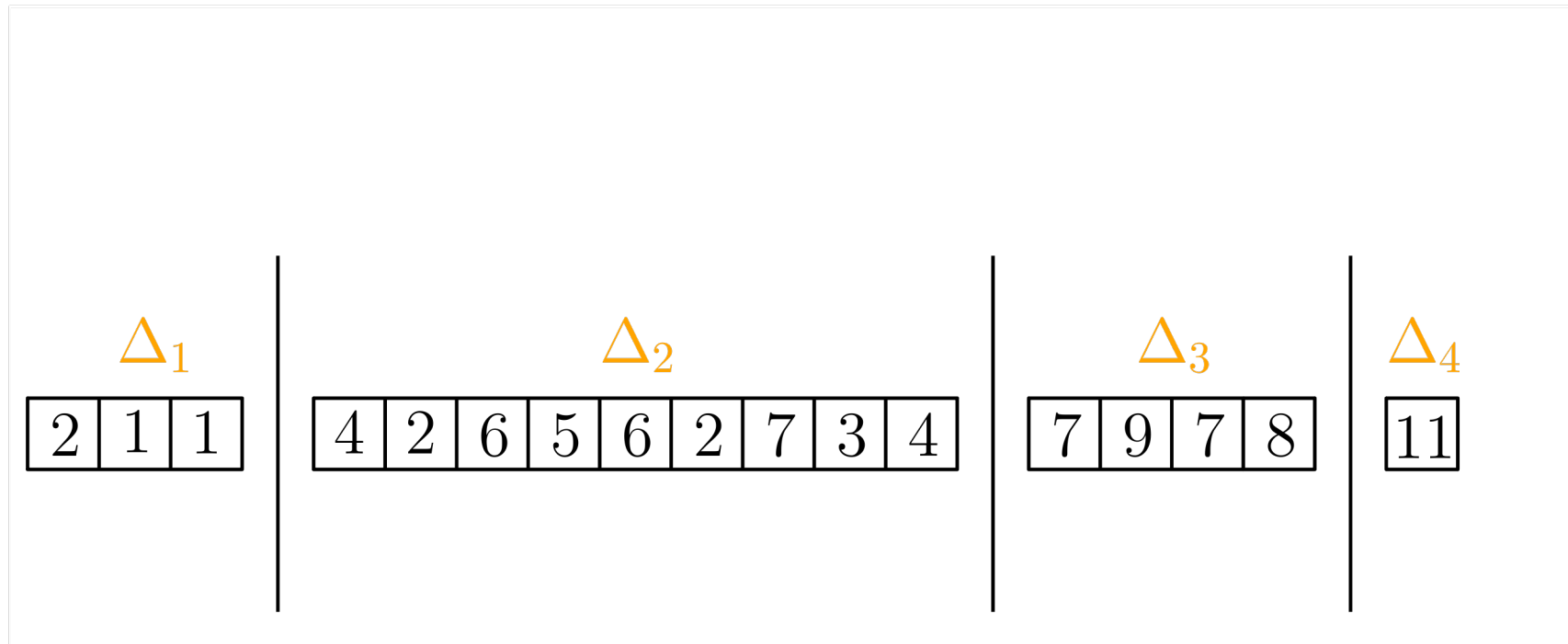
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



Inserted elements are placed into a gap that respects the partition.

Our Model

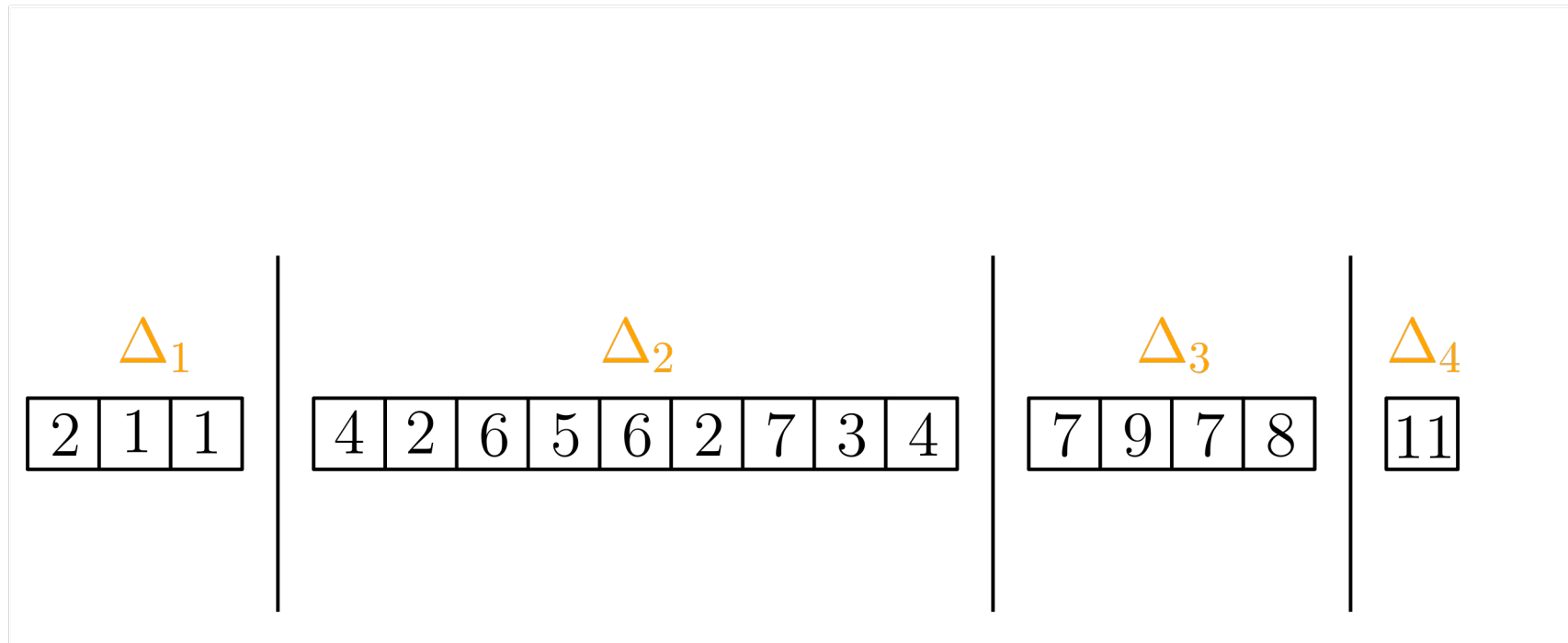
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



Inserted elements are placed into a gap that respects the partition.

Our Model

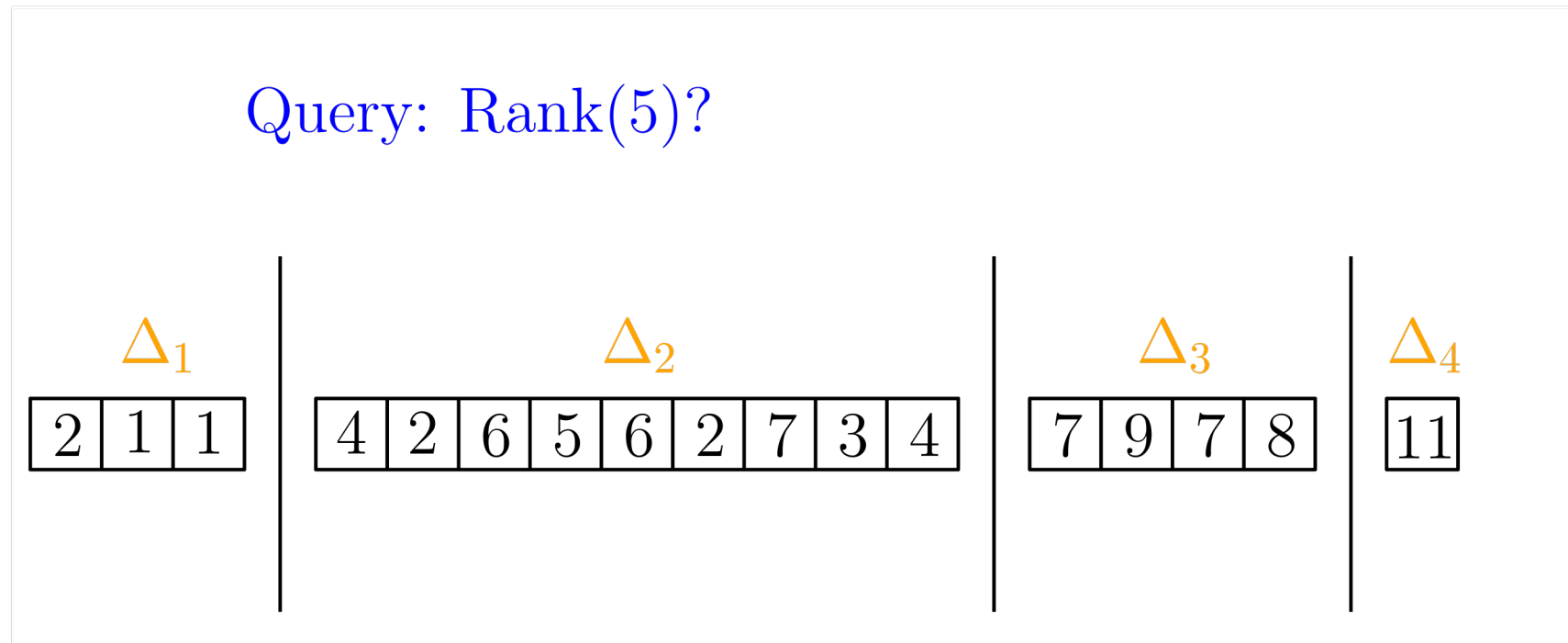
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation.

Our Model

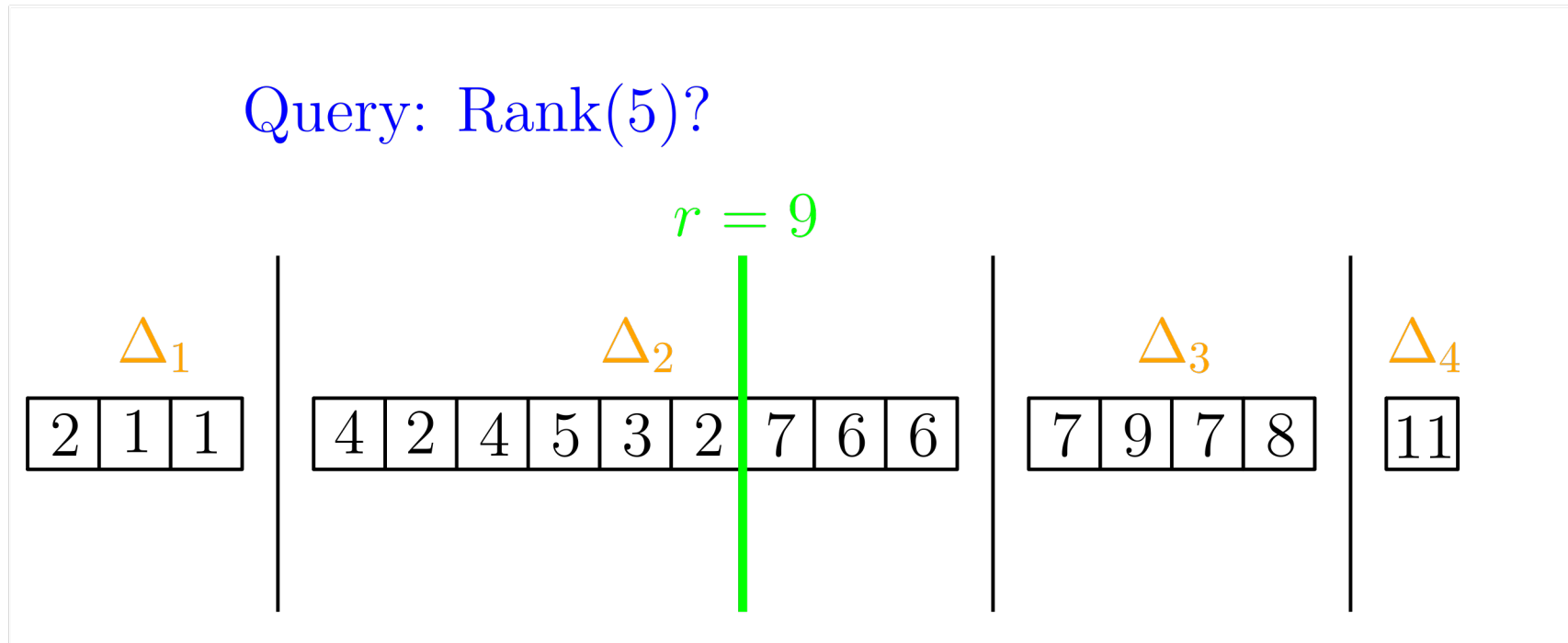
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation.

Our Model

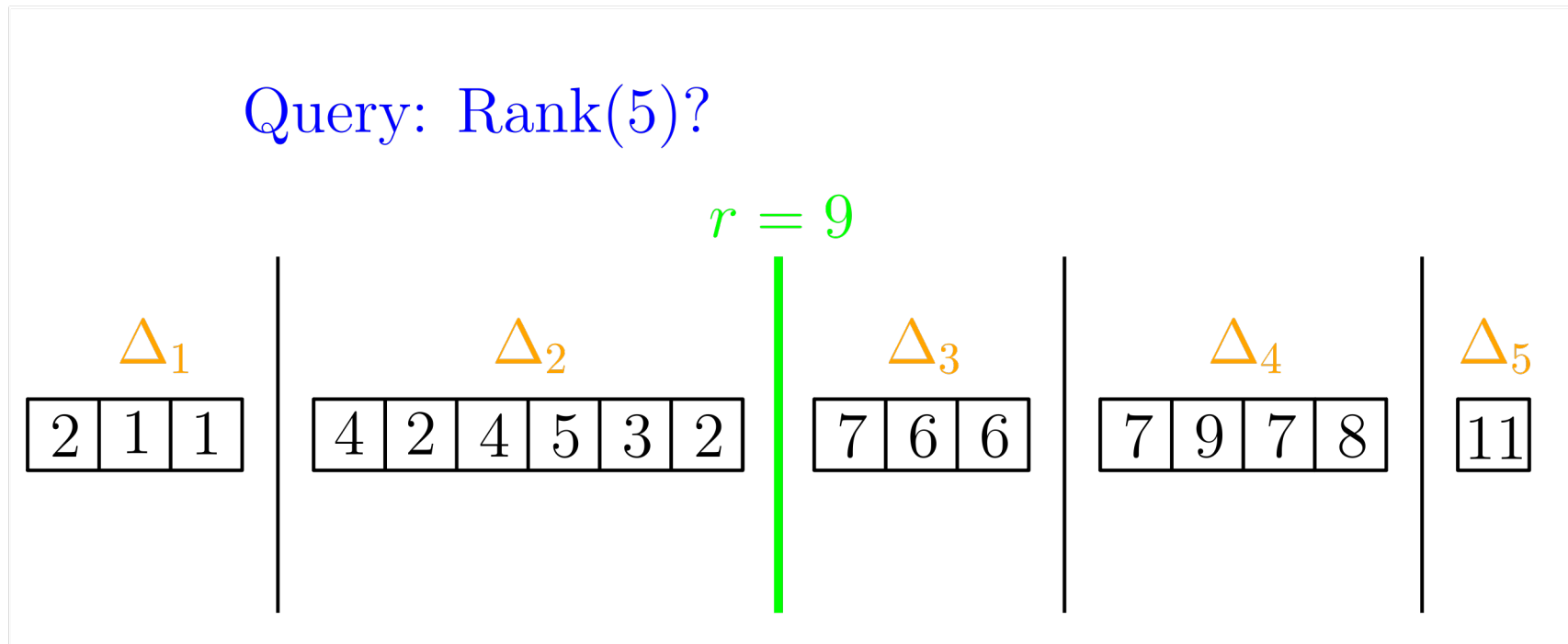
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation.

Our Model

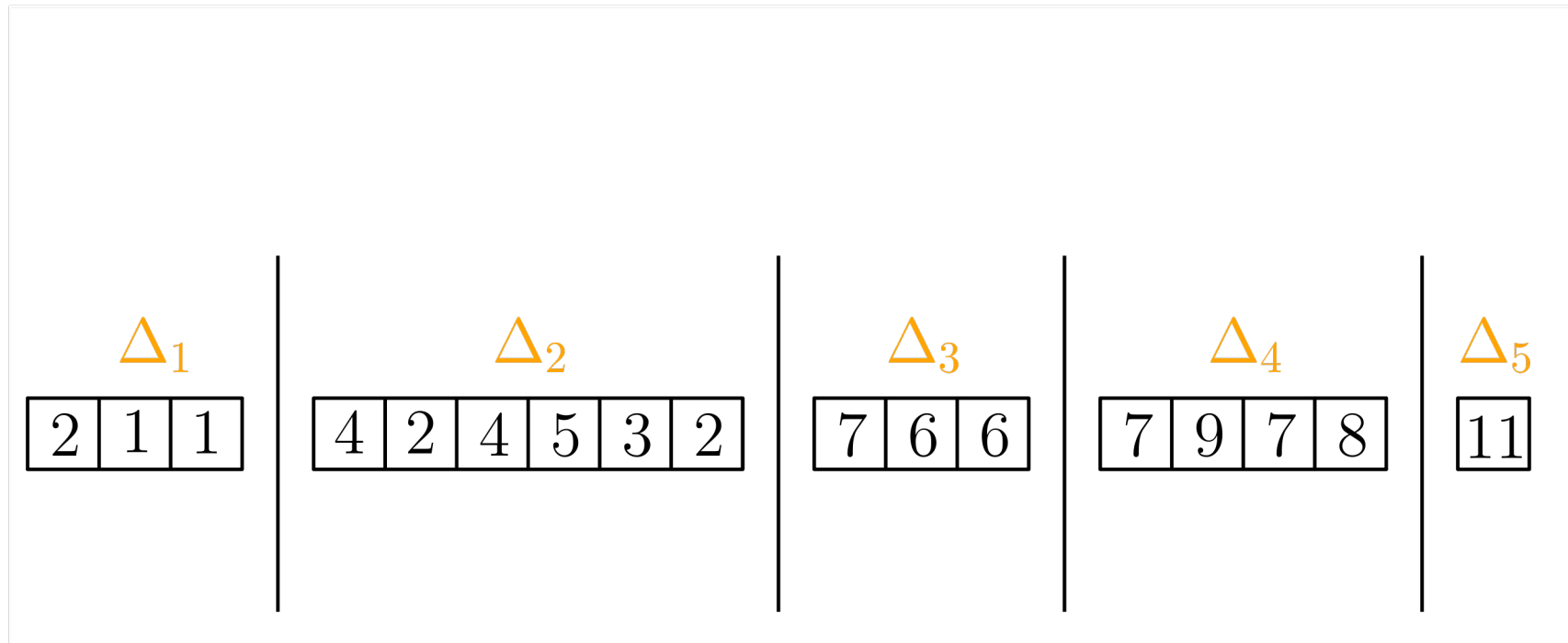
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation.

Our Model

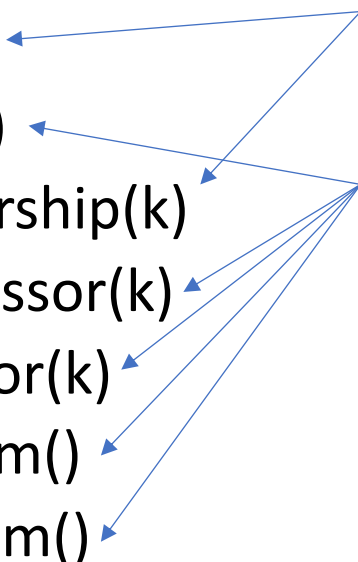
Partition elements into a set of gaps $\{\Delta_i\}$ based on rank.



A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation.

How to Choose Rank r ?

Queries:

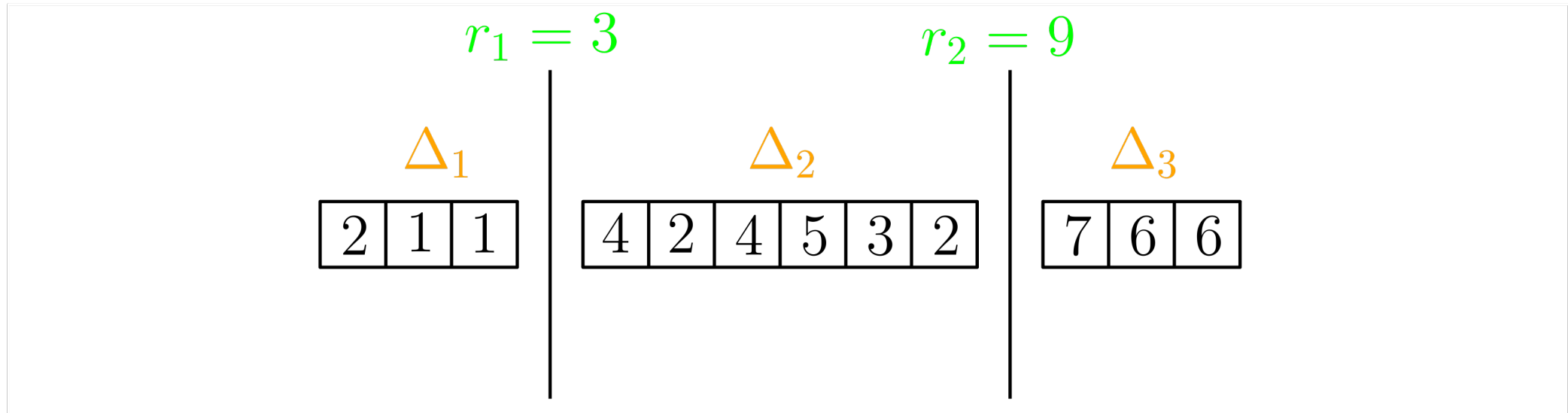
- Rank(k)
 - Select(r)
 - Membership(k)
 - Predecessor(k)
 - Successor(k)
 - Minimum()
 - Maximum()
- r is the rank of the queried element, k .
- r is the rank of the element returned.
- 
- The diagram consists of blue arrows pointing from the variable r in the query names to the explanatory text. Specifically, arrows point from r in Rank(k) and Select(r) to the first text block, and from r in Membership(k), Predecessor(k), Successor(k), Minimum(), and Maximum() to the second text block.

Updates:

- Split(r)
- r is the splitting rank.
- 
- The diagram consists of a single blue arrow pointing from the variable r in Split(r) to the explanatory text.

Intuition

Multiple selection fits directly into the framework.



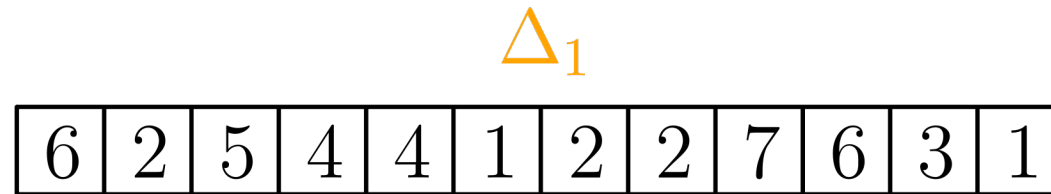
Intuition

Binary search trees correspond to each element having its own gap.



Intuition

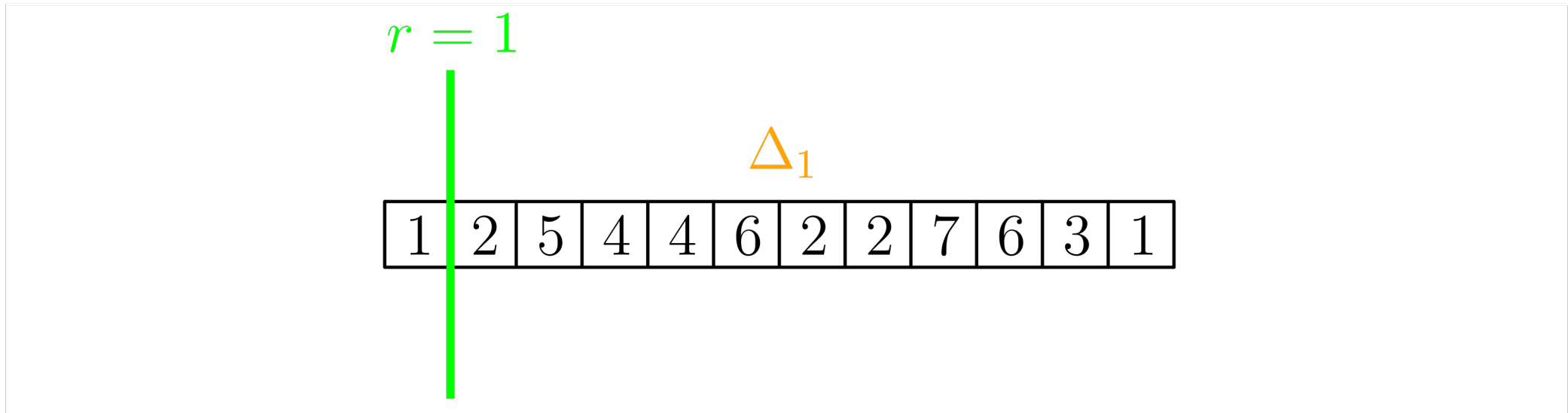
Priority queues correspond to a single gap Δ_1 .



Extract-minimum queries work as follows.

Intuition

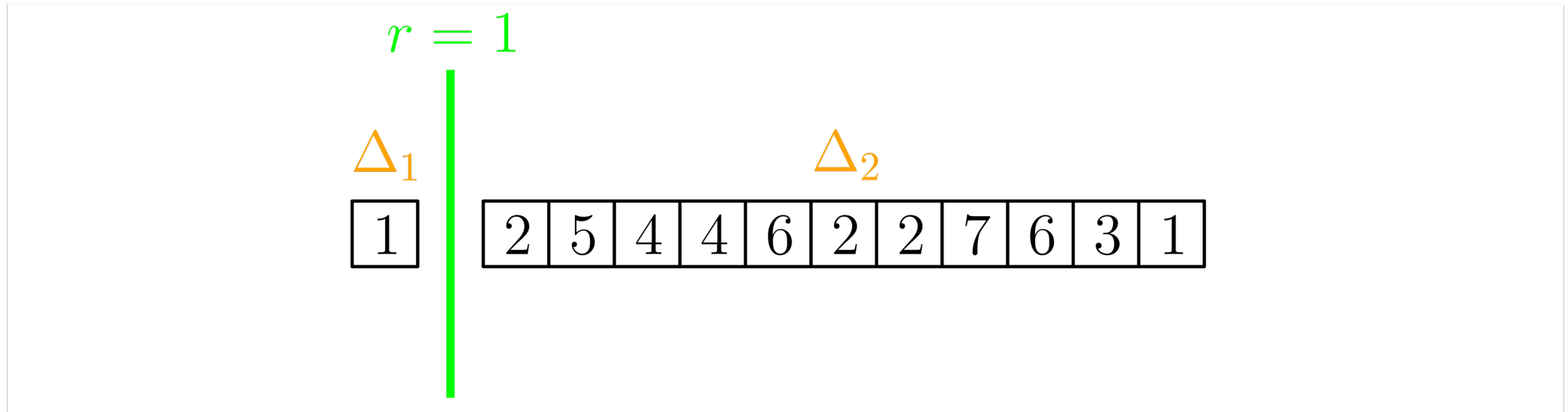
Priority queues correspond to a single gap Δ_1 .



We first perform a minimum query with rank $r = 1$.

Intuition

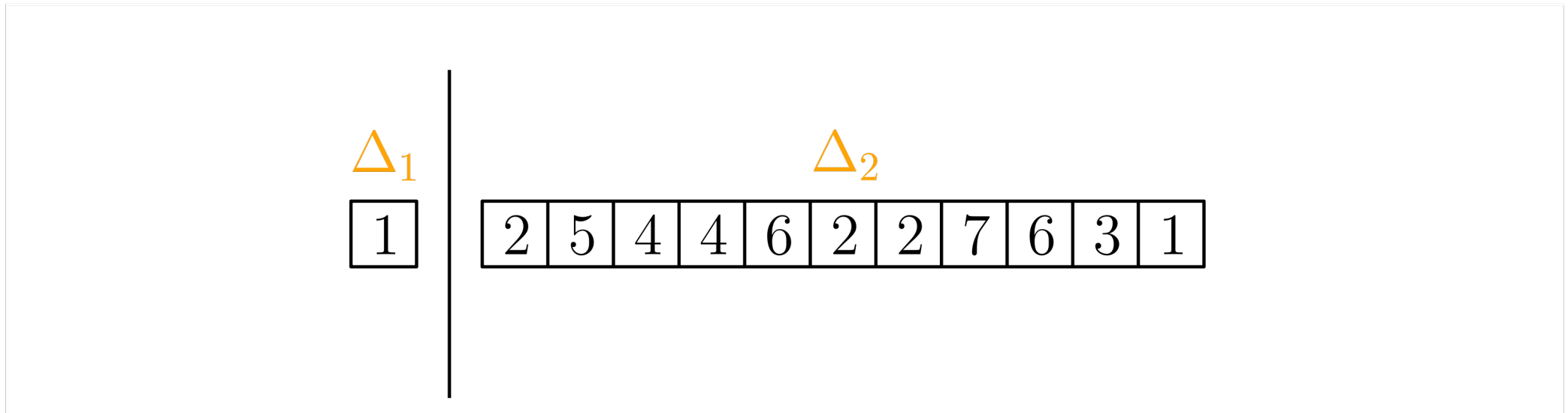
Priority queues correspond to a single gap Δ_1 .



We first perform a minimum query with rank $r = 1$.

Intuition

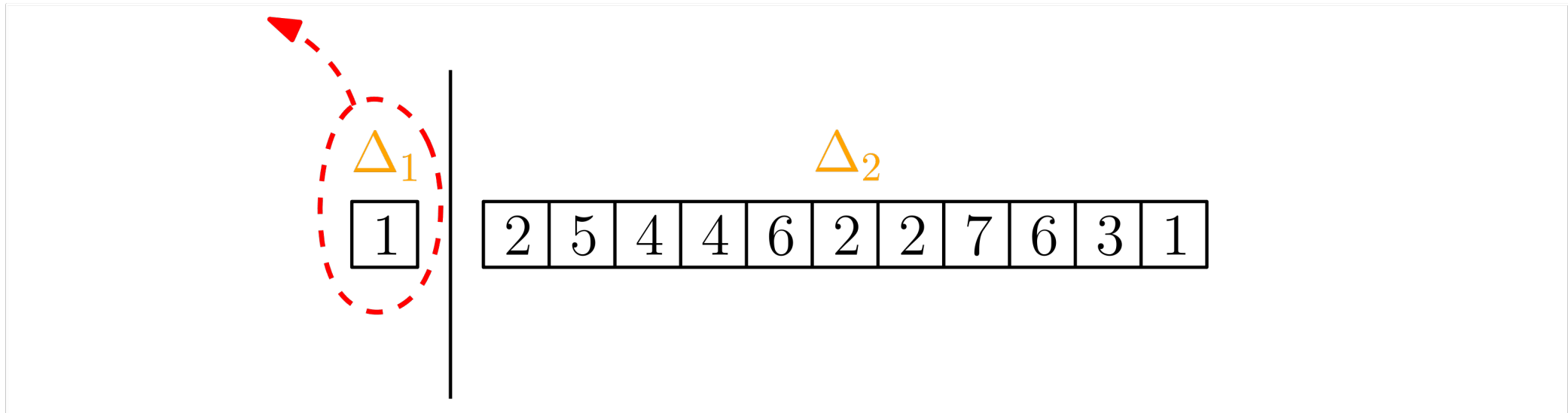
Priority queues correspond to a single gap Δ_1 .



This splits existing gap Δ_1 into two new gaps Δ_1 and Δ_2 , where $|\Delta_1| = 1$.

Intuition

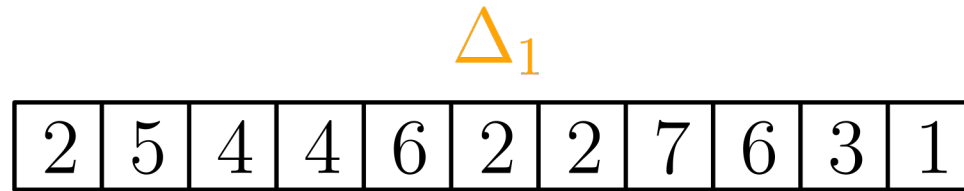
Priority queues correspond to a single gap Δ_1 .



The minimum, now in gap Δ_1 , is then removed.

Intuition

Priority queues correspond to a single gap Δ_1 .



Complexity Goal

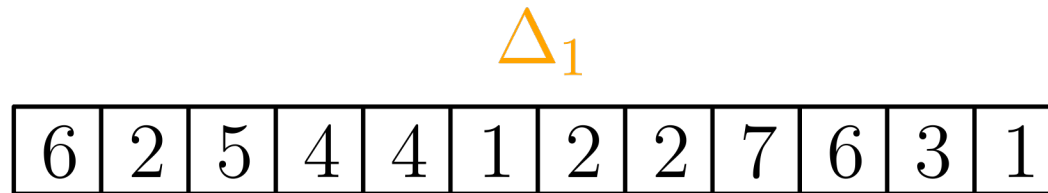
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- B characterizes the amount of information (work) present in the current gap partition.



$$B = \Theta(n \log n)$$

Complexity Goal

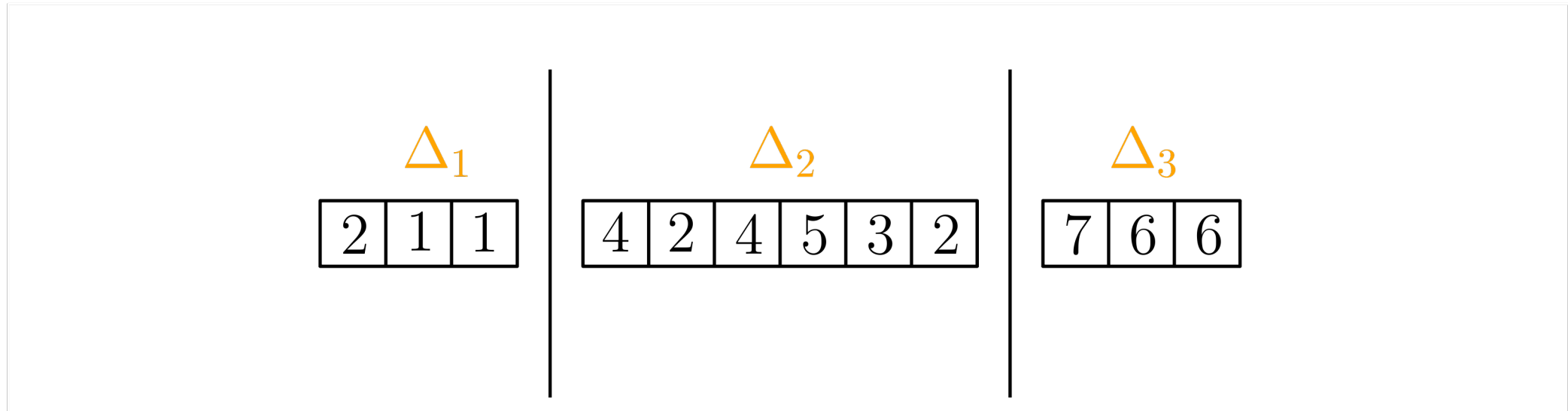
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- B characterizes the amount of information (work) present in the current gap partition.



$$B = 0$$

Complexity Goal

- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- B characterizes the amount of information (work) present in the current gap partition.



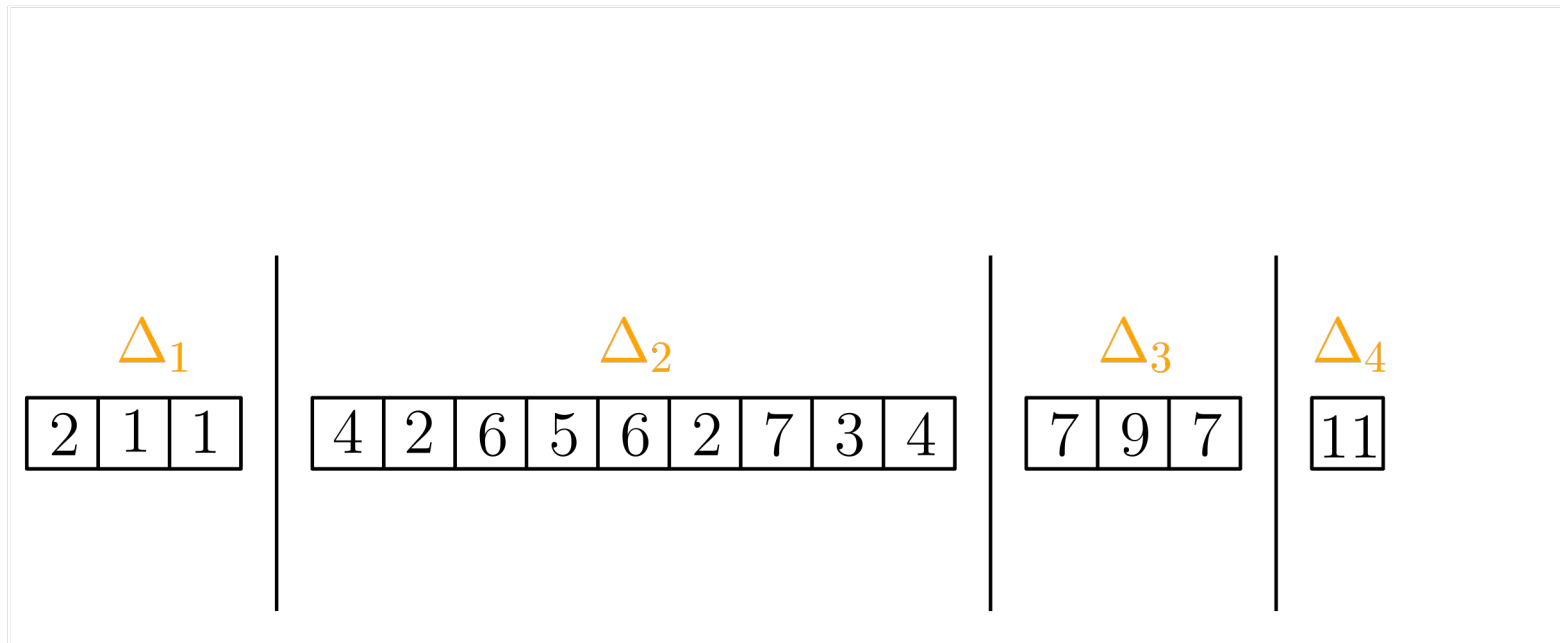
$$B = \omega(1) \text{ and } B = O(n \log q)$$

Complexity Goal

- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- **Goal:** Achieve $O(B + n)$ time on an operation sequence resulting in gaps $\{\Delta_i\}$.
- Multiple selection implies this is optimal.

Technical Overview

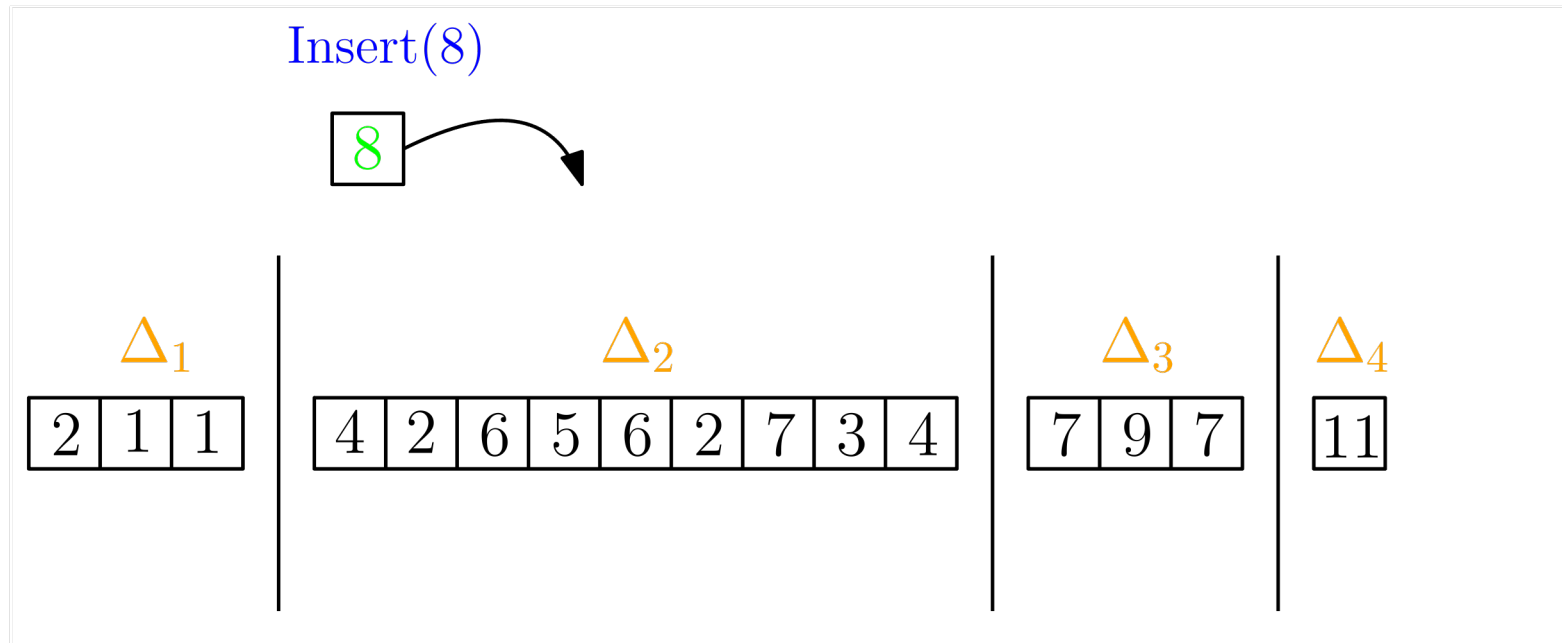
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 1:** The change in B due to insertion into Δ_i is $\Omega \left(\log \left(\frac{n}{|\Delta_i|} \right) \right)$.



$$B \text{ before: } 3 \log_2 \left(\frac{16}{3} \right) + 9 \log_2 \left(\frac{16}{9} \right) + 3 \log_2 \left(\frac{16}{3} \right) + \log_2(16)$$

Technical Overview

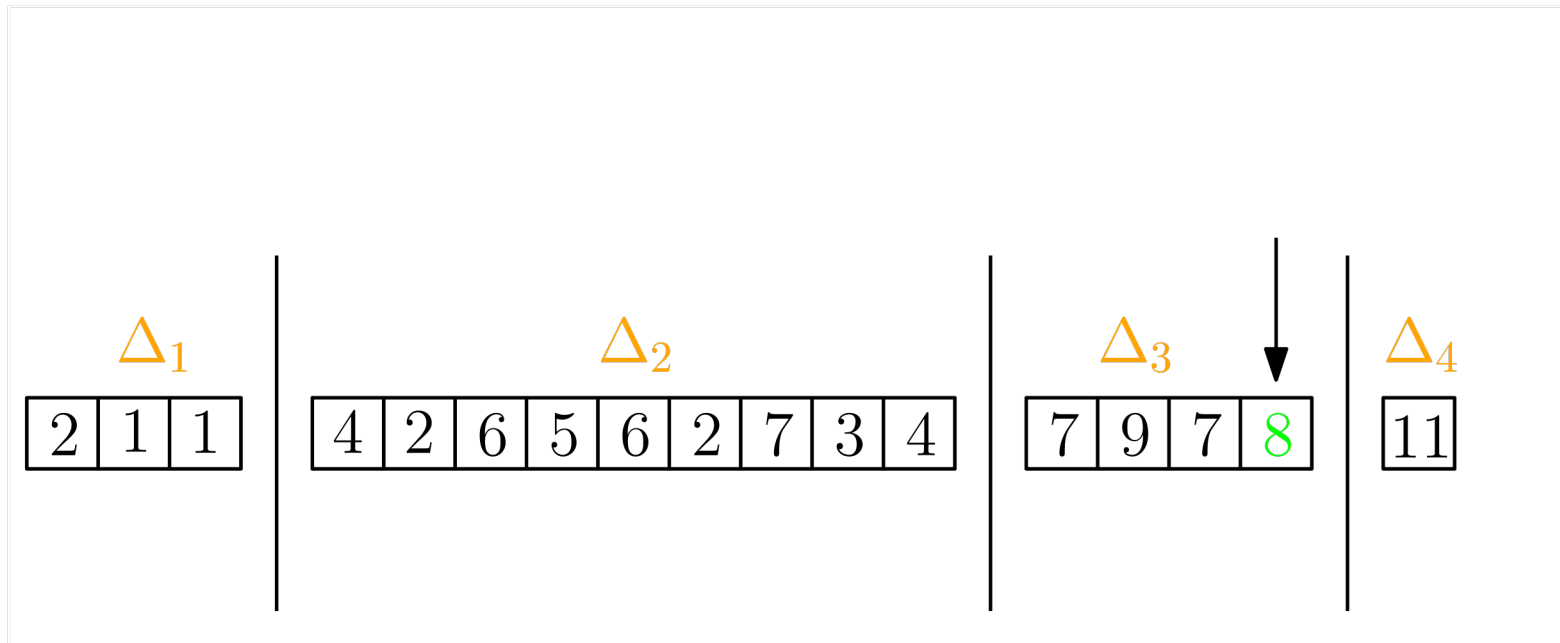
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 1:** The change in B due to insertion into Δ_i is $\Omega \left(\log \left(\frac{n}{|\Delta_i|} \right) \right)$.



$$B \text{ before: } 3 \log_2 \left(\frac{16}{3} \right) + 9 \log_2 \left(\frac{16}{9} \right) + 3 \log_2 \left(\frac{16}{3} \right) + \log_2(16)$$

Technical Overview

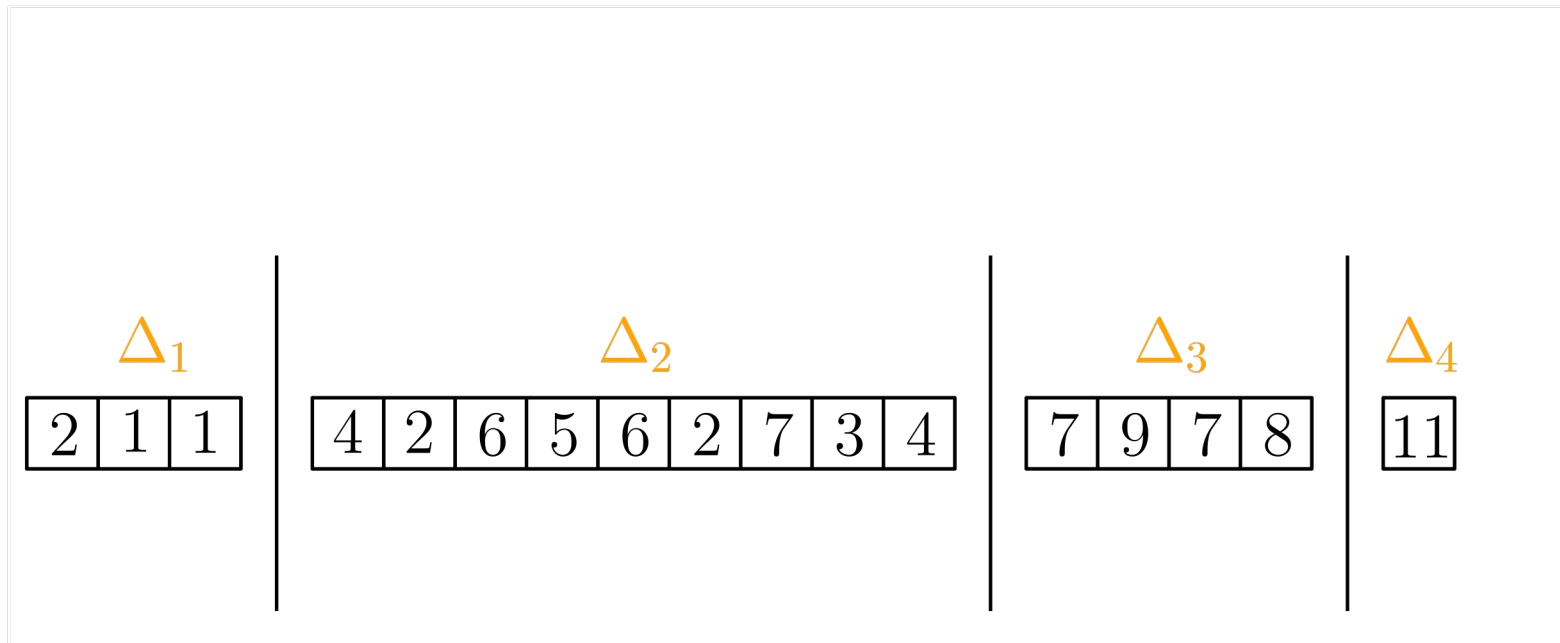
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 1:** The change in B due to insertion into Δ_i is $\Omega \left(\log \left(\frac{n}{|\Delta_i|} \right) \right)$.



$$B \text{ before: } 3 \log_2 \left(\frac{16}{3} \right) + 9 \log_2 \left(\frac{16}{9} \right) + 3 \log_2 \left(\frac{16}{3} \right) + \log_2(16)$$

Technical Overview

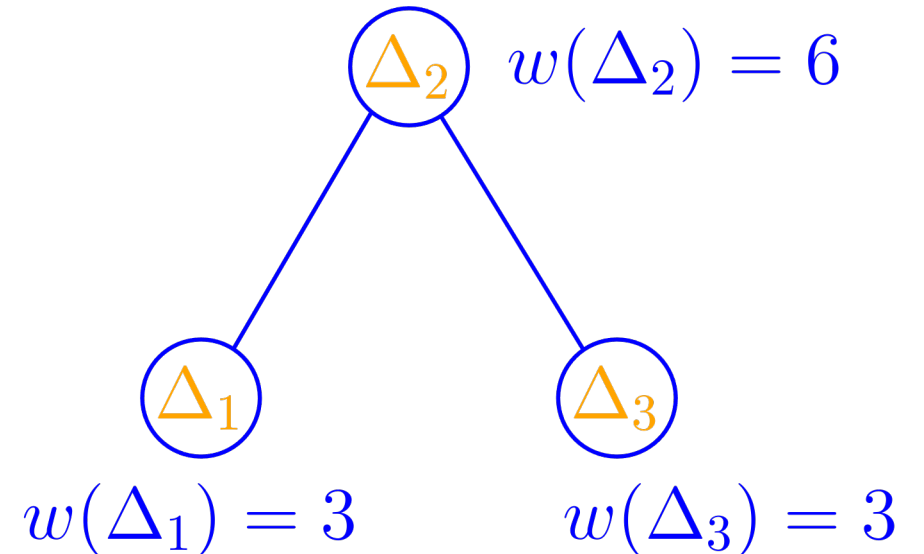
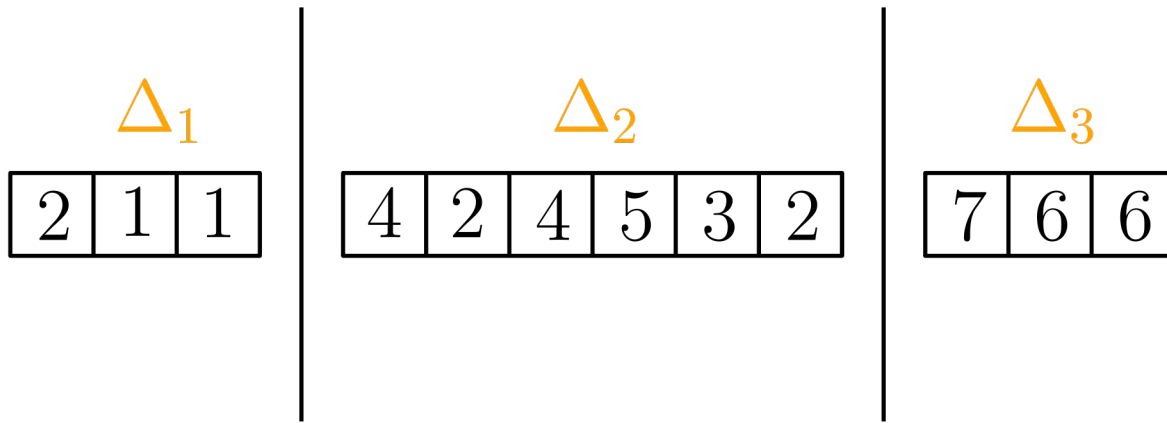
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 1:** The change in B due to insertion into Δ_i is $\Omega \left(\log \left(\frac{n}{|\Delta_i|} \right) \right)$.



$$B \text{ after: } 3 \log_2 \left(\frac{16}{3} \right) + 9 \log_2 \left(\frac{16}{9} \right) + 4 \log_2 \left(\frac{16}{4} \right) + \log_2(16)$$

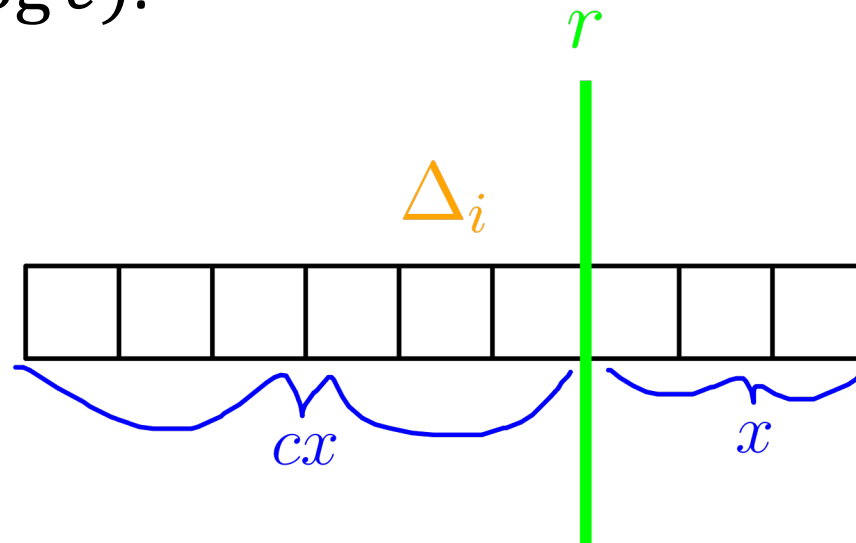
Technical Overview

- Store the set of gaps $\{\Delta_i\}$ in a **biased search tree** where gap Δ_i gets weight $|\Delta_i|$.
- Access to gap Δ_i takes time $O\left(\log\left(\frac{n}{|\Delta_i|}\right)\right)$. ✓



Technical Overview

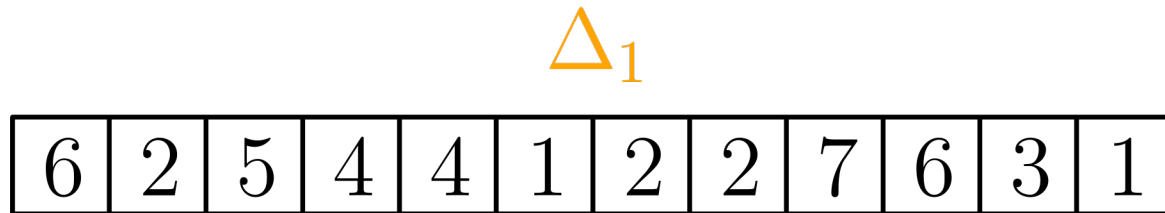
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 2:** The change in B due to splitting Δ_i into gaps of size cx and x for $c \geq 1$ is $\Omega(x \log c)$.



Proof: $\log \binom{(c+1)x}{x} = \Omega(x \log c)$.

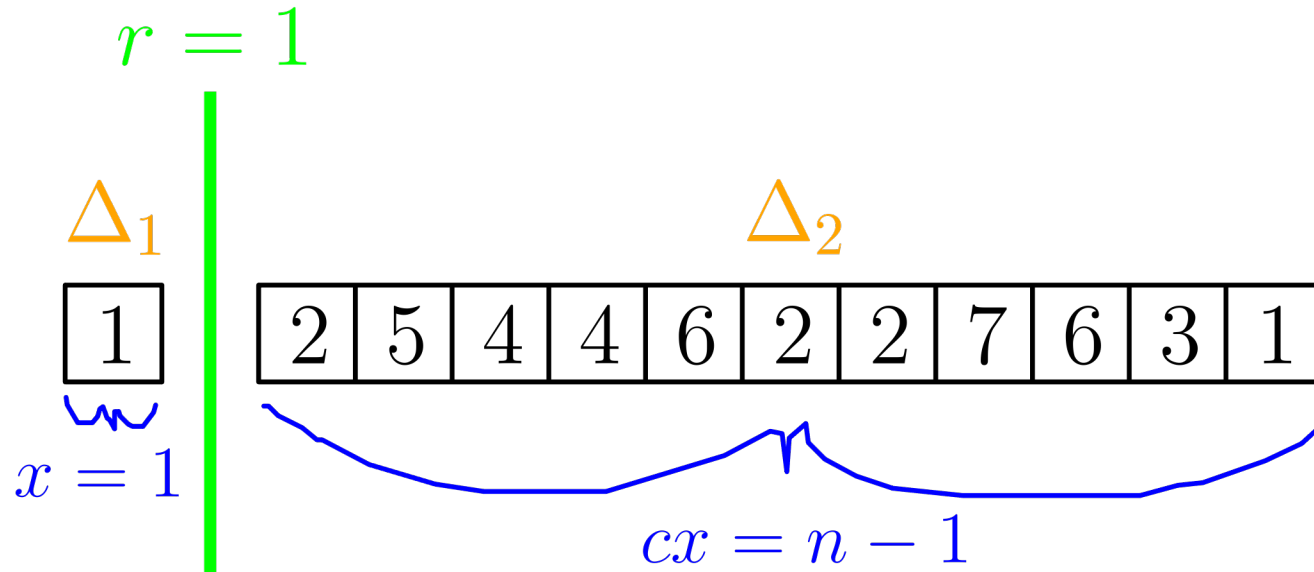
Priority Queue Complexities

- Only one gap Δ_1 , so insertion takes $O\left(\log\left(\frac{n}{|\Delta_1|}\right)\right) = O(1)$ time.



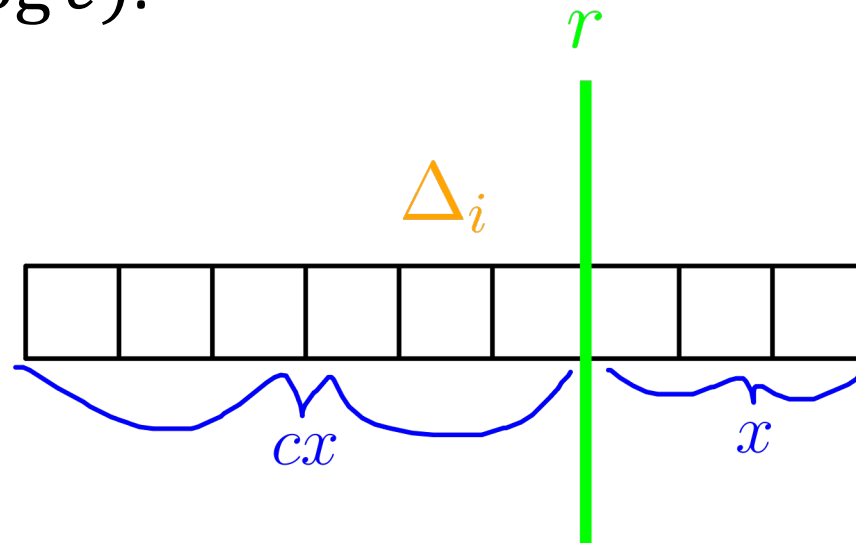
Priority Queue Complexities

- Only one gap Δ_1 , so insertion takes $O\left(\log\left(\frac{n}{|\Delta_1|}\right)\right) = O(1)$ time.
- Removal of the minimum has $x = 1$, $c = \Theta(n)$, and so should take $O(x \log c) = O(\log n)$ time.



Technical Overview

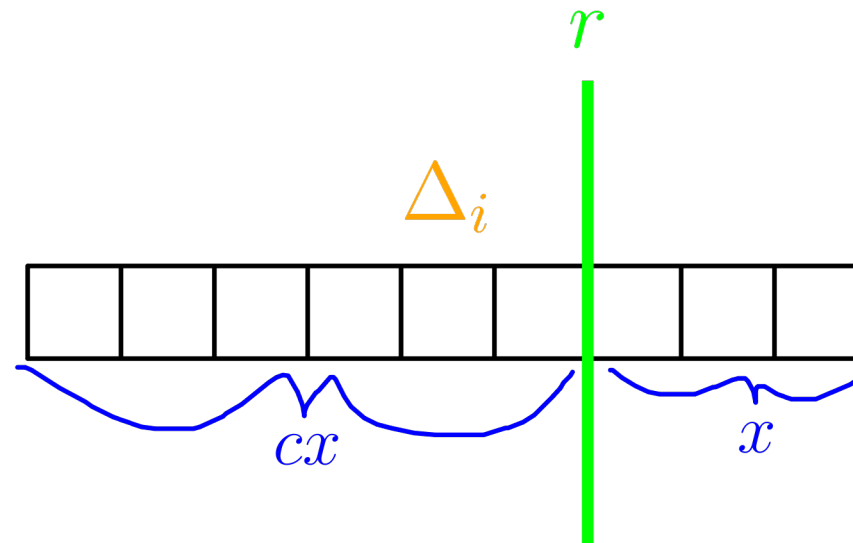
- Recall $B = \sum_i |\Delta_i| \log_2 \left(\frac{n}{|\Delta_i|} \right)$.
- Lemma 2:** The change in B due to splitting Δ_i into gaps of size cx and x for $c \geq 1$ is $\Omega(x \log c)$.



Proof: $\log \binom{(c+1)x}{x} = \Omega(x \log c)$.

Technical Overview

- Supporting the ability to split a gap Δ_i into gaps of size cx and x for $c \geq 1$ in $O(x \log c)$ time is the **main technical challenge**.
- To do so we add $O(\log \log n)$ to insertion time.

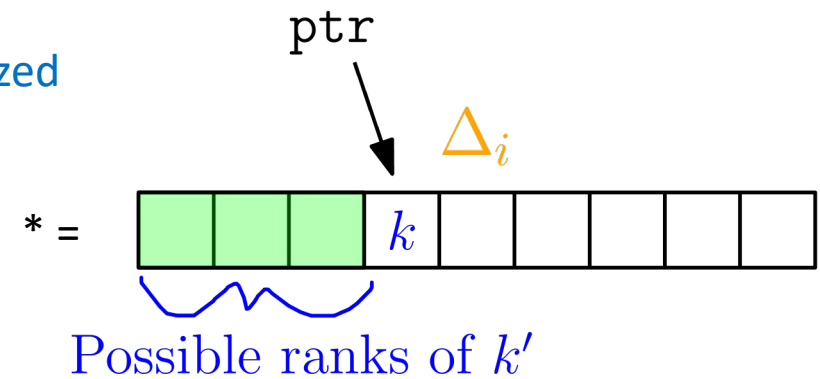


High-Level Results

- **Theorem 1:** Over a sequence of n insertions and q unique queries, performance is $O(B + \min(n \log \log n, n \log q))$.
- **Theorem 2:** $\Omega(B + n)$ is a lower bound.
- **Theorem 3:** Only $O(\min(q, n))$ pointers are required.
- **Theorem 4:** By using a splay tree as the biased search tree over the set of gaps $\{\Delta_i\}$, we can achieve its efficient access theorems automatically.

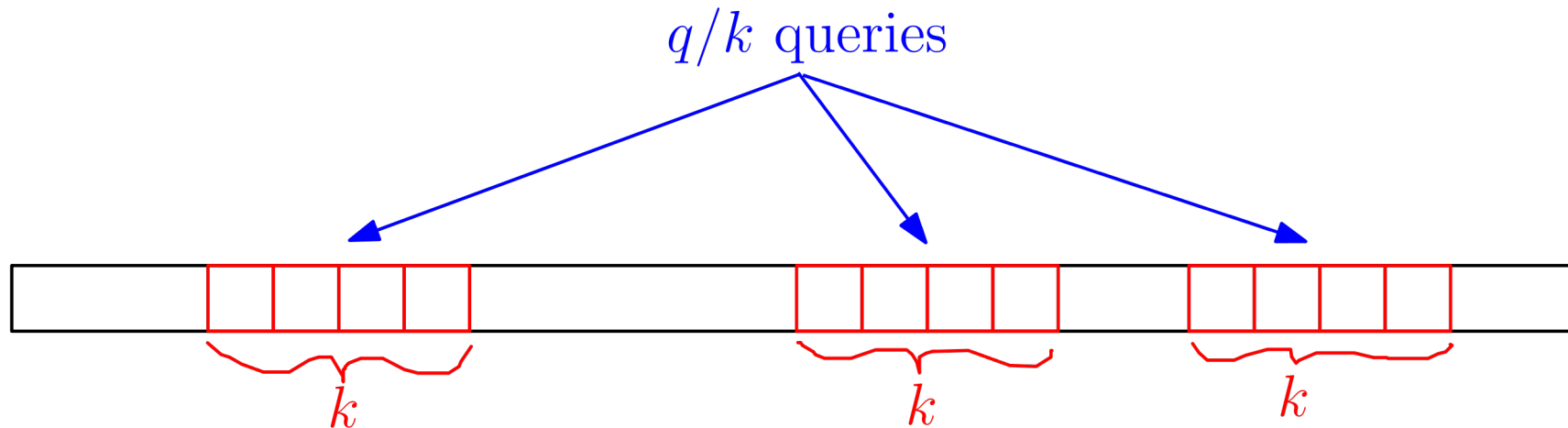
Specific Results

- Worst Case
- Insert(k) where $k \in \Delta_i$ in $O\left(\log\left(\frac{n}{|\Delta_i|}\right) + \log \log n\right)$.
 - Query in gap Δ_i resulting in gaps of size x and cx ($c \geq 1$) in $O(x \log c + \log n)$.
 - Delete(ptr) in $O(\log n)$.
 - ChangeKey(ptr, k') in $O(\log \log n)^*$.
 - Construction(S) in $O(n)$.
 - Split(r) in time as in Query.
 - Merge(T_1, T_2) where $T_1 \leq T_2$ in $O(\log n)$.
- Amortized



Applications

- BST replacement that serves n insertions and q queries in $O(n \log q + q \log n)$.
- Can serve n insertions and q/k queries for k consecutive keys in $O\left(n \log\left(\frac{q}{k}\right) + q \log n + n \log \log n\right)$.



Applications

- BST replacement that serves n insertions and q queries in $O(n \log q + q \log n)$.
- Can serve n insertions and q/k queries for k consecutive keys in $O\left(n \log \left(\frac{q}{k}\right) + q \log n + n \log \log n\right)$.
- PQ or double-ended PQ with $O(\log \log n)$ time insert and decrease-key that supports a general operation set.
- Data structure for online dynamic multiple selection.
- Incremental quicksort.
 - Return the q smallest elements in sorted order in $O(n + q \log n)$.
 - Or sort any part of the array in optimal $O(B + n)$.

Thanks!

Questions?