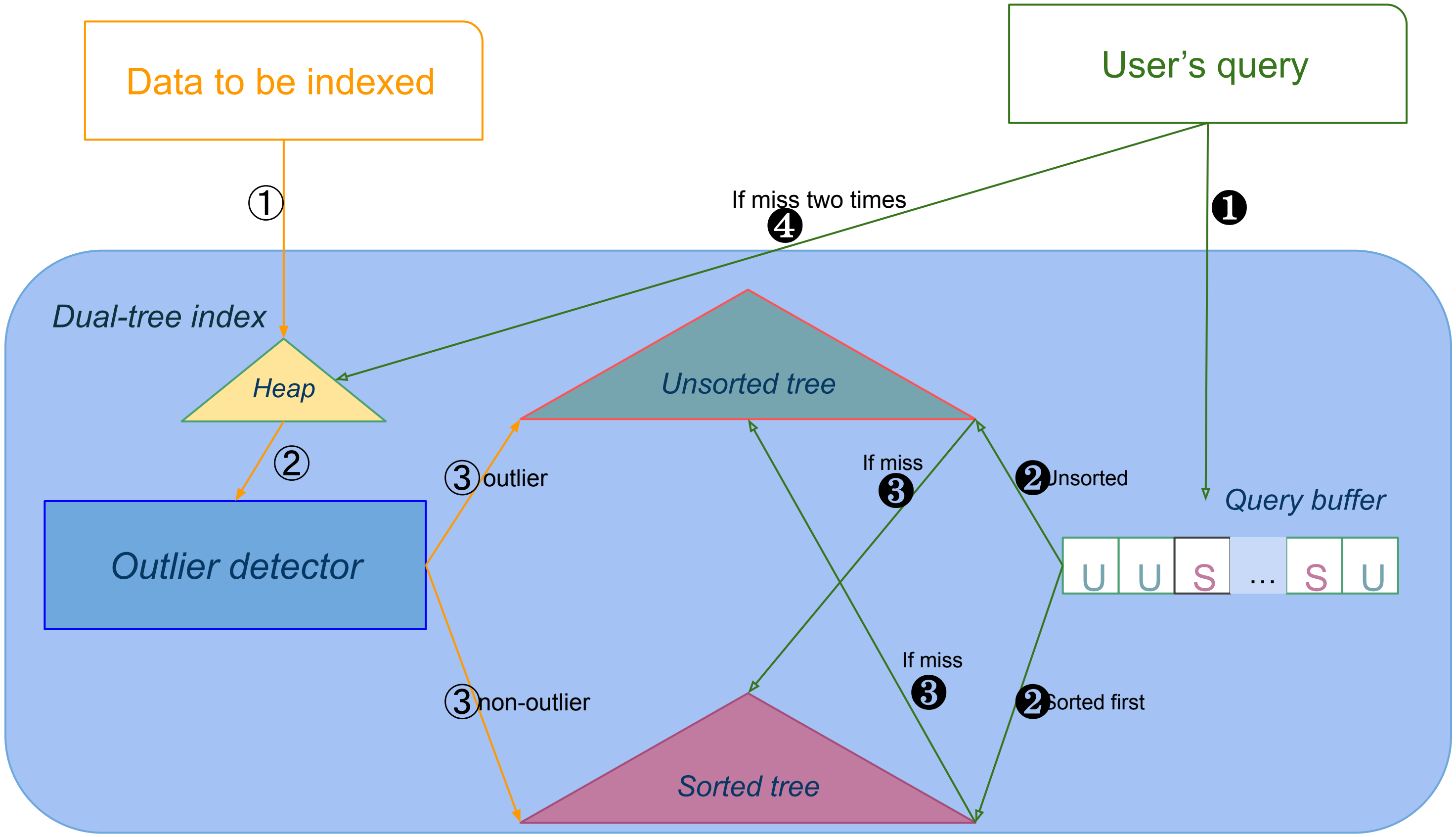
CS561: Dual B+ Tree Presentation

Lanfeng Liu, Ning Wang, Jianqi Ma

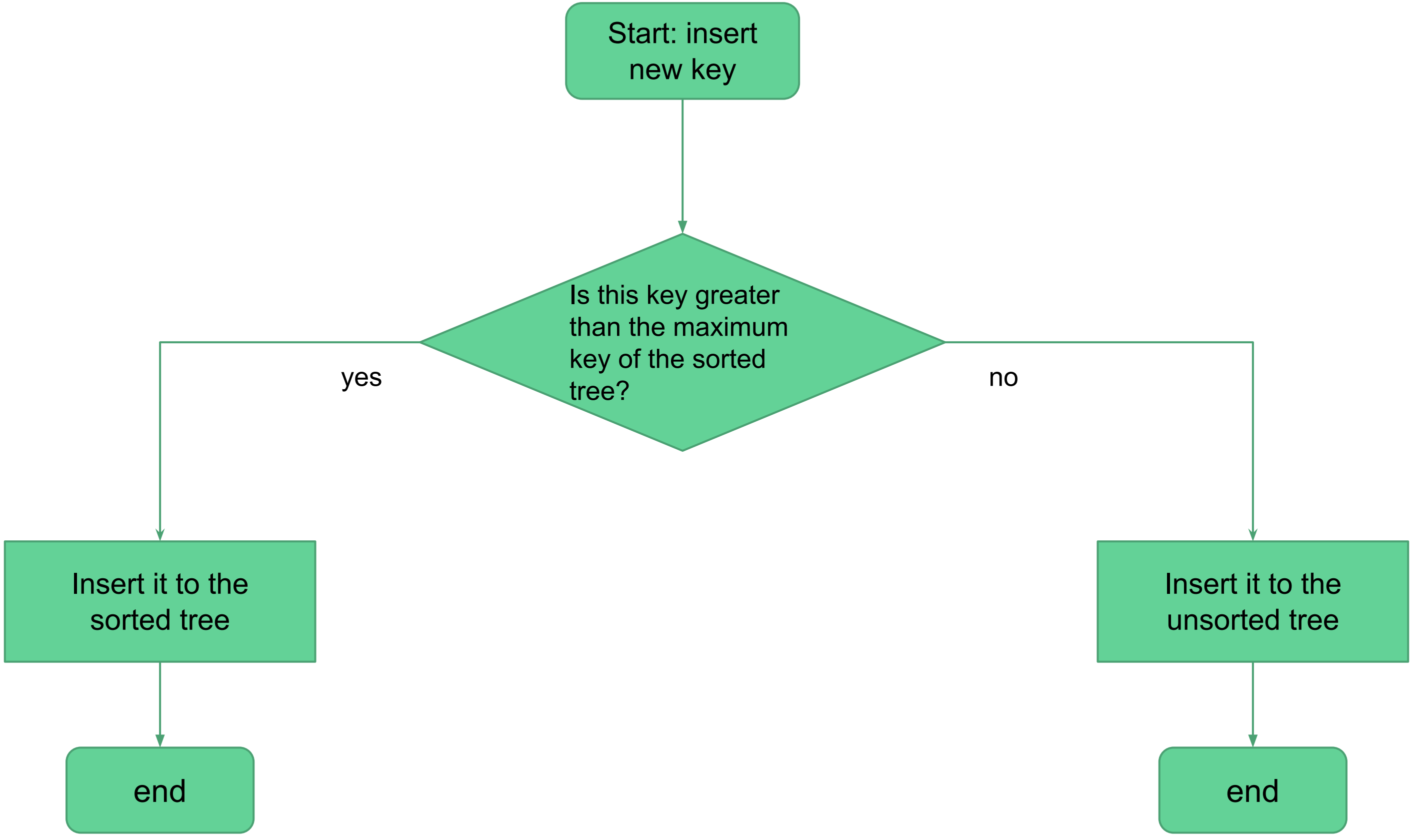
Overview

Overview of the dual-tree system

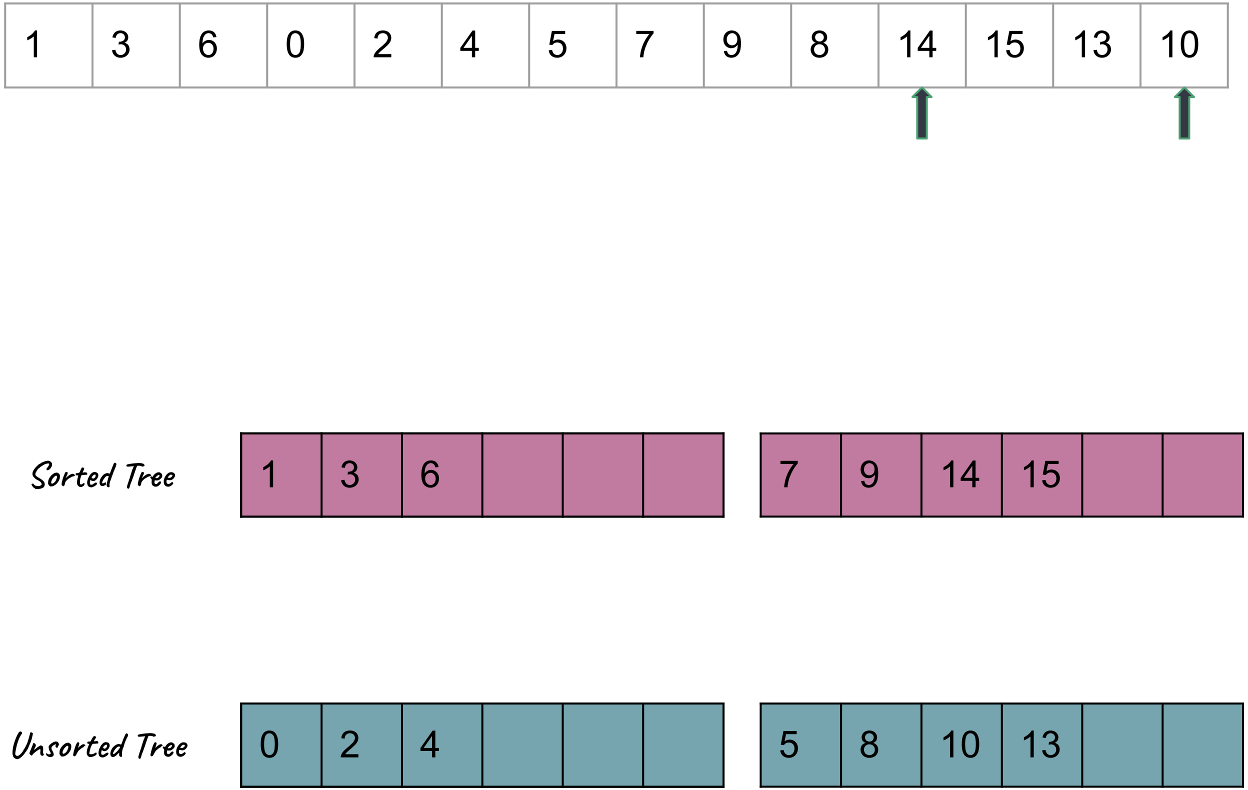


Insertion optimization

Basic insertion



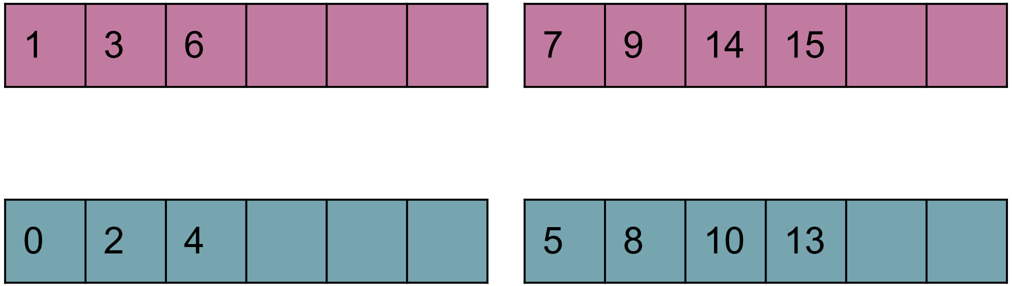
Basic insertion: example



Basic insertion: drawbacks

- The space utility is low: all nodes except the tail leaf node are at most half-full

- A big key can prevent many other keys being inserted into the sorted one.



2 simple optimizations of the insertion(2 tuning knobs)

- The space utility is low: all nodes except the tail leaf node are at most half-full

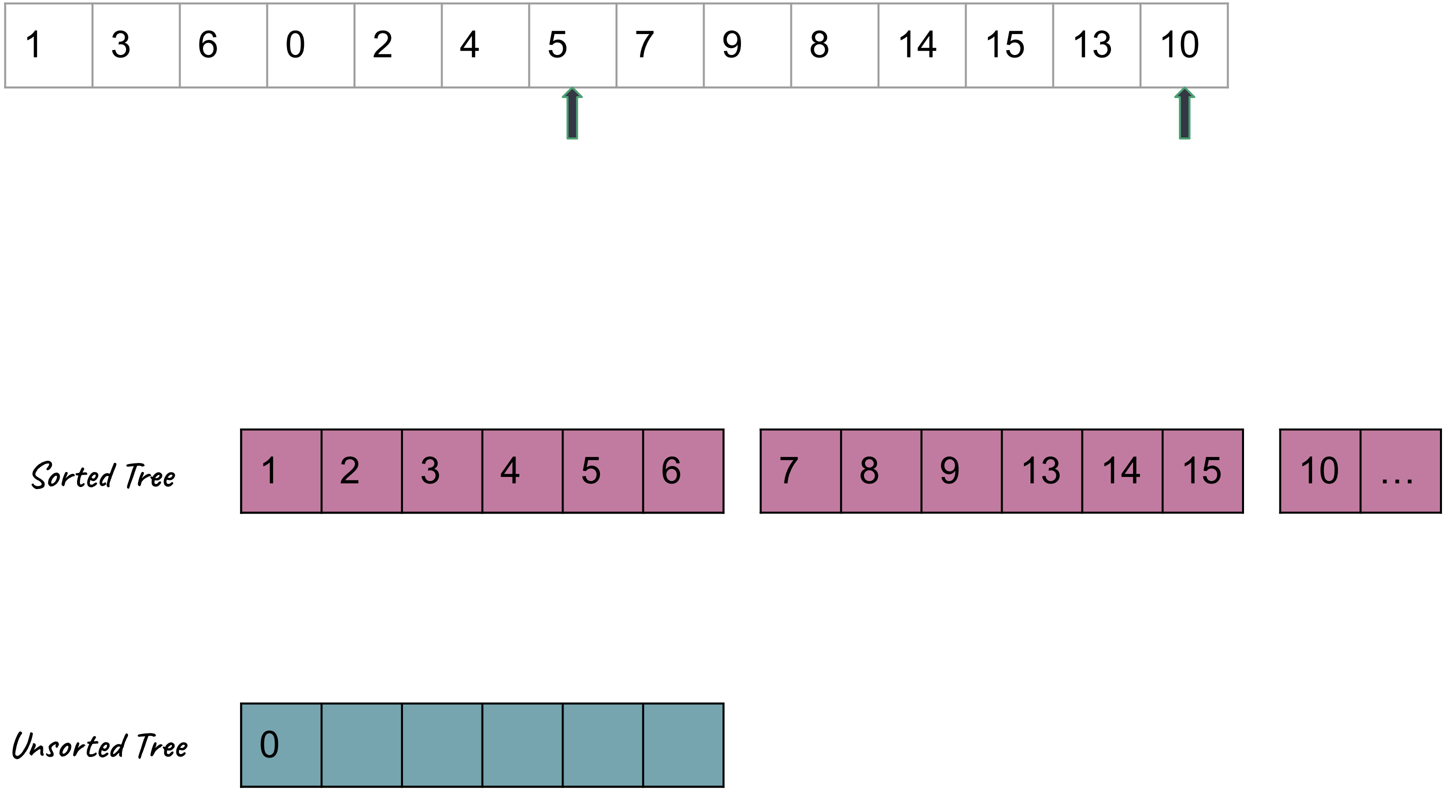
- Split nodes unevenly

- A big key can prevent many other keys inserting into the sorted one.

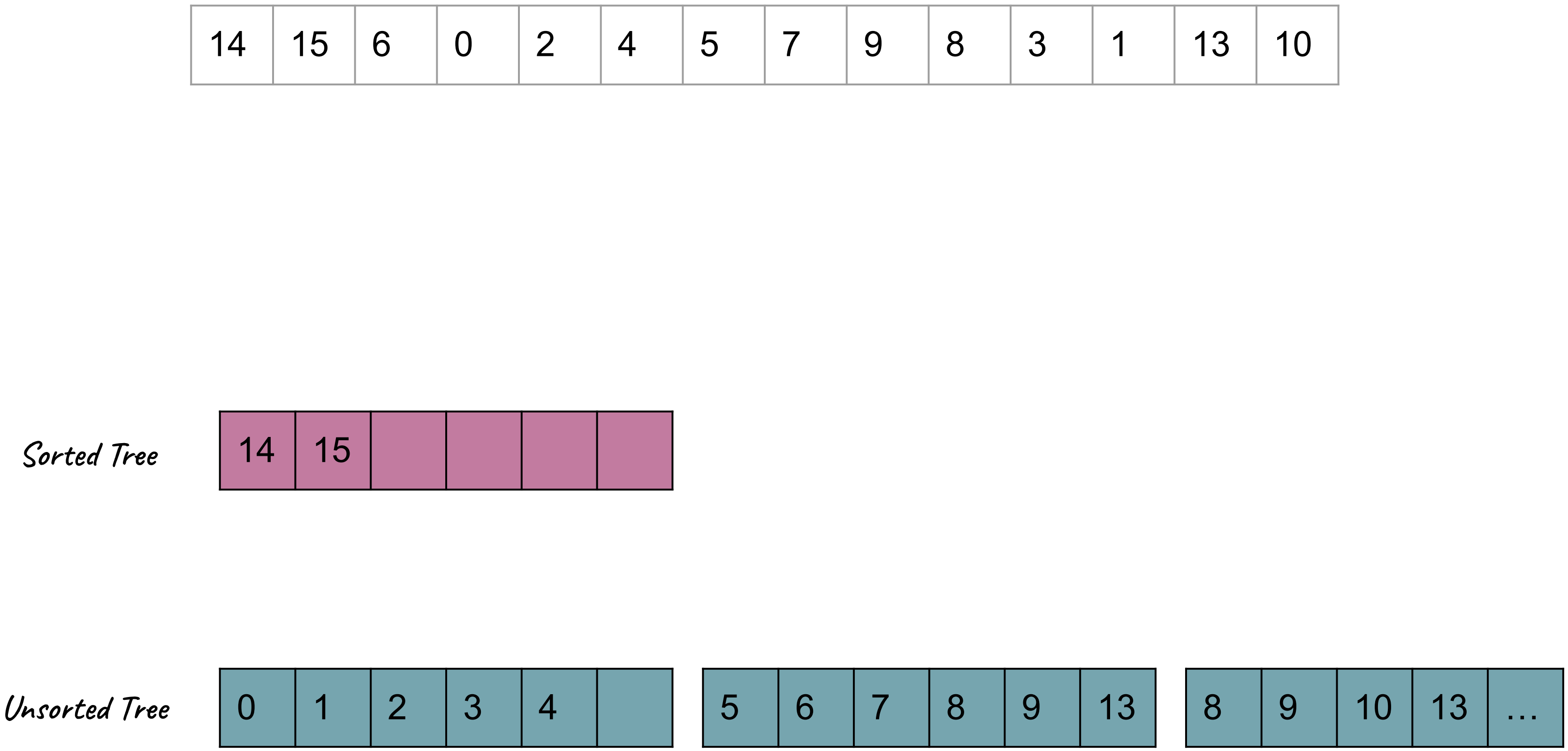
- Allow insertion to the tail leaf

Knob name	Function	Domain
SORTED_TREE_SPLIT_FRAC	Decide how many keys remain in the original node after splitting	[0.5, 1)
ALLOW_SORTED_TREE_INSERTION	Allow insertion to the tail leaf of the sorted tree.	{true, false}

Optimized insertion: example



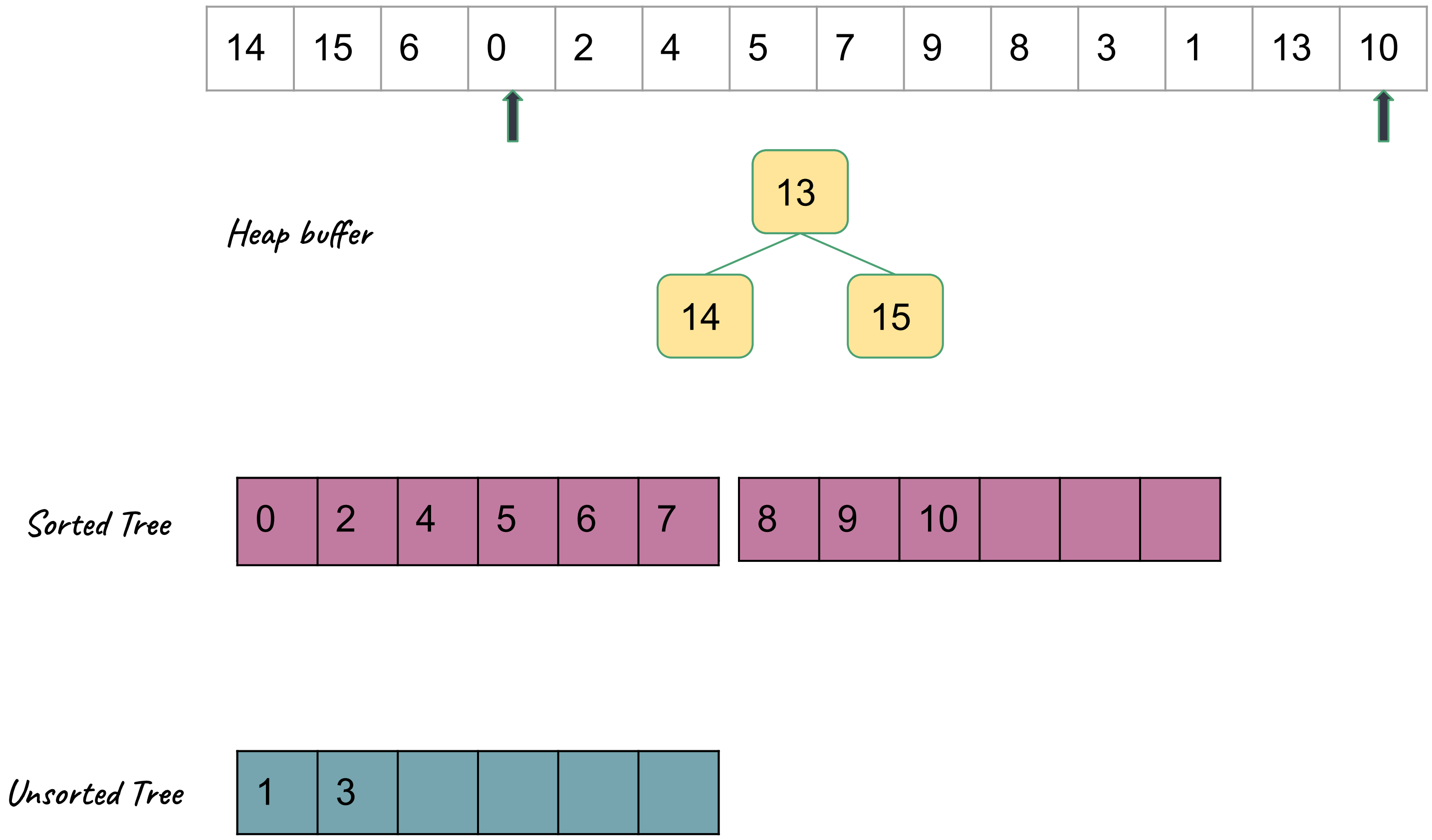
However...


14 15 6 0 2 4 5 7 9 8 3 1 13 10
Sorted Tree
14 15
Unsorted Tree
0 1 2 3 4
5 6 7 8 9 13
8 9 10 13 …

Insertion with a heap buffer(1 tuning knob)

Knob name	Function	Domain
HEAP_SIZE	Define the size of the heap buffer, 0 means no heap buffer is used.	Non-negative integers

Insertion with a minimum heap buffer

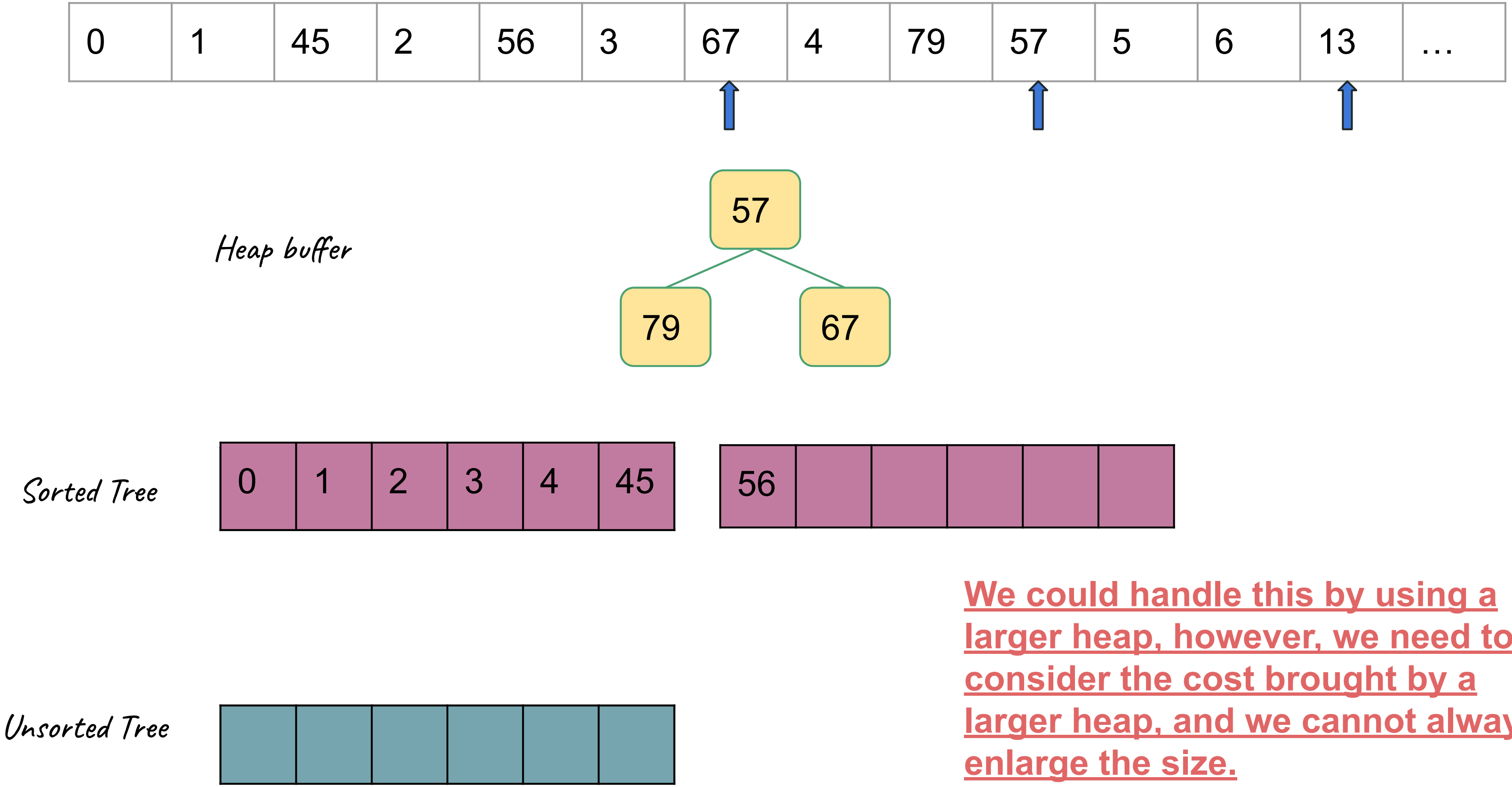


Insertion with a heap buffer(1 tuning knob)

Knob name	Function	Domain
HEAP_SIZE	Define the size of the heap buffer, 0 means no heap buffer is used.	Non-negative integers

The size of heap buffer should not be too large, because the cost of maintaining a heap buffer is non-negligible.

However, again...



We could handle this by using a larger heap, however, we need to consider the cost brought by a larger heap, and we cannot always enlarge the size.

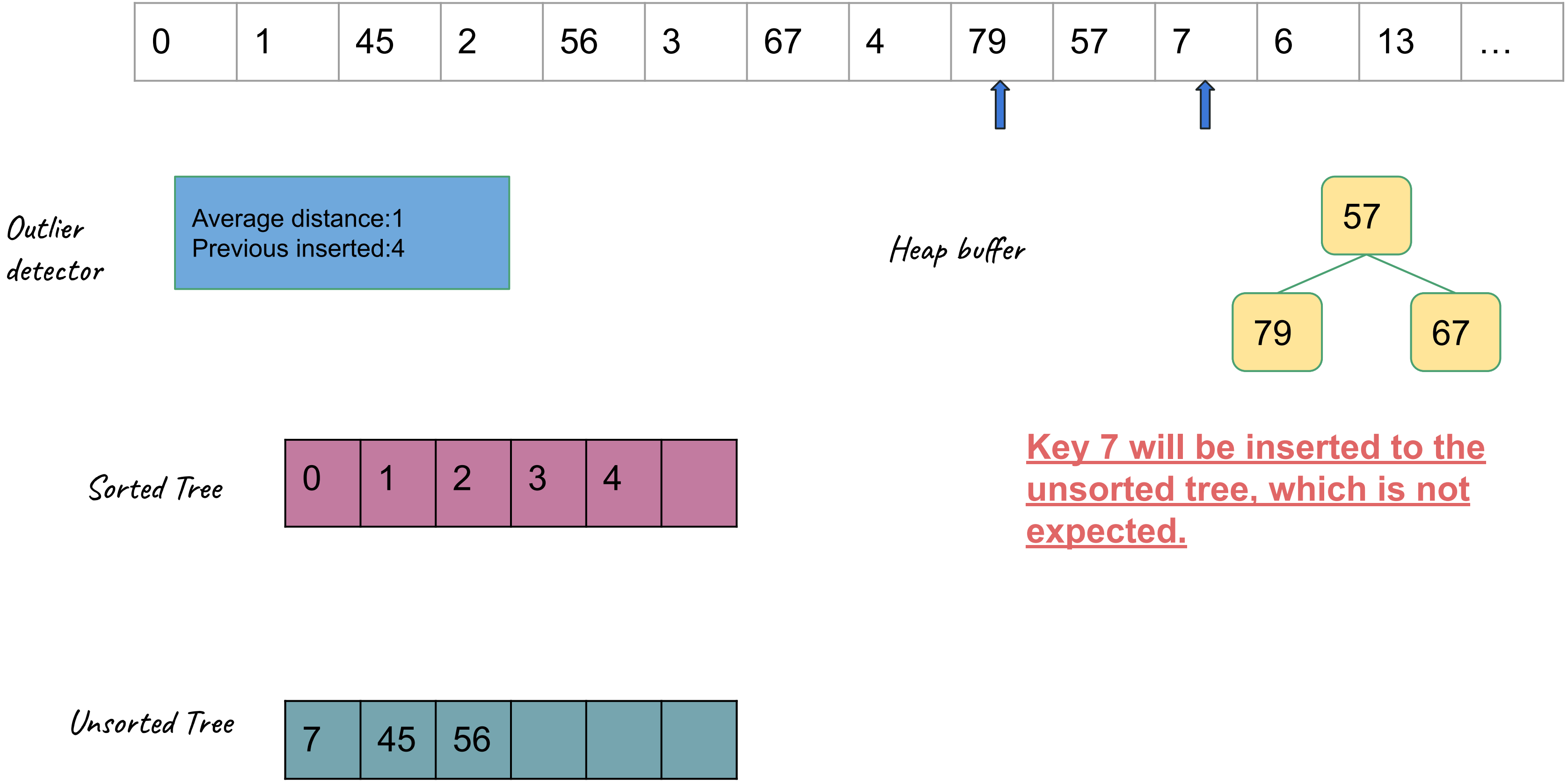
Insertion with the outlier detector

- Metric: The average distance between every two consecutive keys of the sorted tree

- How to use the metric?
    - The easiest way is to compare the average distance(dist_avg) with the distance between a new key and the maximum key of the sorted tree(dist_new). If dist_avg is greater or equal to dist_new, then insert the new key into the sorted tree.

$$\text{dist\_new} \leq \text{dist\_avg}$$

Insertion + heap buffer + outlier detector(easiest)



Key 7 will be inserted to the unsorted tree, which is not expected.

Insertion with the outlier detector

- Metric: The average distance between every two consecutive keys of the sorted tree

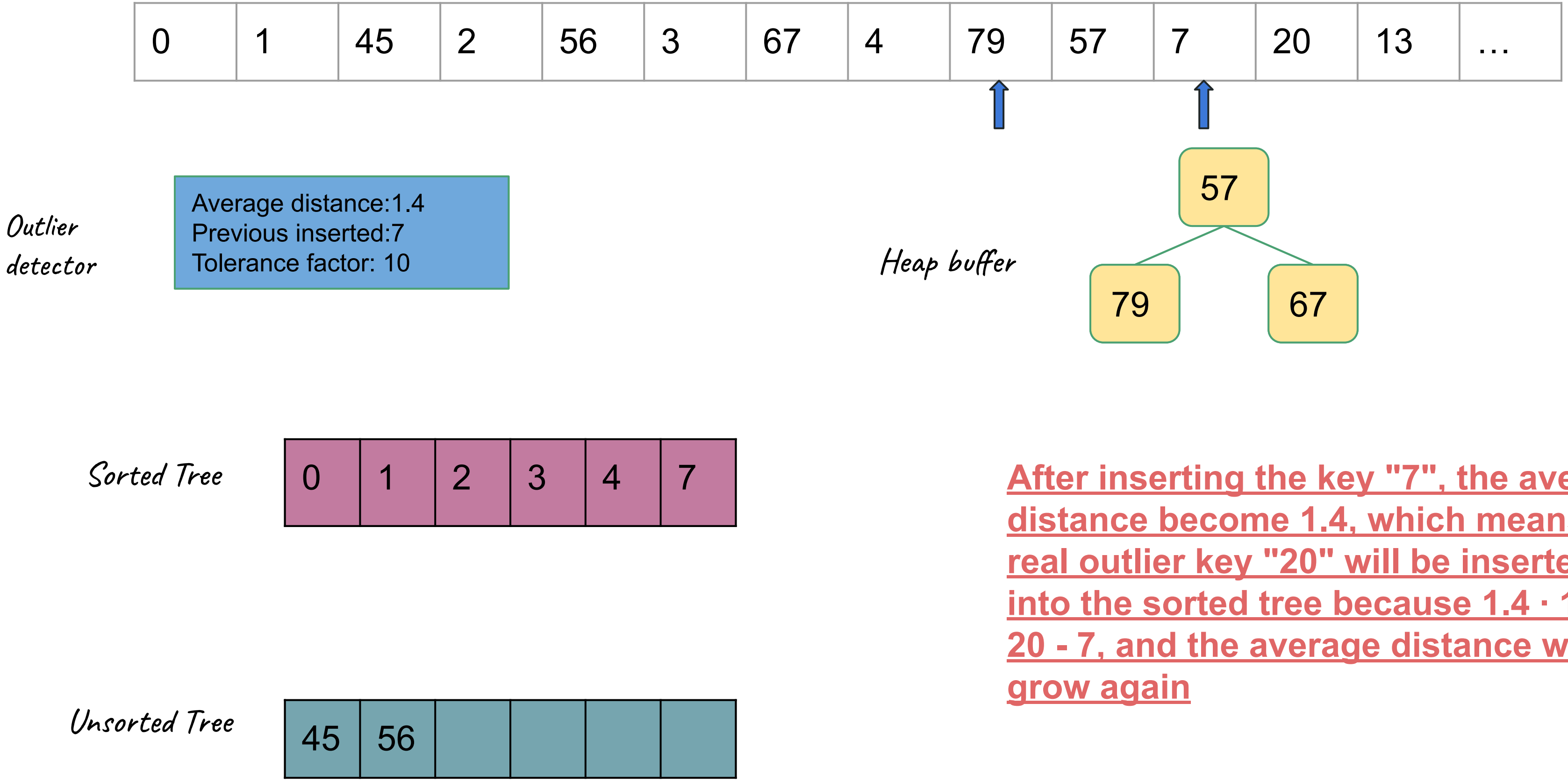
- How to use the metric?
  - <u>The easiest way</u> is to compare the average distance(dist_avg) with the distance between a new key and the maximum key of the sorted tree(dist_new). If dist_avg is greater or equal to dist_new, then insert the new key into the sorted tree.

    $$\textbf{dist\_new} \leq \textbf{dist\_avg}$$

  - <u>Tolerate "small" gaps</u> between every two tuples using a tolerance_factor.

    $$\textbf{dist\_new} \leq \textbf{dist\_avg} \cdot \textbf{tolerance\_factor}$$

Insertion + heap buffer + outlier detector(fixed tolerance)



After inserting the key "7", the average distance become 1.4, which means the real outlier key "20" will be inserted into the sorted tree because 1.4 · 10 > 20 - 7, and the average distance will grow again

Insertion with the outlier detector

- Metric: The average distance between every two consecutive keys of the sorted tree

- How to use the metric?
  - The easiest way is to compare the average distance(dist_avg) with the distance between a new key and the maximum key of the sorted tree(dist_new). If dist_avg is greater or equal to dist_new, then insert the new key into the sorted tree.

$$\text{dist\_new} \leq \text{dist\_avg}$$

  - Tolerate "small" gaps between every two tuples using a tolerance_factor.

$$\text{dist\_new} \leq \text{dist\_avg} \cdot \text{tolerance\_factor}$$

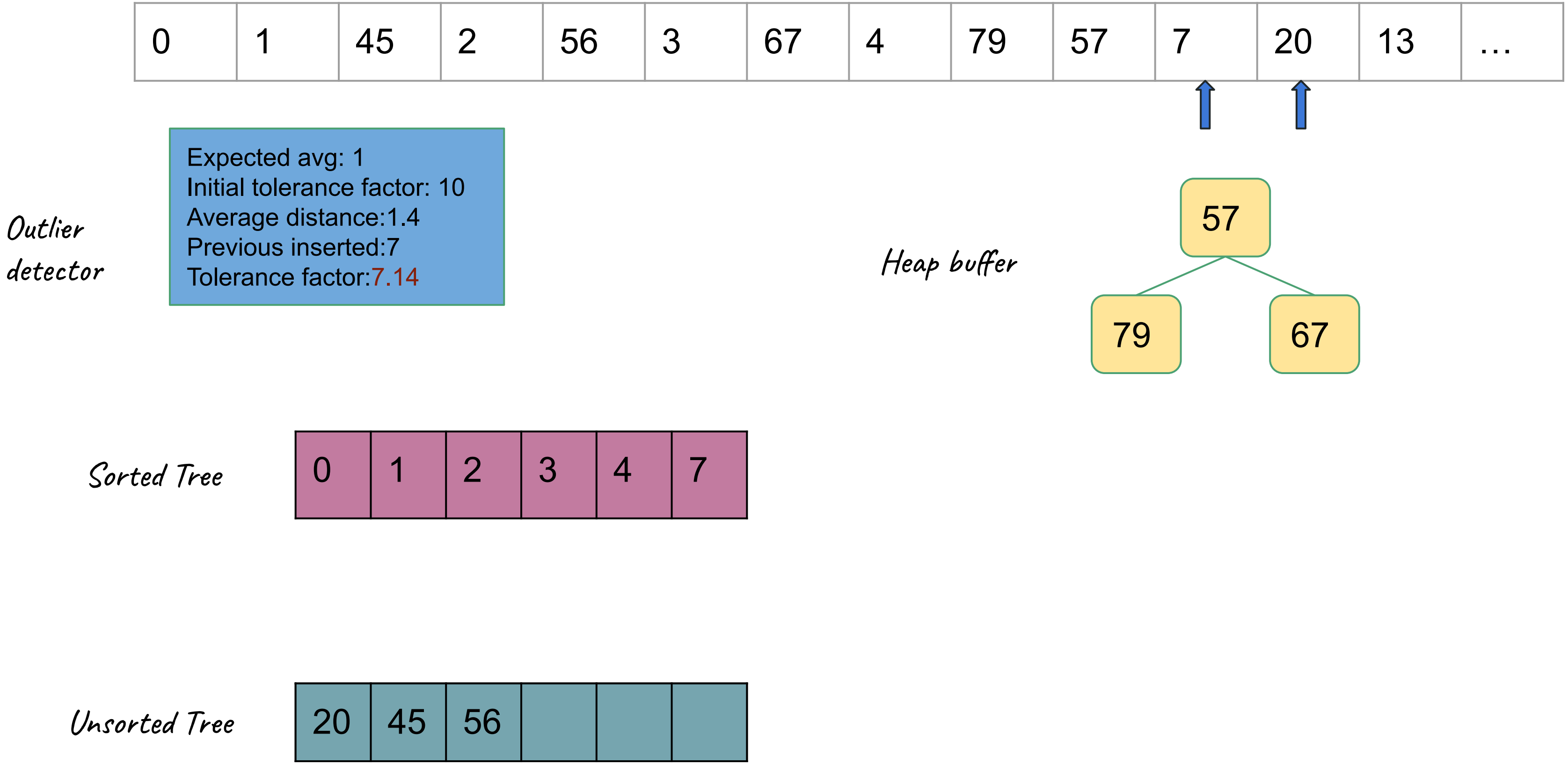
- Update tolerance_factor during the process according to a expected average distance.

$$\text{dist\_new} \leq \text{dist\_avg} \cdot \text{tolerance\_factor}$$

$$+$$

$$\text{tolerance\_factor}' = \text{tolerance\_factor} \cdot \frac{\text{expected\_avg\_distance}}{\text{avg\_distance}}$$

Insertion + heap buffer + outlier detector(elastic tolerance)

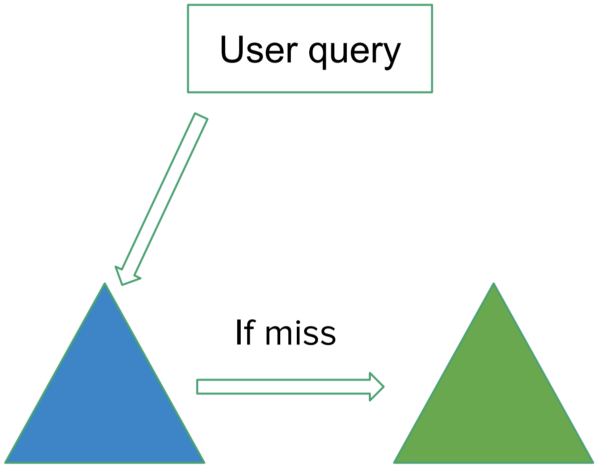


Insertion with outlier detector(2 tuning knob)

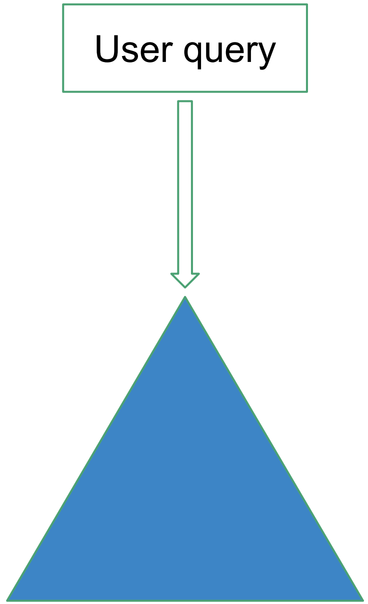
Knob name	Function	Domain
INIT_TOLERANCE_FACTOR	Define initial tolerance factor. If it is 0, then the outlier detector is disabled.	Float numbers greater than 0.
EXPECTED_AVG_DISTANCE	The expectation of the average distance of the sorted tree. If it is less or equal to 1, the tolerance factor is fixed.	Float numbers greater than 1.

Query optimization

Basic Query



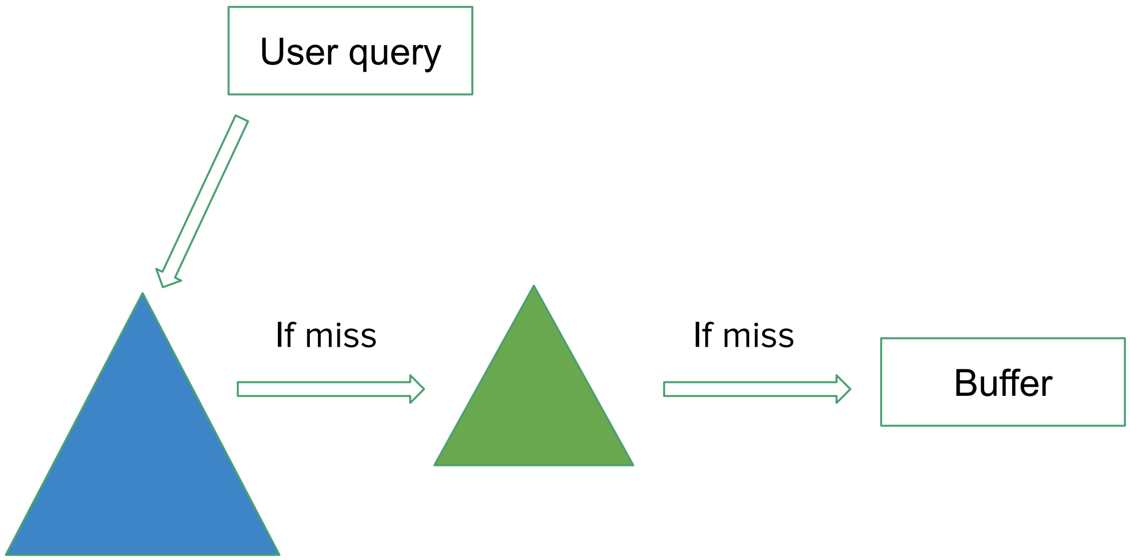
Dual B+ tree



B+ tree

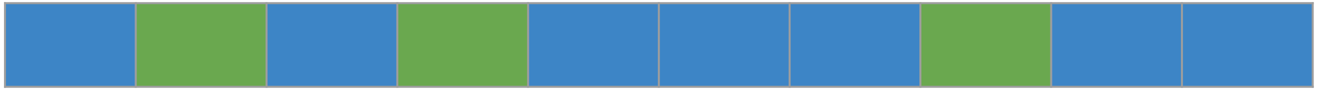
Simple Query Optimization

- Query larger tree first



MRU (most recently used) query

- Keep a buffer for the results of past n queries
- First search the tree that's been queried the most frequently



- A new query comes, search blue tree first, update buffer

Experiment

Sortedness Representation

- k: noise percentage
- l: window size

0	1	2	3	4	5	6	7	8	9

K = 0%, l = 0%

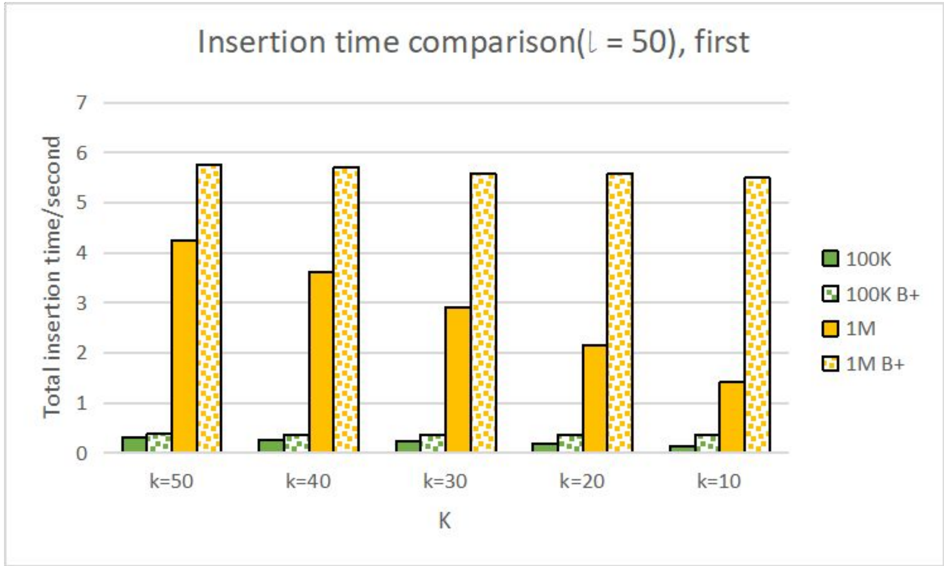
0	1	5	3	4	2	6	7	8	9

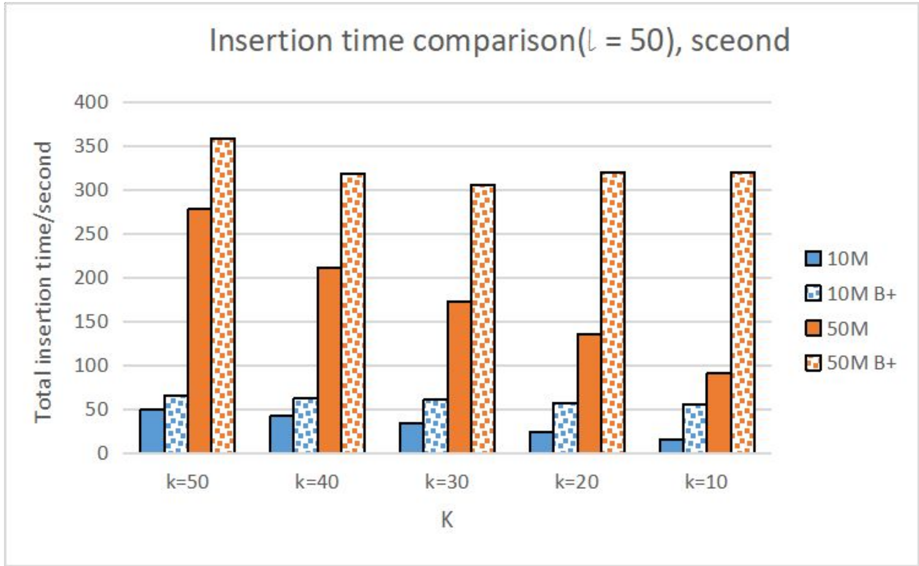
K = 20%, l = 30%

Insertion Benchmark

- Baseline: single B+ tree
- Data size: 100K, 1M, 10M, 50M
- Dual B+ tree tuning knobs:
  - Sorted tree split fraction = 0.9
  - Unsorted tree split fraction = 0.5
  - Heap buffer size = 16
  - Initial outlier tolerance factor = 100
  - Minimum outlier tolerance factor = 20
  - Expected average distance = 2.5
  - Allow sorted tree insertion = 1
  - Query Buffer Size = 20

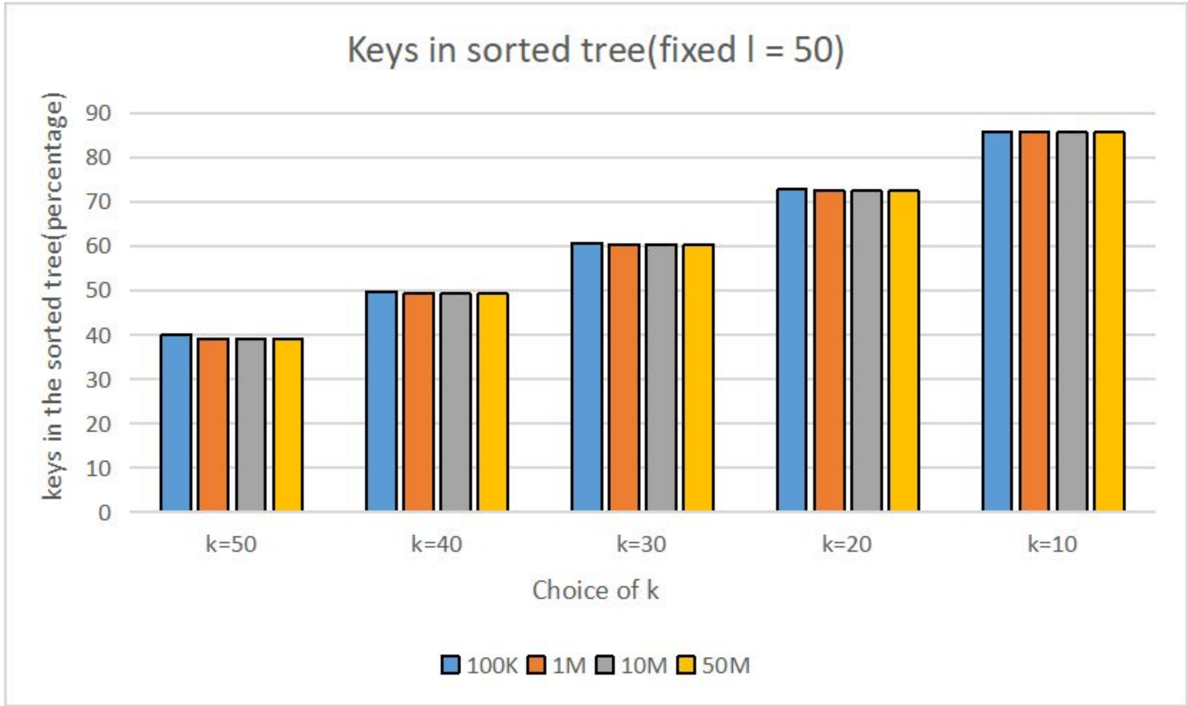
Insertion benchmark: comparison with single B+- tree





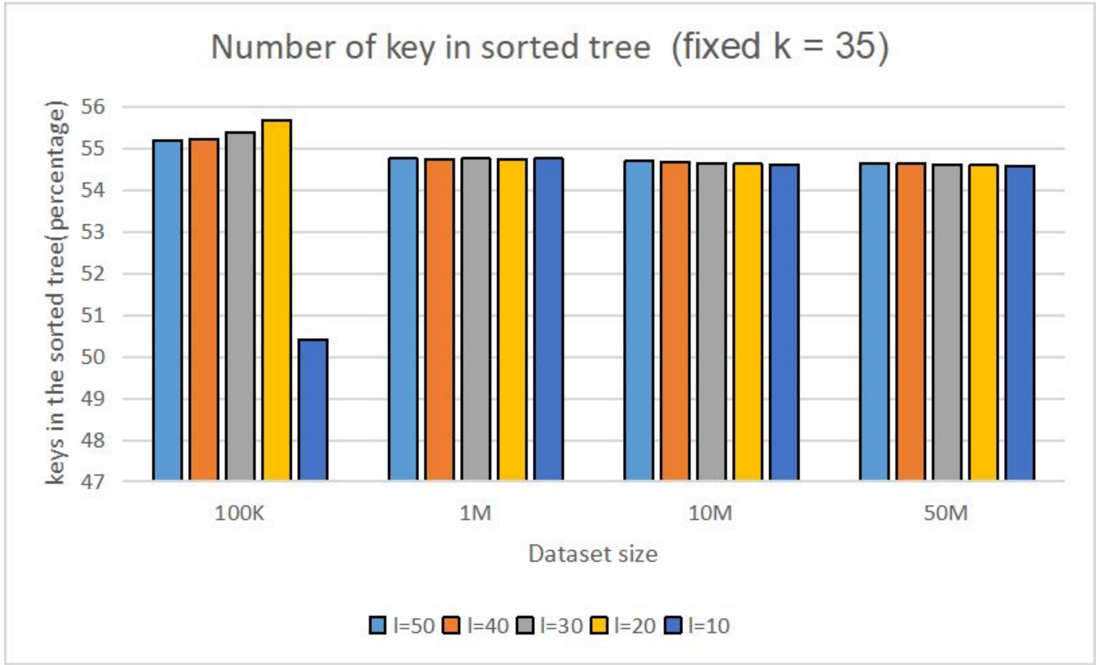
- The insertion performance of the dual-tree system completely outperforms that of single B+- tree.
- Our dual-tree system do make good use of the sortedness in the dataset.

Insertion benchmark: Number of keys in the sorted tree with different K



- As the value of k decrease, the number of keys in the sorted tree increases.
- Even though k is 50(half of the keys are out of order), the sorted tree still contains almost 40% of all keys.

Insertion benchmark: Number of keys in the sorted tree with different L

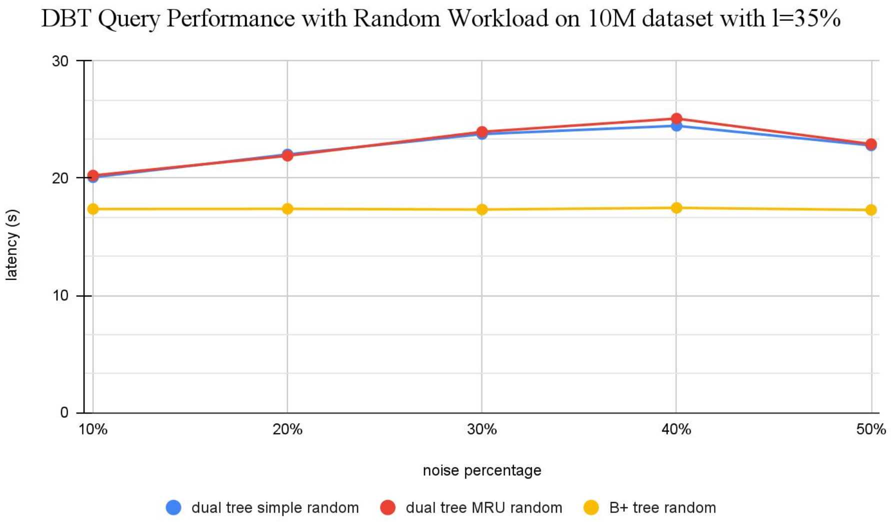


- The change of the value of l hardly influence the performance .
- However there is an immediate drop when dataset size is 100K(l = 10). The possible reason is that the initial tolerance factor is too large.

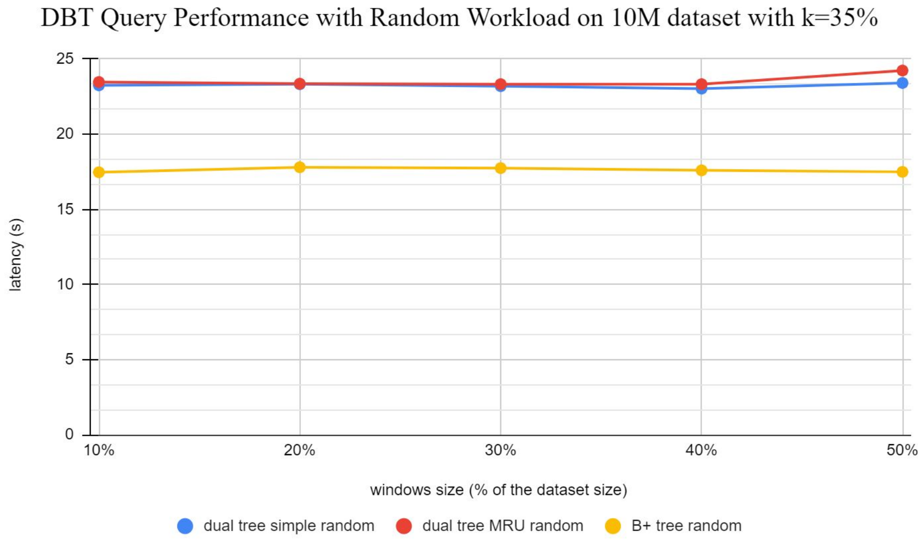
Query Benchmark

- Baseline: single B+ tree
- Data size: 100k, 1M, 10M, 50M
- Query workload: random, sequential
- Metric: cumulative query response time
- Dual B+ tree tuning knobs: same as query benchmark

Query Benchmark - increasing noise percentage



Query Benchmark - increasing window size



Conclusion

- Dual B+ tree
  - Sorted tree: insert in order elements
  - Unsorted tree: insert out of order elements
- Insert optimization
  - Heap buffer
  - Outlier detection
- Query optimization
  - MRU buffer
- Future work
  - Parallel query
  - Individual insertion and query time
  - Query Experiment on other type of workloads