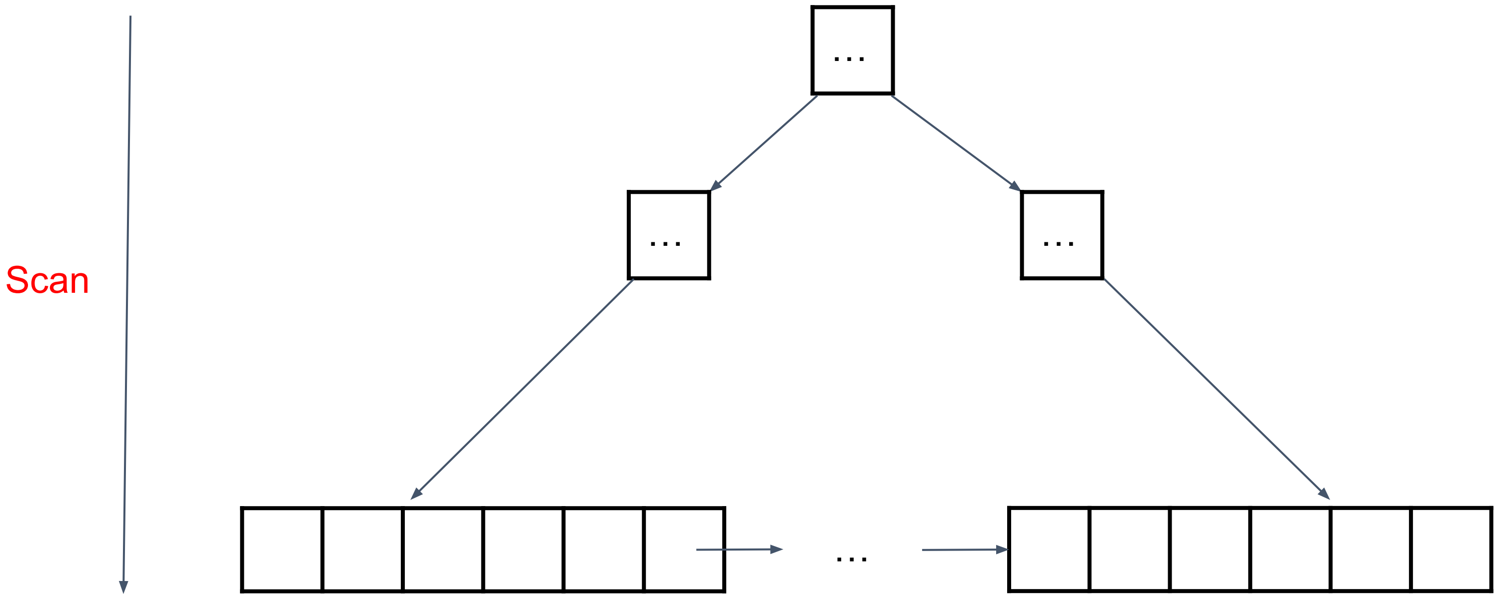
Benchmarking Dual B+ Tree

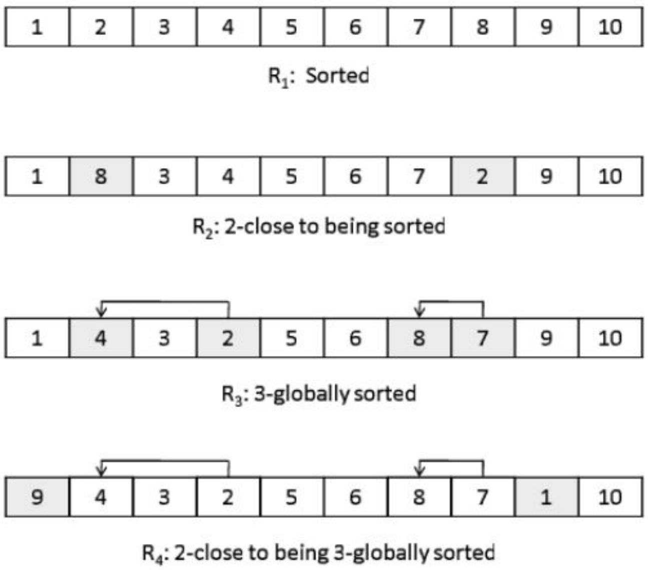
Jingyi Huang, Shaolin Xie, Meng-Heng Lee

Boston University College of Arts and Sciences

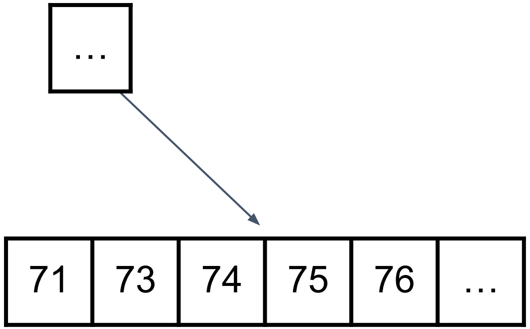
Problem Statement



Nearly-Sorted Workload

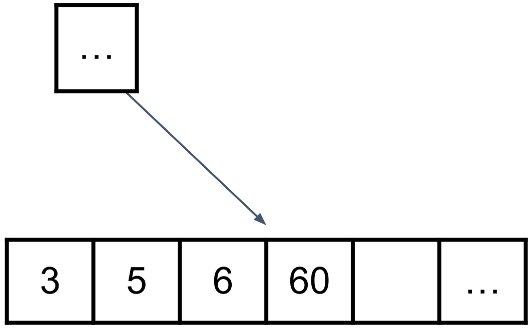


Min Max Dual B+ Tree



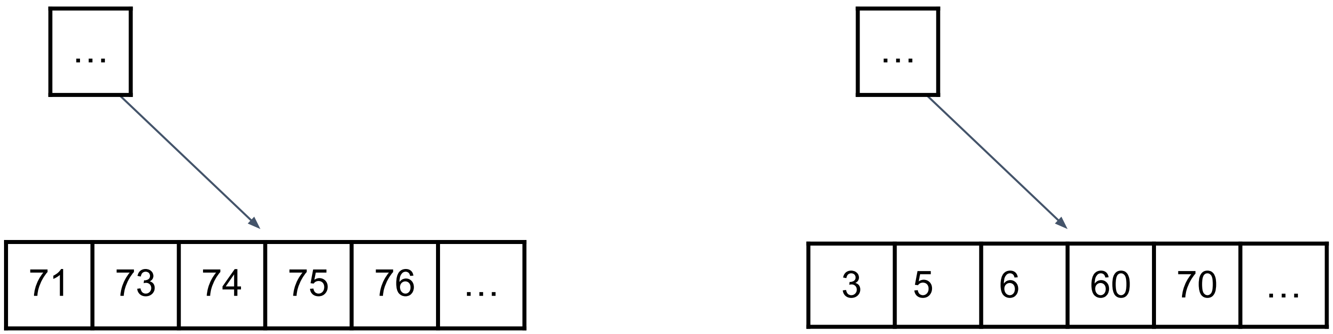
In-order Tree

- insert_by_max()
- insert_by_min()



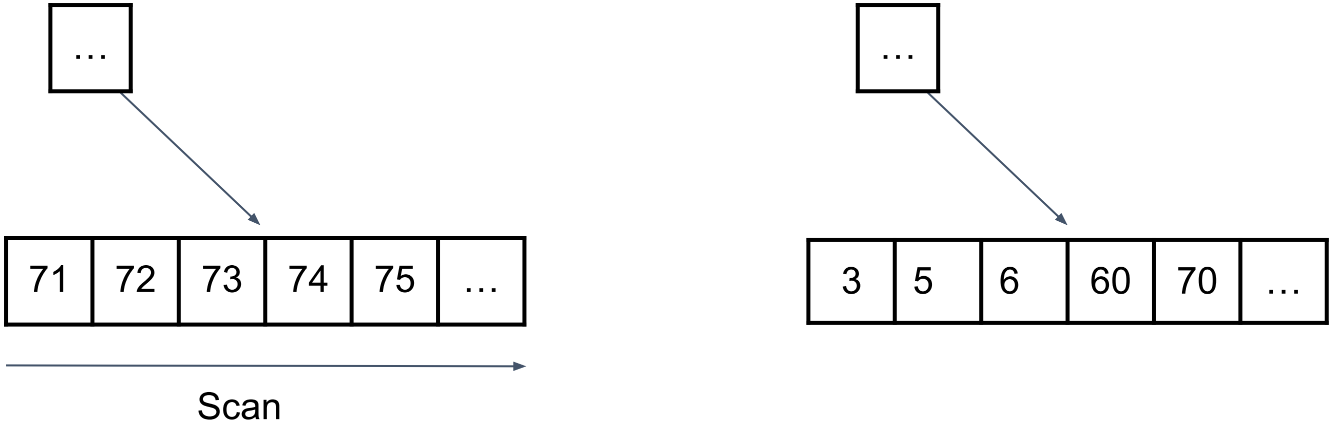
Out of order Tree

Min Max Dual B+ Tree

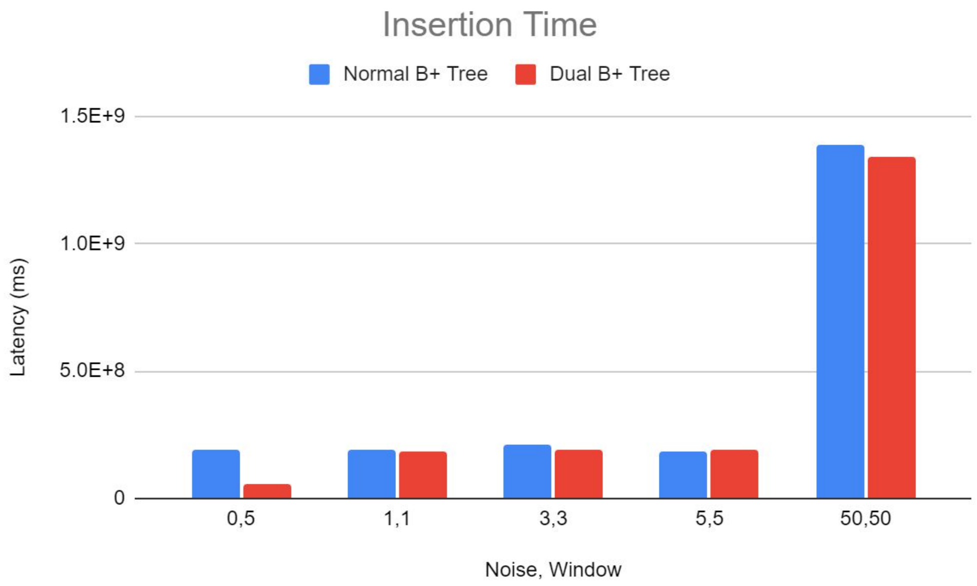


Insertion: 72

Min Max Dual B+ Tree



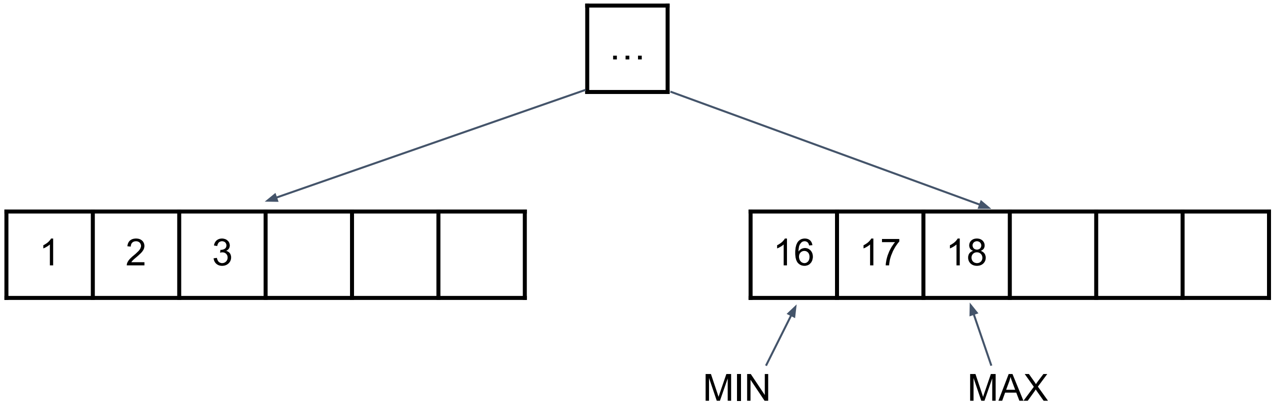
Insertion: 72


Insertion Time
Normal B+ Tree
Dual B+ Tree
1.5E+9
1.0E+9
5.0E+8
0
Latency (ms)
0,5
1,1
3,3
5,5
50,50
Noise, Window

Noise, Window	Sorted Tree Size	Out of Order Tree Size
0,1	100000000	0
1,1	128767	9871233
3,3	98782	9901218
5,5	89818	9910182
50,50	51901	9948099

Case

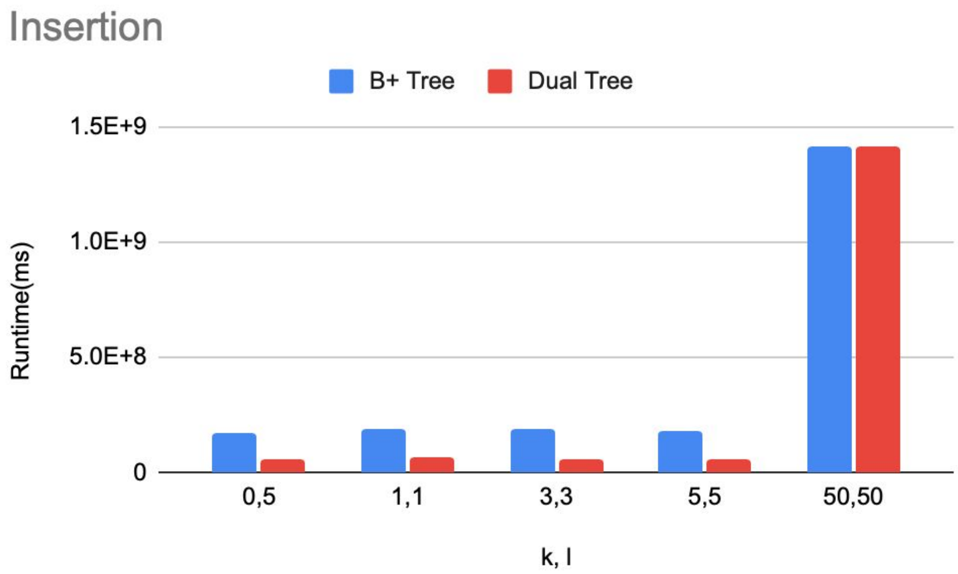
Insertion: 1, 2, 3, 16, 17, 18, 4, 5, 6, 7



Outlier Detection: Standard Deviation

Bound = tree_max + stddev.

Outlier Detection Insertion Time



Outlier Detection Keys Distribution

Noise, Window	Sorted Tree	Sorted Percent
0,5	10000000	100%
1,1	9851345	98.5%
3,3	9556201	95.5%
5,5	9265949	92.6%
50,50	1	0.00001%

Dual B+ Tree Insertion: Outlier Removal

Algorithm 1 Dual B+ Tree Insertion

if $K \geq tail\_max$ **then**
  sorted.insertToTailEnd(K, V)
else if $K \geq tail\_min$ **then**
  outlier $\leftarrow$ sorted.replaceOutlier(K, V)
  unsorted.insert(outlier)
else if $K < tail\_min$ **then**
  unsorted.insert(K)
end if

K: key to insert

tail_max: maximum (last) key of tail node in sorted tree

tail_min: minimum (first) key of tail node in sorted tree

Outlier Replacement

Algorithm 2 Lazy Move Strategy

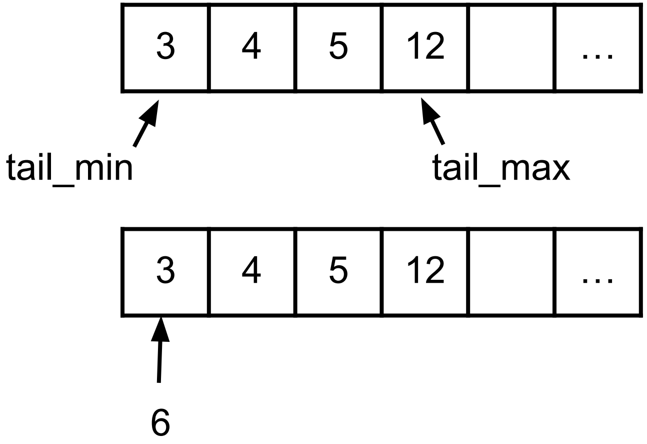
Require: K

```
i = 0
data = sorted.tail.data
while data[i] < K do
    i ← i + 1
end while
outlier ← data[i]
data[i] ← K
return outlier
```

K: key to insert

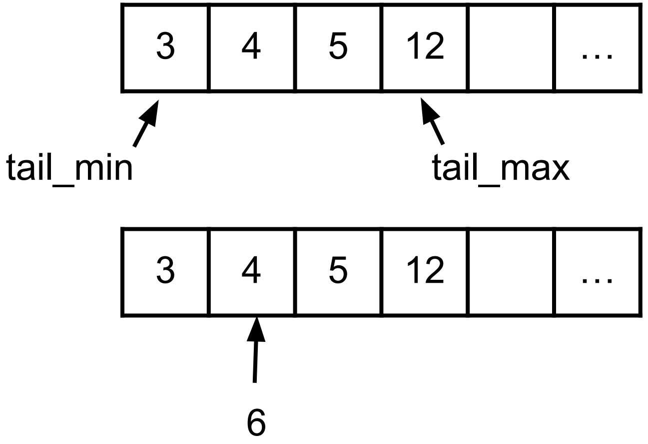
Lazy Move Strategy

Insert key: 6



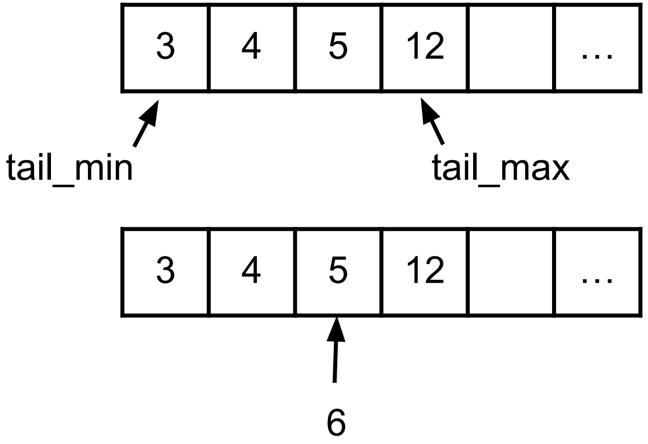
Lazy Move Strategy

Insert key: 6



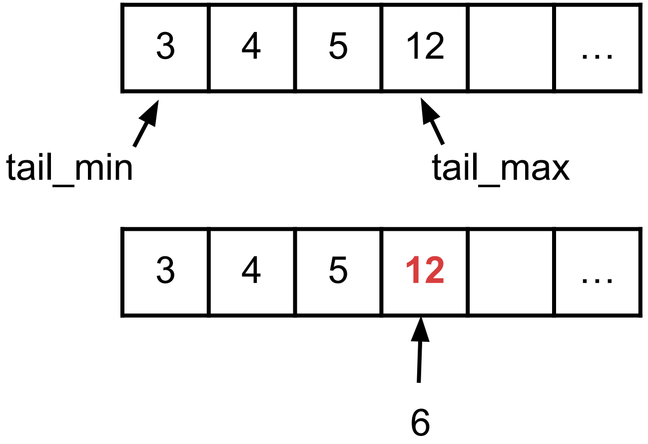
Lazy Move Strategy

Insert key: 6



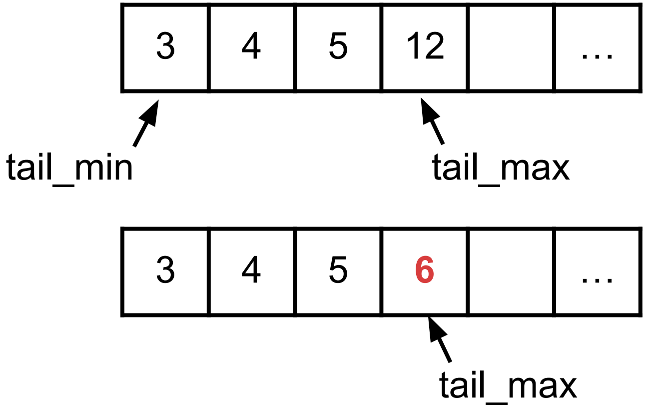
Lazy Move Strategy

Insert key: 6



Lazy Move Strategy

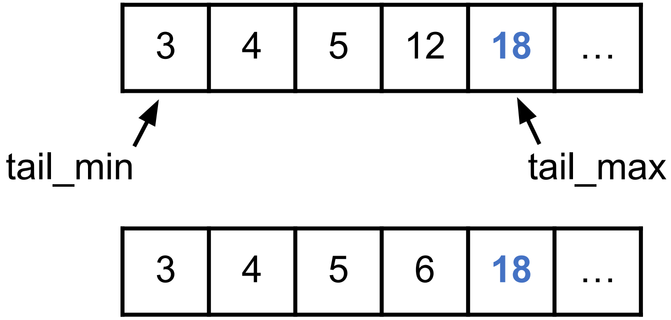
Insert key: 6



Get outlier: 12
Insert 12 to out-of-order tree

Lazy Move Strategy

Insert key: 6

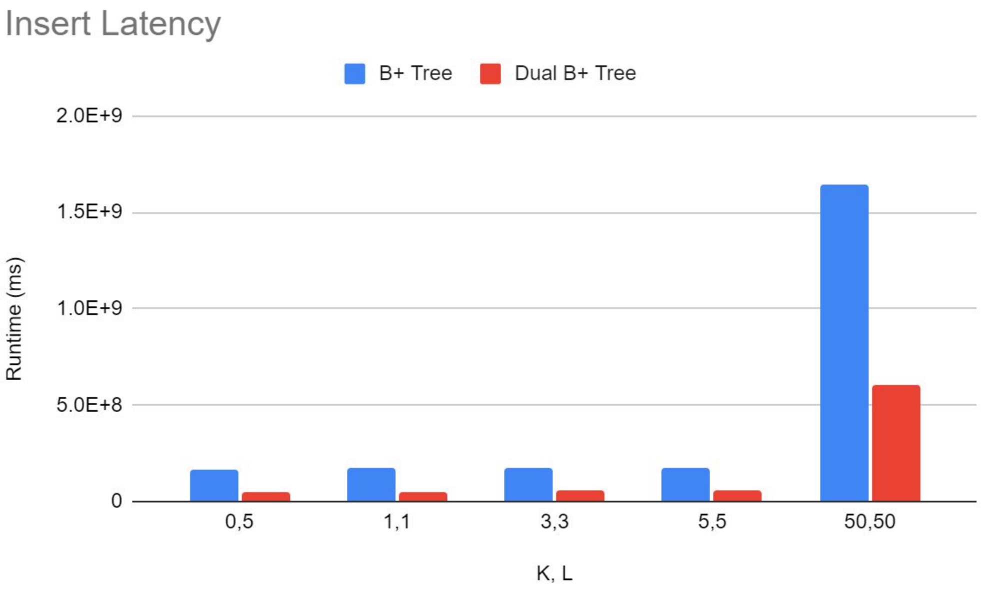


Get outlier: 12
Insert 12 to out-of-order tree

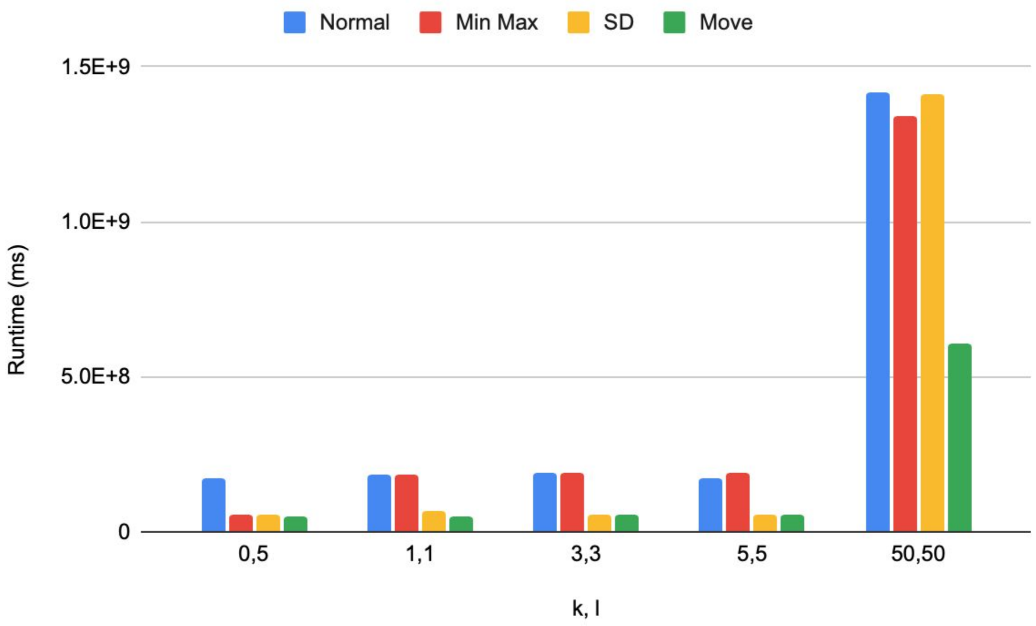
Evaluation: Lazy Move

Noise, Window	Sorted Tree	Sorted Percent
0,5	10000000	100%
1,1	9851028	98.5%
3,3	9555890	95.6%
5,5	9265637	92.7%
50,50	3910799	39.1%

Evaluation: Lazy Move



Overall Evaluation



Reflections

Research

↓

Design algorithms

↓

Design experiments

↓

Analyze results

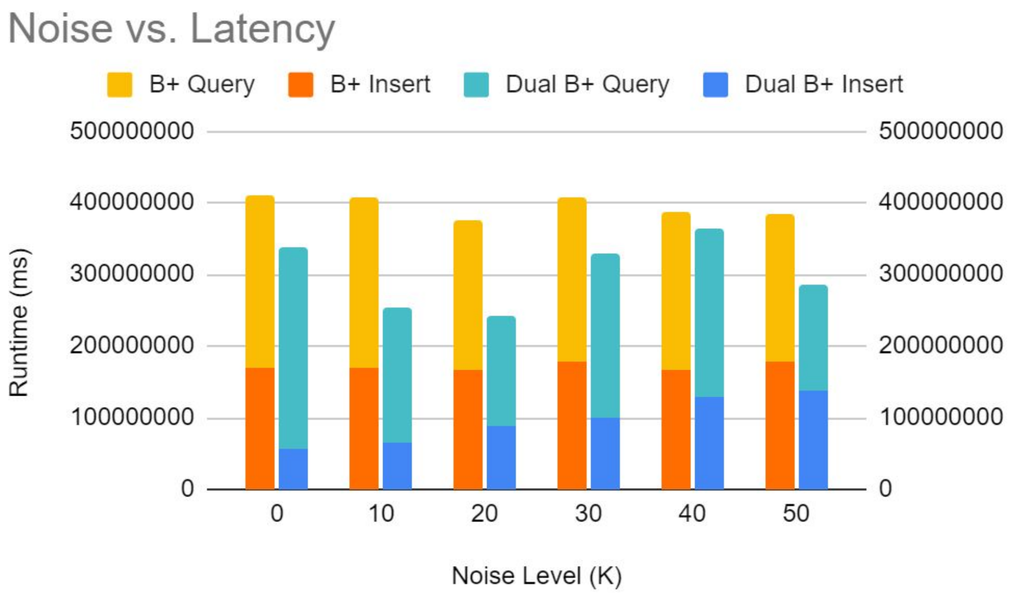
Challenges:

- Modify insertion method
- Designing algorithms

Advice:

- Choose one path to build on
- Talk to mentor!

Future Work


Noise vs. Latency
B+ Query
B+ Insert
Dual B+ Query
Dual B+ Insert
Runtime (ms)
500000000
400000000
300000000
200000000
100000000
0
0
10
20
30
40
50
Noise Level (K)

Thank you!