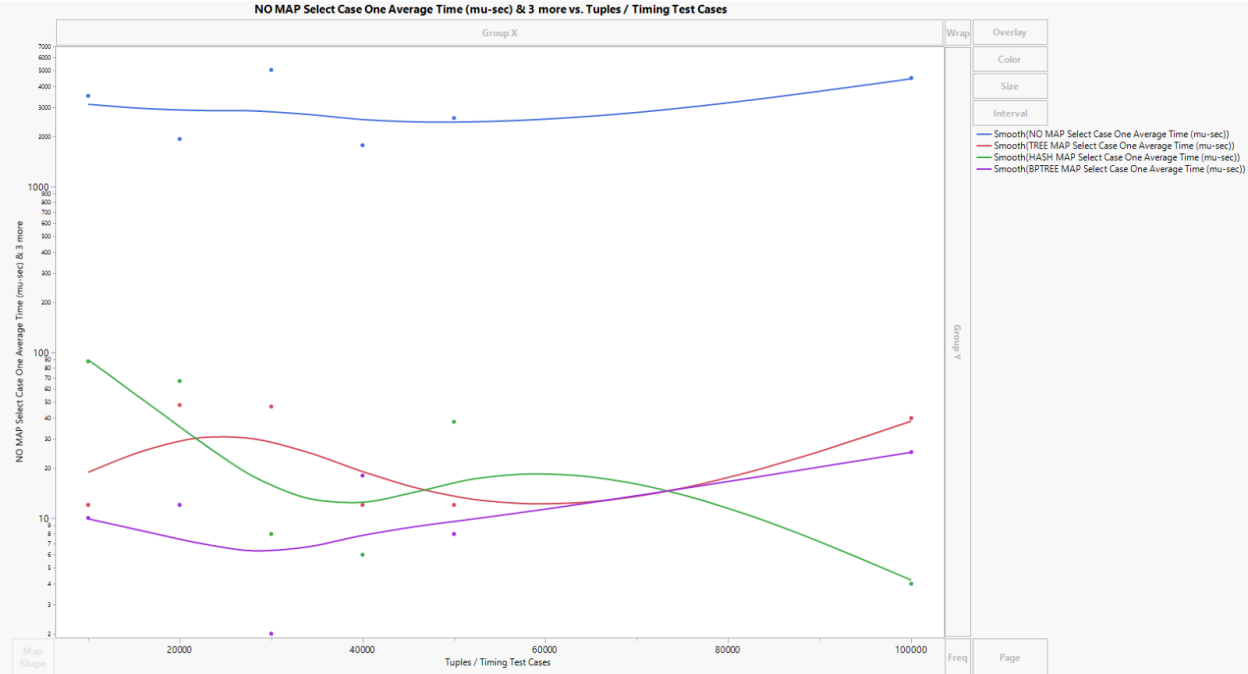
Graph Analysis

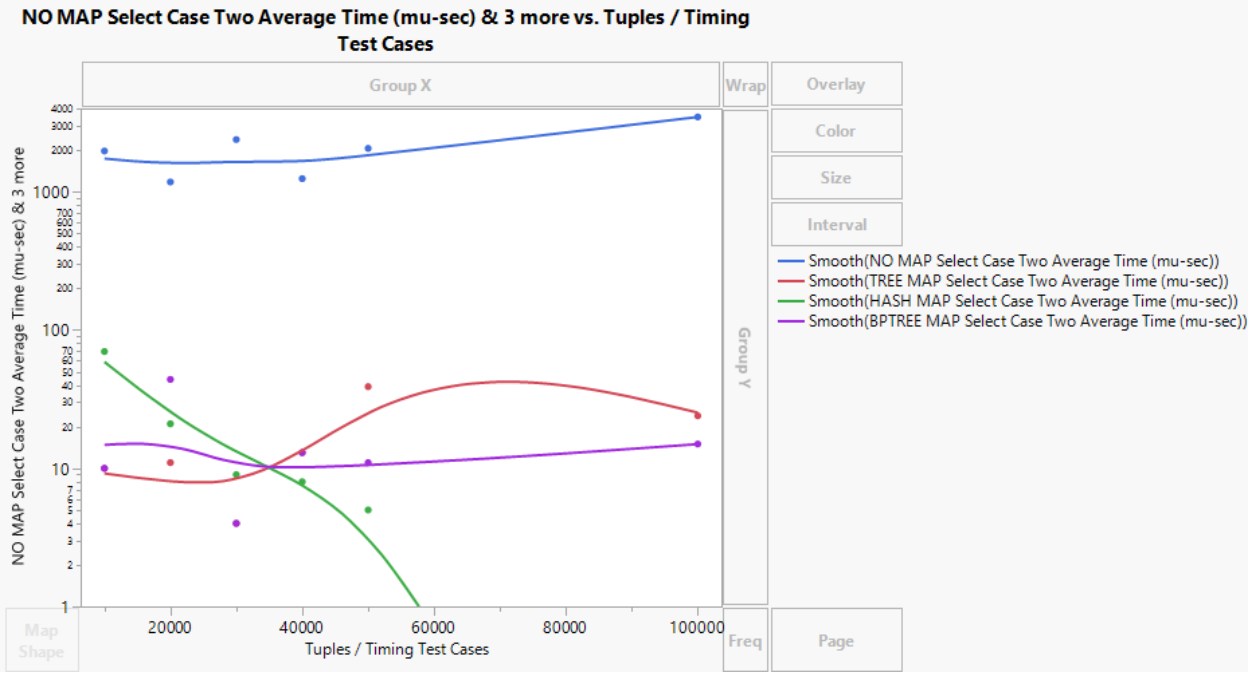


Select Case 1

Select Case 1

- NO_MAP (Blue): Shows a slight upward trend in runtime as the number of tuples increases. This indicates that the NO_MAP implementation becomes slower with larger datasets.
- TREE_MAP (Red): The runtime for TREE_MAP decreases initially but then gradually increases, showing variability in performance. This suggests TREE_MAP might handle small to medium datasets well but faces efficiency issues as tuples grow.
- HASH_MAP (Green): The runtime steadily decreases, which is unusual. It may indicate that the performance improves with larger datasets, possibly due to caching effects or other optimizations.
- BPTREE_MAP (Purple): Shows a stable and efficient performance with a slight increase toward the end. BPTREE_MAP appears to be relatively consistent in handling larger datasets.

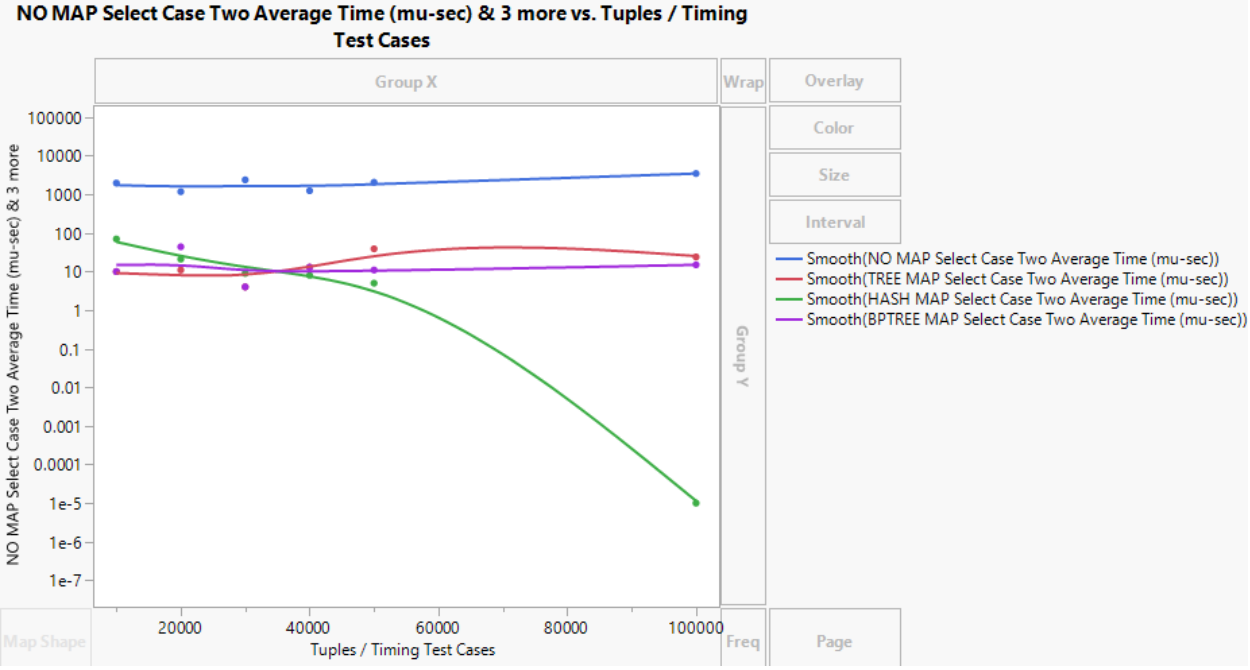
Summary: For Select Case 1, HASH_MAP and BPTREE_MAP perform best as data size increases, while NO_MAP and TREE_MAP become less efficient



Select Case 2

- NO_MAP (Blue): Similar to Case 1, the runtime remains high and gradually increases, highlighting inefficiency with larger data.
- TREE_MAP (Red): The runtime fluctuates but shows an upward trend, indicating inefficiency with higher data volumes.
- HASH_MAP (Green): Shows an initial spike but then stabilizes, suggesting that HASH_MAP adapts well after an initial period.
- BPTREE_MAP (Purple): Consistently maintains low runtime, showcasing strong performance for large datasets.

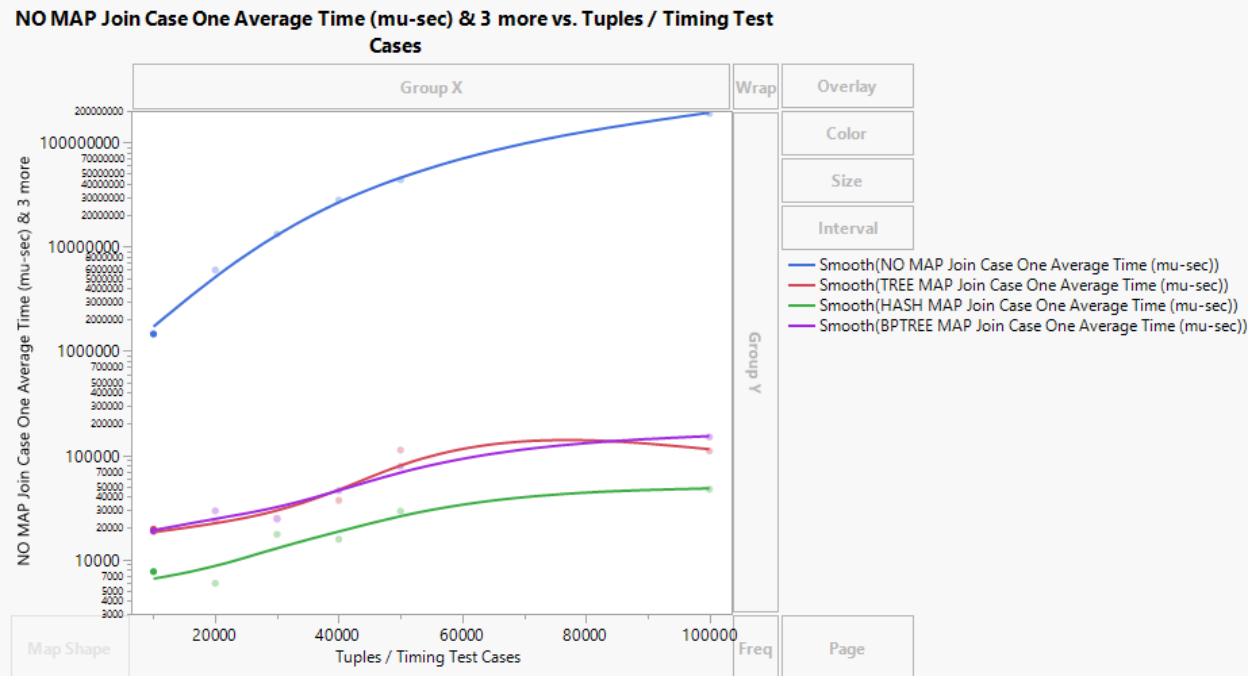
Summary: BPTREE_MAP demonstrates the best stability and efficiency, especially with large tuples. HASH_MAP also performs reasonably well after an initial increase, while NO_MAP and TREE_MAP show increased runtime with dataset growth.



Select Case Version 2

- NO_MAP (Blue): Consistently high runtime with a slight increase, again indicating inefficiency with larger datasets.
- TREE_MAP (Red): Shows a decreasing trend, suggesting it might handle this Select operation better with larger data.
- HASH_MAP (Green): Shows a stable performance and remains efficient as the data size grows.
- BPTREE_MAP (Purple): Exhibits excellent performance, with runtime decreasing as the dataset size increases, indicating optimal handling of large data.

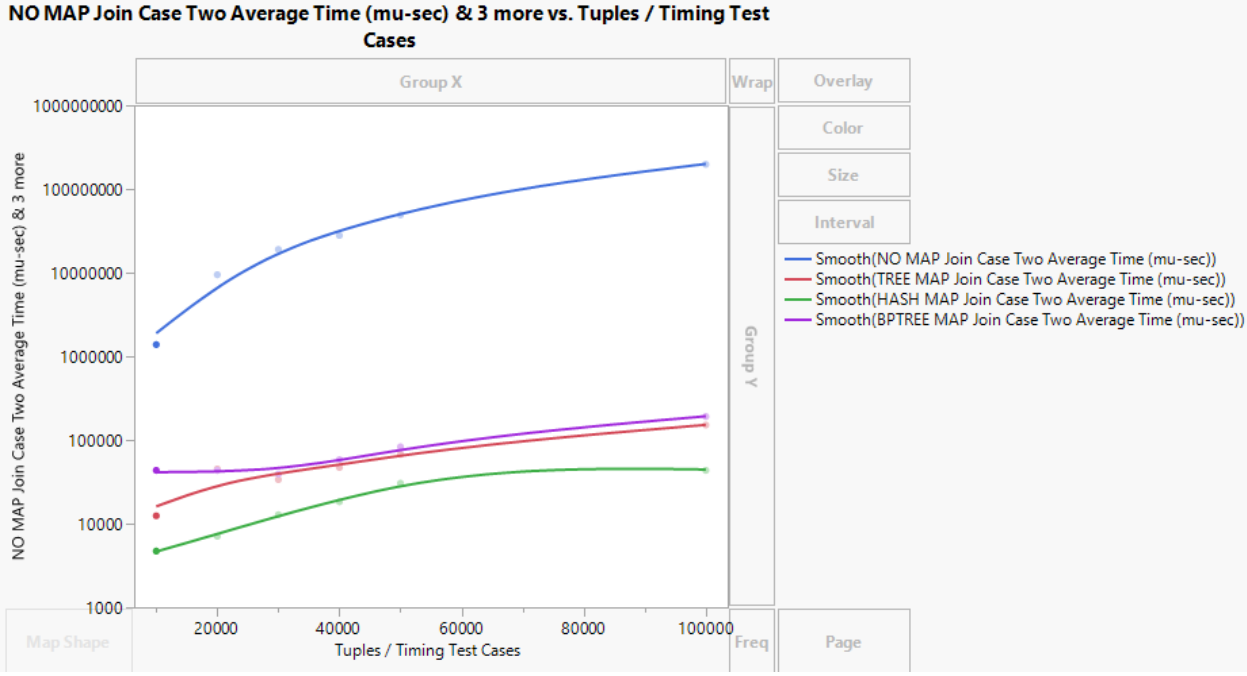
Summary: BPTREE_MAP and HASH_MAP provide the most efficient and stable performance, while NO_MAP remains the least efficient. TREE_MAP shows improvement as data grows but is still less efficient than HASH_MAP and BPTREE_MAP.



Join Case 1

- NO_MAP (Blue): Has the highest runtime with a steep upward curve, showing that it struggles significantly with larger datasets.
- TREE_MAP (Red): Shows a steady increase but remains lower than NO_MAP, indicating it is more efficient for join operations but still slows down with more data.
- HASH_MAP (Green): Displays a moderate upward trend, suggesting relatively efficient handling of joins but with some increase in runtime as data grows.
- BPTREE_MAP (Purple): Maintains the lowest runtime, showing that it's the most efficient for join operations across all data sizes.

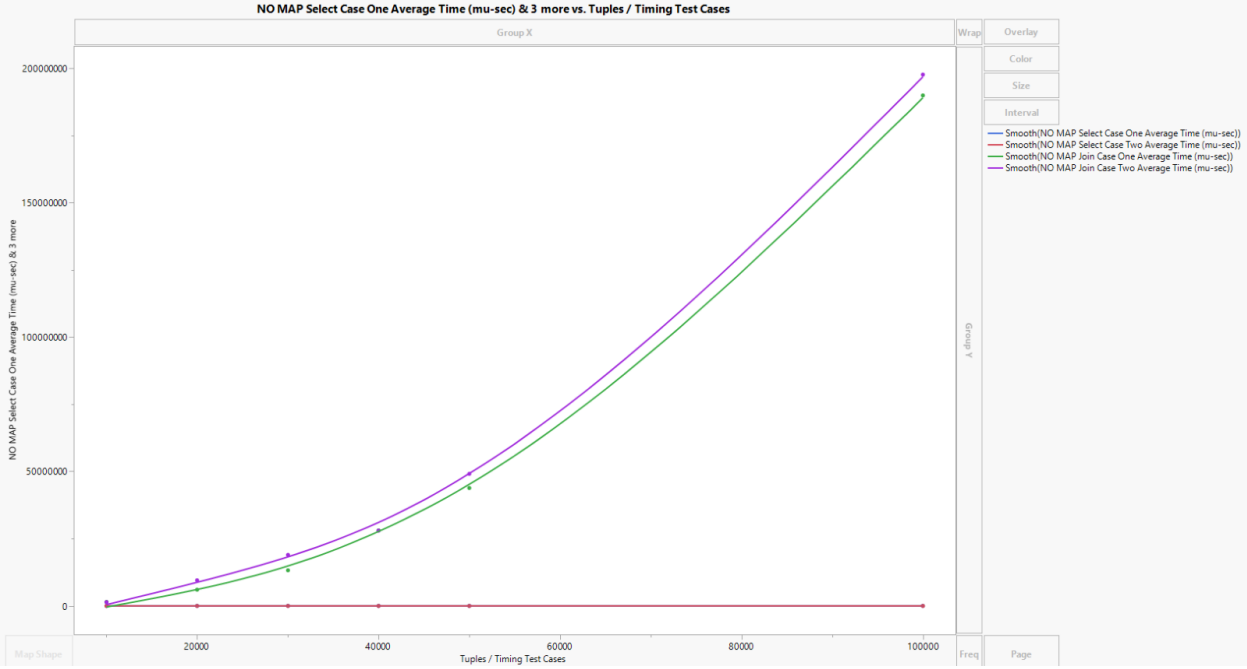
Summary: BPTREE_MAP is the best performer for Join Case 1, handling large datasets efficiently. HASH_MAP performs well but is slightly slower, while NO_MAP is the least efficient and has the steepest increase in runtime.



Join Case 2

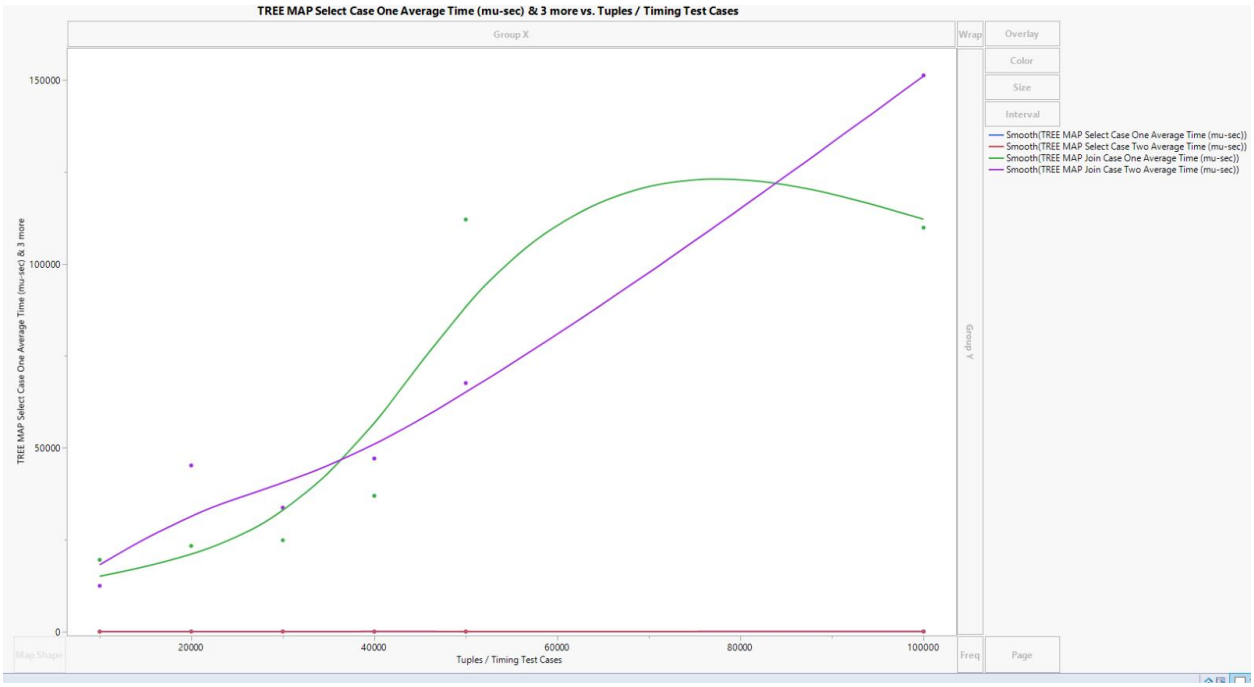
- NO_MAP (Blue): Similar to Join Case 1, NO_MAP has the highest runtime and steep increase, indicating poor scalability for join operations.
- TREE_MAP (Red): Shows a consistent runtime that is lower than NO_MAP, but still increases with larger data sizes, suggesting moderate scalability.
- HASH_MAP (Green): Maintains relatively low runtime with a gradual increase, indicating good efficiency and scalability.
- BPTREE_MAP (Purple): Once again demonstrates the lowest runtime and stable performance, making it the most scalable option for large datasets.

Summary: BPTREE_MAP continues to show the best efficiency and scalability for Join Case 2, while HASH_MAP performs well with moderate scalability. TREE_MAP is acceptable but not as efficient, and NO_MAP performs the worst.



No MAP

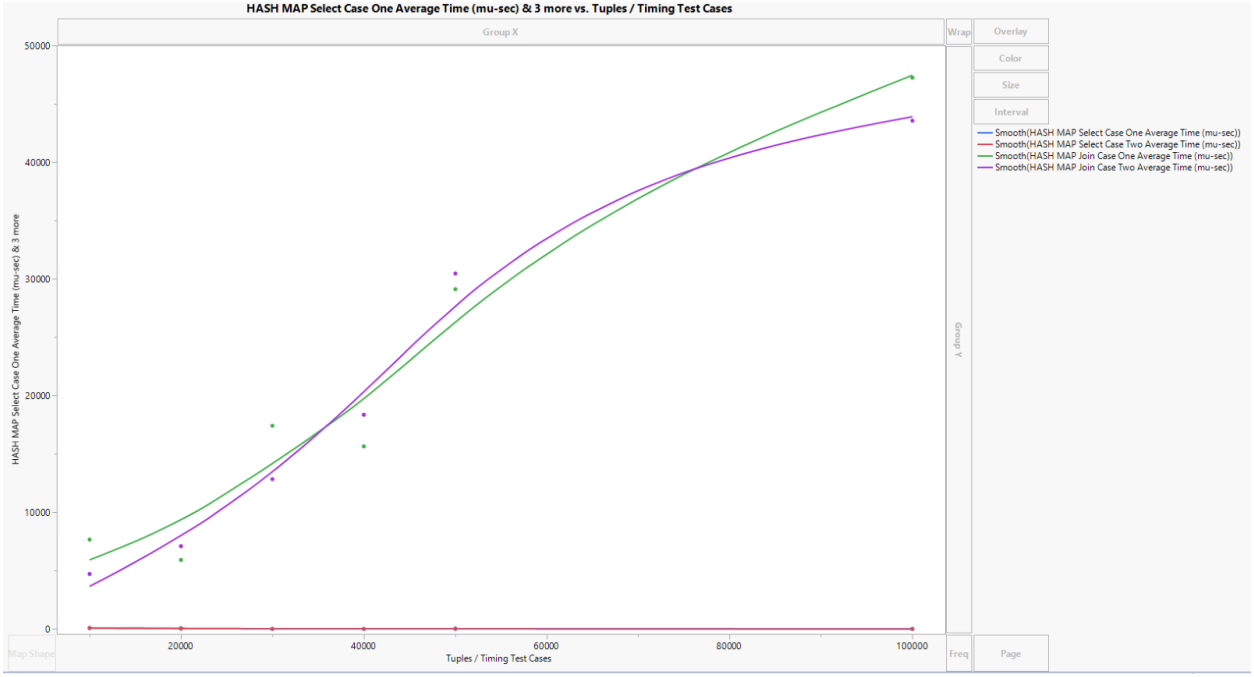
The graph displays a consistent upward trend, showing a rapid increase in average time as the number of tuples/test cases grows. The lines follow a close, slightly curved trajectory with consistent growth, suggesting high time complexity in this setup. The curve appears smoother and more predictable, potentially implying that this structure is more efficient for the types of operations being tested.



TREE MAP

The performance is less consistent than in the "No Map" scenario. The green line in particular shows changing, initially increasing, then decreasing, and eventually rising again. This graph suggests an irregular performance, potentially due to overhead in managing the Tree Map structure, which affects time complexity differently across different test cases. It shows variability, which could imply inefficiency in certain cases but might offer benefits in structured lookups depending on use cases.

In both graphs, the lines seem to follow a polynomial growth trend rather than linear, indicating that both scenarios have high time complexity. However, the Tree Map's changes (non-linear growth) suggest that it may have additional operations that influence timing depending on the size and complexity of data.



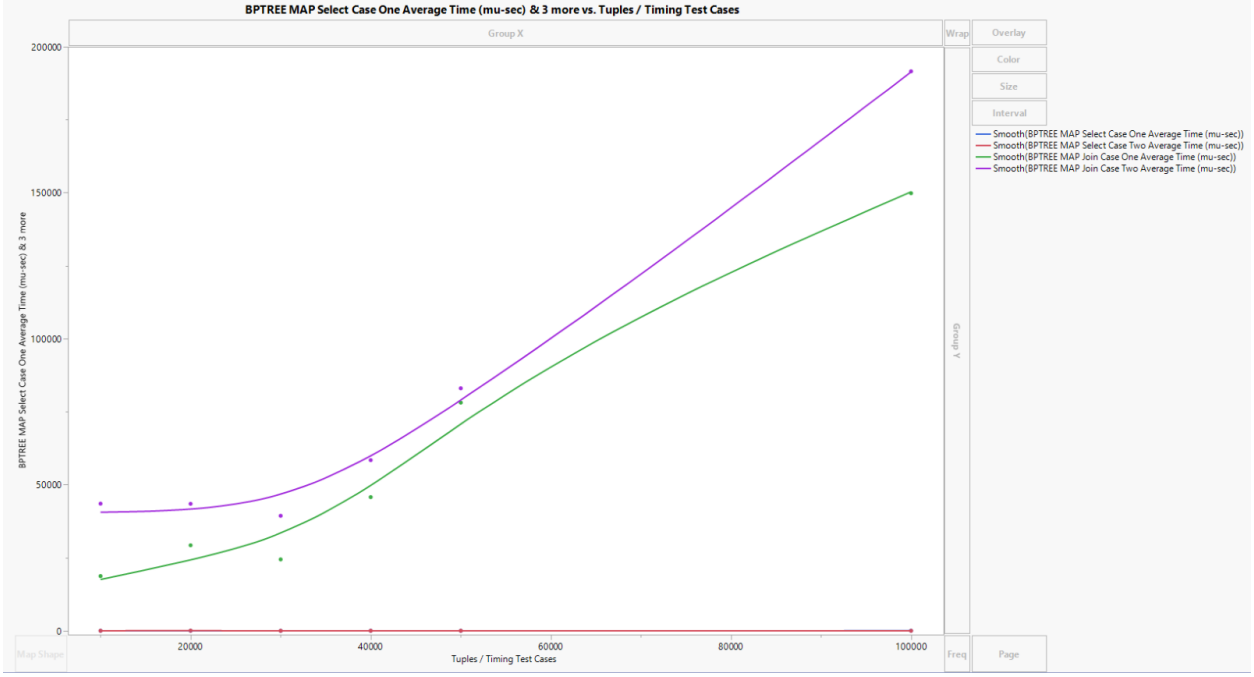
HASH MAP

The graph shows a steady increase in average time as the number of tuples increases. The curve shows a near-linear growth initially, which might transition towards a more curved shape at higher tuple counts. This means that the Hash Map maintains efficient performance even as the dataset grows, although there's a gradual increase in time required for operations.

For smaller numbers of tuples (e.g., up to 20,000), the Hash Map performs well with minimal latency, keeping the average operation time low. This shows the suitability of

Hash Maps for databases with smaller datasets or use cases where rapid lookup and retrieval are crucial.

As the tuple count exceeds around 50,000 to 100,000, the average time starts to grow more noticeably, but it still remains comparatively lower than the BTree Map. This shows that, while Hash Maps handle larger datasets efficiently.



BPTREE MAP

The BTree Map shows a steeper curve in its graph, especially as the tuple count increases. This suggests a faster-than-linear increase in average operation time.

For lower tuple counts (up to 20,000), the BTree Map performs reasonably well, with a minimal increase in operation time. This shows its efficiency with smaller datasets, where the structure's balancing overhead does not significantly impact performance.

At higher tuple counts (beyond 50,000), the BTree Map's average operation time increases considerably. This shows the increasing complexity of managing a balanced tree structure as more nodes (or entries) are added. The steep rise in time suggests that BTree Maps may not be as suitable for very large datasets unless ordered retrieval is critical.

This graph suggests that BTree Maps are better suited for situations where ordering and range queries are necessary, even though they may not provide the fastest access times compared to Hash Maps.