Project 3 Documentation Report: Performance Comparison

Manager: Ridhima Reddy

Team Members: Curt Leonard, Heeya Jolly, Jason Todd Maurer, Thomas Nguyen

Project Overview: In this project, we compared the performance of four data structures—NO_MAP, TREE_MAP, HASH_MAP, and BPTREE_MAP—to see how well they handle Select and Join operations. By measuring how quickly each structure ran these operations on varying data sizes, we identified which structures performed best under different conditions. This simulates how data structures in a database handle large data efficiently.

Project Structure

The project is organized with a clear and logical folder strucute, grouping related files to streamline navigation and development. Our code has detailed comments about complex logic and algorithms. Below is an overview of the main folders and files:

Folders:

timing graphs/: Contains visualizations and timing analysis graphs for Select and Join operations. These graphs illustrate performance comparisons across various indexing techniques (e.g., NO_MAP, TREE_MAP, HASH_MAP, BPTREE_MAP) as tuple sizes increase.

bin/: Holds compiled binary files generated during the build process. This folder contains the executable files needed to run the project without needing to recompile.

src/: The source code folder, containing the Java files for all main functionalities, including:

- **Indexing functions**: Modules that implement different indexing techniques like NO_MAP, TREE_MAP, HASH_MAP, and BPTREE_MAP.
- **Tuple generation**: Code for generating test tuples of various sizes, simulating database records for Select and Join operations.
- **Performance evaluation**: Code that measures and logs the time taken for Select and Join operations under different indexing strategies, providing data for further analysis.

store/: This folder stores generated data files and any temporary or intermediate storage required by the project during testing or runtime.

Files

- **compile**: A file that compiles all Java source files in the src folder. This file ensures that all code is built and up-to-date before running tests or evaluations.
- **run.sh**: A script that says the execution of the project. This file runs the compiled program, orchestrating Select and Join tests across different indexing techniques, and saves outputs to result files.
- **test_results.txt** & **test_results2.txt**: Output files that log the timing and performance results of the tests. These files store the runtime data generated by the performance evaluation scripts for different cases, providing raw data for analysis and graph generation.

Implementation Overview

Tuple Generator

The tuple generator in our project created tuples in incremental datasets increasing in steps for each test case. This generator is used to ensure consistent data input across Select and Join operation, which is important for comparing the four maps.

Data Structures Tested

We evaluated the following data structures for their efficiency in handling Select and Join operations:

1. **NO_MAP**: A structure without mapping.
2. **TREE_MAP**: A tree-based mapping structure.
3. **HASH_MAP**: A hash-based mapping structure.
4. **BPTREE_MAP**: A B+Tree-based mapping structure.

Operations

Two types of operations were tested:

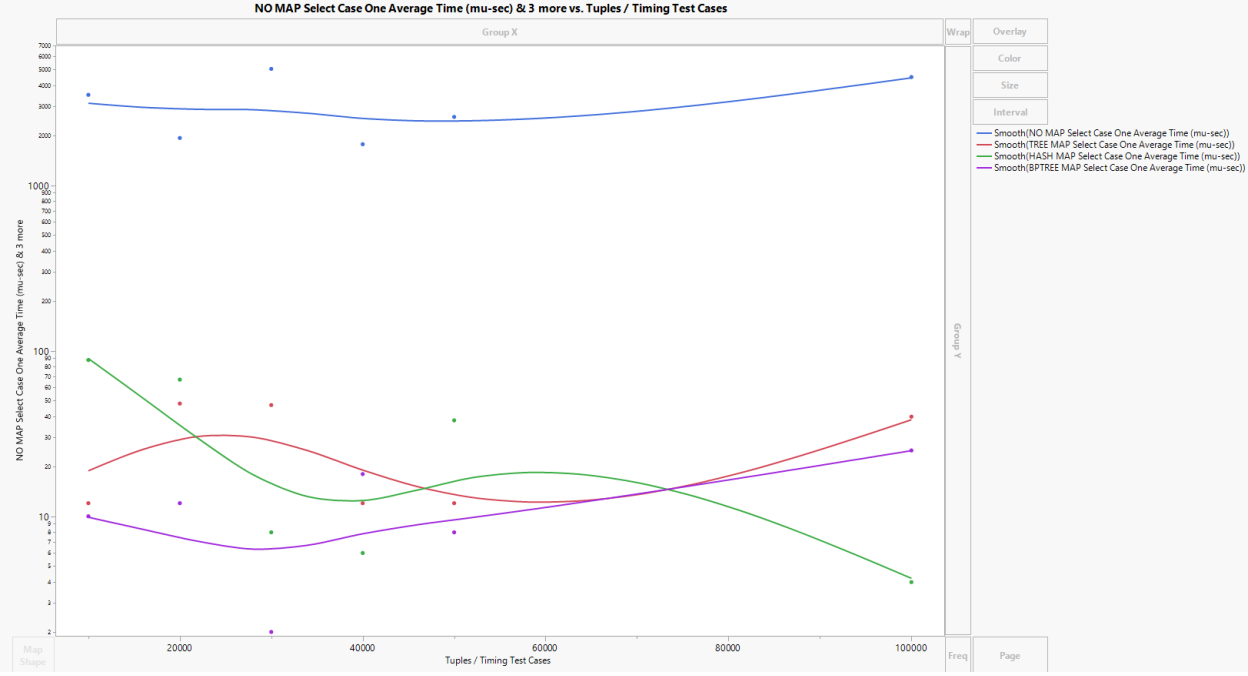
1. **Select Operation**: Querying specific tuples based on set conditions.
2. **Join Operation**: Combining tuples from two relations based on a common attribute.

Each operation was tested in two cases, totaling four performance evaluations. We used tuple generator to create tuple data sets for different amounts of total tuples (10-100K).

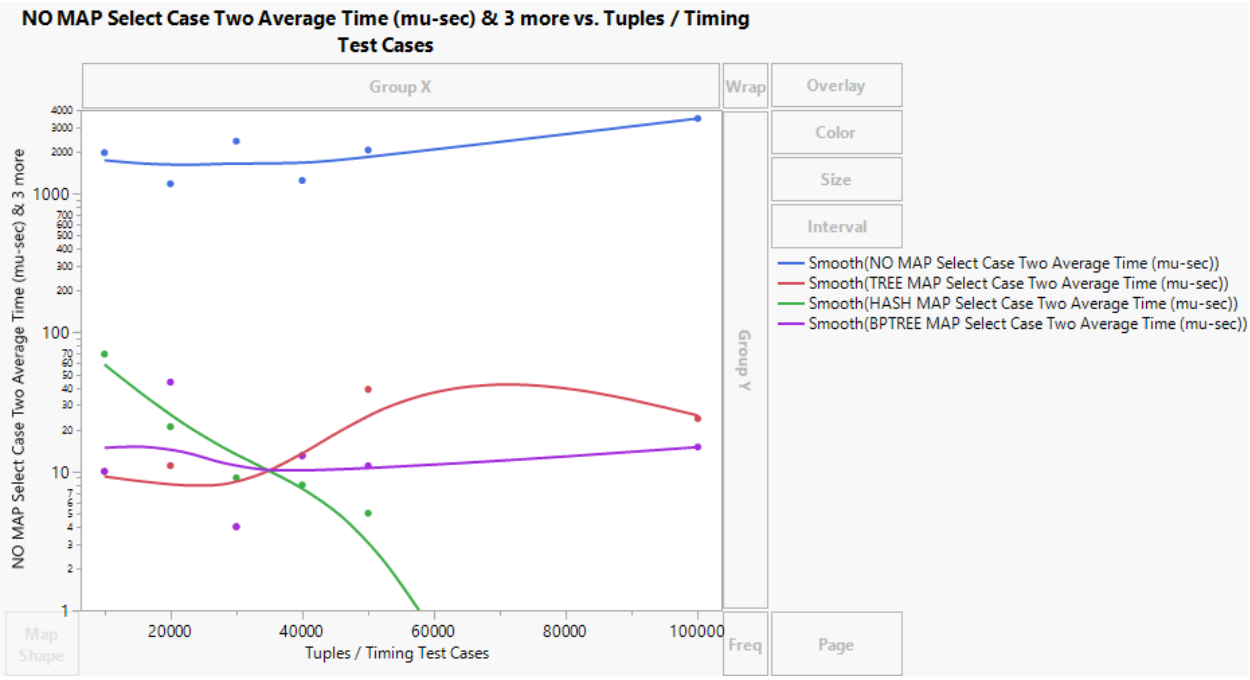
After that, we designed and implemented test cases for select and join operations. Then, ran test cases for each indexing type and each data set size, and documented times.

Performance Evaluation Process

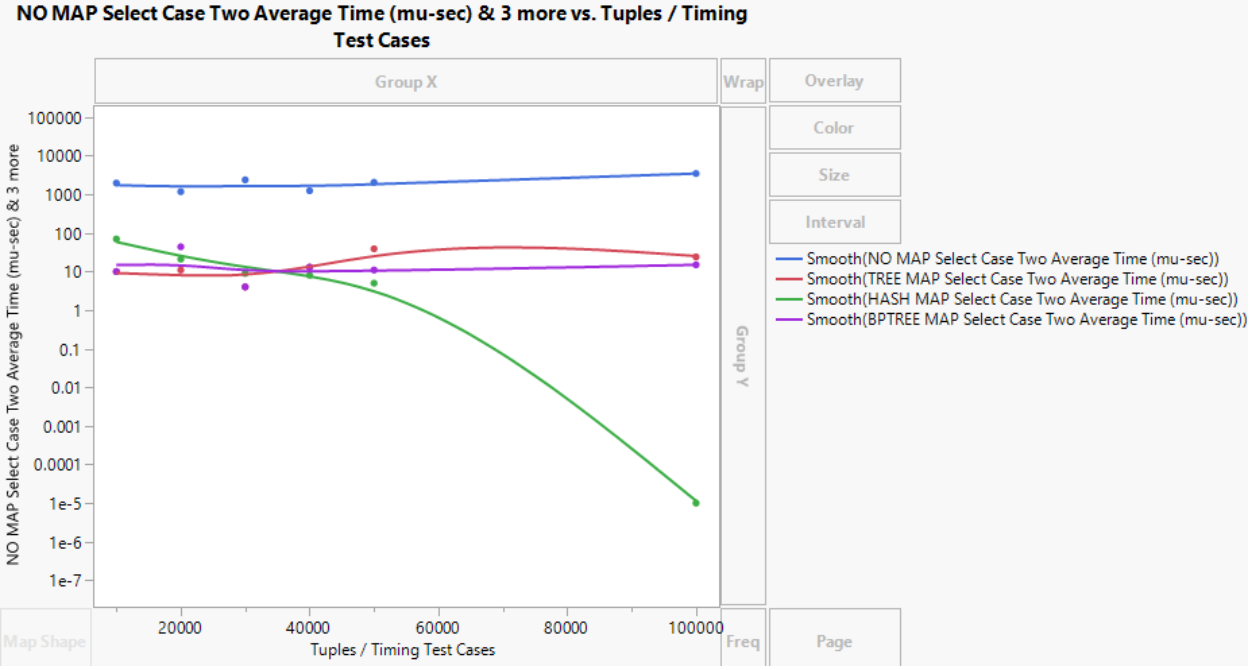
1. **Tuple Generation**:
    a. The provided tuple generator created datasets in increasing sizes to evaluate the data structures under different loads.
2. **Performance Measurement**:
    a. **Measurement Tool**: We used nanoTime() to get runtime measurements in milliseconds
    b. **Iteration Strategy**: To counter Just-In-Time (JIT) optimization effects, the first iteration was skipped. Each test case was run five times, with average runtime recorded.
3. **Plotting**: Results were plotted with runtime (ms) against the number of tuples (in units of 10,000). After, compiling all the provided data and entering it into the software to create graphs. We organized the data into four columns for each map and plotted them on a graph. Additionally, we generated a line of best fit for each case: select case 1, select case 2, join case 1, and join case 2. Then converted all the graphs to a logarithmic scale for best results.
    a. Graph Analysis in graph_analysis.pdf



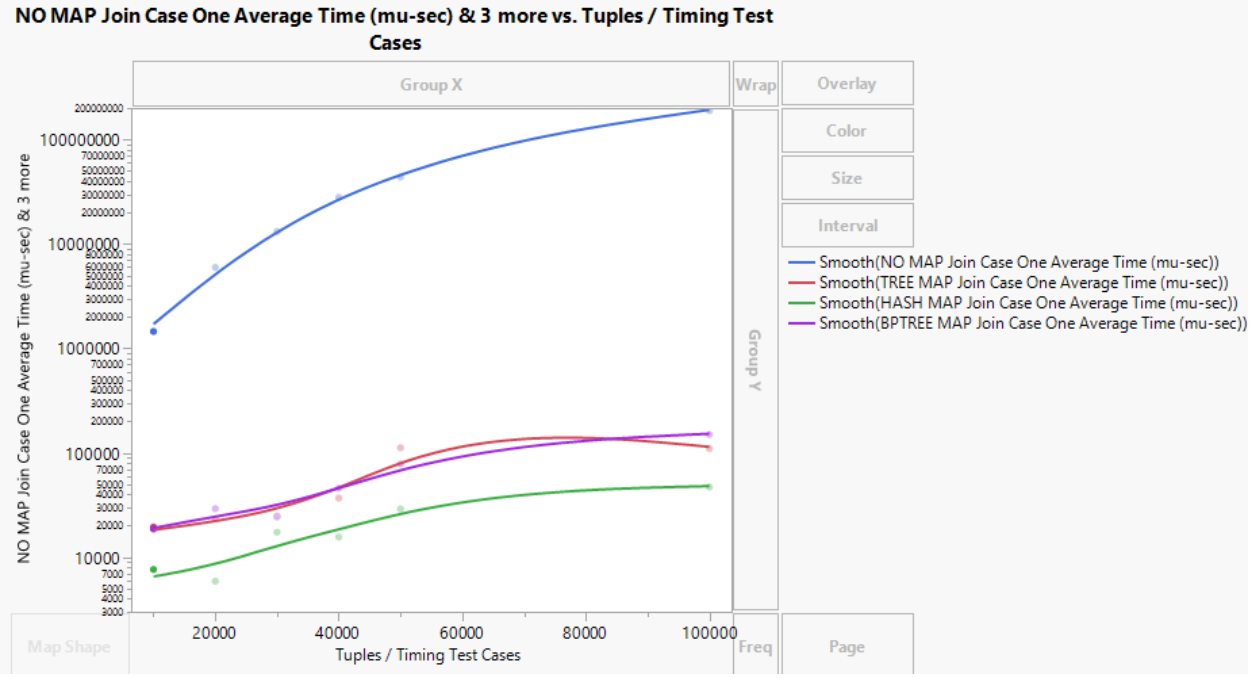
Select Case 1



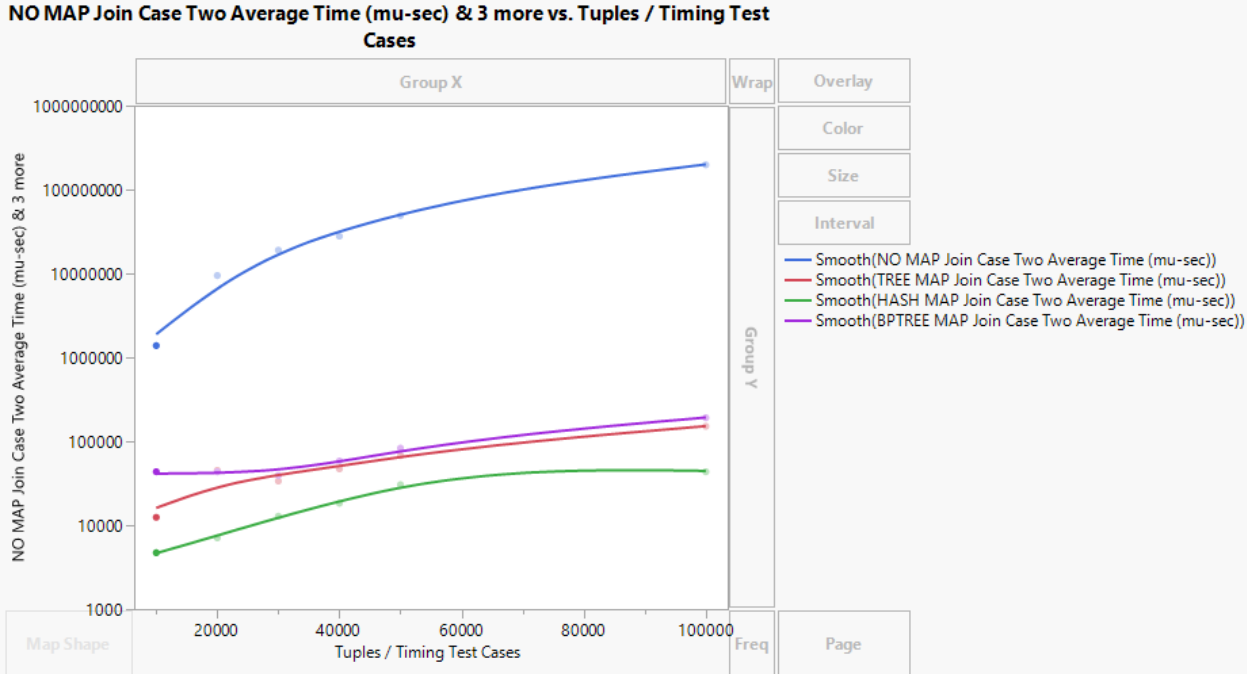
Select Case 2



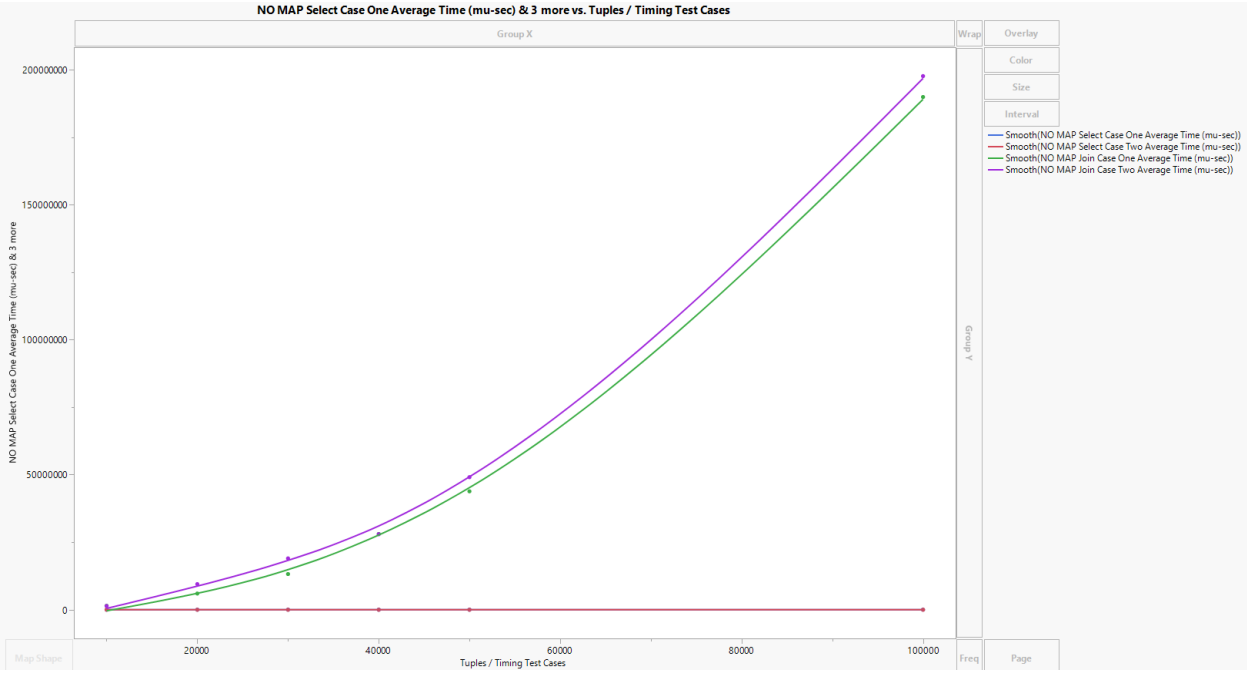
Select Case Version 2



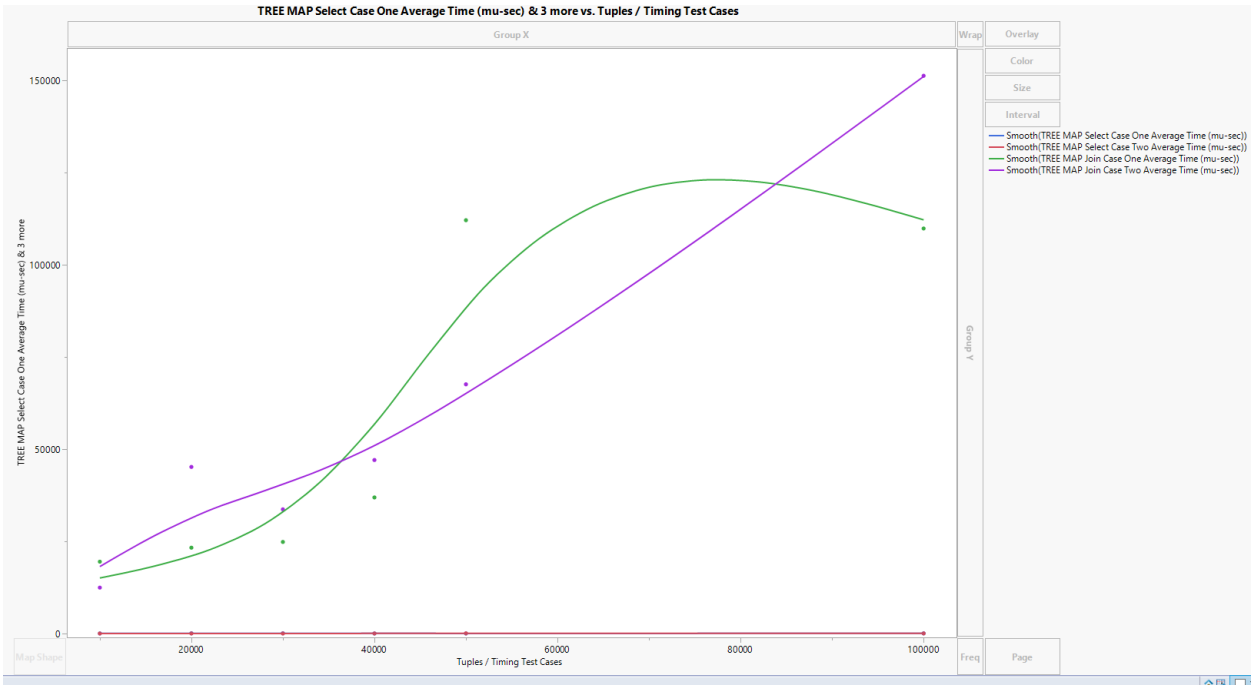
Join Case 1



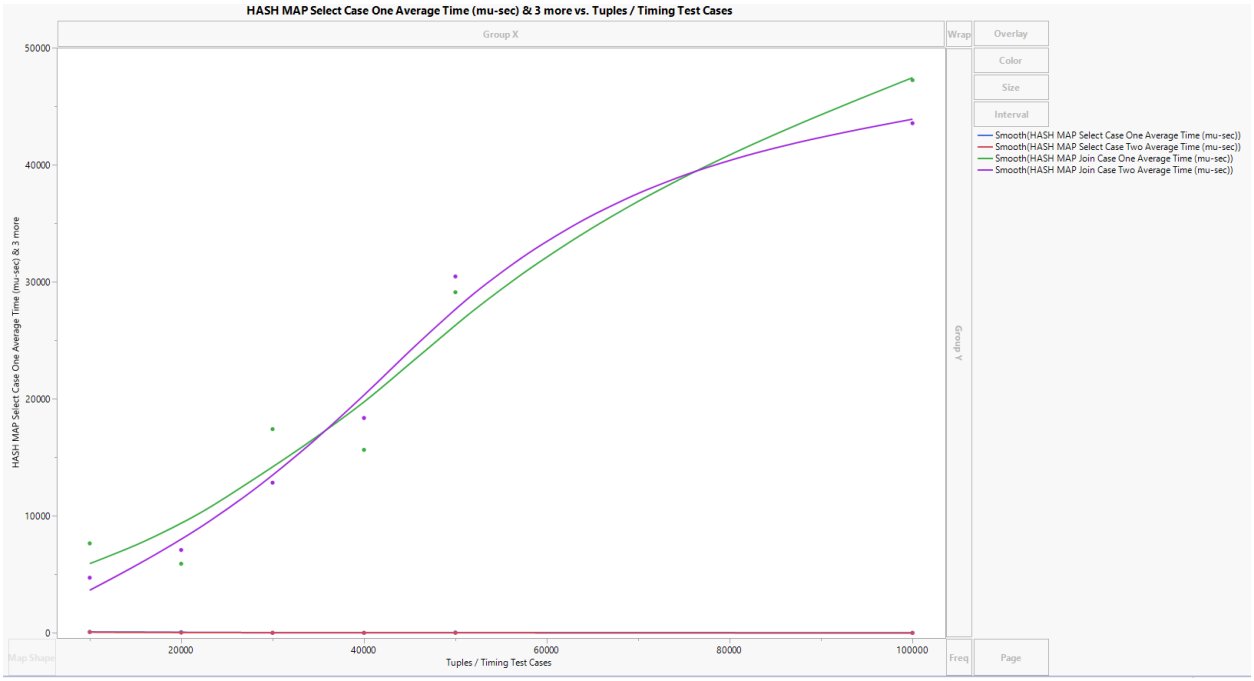
Join Case 2



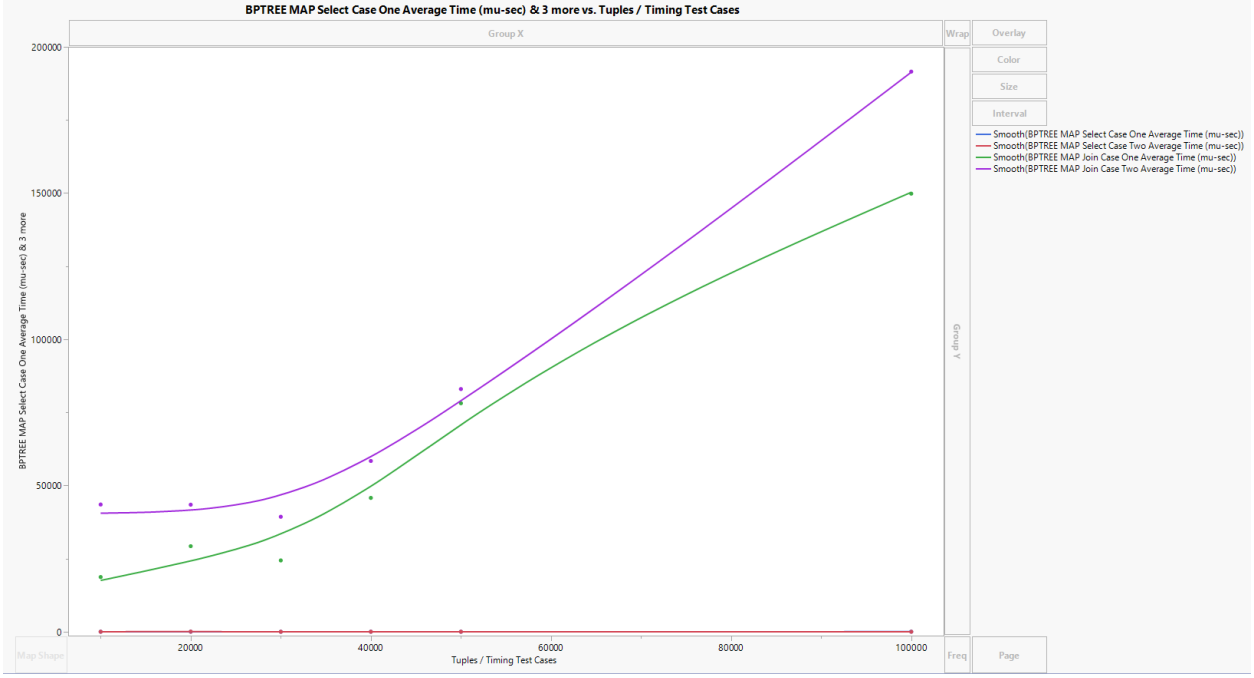
No MAP



TREE MAP



HASH MAP


BPTREE MAP Select Case One Average Time (mu-sec) & 3 more vs. Tuples / Timing Test Cases
Group X
Wrap
Overlay
Color
Size
Interval
Group Y
Map Shape
Freq
Page
200000
150000
100000
50000
0
BPTREE MAP Select Case One Average Time (mu-sec) & 3 more
20000
40000
60000
80000
100000
Tuples / Timing Test Cases
Smooth(BPTREE MAP Select Case One Average Time (mu-sec))
Smooth(BPTREE MAP Select Case Two Average Time (mu-sec))
Smooth(BPTREE MAP Join Case One Average Time (mu-sec))
Smooth(BPTREE MAP Join Case Two Average Time (mu-sec))

BPTREE MAP