Solution Report

1 Dataset Structure

The dataset was specifically designed to "stress test" the Sequential Join algorithm by creating a scenario with a **massive intermediate result** that mostly gets discarded in the final step.

- R_1: Contains 1000 tuples with $A_2 = 5$ and 1000 tuples with $A_2 = 7$.
- R_2: Contains 1000 tuples with $A_2 = 5$ and 1000 tuples with $A_2 = 7$.
- $R_1 \bowtie R_2$:
  - The join on $A_2 = 5$ produces $1000 \times 1000 = 1,000,000$ tuples.
  - The join on $A_2 = 7$ produces $1000 \times 1000 = 1,000,000$ tuples.
  - **Total Intermediate Size**: $\sim 2,000,000$ tuples.
- R_3: Mostly contains random values for A_3 that **do not match** the results from $R_1 \bowtie R_2$. The only matching path is via the specific "needle in the haystack" tuples (e.g., $(2002, 8)$ in R_2 matching $(8, 30)$ in R_3).

2 Comparison Results

I ran both algorithms on the generated dataset. The code can be found in `Solution.js`.

Algorithm	Execution Time	Number of Results
Yannakakis (Problem 2)	~ 1.33 **ms**	1001
Sequential Join (Problem 3)	~ 372.71 **ms**	1001

(Note: Times are from a representative run. The difference is consistently orders of magnitude.)

Do they return the same results?

Yes. Both algorithms correctly identified the 1001 result tuples.

Performance Analysis

Yannakakis is significantly faster (over 200x faster in this test).

Why?

1. **Sequential Join (Left-Deep)**:
   - It blindly computes the full join $R_1 \bowtie R_2$ first.
   - This generates an intermediate dataset of ~ 2 **million tuples**.

- It then attempts to join these 2 million tuples with R_3.
- Most of this work is wasted because the vast majority of these intermediate tuples have no match in R_3. The algorithm pays the cost of generating and processing a huge intermediate relation.

2. **Yannakakis Algorithm**:

   - **Reduction Phase**: It first filters the relations.
     - R_2 is reduced by R_3: The 2000 tuples in R_2 with $A_3 \in [1, 2000]$ are checked against R_3. Since R_3 mostly has $A_3 \geq 2002$, almost all tuples in R_2 are removed immediately! Only the tuple $(2002, 8)$ and potentially a few others survive.
     - R_1 is then reduced by the tiny R_2'.
   - **Join Phase**: By the time the join happens, the relations are tiny. The "explosion" of 1000×1000 never happens because the participating tuples were filtered out *before* the join.
   - This demonstrates the power of **semijoin reduction**: it avoids generating dangling intermediate results, ensuring $O(N + OUT)$ complexity rather than potentially $O(N^2)$ or worse for intermediate steps.