Trial Task: Cont & Kukanov Back-testing

Task Description
Your mission is to back-test and tune a Smart Order Router that follows the static cost model introduced by Cont & Kukanov ("Optimal Order Placement in Limit Order Markets"). The router splits a 5 000-share buy order across multiple venues using three risk parameters—lambda_over, lambda_under, and theta_queue. You must implement the allocator exactly as described in *allocator_pseudocode.txt*, replay a short stream of mocked market data, search for the parameter set that minimises total cost, and benchmark that tuned result against three baselines: (i) a naïve "take the best ask" strategy, (ii) a sixty-second-bucket TWAP, and (iii) a VWAP that weights prices by displayed ask size. All baselines and metrics are computed over the same nine-minute window provided in the data.

Resources
We supply four files:

- **cont_kukanov_excerpt.pdf** – four pages summarising the cost model
- **allocator_pseudocode.txt** – the exact static allocation algorithm (see above)
- **l1_day.csv** – roughly sixty thousand MBP records covering 13 : 36 : 32–13 : 45 : 14 UTC on 1 Aug 2024

You may use Python 3.8 + together with numpy and pandas. No other external data sources or services are allowed.

Input
l1_day.csv is a message-by-message feed. Use the field publisher_id as the venue key and the fields ask_px_00 and ask_sz_00 as the venue's best ask and displayed size (depth = 0, side = ask). For every unique ts_event keep only the first message per publisher_id; that gives you one Level-1 snapshot per venue per timestamp. Feed those snapshots, ordered by ts_event, into your back-test.
Your script should accept or internally define a small grid or random search over **lambda_over**, **lambda_under**, and **theta_queue**. At each snapshot attempt to execute as many shares as the allocator tells you, crossing the posted ask price up to the displayed size. Any unfilled quantity rolls forward to the next snapshot until the 5 000 shares are completed or the file ends.

Output
When your run finishes print one JSON object to stdout. It must contain:

- the best parameters you found,
- total cash spent and average fill price for that parameter set,
- the same two figures for the best-ask, TWAP, and VWAP baselines,
- the savings (in basis points) versus each baseline.

If you produce a cumulative-cost plot, save it as results.png or results.pdf.

Evaluation
We will run your code on a hidden data slice and judge it on:

1. correct implementation of the allocator,
2. clarity and robustness of your back-test loop,
3. sensible search over the three risk parameters,
4. completeness and readability of the JSON (and any plot),
5. the quality of a concise **README.md** explaining your approach, your parameter ranges, and one idea for improving fill realism (slippage, queue position, etc.).

A script that runs in under two minutes on a laptop and beats the best-ask baseline by at least a few basis points will be deemed successful.

What to upload

- backtest.py – standalone, imports only numpy, pandas, and the standard library, and prints the required JSON
- README.md – one page or less covering code structure, search choices, and your suggested improvement
- optional results.png or results.pdf with a cumulative-cost plot

Push the 3 files to a public Git repository and send us the link. If everything runs out of the box and your tuned router clearly outperforms the baselines, you're done. Good luck.