Homework Y

Designing a Trending System Under Uncertainty

Data Structures and Algorithms

Overview

In earlier assignments, you have studied heaps, balanced search trees, hash tables, graphs, shortest paths, streaming ideas, and priority queues.

This homework asks you to use those tools in a setting where the hard part is not only choosing the right data structure, but also deciding what the data structure is trying to represent.

What does it mean for a topic to be trending?

At first glance, this is a real-time top-K maintenance problem. But a ranking system is never just a passive observer of data. It defines what counts as importance, decides what receives attention, and may influence the future behavior that it later measures.

Your task is to design, implement, evaluate, and critique a trending-topic system. There is no unique correct answer. Strong submissions will combine algorithmic precision with thoughtful judgment.

Important principles:

- A simple design with excellent reasoning can be better than a complex design with weak justification.
- Your report matters as much as your code.
- You are encouraged to challenge the scoring rule, the workload model, and even the meaning of "top-K."
- We care less about whether you agree with us and more about whether your claims are precise, testable, and well defended.

System Story

Suppose you are building the "Trending Topics" page for a large social platform. Users continuously create events:

- creating posts,
- liking posts,
- commenting on posts,
- sharing posts,
- following related topics,
- moving between regions or communities.

The platform must answer questions such as:

- What are the current top 10 topics?
- Which topics are growing fastest?
- Is a topic popular globally or only inside a small community?
- Is the ranking robust to spam or automated bot traffic?
- How much accuracy should be sacrificed for speed at large scale?

The assignment has two layers:

1. Build a correct baseline system for a precisely specified event stream.
2. Extend and critique the system under alternative definitions of trending, different workloads, and imperfect information.

What We Are Really Studying

Ranking is an algorithmic definition of importance.

Different definitions lead to different rankings. Different rankings change what users see. What users see can change what users do. Therefore, ranking systems can shape the systems they measure.

> **Guiding question.** If two topics have the same raw popularity score, but one is rapidly growing inside a small community while the other is slowly fading from global attention, which one should be ranked higher? Why?

This homework is about data structures and algorithms, but it is also about the assumptions hidden inside them.

Baseline Scoring Model

Each topic has a score. In the baseline system, events modify the score as follows:

$$Score = 1 \times Post + 2 \times Like + 3 \times Comment + 5 \times Share.$$

Thus, an event contributes:

Event	Score Increment
`POST`	1
`LIKE`	2
`COMMENT`	3
`SHARE`	5

If two topics have exactly the same score, ties are broken by smaller topic id.

Time and Decay

Trending systems should not simply reward old accumulated popularity. Recent activity should matter more than ancient activity.

Let λ be a decay factor:

$$0.95 \leq \lambda \leq 0.9999.$$

Time is now explicit in the input. Each operation has a timestamp t. Timestamps are nondecreasing integers.

Suppose the previous processed operation had timestamp t_{prev}. Before processing the next operation at timestamp t, every active topic score decays by:

$$Score_i \leftarrow \lambda^{t-t_{\text{prev}}} \cdot Score_i.$$

Then the operation at time t is processed. If several operations have the same timestamp, no decay occurs between them. Queries do not create artificial time steps; they simply observe the system at their stated timestamp.

Algorithmic challenge. Naively applying decay once for every unit of elapsed time may be too slow when timestamps are large. Can you implement decay lazily while still answering ranking queries correctly?

You may assume the initial timestamp is 0 and all topic scores are initially 0.

Input Format

The first line contains the decay factor λ. The second line contains the number of operations M. Each of the next M lines is one timestamped operation.

Update Events

```
t POST topic_id
t LIKE topic_id
t COMMENT topic_id
t SHARE topic_id
```

Queries

```
t QUERY K
t RANK topic_id
t SCORE topic_id
```

`t QUERY K` returns the current top-K topics at time t in decreasing score order.

`t RANK topic_id` returns the current rank of a topic at time t. The highest-ranked topic has rank 1. If the topic has never appeared, return `-1`.

`t SCORE topic_id` returns the current score of a topic at time t, rounded to three digits after the decimal point. This query is included mainly to help testing and debugging.

Example

Input

```
0.99
10
1 POST 1
2 LIKE 1
3 SHARE 2
4 QUERY 2
5 COMMENT 1
6 LIKE 2
7 RANK 1
8 POST 3
9 QUERY 3
9 SCORE 1
```

One Possible Output

```
2 1
2
2 1 3
5.669
```

This example is intentionally small. It is not a performance benchmark. You should construct much larger and more adversarial test cases.

Required Programming Task

Implement the baseline system.

Your implementation must support:

1. score updates from events,
2. time decay,
3. top-K queries,
4. rank queries,
5. score queries,
6. deterministic tie-breaking.

Accepted languages:

- C,
- C++,
- Java,
- Python.

You may choose any reasonable internal representation. Possible ingredients include heaps, indexed heaps, balanced binary search trees, skip lists, hash tables, bucket structures, segment trees, or custom hybrids.

Scale Targets

Your report should discuss how your design behaves under the following settings.

Scale	Operations M	Distinct Topics T	Typical K
Small	10^4	10^3	10
Medium	10^6	10^5	10^2
Large	10^7	10^7	10^3

You are not required to make Python handle the largest setting, but you must honestly explain what would fail and how a production system might change.

Core Design Questions

Answer the following questions in your report. These questions are the heart of the homework.

Question 1: Choosing the Ranking Data Structure

What data structure do you use to maintain the ranking?

Discuss:

- update time,
- `QUERY K` time,
- `RANK` time,
- memory usage,
- implementation complexity,
- behavior under many equal or nearly equal scores.

Question 2: Lazy Decay

Can score decay be represented without explicitly updating every topic whenever time advances?

If yes, give the invariant that makes your method correct. If no, explain which operation prevents it.

Question 3: Query-Heavy vs. Update-Heavy Workloads

Suppose first that queries are much more frequent than updates:

$$\#Queries \gg \#Updates.$$

Suppose instead that updates are much more frequent than queries:

$$\#Updates \gg \#Queries.$$

Would you use the same design in both cases? Why or why not?

Question 4: Exact Rank Queries

Supporting `RANK topic_id` is often harder than supporting `QUERY K`.

What extra information does your structure need in order to answer rank queries efficiently? How does this change update complexity?

Question 5: Regional and Community Rankings

Now suppose each event also has a region:

```
LIKE topic_id region_id
QUERY K region_id
```

How would you support both global trending and regional trending?

Consider the case where there are many small regions, a few huge regions, and topics that cross regional boundaries.

Question 6: Throughput Is Not One Number

Is it possible that the data structure with the fastest asymptotic update time does not give the highest end-to-end throughput?

Discuss cache locality, constants, memory allocation, batching, and the distribution of K.

Alternative Definitions of Trending

The baseline scoring rule is intentionally simple. You are encouraged to question it.

Definition A: Raw Popularity

$$Popularity(t) = Score(t).$$

This favors topics with large total engagement.

Definition B: Velocity

$$Velocity(t) = Score(t) - Score(t - \Delta t).$$

This favors topics that are growing quickly.

Definition C: Acceleration

$$Acceleration(t) = Velocity(t) - Velocity(t - \Delta t).$$

This favors topics whose growth is itself increasing.

Definition D: Engagement Quality

Not all interactions are equally meaningful.

Should a long comment count more than a like? Should a share from a trusted user count more than many likes from new accounts? Should repeated actions from the same user be discounted?

Definition E: Diversity-Aware Ranking

Should ten nearly identical topics occupy all ten positions?

Design a rule that balances relevance and diversity. You may use clustering, topic similarity, graph distance, or any other reasonable model.

Definition F: Fairness-Aware Ranking

Should small communities have a chance to appear on the trending page? Should popularity alone determine visibility?

Give at least one fairness-aware ranking rule and discuss one potential downside of your rule.

> **Required reflection.** Choose one alternative definition above. Explain how the data structure would change if that definition replaced raw popularity.

Experimental Evaluation

Construct your own workloads. At minimum, evaluate:

- uniform topic distribution,
- Zipfian topic distribution,
- query-heavy workload,
- update-heavy workload,
- bursty workload where one topic suddenly becomes popular,
- adversarial workload with many tie scores or repeated score changes.

Measure:

- total running time,
- average update time,
- average query time,
- memory usage,
- how performance changes as K grows,
- correctness on small tests compared with a slow reference solution.

Your experiments do not need to be enormous. They do need to be honest, reproducible, and connected to your design claims.

Approximate and Large-Scale Systems

At billion-user scale, exact global ranking may be too expensive.

Discuss whether you would consider:

- sampling,
- Count-Min Sketch,
- Space-Saving or Misra-Gries heavy-hitter algorithms,
- reservoir sampling,
- sharded top-K aggregation,

- periodic batch recomputation,
- approximate quantiles or score buckets.

For one approximate method, state:

- what guarantee it gives,
- what error it may introduce,
- what memory it uses,
- which topics are most likely to be misranked.

Theory Corner

You are not required to give a full formal proof, but your report should include at least one theoretical claim with justification.

Examples:

- Any exact comparison-based structure supporting arbitrary score increments and exact rank queries appears to need $\Omega(\log T)$ update time in the worst case.
- If only top-K queries are needed for small fixed K, exact rank maintenance may be unnecessary.
- Under a streaming memory limit $o(T)$, exact top-K over arbitrary streams cannot always be recovered.
- Decay can be handled lazily if all scores are multiplied by the same global factor between updates.

Your claim may be a proof sketch, a reduction, an invariant argument, or a carefully stated counterexample.

Robustness and Manipulation

A ranking system can be attacked.

Discuss at least two of the following:

1. bot-generated traffic,
2. coordinated spam campaigns,
3. repeated actions by the same user,
4. fake engagement farms,
5. sudden celebrity amplification,
6. cold-start topics,
7. brigading across communities,
8. public-interest events that are important but not initially popular.

For each one you choose, explain:

- the threat model,
- how the attack changes the event stream,
- whether your data structure notices anything unusual,
- one possible defense,
- the computational cost of that defense.

Bonus Challenges

This section is optional. Strong submissions may receive bonus credit.

Bonus 1: Bot-Resistant Ranking

Design a ranking algorithm that is difficult for automated bots to manipulate.

Your answer should include:

- a threat model,
- an attack strategy,
- a defense mechanism,
- complexity analysis,
- one way your defense could fail.

Bonus 2: Distributed Top-K

Suppose events are processed by S independent shards. Each shard can send only a small summary to a central server.

Design a distributed algorithm for approximate global top-K. Analyze communication cost and failure cases.

Bonus 3: Lower Bound

Give a convincing argument for a lower bound related to exact top-K, exact rank queries, or streaming memory.

A complete proof is not required, but the statement must be precise enough to be falsifiable.

Bonus 4: A Better Definition of Trending

Propose your own definition of trending that is not listed in this handout.

It should include:

- a mathematical scoring rule,
- an efficient data structure or approximation,
- one reason the rule is socially or practically meaningful,
- one example where the rule behaves better than raw popularity,
- one example where it behaves worse.

Deliverables

Submit the following.

Submission Instructions

Name your submitted package using the following format:

`HWY-ClassXX-StudentID-Name`

For example:

`HWY-Class01-12345678-ZhangSan`

Submit your homework by email to:

`chengyuma@stu.xjtu.edu.cn`

Deadline:

July 12, 2026, 24:00

Part 1: Source Code

Submit your implementation and a short README explaining how to run it.

Part 2: Technical Report

Maximum length:

6 pages

Suggested structure:

1. Problem understanding and chosen interpretation.
2. Baseline data structure design.
3. Correctness argument and invariants.
4. Complexity analysis.
5. Experimental evaluation.
6. Alternative definition of trending.
7. Robustness, fairness, or manipulation discussion.
8. Reflection.

Part 3: Test Cases

Submit at least five test cases:

- one tiny hand-checkable test,
- one tie-breaking test,
- one decay-focused test,
- one query-heavy test,
- one stress or adversarial test.

What We Are Looking For

The teaching staff is not searching for a single solution.

We value:

1. clear reasoning,
2. correct implementation of the baseline task,
3. sound complexity analysis,
4. awareness of workload-dependent tradeoffs,
5. meaningful experiments,
6. careful discussion of approximation and uncertainty,
7. creativity that is grounded in evidence.

A sophisticated system that cannot be explained will not receive full credit. A modest system with clear invariants, honest experiments, and thoughtful tradeoff analysis can receive an excellent grade.

Grading Rubric

Component	Weight
Baseline correctness	20%
Data structure and algorithm design	20%
Complexity analysis and correctness reasoning	15%
Experimental evaluation	15%
Alternative trending definition and critique	15%
Report clarity and intellectual honesty	15%

TA Evaluation Guide

Excellent

An excellent submission has a correct implementation, clear invariants, strong complexity analysis, meaningful experiments, and an insightful discussion of tradeoffs. It goes beyond lecture material in at least one interesting direction without losing precision.

Good

A good submission has a mostly correct implementation, reasonable complexity analysis, basic experiments, and some discussion of alternative designs.

Fair

A fair submission has a partially correct implementation and limited analysis. It may work on simple cases but lacks convincing evidence or misses important tradeoffs.

Poor

A poor submission has missing or substantially incorrect implementation, unsupported claims, little evidence of testing, or a report that does not show understanding of the problem.

Academic Integrity

Discussion is encouraged. Copying is not.

You may discuss ideas with classmates, but all submitted code, experiments, and writing must be your own work. If you use external resources, cite them appropriately.

We care far more about intellectual honesty than obtaining a particular answer.

Research Corner

Many active research areas emerge naturally from this homework:

- streaming algorithms,
- heavy-hitter detection,
- online algorithms,
- approximation algorithms,
- distributed data structures,
- fair ranking,
- recommendation systems,
- information retrieval,
- mechanism design,
- adversarial machine learning,
- computational social choice.

If this homework sparks your curiosity, consider exploring one of these topics further.

Final Thought

Throughout this course, we have studied many data structures: stacks, queues, trees, heaps, hash tables, graphs, and more.

At times, it may seem that data structures are merely tools for making programs faster.

This homework asks you to discover something deeper.

Every data structure embodies a particular view of the world.
Every algorithm encodes a set of priorities.
Every ranking system decides what deserves attention.

As (future) AI scientists, we do not merely build systems. We shape how information flows.

Design carefully.