<https://leetcode.com/problems/minimum-time-to-complete-trips/description/>

You are given an array time where time[i] denotes the time taken by the ith bus to complete one trip.

Each bus can make multiple trips successively; that is, the next trip can start immediately after completing the current trip. Also, each bus operates independently; that is, the trips of one bus do not influence the trips of any other bus.

You are also given an integer totalTrips, which denotes the number of trips all buses should make in total. Return the minimum time required for all buses to complete at least totalTrips trips.

**Example 1:**

**Input:** time = [1,2,3], totalTrips = 5

**Output:** 3

**Explanation:**

- At time t = 1, the number of trips completed by each bus are [1,0,0]. The total number of trips completed is 1 + 0 + 0 = 1.

- At time t = 2, the number of trips completed by each bus are [2,1,0]. The total number of trips completed is 2 + 1 + 0 = 3.

- At time t = 3, the number of trips completed by each bus are [3,1,1]. The total number of trips completed is 3 + 1 + 1 = 5.

So the minimum time needed for all buses to complete at least 5 trips is 3.

**Example 2:**

**Input:** time = [2], totalTrips = 1

**Output:** 2

**Explanation:**

There is only one bus, and it will complete its first trip at t = 2.

So the minimum time needed to complete 1 trip is 2.

**Constraints:**

1 <= time.length <= 10^5

1 <= time[i], totalTrips <= 10^7

**Attempt 1: 2024-12-09**

**Solution 1: Binary Search + Greedy (10 min)**

**Style 1: canCompleteTrips**

class Solution {

    public long minimumTime(int[] time, int totalTrips) {

        long lo = 1;

        // The largest possible time is min⁡(time) × totalTrips,

        // assuming only the fastest bus is used for all trips.

// (Don't make mistake as assuming only the slowest bus is used for all

// trips as max(time) \* totalTrips)

        // long right = (long) Arrays.stream(time).min().getAsInt() \* (long) totalTrips;

        long hi = (long) 1e7;

        for(int t : time) {

            hi = (long) Math.min(hi, (long) t) \* totalTrips;

        }

        // Find lower boundary (since return the minimum time required

        // for all buses to complete at least totalTrips trips)

        while(lo <= hi) {

            long mid = lo + (hi - lo) / 2;

            // If able to complete trips with current minimum time,

            // we can move 'hi' backward to 'mid - 1' to attempt

            // smaller minimum time, otherwise if cannot complete,

            // we can move 'lo' forward to 'mid + 1' to attempt

            // larger minium time

            if(canCompleteTrips(time, totalTrips, mid)) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        return lo;

    }

    private boolean canCompleteTrips(int[] time, int totalTrips, long minTime) {

        long trips = 0;

        for(int t : time) {

            // Number of trips each bus can complete in givenTime

            trips += minTime / t;

            // Early exit if we meet the requirement

            if(trips >= totalTrips) {

                return true;

            }

        }

        return false;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Style 2: cannotCompleteTrips**

class Solution {

    public long minimumTime(int[] time, int totalTrips) {

        long lo = 1;

        // The largest possible time is min⁡(time) × totalTrips,

        // assuming only the fastest bus is used for all trips.

        // (Don't make mistake as assuming only the slowest bus is used for all

        // trips as max(time) \* totalTrips)

        // long right = (long) Arrays.stream(time).min().getAsInt() \* (long) totalTrips;

        long hi = (long) 1e7;

        for(int t : time) {

            hi = (long) Math.min(hi, (long) t) \* totalTrips;

        }

        // Find lower boundary (since return the minimum time required

        // for all buses to complete at least totalTrips trips)

        while(lo <= hi) {

            long mid = lo + (hi - lo) / 2;

            // If not able to complete trips with current minimum time,

            // we can move 'lo' forward to 'mid + 1' to attempt

            // larger minimum time, otherwise if cannot complete,

            // we can move 'hi' backward to 'mid - 1' to attempt

            // smaller minium time

            if(cannotCompleteTrips(time, totalTrips, mid)) {

                lo = mid + 1;

            } else {

                hi = mid - 1;

            }

        }

        return lo;

    }

    private boolean cannotCompleteTrips(int[] time, int totalTrips, long minTime) {

        long trips = 0;

        for(int t : time) {

            // Number of trips each bus can complete in givenTime

            trips += minTime / t;

        }

        return trips < totalTrips;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to chatGPT**

**Approach:**

This is a **binary search on the answer** problem. The task is to minimize the time required to complete the trips by leveraging all buses.

**Key Idea:**

Binary Search:

The smallest possible time is 1.

**The largest possible time is min(time) × totalTrips, assuming only the slowest bus is used for all trips. (Don't make mistake as max(time) \* totalTrips)**

Feasibility Check:

For a given time t, calculate the total number of trips all buses can complete within t.

If the total trips ≥ totalTrips, then t is feasible.

class Solution {

public long minimumTime(int[] time, int totalTrips) {

// Binary search bounds

long left = 1;

long right = (long) Arrays.stream(time).min().getAsInt() \* (long) totalTrips;

while (left < right) {

long mid = left + (right - left) / 2;

if (canCompleteTrips(time, totalTrips, mid)) {

right = mid; // Try for a smaller time

} else {

left = mid + 1; // Increase the time

}

}

return left;

}

private boolean canCompleteTrips(int[] time, int totalTrips, long givenTime) {

long trips = 0;

for (int t : time) {

trips += givenTime / t; // Number of trips each bus can complete in givenTime

if (trips >= totalTrips) return true; // Early exit if we meet the requirement

}

return trips >= totalTrips;

}

}

**Explanation:**

Binary Search:

Start with left=1 and right=min⁡(time)×totalTrips.

Use the mid-point of the current range to test if it's feasible to complete totalTrips within mid time.

canCompleteTrips **Function**:

For each bus, calculate the number of trips it can complete within givenTime.

Accumulate these trips and return true if the total ≥ totalTrips, otherwise false.

Optimization:

Exit early in the feasibility check if the total trips already meet the requirement.

**Refer to**

[L410.Split Array Largest Sum (Ref.L1011,L704,L1482)](note://WEB4d9d6f26a965381b137615128e2648be)

[L774.Minimize Max Distance to Gas Station (Ref.L410,L1011,L1283,L1482)](note://WEB28e37ffbbab2b651254c65af5334111d)

[L1011.Capacity To Ship Packages Within D Days (Ref.L410,L1482)](note://WEB87ca8f9dd9a14fb4535856a4e0f8f4b1)

[L1482.Minimum Number of Days to Make m Bouquets (Ref.L410,L1011)](note://WEBd5ce394160da45a79e92dbbc806111ec)

[L1552.Magnetic Force Between Two Balls (Ref.L1802,L2226)](note://WEB4d72055c8eee671bf55dca2e0a3f951d)

[L1760.Minimum Limit of Balls in a Bag (Ref.L410,L1011)](note://WEB63c2447a0b64e5d381d55dd4e94b7f0f)

[L1802.Maximum Value at a Given Index in a Bounded Array (Ref.L410)](note://WEB7d060d486a37a8c92696be41f18fb27c)

[L2064.Minimized Maximum of Products Distributed to Any Store (Ref.L410)](note://WEB2ca95ff77a9fee4bcced70f3e62a41df)

[L2226.Maximum Candies Allocated to K Children (Ref.L1802,L1552)](note://WEBbc250dc9fe15c478547298491ba2386f)