Coding with AI Assignment

Ali Muhammad Asad
aa07190

1. Problem Statement

Finding Median from a Data Stream: The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values. So the problem was to continously find the median given an array of integers, and a stream of instructions, such as "MedianFinder", "addNum", and "findMedian", where "addNum" adds a number to the array, and "findMedian" returns the median of the array, while "MedianFinder" initializes the MedianFinder object.
The image below describes an example input, output, and its explanation.

Example 1:

Input
```
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
[[], [1], [2], [], [3], []]
```
Output
```
[null, null, null, 1.5, null, 2.0]
```

Explanation
```
MedianFinder medianFinder = new MedianFinder();
medianFinder.addNum(1);    // arr = [1]
medianFinder.addNum(2);    // arr = [1, 2]
medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3);    // arr[1, 2, 3]
medianFinder.findMedian(); // return 2.0
```

2. My Solution

If we tackle this problem using the brute force approach, we can simply add the number to the array, sort the array, and then find the median. However, this approach is not efficient, as the time complexity of sorting the array is $O(n \log n)$, where n is the number of elements in the array, and we do this each time a new element is added which is not efficient and would bring the time complexity to $O(n^2 \log n)$.

Thus, I approached this problem by maintaining two heaps; a max heap (which would store the larger half of the array), and a min heap (which would store the smaller half of the array). Since the median is the middle value, by maintaining two heaps, we ensure that the median is always at the top of the heaps; if the total number of elements is odd, then the median of the two elements is on top of the max heap, and if the total number of elements is even, then the median is the average of the top elements of the two heaps.

This approach is efficient as the time complexity of adding an element to the heap is $O(\log n)$, and since we add n elements, the time complexity of the entire operation is $O(n \log n)$, as taking

the top element of the heap is $O(1)$. Further, the space complexity of this approach is $O(n)$, as we are storing n elements in the heap.

My solution passed all the test cases, ran in 424ms, beating 72.76% of the submissions, while taking a memory of 35.44 MB.

3. LLM 1 - ChatGPT 4o

I passed the entire prompt on Leetcode to ChatGPT 4o and then ran its provided code to see if it could solve the problem. ChatGPT had the same idea of maintaining two heaps, and its code was similar to mine, however, it had more conditional checks than mine hence took more time than mine to run.

ChatGPT's code was more easier to follow through since it also provided comments at every step, and made it more readable. Since the approach is the same, the time complexity of ChatGPT's code is also $O(n \log n)$, and the space complexity is $O(n)$.

GPT's solution passed all the test cases, ran in 490ms, beating 42.99% of the submissions, while taking a memory of 35.58 MB.

4. LLM 2 - Claude 3.5 Sonnet

I passed the entire prompt on Leetcode to Claude 3.5 Sonnet and then ran its provided code to see if it could solve the problem. Claude also had the same idea of maintaining two heaps, and its code was similar to mine, however, it had more conditional checks than mine hence took more time than mine to run.

Claude's code was more easier to follow through since it also provided comments at every step, and made it more readable. Since the approach is the same, the time complexity of Claude's code is also $O(n \log n)$, and the space complexity is $O(n)$.

Claude's solution passed all the test cases, ran in 490ms, beating 42.99% of the submissions, while taking a memory of 35.58 MB. It solution was similar to ChatGPT's solution, and also ran in the same time and memory.

5. Comparison and Analysis

My solution ran the fastest, beating both ChatGPT and Claude, while taking the same memory as Claude. The reason for this could be that my code had less conditional checks than ChatGPT and Claude, and was more concise. However, considering readability and understandability, ChatGPT and Claude's code was more readable and understandable than mine, as they provided comments at every step, and made the code more readable.

6. Reflection

The LLMs were able to solve the problem efficiently, and their solutions were similar to mine, however, they had more conditional checks than mine, which made their code less efficient. However, their code was more readable and understandable than mine, as they provided comments at every step, and made the code more readable. I think that LLMs can complement human problem-solving by providing a different perspective to the problem, and can challenge human problem-solving by providing a more efficient solution than the human, or by providing a more readable and understandable solution than the human.