LeetCode IPO Problem - Detailed Report

Kushal Chandani (kc07535)

1 Problem Statement

IPO: Given at most k projects, each with a profit and a minimum capital requirement, choose projects so that you maximize your final capital. You start with an initial capital w.

Example:

- $k = 2$
- $w = 0$
- `profits` = [1,2,3]
- `capital` = [0,1,1]

In the example, the optimal strategy is to pick project 0 (profit 1, capital 0) and then project 2 (profit 3, capital 1), resulting in a final capital of 4.

For the following solutions, I have included a Jupyter notebook file which contains all the code for each of the solutions.

2 Manual Solution

2.1 Approach

I used a straightforward method to solve the problem, i.e. to first pair each project's capital requirement with its corresponding profit and sort the list based on the required capital. This ensures that projects with lower capital requirements are considered first. Then, in an iterative process, we evaluate the projects within our capital limits and choose the most profitable one at each step. We repeat this process up to k times, ensuring that each selected project increases our available capital, allowing us to unlock more profitable projects in subsequent iterations.

To track selected projects, a boolean array can be used to mark completed ones. The overall approach relies on scanning through the project list repeatedly to find the most profitable and affordable option.

2.2 Complexity Analysis

The manual approach, in the worst case, requires scanning through all n projects in each of the k iterations. This results in a time complexity of $\mathcal{O}(k \cdot n)$, which may be acceptable for small inputs but becomes inefficient as the number of projects increases. The space complexity remains at $\mathcal{O}(n)$, mainly for storing project details and tracking selections.

2.3 Observations

While the manual solution produces correct results for some cases, its inefficiency on larger inputs is a significant drawback. The repeated full scans introduce performance bottlenecks.

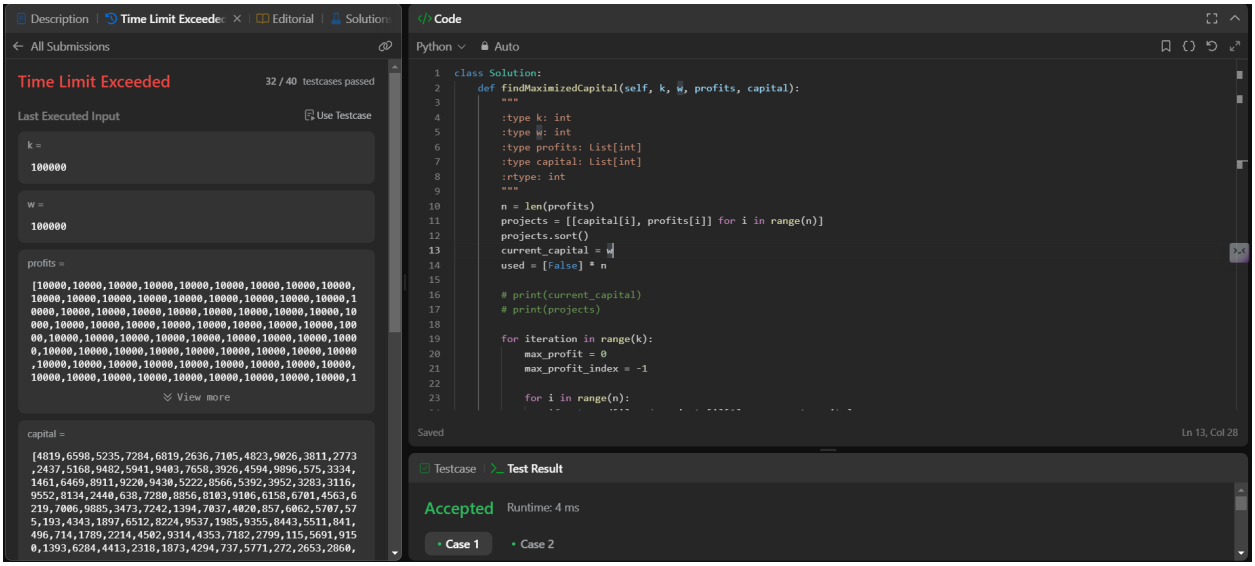

Figure 1: Manual Solution

3 LLM #1 Solution (ChatGPT 4o-mini)

3.1 Approach

The first LLM-generated solution improves upon the manual approach by incorporating sorting and a max-heap. The projects are initially sorted by their capital requirements, ensuring that at each step, we only need to consider those projects we can afford. Then, a max-heap is used to efficiently retrieve the most profitable project among the ones available.

During each iteration, projects that can be afforded are added to the heap, and the most profitable one is selected. This significantly reduces the number of scans required, optimizing performance.

3.2 Complexity Analysis

Sorting the projects requires $\mathcal{O}(n \log n)$ time while inserting elements into and extracting them from the heap takes $\mathcal{O}(\log n)$. As a result, the overall complexity of the approach is $\mathcal{O}(n \log n)$, which is a significant improvement over the manual solution. The space complexity remains at $\mathcal{O}(n)$ due to storage requirements for the heap.

3.3 Observations

This approach passes all test cases and demonstrates a clear performance advantage. The use of a max-heap allows efficient selection of the best project at each step.

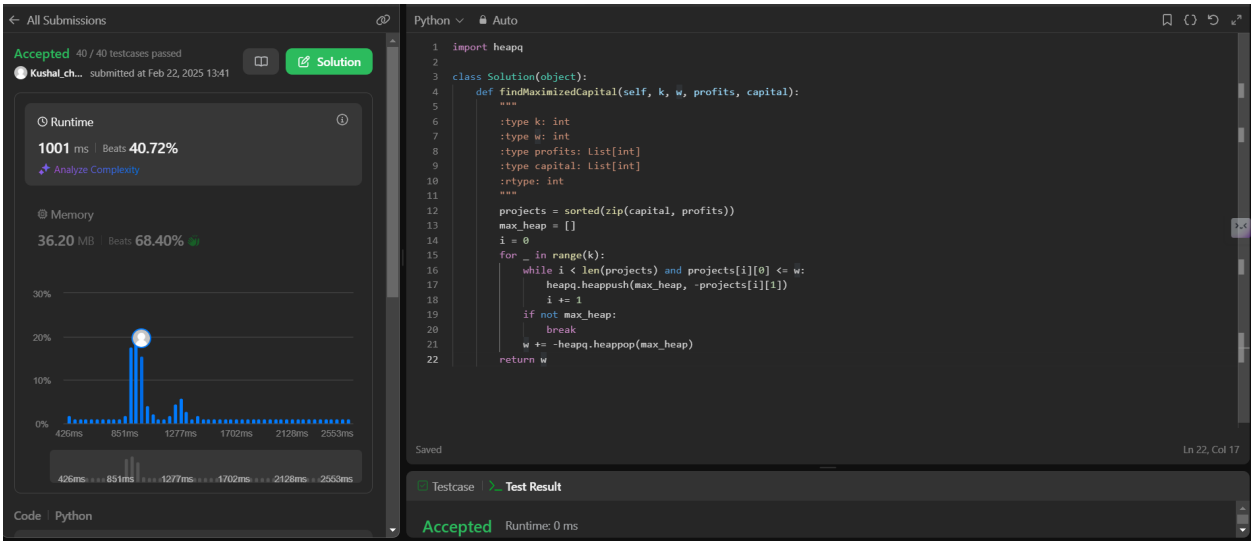

Figure 2: GPT Solution

4 LLM #2 Solution (Claude 3.5 Sonnet)

4.1 Approach

The second LLM-generated solution follows a similar strategy as the first, leveraging a max-heap for optimized selection. It also sorts the projects by capital requirement and efficiently selects the most profitable project using a heap structure. Additionally, this solution incorporates error handling to ensure robustness in cases where no projects can be selected at a given step.

4.2 Complexity Analysis

Just like the first LLM approach, this method also runs in $\mathcal{O}(n \log n)$ time, leveraging sorting and heap operations for efficiency. The space complexity remains at $\mathcal{O}(n)$.

4.3 Observations

This solution is well-structured and easy to follow. It correctly handles edge cases and maintains efficiency. The readability and clarity of the implementation make it a strong alternative to the first LLM solution.

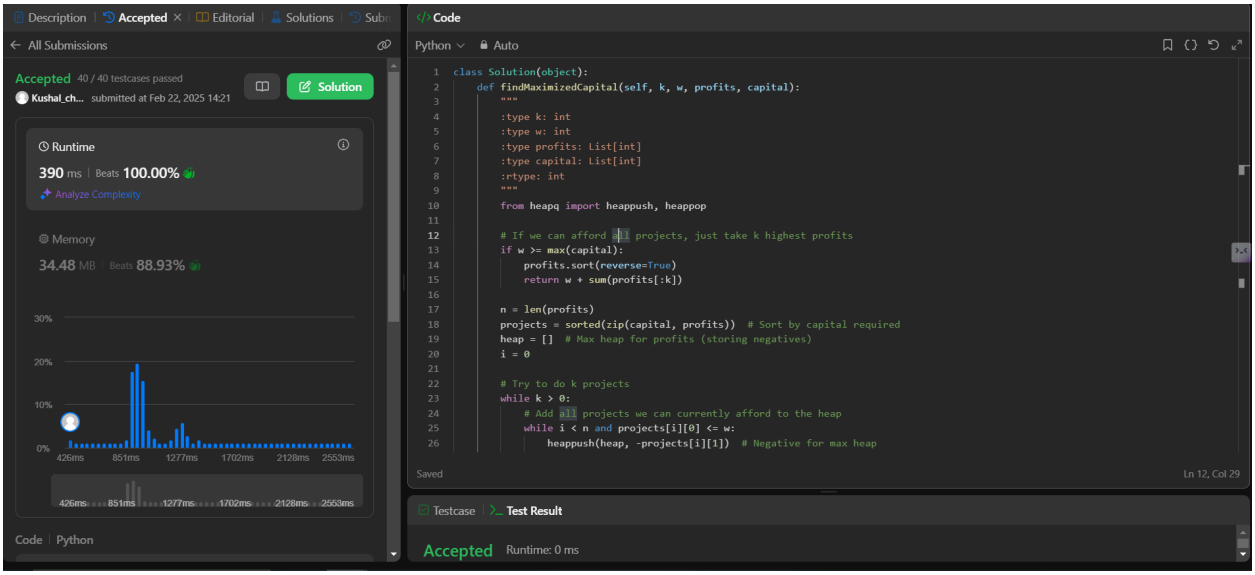

Figure 3: Claude Solution

5 Comparison & Analysis

5.1 Correctness

All three solutions correctly solve the problem for given test cases. However, while the manual approach may be viable for small inputs, its inefficiencies become apparent with larger datasets. The LLM solutions, by contrast, maintain efficiency even at scale.

5.2 Efficiency

The manual solution operates at $\mathcal{O}(k \cdot n)$, which can be slow for large values of n and k. On the other hand, both LLM solutions use a combination of sorting and heap operations to achieve $\mathcal{O}(n \log n)$ complexity, making them significantly more efficient.

5.3 Readability & Structure

The manual approach is easy to understand but lacks optimization. The LLM solutions are both well-structured and efficient, offering concise yet powerful implementations. Their use of data structures like heaps enhances performance without sacrificing clarity.

6 Takeaways

I think the comparison highlights the importance of optimization in problem-solving. Using a max-heap, as seen in both LLM solutions, improves performance, making an otherwise inefficient approach scalable. Additionally, using AI-generated solutions can provide insights into better ways to tackle algorithmic challenges. Critical assessment of solutions, considering correctness, efficiency, and readability, is essential in developing robust problem-solving skills. Ultimately, blending manual efforts with LLM-generated insights can enhance coding practices and lead to more effective solutions.