📓 502. IPO — Complete Documentation

1. Problem Statement

You are given n projects. Each project requires a certain minimum capital to start (capital[i]) and offers a pure profit (profits[i]) upon completion.

You start with an initial capital w. You can complete at most k distinct projects. After completing a project, the profit is added to your current capital.

Your goal is to choose at most k projects such that your final capital is maximized.

- Input:
  - k (int): Max number of projects you can do.
  - w (int): Initial capital.
  - profits (List[int]): Profit of each project.
  - capital (List[int]): Capital needed for each project.
- Output:
  - Return the maximum capital you can obtain after completing up to k projects.

2. Intuition

We always want to pick the most profitable project we can afford at the current time. Therefore, we:

- Use a min-heap to track available projects by their capital requirement.
- Use a max-heap to always pick the most profitable among affordable projects.

3. Key Observations

- We can't do all projects — only those we can afford with current capital.
- Profits earned from earlier projects can unlock new projects.
- Greedily selecting the highest-profit project from the affordable set gives optimal results.

4. Approach

Step-by-Step:

- Heapify all projects into a min-heap by capital.
- Initialize a max-heap for profits (simulate max-heap using negatives).
- For up to k iterations:
  - Move all projects that can be started with current capital into the max-heap.
  - If none are affordable, break.
  - Else, pick the most profitable one and update current capital.

5. Edge Cases

- If all projects need more capital than we have — return initial capital.
- If k == 0 — no projects can be done.
- If we run out of affordable projects before reaching k, stop early.

6. Complexity Analysis

◈ Time Complexity:

- O(n log n + k log n)
  - O(n log n) to heapify and insert projects.
  - O(k log n) for picking max profit k times.

◈ Space Complexity:

- O(n) for the heaps.

7. Alternative Approaches

- Naive Approach: Iterate through all projects k times, checking which are affordable and most profitable — results in O(k*n) time, which is too slow for large inputs.
- Sorting + Two Pointers: Not flexible for dynamic capital changes — using heaps is more optimal.

8. Test Cases

✅ Example 1:

k = 2, w = 0

profits = [1,2,3]

capital = [0,1,1]

Output: 4

Explanation: Do project 0 (profit 1), then project 2 (profit 3)

✅ Example 2:

k = 3, w = 0

profits = [1,2,3]

capital = [0,1,2]

Output: 6

✅ Edge Case:

k = 1, w = 0

profits = [1]

capital = [1]

Output: 0

9. Final Thoughts

- This problem combines greedy strategy and efficient data structures (heaps).
- Picking the best among the affordable projects at every step is key.
- It demonstrates a practical optimization problem seen in investment planning, resource allocation, and startup growth scenarios.