2.7 Project Selection

Andre Monteiro

Term 3, 2025

Problem Summary

We are given n personal coding projects to choose from. Each project i has:

- A minimum knowledge requirement r_i
- A knowledge gain g_i

Initally, knowledge level is s, and we only have time to complete exactly k projects, where $k < n$. Your goal is to select k projects to maximize your final knowledge level.

Question A)

We are asked to find an efficient algorithm to select a sequence of k projects (possibly with repetitions) that maximizes your final knowledge. At each step, the only restriction is that you can select a project if your current knowledge is at least r_i.

Approach

To solve this, a greedy algorithm using sorting and linear search can be used.

(a) First, we sort the projects in ascending order of r_i. This allows us to quickly find all available projects at each knowledge level.

(b) For each of the k steps:

  (i) Linearly scan the list of projects from the start

  (ii) Select the project with the highest g_i such that $r_i \leq$ current knowledge

  (iii) Increase your knowledge by g_i

This ensures that, at every step, you always pick the best available project in terms of knowledge gain.

Time Complexity

- Sorting the list by r_i: $O(n \log n)$ (mergesort)
- For each of the k steps, scanning up to n projects: $O(kn)$

Thus, the total time complexity is:

$$O(n \log n + kn)$$

Since $k < n$, this is bounded above by $O(n^2)$.

$$\boxed{\text{Total Time Complexity: } O(n \log n + kn)}$$

Discussion of Correctness

This greedy approach is justified by the following intuition:

- At every step, we can only choose from projects that are *currently available* and, whose requirement r_i is less than or equal to our current knowledge
- Among those, it always makes sense to choose the project with the highest knowledge gain g_i
- Since repetition is allowed, we are not penalized for choosing the same project again if it remains optimal

There may exist multiple optimal sequences, but this method guarantees one such sequence, by always taking the greedy best move at each point.

Question B)

Now, assume that projects cannot be repeated. You may still only complete exactly k projects, and your goal remains to maximize the final knowledge level after those k steps.

We must now design an efficient algorithm that runs in:

$$\boxed{O(n \log n)}$$

Algorithm

The algorithm begins by sorting all projects in ascending order of their knowledge requirement r_i. We initialize a priority queue to keep track of the gains g_i of all projects that are currently available,(those with $r_i \leq s$), where s is the current knowledge level. Starting with a pointer $i = 0$ and initial knowledge s, we iterate for k rounds. In each round, we add all newly available projects (those with $r_i \leq s$) into the queue and increment the pointer i accordingly. If the queue is empty at any point, we terminate early since no further projects can be undertaken. Otherwise, we select the project with the highest g_i from the queue, remove it, and increase our knowledge level by g_i. This process ensures that at each step we choose the most rewarding project from the set of available, unused projects.

(a) Sort all projects by increasing requirement r_i.

(b) Initialize a priority queue to store available projects by gain g_i.

(c) Set a pointer $i = 0$ and start with initial knowledge s.

(d) Repeat for k steps:

- While $r_i \leq s$, add project i to the queue and increment i.
- If the queue is empty, terminate early (no available project).
- Otherwise, pop the project with the highest g_i and add g_i to your knowledge.

Time Complexity

- Sorting the projects: $O(n \log n)$
- Each project is pushed into and popped from the queue at most once:

$$O(n \log n)$$

- Each of the k rounds performs at most one pop and multiple pushes (in total, n pushes).

Overall time complexity:

$$\boxed{O(n \log n)}$$

Question C)

We aim to prove that the greedy algorithm described in question b) selects, at each step, the highest gain project currently available based on the current knowledge level producing an optimal final knowledge total when no project is repeated.

Let us suppose, for the sake of contradiction, that there exists an optimal sequence A of k distinct projects that yields a higher final knowledge level than the greedy sequence G.

Let both A and G consist of k valid (non-repeating) project selections. Suppose the two sequences agree for the first t steps, but differ at step $t+1$. Let project p be chosen by G at this step, and project q be chosen by A, with both being available. Since the greedy algorithm always selects the project with the highest possible gain among available options, we must have:

$$g_p \geq g_q$$

Now, construct a new sequence A' by replacing q with p in sequence A at position $t+1$. Since p was available at this step and was not used in A previously, A' remains valid (still consists of k distinct projects).

Additionally, the total knowledge sum of A' is at least as large as that of A:

$$\boxed{\sum(A') \geq \sum(A)}$$

We can repeat this exchange argument step-by-step, continuing to align sequence A' with the greedy sequence G from left to right. After at most k such exchanges, we obtain a sequence identical to G, with a total knowledge gain at least equal to that of A.

Therefore, the greedy sequence G must be optimal.