Algorithms

Dynamic Programming Approach: The fault of this part is that it can take more than one element from each level. For example, it can add two pawns to the army. I could not correctly apply the condition of adding an element from each level while processing the data to the table.I correctly returned from the last step to the first step and found the selected elements from the table.

Greedy Approach: Before calling the function, I sorted the data (AttackPoint/ Gold) from smallest to largest. According to the algorithm described in the lecture, the array should be ordered from largest to smallest, but instead I reversed the operations inside the function. The function (AttackPoint / Gold) takes the warriors in order, starting from the one with the highest rate until the amount of gold in its hand is exhausted.

Random Approach: A random number of 1 or 0 is generated at each level. If 0 is obtained, no element is selected from that level. If 1 is obtained, a new random number is generated to select an element at that level. If there are enough gold to buy that member, it joins the army.

Run times

The slowest random approach function among the 3 algorithms. While the speed order varies between Dynamic Programing and Greedy Approach, generally Greedy Approach runs faster.

External Libraries

```
io.Fil ,io.IOException ,java.nio.charset.StandardCharsets, ArrayList,
Random ,Scanner
```

References

I used the knapsack code given in the lab lesson for dynamic Programming Approach. For greedy Approach, I used the fractional knapsack method, whose algorithm is described in the lesson. also this example helped me develop idea https://www.geeksforgeeks.org/job-sequencing-problem/. For knapsack https://www.geeksforgeeks.org/printing-items-01-knapsack/.