Complex problem solving using C++ [defeating n number of monsters]

Project Overview

The "Monster Defeat" algorithm is a coding challenge focused on determining the minimum number of moves required to defeat a monster based on given constraints. This project, implemented in C++, demonstrates advanced algorithm design and problem-solving skills.

Problem statement :

While playing an RPG game, you were assigned to complete one of the hardest quests in this game. There are n monsters you'll need to defeat in this quest. Each monster i is described with two integer numbers – power and bonus. To defeat this monster, you'll need at least power experience points. If you try fighting this monster without having enough experience points, you lose immediately. You will also gain bonus experience points if you defeat this monster. You can defeat monsters in any order. The quest turned out to be very hard - you try to defeat the monsters but keep losing repeatedly. Your friend told you that this quest is impossible to complete. Knowing that, you're interested, what is the maximum possible number of monsters you can defeat?

Input: The first line contains an integer, n, denoting the number of monsters. The next line contains an integer, e, denoting your initial experience. Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, power which represents power of the corresponding monster. Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, bonus, which represents bonus for defeating the corresponding monster.

Sample cases: Input

2

123

78

130

10

0

Output : 2

Output description : Initial experience level is 123 points. Defeat the first monster having power of 78 and bonus of 10. Experience level is now 123+10=133. Defeat the second monster

Input 2 :

3

100

101

100

304

100

1

524

Output ; 2

Output description : Initial experience level is 100 points. Defeat the second monster having power of 100 and bonus of 1. Experience level is now 100+1=101. Defeat the first monster having power of 101 and bonus of 100. Experience level is now 101+100=201. The third monster can't be defeated.

Approach

1. Input Handling:
    - Number of Monsters: Read the total number of monsters.
    - Initial Experience: Read the initial experience level.
    - Monster List: Read the power and bonus values for each monster.
2. Algorithm Design:
    - Sorting: Sort the list of monsters based on their power in ascending order.
    - Defeating Monsters: Iterate through the sorted list and defeat as many monsters as possible based on the current experience level.

Explanation:

3. Sorting: Monsters are sorted by their power to ensure the algorithm attempts to defeat weaker monsters first, maximizing the number of defeats.
4. Defeating Logic: For each monster, check if the current experience is sufficient to defeat the monster. If so, increase the experience by the monster's bonus and count the defeat. Stop if the current experience is insufficient for the next monster.

Testing:

- Validation: The algorithm is tested with various inputs to ensure that it correctly calculates the number of monsters that can be defeated based on experience and bonuses.
- Test Cases: Include different numbers of monsters, varying power and bonus values, and different initial experience levels.

Features

- Efficiency: Utilizes sorting and iteration to efficiently handle the problem.
- Modularity: Clear and organized C++ code with functions for key operations.
- Documentation: Code includes comments and explanations for clarity.

Challenges and Learnings

- Challenges: Ensuring the algorithm handles different constraints and edge cases efficiently.
- Learnings: Improved skills in sorting algorithms, vector manipulation, and C++ programming.

Conclusion

This project highlights the ability to solve practical problems using C++ and demonstrates skills in algorithm design and implementation. It shows the capability to develop efficient solutions and handle real-world constraints.