1. Mô tả thuật toán

Subtask 1 ($n \leq 20$)

- **Thuật toán sử dụng: Vét cạn (Brute Force)**, cụ thể là phương pháp **vét cạn theo tập con (subset brute force)**, có thể cài đặt bằng quay lui (backtracking) hoặc bitmask.
- **Lý do sử dụng:** Với $n \leq 20$, số lượng chiến binh là rất nhỏ. Tổng số tập con (các cách chọn chiến binh) có thể có là 2^n. Với $n = 20$, $2^{20} \approx 1 \text{ triệu}$, đây là một số lượng phép toán đủ nhỏ để máy tính thực thi trong thời gian giới hạn cho phép (thường là 1-2 giây).
- **Mô tả các bước:**
  i. Duyệt qua tất cả 2^n tập con có thể có của n chiến binh. (Sử dụng 2 vòng lặp `for` lồng nhau nếu dùng quay lui, hoặc 1 vòng lặp từ 0 đến $2^n - 1$ nếu dùng bitmask).
  ii. Với mỗi tập con được tạo ra:
    - **Kiểm tra tính hợp lệ:** Kiểm tra xem tập con này có thỏa mãn điều kiện của đề bài hay không. Cách kiểm tra hiệu quả: Sắp xếp các chiến binh trong tập con theo `Age` tăng dần, nếu `Age` bằng nhau thì là theo 'Intelligence' tăng dần. Sau đó, duyệt qua tập con đã sắp xếp và đảm bảo rằng `Intelligence` cũng là một dãy không giảm.
    - **Tính tích:** Nếu tập con hợp lệ, tính tích `Power` của tất cả các chiến binh trong tập con đó.
  iii. Lưu lại giá trị tích `Power` lớn nhất tìm được trong tất cả các tập con hợp lệ.
  iv. Sau khi duyệt hết 2^n tập con, giá trị lớn nhất đã lưu chính là đáp án.

Subtask 2 ($n \leq 1000$)

- **Áp dụng thuật toán của Subtask 1?**
  - **Không** thể áp dụng.
  - **Giải thích:** Nếu $n = 1000$, độ phức tạp $O(2^n)$ của thuật toán vét cạn sẽ trở thành $O(2^{1000})$, một con số khổng lồ không thể tính toán được. Chúng ta cần một phương pháp hiệu quả hơn.
- **Phương pháp thay thế: Quy hoạch động (Dynamic Programming - DP)**.
- **Giải thích:**
  - Bài toán này có cấu trúc con tối ưu (optimal substructure) và các bài toán con gối nhau (overlapping subproblems). Cụ thể, kết quả tối ưu để chọn một dãy chiến binh kết thúc ở vị trí i có thể được xây dựng từ các kết quả tối ưu của các dãy kết thúc ở vị trí j (với $j < i$).

- Đây là một biến thể của bài toán "Dãy con tăng dài nhất" (Longest Increasing Subsequence - LIS). Tuy nhiên, vì `Power` có thể là số âm, việc nhân thêm một số âm có thể biến một tích âm (tệ nhất) thành một tích dương (tốt nhất) và ngược lại.
- Do đó, tại mỗi bước, chúng ta không chỉ lưu tích `Power` **lớn nhất** mà còn phải lưu cả tích `Power` **nhỏ nhất** (âm nhiều nhất) có thể đạt được.

- **Các bước thực hiện:**
  i. **Sắp xếp:** Sắp xếp danh sách n chiến binh theo `Age` tăng dần. Nếu `Age` bằng nhau, sắp xếp tiếp theo `Intelligence` tăng dần.
  ii. **Khởi tạo:** Tạo 2 mảng quy hoạch động `dp_max_positive` và `dp_min_negative` với độ dài n.
    - `dp_max_positive[i]` sẽ lưu tích `Power` **dương lớn nhất** của một dãy con hợp lệ (thỏa mãn điều kiện `Intelligence` không giảm) kết thúc tại chiến binh i.
    - `dp_min_negative[i]` sẽ lưu tích `Power` **âm nhỏ nhất** của một dãy con hợp lệ kết thúc tại chiến binh i.
  iii. **Vòng lặp DP:** Lặp i từ 0 đến $n-1$:
    - **Trường hợp cơ sở:** Khởi tạo `dp_max_positive[i] = fighters[i].power` và `dp_min_negative[i] = fighters[i].power` (coi như dãy con chỉ có 1 mình chiến binh i).
    - **Vòng lặp lồng:** Lặp j từ 0 đến $i-1$:
      - **Kiểm tra điều kiện:** Nếu `fighters[j].intel <= fighters[i].intel` (thỏa mãn điều kiện, vì `Age` đã được sắp xếp trước đó), ta có thể mở rộng các dãy con kết thúc tại j.
      - **Tính toán tích mới:** Lấy `current_power = fighters[i].power`. Tính 2 giá trị tích có thể có:
        - `prod1 = dp_max_positive[j] * current_power`
        - `prod2 = dp_min_negative[j] * current_power`
      - **Cập nhật DP:**
        - `dp_max_positive[i] = max(dp_max_positive[i], prod1, prod2)`
        - `dp_min_negative[i] = min(dp_min_negative[i], prod1, prod2)`
  iv. **Kết quả:** Đáp án cuối cùng không nhất thiết phải nằm ở `dp_max_positive[n-1]`. Nó là giá trị lớn nhất trong toàn bộ mảng `dp_max_positive`. Khởi tạo `global_max_product = -inf` và cập nhật nó với `dp_max_positive[i]` sau mỗi vòng lặp i.
  v. `global_max_product` chính là kết quả cần tìm.

2. Cài đặt thuật toán

Đây là code cài đặt cho Subtask 2 (đã được chú thích rõ ràng):

```python
import sys

class Warrior:
    def __init__(self, age, intel, power):
        self.age = age
        self.intel = intel
        self.power = power

def solve():
    n_line = sys.stdin.readline()
    n = int(n_line)

    fighters = []
    for _ in range(n):
        a, i, p = map(int, sys.stdin.readline().split())
        fighters.append(Warrior(a, i, p))

    # Sắp xếp danh sách theo Age rồi theo Intelligence
    fighters.sort(key=lambda warrior: (warrior.age, warrior.intel))

    # Khởi tạo mảng DP để lưu kết quả tốt nhất và kết quả tệ nhất đạt được
    dp_max_positive = [-float('inf')] * n
    dp_min_negative = [float('inf')] * n

    # Biến lưu kết quả cuối cùng
    global_max_product = -float('inf')

    # Vòng lặp DP chính, O(n^2)
    for i in range(n):
        # Base case: Dãy con có thể chỉ chứa chính chiến binh i
        dp_max_positive[i] = fighters[i].power
        dp_min_negative[i] = fighters[i].power

        # Thử mở rộng các dãy con kết thúc tại j (với j < i)
        for j in range(i):
            # Kiểm tra điều kiện (Age đã được sắp xếp, chỉ cần check Intelligence)
            if fighters[j].intel <= fighters[i].intel:

                # Lấy các trạng thái ở bước j
                prev_max = dp_max_positive[j]
                prev_min = dp_min_negative[j]

                # Tính 2 trạng thái DP hiện tại có thể có
```

```
                prod1 = prev_max * fighters[i].power
                prod2 = prev_min * fighters[i].power

                # Cập nhật giá trị tốt nhất và tệ nhất cho i
                dp_max_positive[i] = max(dp_max_positive[i], prod1, prod2)
                dp_min_negative[i] = min(dp_min_negative[i], prod1, prod2)

        # Kết quả lớn nhất có thể kết thúc tại i
        global_max_product = max(global_max_product, dp_max_positive[i])

    print(int(global_max_product))

if __name__ == "__main__":
    solve()
```

3. Phân tích độ phức tạp

Subtask 1 (Vét cạn - Bitmask)

- **Độ phức tạp thời gian (Time Complexity):** $O(2^n \times n \log n)$
  - $O(2^n)$: Để duyệt qua tất cả các tập con.
  - $O(n \log n)$: Với mỗi tập con (có tối đa n phần tử), ta cần $O(n \log n)$ để sắp xếp theo `Age` và $O(n)$ để kiểm tra `Intelligence`, do đó tổng cộng là $O(n \log n)$ cho mỗi tập con.
- **Độ phức tạp không gian (Space Complexity):** $O(n)$
  - Cần không gian $O(n)$ để lưu trữ tạm thời một tập con đang xét.

Subtask 2 (Quy hoạch động)

- **Độ phức tạp thời gian (Time Complexity):** $O(n^2)$
  - $O(n \log n)$: Cho bước sắp xếp ban đầu.
  - $O(n^2)$: Cho 2 vòng lặp `for` lồng nhau của quy hoạch động (vòng lặp i chạy n lần, vòng lặp j chạy trung bình $n/2$ lần).
  - Tổng thời gian là $O(n \log n + n^2)$, rút gọn còn $O(n^2)$ vì n^2 lớn hơn.
- **Độ phức tạp không gian (Space Complexity):** $O(n)$
  - Cần $O(n)$ để lưu danh sách `fighters` đã sắp xếp.
  - Cần $O(n)$ cho mảng `dp_max_positive` và $O(n)$ cho mảng `dp_min_negative`.
  - Tổng không gian là $O(n + n + n) = O(n)$.