1 UVA Problem 10154: Weights and Measures

Background

I know, up on top you are seeing great sights, But down at the bottom, we, too, should have rights. We turtles can't stand it. Our shells will all crack! Besides, we need food. We are starving!" groaned Mack. Mack, in an effort to avoid being cracked, has enlisted your advice as to the order in which turtles should be dispatched to form Yertle's throne. Each of the ve thousand, six hundred and seven turtles ordered by Yertle has a different weight and strength. Your task is to build the largest stack of turtles possible

Input

Standard input consists of several lines, each containing a pair of integers separated by one or more space characters, specifying the weight and strength of a turtle. The weight of the turtle is in grams. The strength, also in grams, is the turtle's overall carrying capacity, including its own weight. That is, a turtle weighing 300g with a strength of 1000g could carry 700g of turtles on its back. There are at most 5,607 turtles.

Output

Your output is a single integer indicating the maximum number of turtles that can be stacked without exceeding the strength of any one.

1.1 Mathematical Formulation

Given an input of N turtles in the format w_i, s_i for $1 \leq i \leq N$ where w_i is the wieght and s_i the strength of the i^{th} trutle. Let then the difference d_i be $w_i - s_i$ for each turtle and W.L.O.G. say that the turtles are sorted based off of their d_i in increasing order. i.e. $d_1 \leq d_2 \leq ... \leq d_N$. This algorithm will use the Look-Ahead greedy approach to determine the maximum number of turtles that can be stacked on top of eachother.

1.2 Solution

Important Confusing Data Structures:

- int[ ] **allWeights** : All of the correspoding weights of the turtles
- int[ ] **allDifferences** : All of the corresponding differences of the turtles
- int **space** : The least amount of difference currently available

The main functionality of this algorithm will be to

Algorithm 1 Method

```
procedure MAIN
    allWeights ← from input
    allDifferences ← from input
    intspace ← allDifferences[0]
    intc ← 1
    for i ∈ 1, N do
        intw ← allWeights[i] // current turtle weight
        if w ≤ space then
            if (space − w) ≤ allDifferences[i] then
                space -= w
            else
                space ← allDifferences[i]
        else
            continue;
    print(c);
```

1.3 Correctness

Proposition 1.

I know this is incorrect. Figured it out after writing everything out.

Proof. Counter Example: $t_1 = (100, 300)$, $t_2 = (250,350) \implies d_1 = 200$ and $d_2 = 100$. Going through this algorithm we will get this order as the sorted order and then see that we cannot put t_2 on top of t_1 therefore we have only 1, however we can do t_2 on top of t_1. There-in-lies the contradiction. □

1.4 Analysis

Proposition 2. *The space complexity of this algorithm is* ***O(N)***

Proof.
This is due to the fact that all of our data is stored in data structures:

- int[ ] **allWeights** : All of the correspoding weights of the turtles **O(N)**
- int[ ] **allDifferences** : All of the corresponding differences of the turtles **O(N)**
- int **space** : The least amount of difference currently available **O(1)**

Giving us a space complexity of **O(N)**

□

Proposition 3. *The time complexity of this algorithm is* ***O(N·log(N))***

Proof. This is the case because our algorithm will sort in $N \cdot log(N)$ time and then it will do one linear search (N) to determine the number of valid.

Giving us a time complexity of **O(N·log(N))**

□

2 Book Problem 6.2, hackers jobs

2.1 Part A

Counter Example: let (10, 1, 10, 10) ∈ L and (5,50,100,1) ∈ H ∴ we would pick 0 + 50 + 10 + 10 = 70 as our income where the best would be 10 + 0 + 100 + 10 = 120.

2.2 Part B

Algorithm:

3 Book Problem 6.7

A quick look here and we will note that we should be solving for the subsection of what is the optimal days to buy and sell within interval (1,i) where $i \leq N$. Therefore we must calculate for the best day to buy before selling on a particular day. i.e. min(i) then using this we can create an optimal sell value list for what value can you get the most money if you sell on this day. Then searching through the sell list we find that of the greatest value and trace back to its best buy date. All of these are linear passes.

4 hw05

Completed Maximum sum, can discuss this.