<https://leetcode.com/problems/maximum-units-on-a-truck/description>

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]:

numberOfBoxesi is the number of boxes of type i.

numberOfUnitsPerBoxi is the number of units in each box of the type i.

You are also given an integer truckSize, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.

Return *the****maximum****total number of****units****that can be put on the truck.*

**Example 1:**

**Input:** boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4

**Output:** 8

**Explanation:** There are:

- 1 box of the first type that contains 3 units.

- 2 boxes of the second type that contain 2 units each.

- 3 boxes of the third type that contain 1 unit each.

You can take all the boxes of the first and second types, and one box of the third type.

The total number of units will be = (1 \* 3) + (2 \* 2) + (1 \* 1) = 8.

**Example 2:**

**Input:** boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10

**Output:** 91

**Constraints:**

1 <= boxTypes.length <= 1000

1 <= numberOfBoxesi, numberOfUnitsPerBoxi <= 1000

1 <= truckSize <= 10^6

**Attempt 1: 2025-05-04**

**Solution 1: Sorting + Greedy (10 min)**

class Solution {

    public int maximumUnits(int[][] boxTypes, int truckSize) {

        Arrays.sort(boxTypes, (a, b) -> b[1] - a[1]);

        int totalUnits = 0;

        int remain = truckSize;

        for(int[] boxType : boxTypes) {

            if(remain <= 0) {

                break;

            }

            int boxTaken = Math.min(remain, boxType[0]);

            totalUnits += boxTaken \* boxType[1];

            remain -= boxTaken;

        }

        return totalUnits;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to Deepseek**

This problem requires us to maximize the number of units we can fit on a truck given boxes of different unit counts. The optimal strategy is to greedily pick boxes with the highest units first.

**Approach**

Sort the box types array in descending order based on the number of units per box.

Iterate through the sorted array and take as many boxes as possible (either all available or until truck is full).

Calculate the total units added to the truck.

**Solution Code**

class Solution {

public int maximumUnits(int[][] boxTypes, int truckSize) {

// Sort the boxTypes array in descending order of number of units per box

Arrays.sort(boxTypes, new Comparator<int[]>() {

public int compare(int[] a, int[] b) {

return b[1] - a[1];

}

});

int totalUnits = 0;

int remainingSpace = truckSize;

for (int[] boxType : boxTypes) {

int boxes = boxType[0];

int unitsPerBox = boxType[1];

if (remainingSpace <= 0) break;

int boxesTaken = Math.min(boxes, remainingSpace);

totalUnits += boxesTaken \* unitsPerBox;

remainingSpace -= boxesTaken;

}

return totalUnits;

}

}

**Explanation**

Sorting: We sort the boxTypes array in descending order based on the units per box. This ensures we always consider the most valuable boxes first.

Greedy Selection: We iterate through the sorted array, taking as many boxes as we can from each type (either all available boxes or until the truck is full).

Tracking: We keep track of the remaining space on the truck and add the units from each box we take to our total.

Termination: The loop breaks early if we fill the truck before processing all box types.

This approach efficiently maximizes the units on the truck by always prioritizing boxes with higher unit counts, which is optimal for this problem. The time complexity is dominated by the sorting step, which is O(n log n), where n is the number of box types.

**Refer to**

[L2279.Maximum Bags With Full Capacity of Rocks (Ref.L1710)](note://WEB0e1a5d6315e1ec88d51fabd87b258e3d)