Solution 1

```swift
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Program {
    // O(n^2) time | O(1) space
    func diskStacking(disks: inout [[Int]]) -> [[Int]] {
        disks.sort(by: { $0[2] < $1[2] })

        var heights = disks.map { $0[2] }
        var previousIndices = Array(repeating: -1, count: disks.count)
        var maximumHeightIndex = 0

        for i in 1 ..< disks.count {
            let currentDisk = disks[i]

            for j in 0 ..< i {
                let previousDisk = disks[j]

                if areValidDimensions(previousDisk, currentDisk) {
                    if heights[i] <= heights[j] + currentDisk[2] {
                        heights[i] = heights[j] + currentDisk[2]
                        previousIndices[i] = j
                    }
                }
            }

            if heights[i] >= heights[maximumHeightIndex] {
                maximumHeightIndex = i
            }
        }

        return buildSequence(disks, previousIndices, &maximumHeightIndex)
    }

    func areValidDimensions(_ previousDisk: [Int], _ currentDisk: [Int]) -> Bool {
        return previousDisk[0] < currentDisk[0] && previousDisk[1] < currentDisk[1] && previousDisk[2] < currentDisk[2]
    }

    func buildSequence(_ disks: [[Int]], _ previousIndices: [Int], _ currentIndex: inout Int) -> [[Int]] {
        var sequence = [[Int]]()

        while currentIndex != -1 {
            sequence.insert(disks[currentIndex], at: 0)
            currentIndex = previousIndices[currentIndex]
        }

        return sequence
    }
}

```