Solution 1

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

import java.util.*;

class Program {
  // O(n^2) time | O(n) space
  public static List<Integer[]> diskStacking(List<Integer[]> disks) {
    disks.sort((disk1, disk2) -> disk1[2].compareTo(disk2[2]));
    int[] heights = new int[disks.size()];
    for (int i = 0; i < disks.size(); i++) {
      heights[i] = disks.get(i)[2];
    }
    int[] sequences = new int[disks.size()];
    for (int i = 0; i < disks.size(); i++) {
      sequences[i] = Integer.MIN_VALUE;
    }
    int maxHeightIdx = 0;
    for (int i = 1; i < disks.size(); i++) {
      Integer[] currentDisk = disks.get(i);
      for (int j = 0; j < i; j++) {
        Integer[] otherDisk = disks.get(j);
        if (areValidDimensions(otherDisk, currentDisk)) {
          if (heights[i] <= currentDisk[2] + heights[j]) {
            heights[i] = currentDisk[2] + heights[j];
            sequences[i] = j;
          }
        }
      }
      if (heights[i] >= heights[maxHeightIdx]) {
        maxHeightIdx = i;
      }
    }
    return buildSequence(disks, sequences, maxHeightIdx);
  }

  public static boolean areValidDimensions(Integer[] o, Integer[] c) {
    return o[0] < c[0] && o[1] < c[1] && o[2] < c[2];
  }

  public static List<Integer[]> buildSequence(
      List<Integer[]> array, int[] sequences, int currentIdx) {
    List<Integer[]> sequence = new ArrayList<Integer[]>();
    while (currentIdx != Integer.MIN_VALUE) {
      sequence.add(0, array.get(currentIdx));
      currentIdx = sequences[currentIdx];
    }
    return sequence;
  }
}

```