COT 4400: Analysis of Algorithm
Dynamic Programming Group Project (100 pts)
Due: March 20, 2014

This project requires you to first theoretically solve the dynamic programming problem below[1] and then write a program that implements your solution.

You are not allowed to use the Internet. You may only consult approved references[2].
The only people you can work with on this project are your group members.

1. *Problem Description*:

   Suppose you are managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets). The basic question each week is whether to take on a low-stress job or a high-stress job.

   If you select a low-stress job for you team in week i, then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for the team to take on a high-stress job in week i, it is required that they do not job (of either type) in week $i-1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it is okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i-1$.

   So, given a sequence of n weeks, a *plan* is specified by a choice of "low-stress", "high-stress", or "none" for each of the n weeks, with the property that if "high-stress" is chosen for week $i > 1$, then "none" has to be chosen for week $i-1$. (*It is okay to choose a high-stress job in week 1.*) The *value* of the plan is determined in the natural way: for each i, you add l_i to the value if you choose "low-stress" in week i, and you add h_i to the value if you choose "high-stress" in week i. (*You add 0 if you choose "none" in week i.*)

   Given a set of values $l_1, l_2, \ldots, l_n$ and $h_1, h_2, \ldots, h_n$, find a plan of maximum value. (Such a plan will be called *optimal.*)

   Example:

   Suppose $n = 4$, and the values of l_i and h_i are given by the following table.

i	1	2	3	4
l	10	1	10	10
h	5	50	5	1

   The plan of maximum value would be to choose "none" in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of this plan would be 0+50+10+10=70.

   Input/Output

   - The Input: (`input.txt`)
     The input file contains multiple sets of inputs. The input file begins with a single positive integer on a line by itself indicating the number of cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between consecutive inputs.

     The first line of each input contains an integer n, which represents the number of weeks for this instance. The next next line contains n integers, separated by spaces. The i^{th} integer in this line represents the revenue for selecting a low-stress job on day i (i.e., l_i). The next next line contains n integers, separated by spaces. The i^{th} integer in this line represents the revenue for selecting a high-stress job on day i (i.e., h_i).

   - The Output: (`<studentLastName>.txt`)
     The output file should contain two lines for each input.

     (a) The first output line should indicate the value of the plan

     (b) The second output line should indicate the plan. This line should have n characters with one space between each character. The i^{th} character represents the choice made for week i, where L indicates low-stress job, H indicates high-stress job, and N indicates no job.

     There should be a blank line between consecutive outputs.

     Note that if more than one plan yields the the optimal solution then any one will do.

[1]The problem has been adapted from Algorithm Design, by Kleinberg and Tardos.
[2]You must cite all references used.

2. *Deliverables*:

   Please submit a **hard copy**, in-class, of all of the items requested below. Please don't send this by e-mail as I would have to print it out anyways. Please also submit your *code*[3] and your typed project report[4] to Canvas.

   (a) **Theory [60 pts]:** Devise an efficient[5] dynamic programming algorithm that solves the problem. Demonstrate each of the following dynamic programming steps and derive the complexity of your algorithm:

      i. **[20 pts]:** Describe the main idea of your approach including how you break the problem into smaller sub-problems and obtain the principle of optimality.

      ii. **[15 pts]:** Write pseudocode for your dynamic programming algorithm (either a recursive algorithm with memoization OR a bottom-up algorithm that explicitly uses a table).

      iii. **[15 pts]:** Describe a backtracking algorithm that returns a plan with the maximum value.

      iv. **[10 pts]:** Derive the complexity of your dynamic programming algorithm in terms of n and any other appropriate parameters. (prove this)
      For full credit you must present a formal proof justifying the worst-case time complexity and the worst-case space complexity of your algorithm. Note that in a formal proof you must justify each step of the proof sequence. The bound on the worst-case complexity of your algorithm must be as tight as possible.

      Note that for full credit your descriptions must be clear enough that any competent programmer will understand how the algorithm works and can implement your algorithm in their preferred programming language.

   (b) **Implementation [20 pts]:**

      i. **[10 pts]:** Submit a print-out of your code. If you are unable to get your code to compile/run, please state this explicitly.
      Code Requirements[6]:
      - Read the input from a file in the described format.
      - Output the solution in the described format.
      - README file describing how to compile and run your code. If your code requires more than a simple command to compile and run then you must also provide a Makefile and/or shell script.
      - *Optional: You may also submit an executable for the C4 Linux Lab machines*

      ii. **[10 pts]:** Demonstrate your algorithm works correctly by walking through a small example.
      For the walk-through you should select a small example and discuss step-by-step (line-by-line of the pseudocode) how your algorithm produces the output. The example used for your walk-through should be large enough to demonstrate all important aspects of your algorithm, but small enough to be understood by the grader.

   (c) **Results [20 pts]:**

      i. Submit your code to Canvas to be run on a set of inputs devised by the grader.
      Your grade is based on whether or not your code produces the correct answer for all inputs.

   (d) Submit group effort percentages and describe who did what. The effort percentages must be agreed upon by the group. See syllabus to determine how we will use this to compute your individual grade.
   Note that we reserve the right to change your group effort percentages. Group effort percentages will be changed if we think they do not accurately represent the contribution of each student.

[3]Your code must compile and run on the C4 Linux Lab machines

[4]Project report and code will be checked for plagiarism

[5]You will lose points for both your algorithm design grade (i.e., lose points for poor design) and your results grade (i.e., lose points because you are not able to process all inputs) if your algorithm is not efficient.

[6]Failure to follow these requirements may result in a 0 for the code portion of your grade AND a 0 in the results portion of your grade