Homework 4 – solutions

1. Programming assignment

```
def mwis (n, weights):

    #opt_list[i] represents the optimal solution to
    # the problem with weights(1), ..., weights(i+1)

    opt_list = [weights[0], max(weights[0], weights[1])] #base cases.

    #fill in DP table
    for i in range(2,n):
        opt_list.append(max(opt_list[i - 1], opt_list[i - 2] + weights[i]))

    #find the value of the best solution
    sol_value = opt_list[n-1]

    #traceback to find the optimal solution
    sol = []
    i  = n - 1
    while i  > 1:
        if opt_list[i - 1] < opt_list[i - 2] + weights[i] :
            sol.append(i)
            i  -= 2
        else:
            i  -= 1

    if i == 0 :
        sol.append(0)

    if i == 1 :
        if weights[0] > weights[1] :
            sol.append (0)
        else :
            sol.append (1)

    return (opt_list, sol_value, sorted(sol))
```

2. **(Minimarts)** You are managing the construction of minimarts along a stretch of road. The possible sites for the minimarts are given by real numbers $x_1, ..., x_n$, each of which specifies the position along the road measured in miles from its beginning. Assume that the road is a straight line. If you place a minimart at location x_i, your company will make a profit of $r_i > 0$ dollars.

   Regulations require that every pair of minimarts be at least 5 miles apart. You'd like to place minimarts at a subset of the sites so as to maximize total profit, subject to this restriction. The input is given as a list of n pairs $(x_1, r_1), ..., (x_n, r_n)$ where the x_i's are sorted in increasing order.

   Note: *This problem is actually a special case of weighted interval scheduling (that we saw in class). That is, one could solve the problem by setting up a set of intervals, each five miles long, centered at the minimart locations, where the weight of an interval is the profit associated with the corresponding bollboard. This was not the intended solution, but because it is correct, it received full credit. The solution shown below is the one from first principles, and will also receive full credit.*

   (a) You first consider some very simple approaches to the problem, which you realize will not work well. For each of the following approaches, give an example of an input on which the approach does not find the optimal solution:

      i. *Next available location:* put a minimart at $i = 1$. From then on, put a minimart at the smallest index i which is more than five miles from your most recently placed minimart. Here $L = 3$.

Minimart #	1	2
Distance	1	2
Profit	1	100

      ii. *Most profitable first*: Put a minimart at the most profitable location. From then on, place a minimart at the most profitable location not ruled out by your current minimarts. Here $L = 10$.

Minimart #	1	2	3
Distance	2	5	8
Profit	2	3	2

   (b) Give a dynamic programming algorithm for this problem. Analyze the time complexity of your algorithm. To make it easier to present your answer clearly, follow the steps below:

      i. Clearly define the subproblems that you will solve recursively.
      Assume that minimarts are sorted in increasing order of distance from the start of the highway.
      OPT(j) = maximum profit you can obtain using only minimarts $1, 2, ..., j$. For simplicity, define $OPT(0) = 0$.

      ii. Give a recursive formula for the solution to a given subproblem in terms of smaller subproblems. Explain why the formula is correct. For every index j, let $p(j)$ be the highest index of a minimart that is five or more miles closer to the beginning of the highway than minimart j. That is, $p(j) = \max\{i : x_i \leq x_j - 5\}$. When there are no such minimarts , define $p(j)$ to be 0.
      Then we can define $OPT(j)$ recursively:

$$OPT(j) = \max(OPT(j-1), r_j + OPT(p(j)))$$

The formula holds because one of two cases always holds: either j is not in the optimal solution, in which case we are free to choose the best solution among billboards $\{1, ..., j-1\}$, or j is in the optimal solution, in which case all the minimarts between $p(j)+1$ and $j-1$ are ruled out by the presence of minimart j, but we are free to choose the best solution among minimarts $\{1, ..., p(j)\}$ (and we add to that optimum the profit obtained from minimart j).

iii. Give a bottom-up pseudocode for an algorithm that calculates the profit of the optimal solution. Analyze the time complexity of your algorithm. This algorithm computes the recursive formula above, bottom up. Note that we store the optimum as well as whether or not the optimum was obtained by including j.

Algorithm 1: MINIMART-PROFIT$((x_1, r_1), ..., (x_n, r_n))$

```
▷ OPT is an array of integers, Include is an array of booleans.;
▷ Include[j] indicates whether minimart j was used in the solution with value OPT[j].;
Compute p[1], ..., p[n]. ;
for j ← 1 to n do
    if OPT[j − 1] > r_j + OPT[p[j]] then
        OPT[j] ← OPT[j − 1];
        Include[j] ← FALSE;
    else
        OPT[j] ← r_j + OPT[p[j]];
        Include[j] ← TRUE ;
return OPT[n];
```

The algorithm is correct because of the recursive formula above: the important point is that all the information needed to compute $OPT(j)$ is prepared before execution j of the for loop.

The algorithm takes $\Theta(n)$ time when the input is sorted by x_i. On an unsorted input, the sorting makes the total time be $\Theta(n \log n)$.

iv. Give pseudocode for an algorithm that uses the information computed in the previous part to output an optimal solution. Analyze the time complexity of your algorithm.

Algorithm 2: MINIMART$((x_1, r_1), ..., (x_n, r_n))$

```
Compute p, OPT, Include as in previous algorithm.;
final ← empty list;
j ← n;
while j > 0 do
    if Include[j] == TRUE then
        Add j to final ▷ Include j and skip to next available minimart ;
        j ← p[j];
    else
        j ← j − 1 ▷ Don't include j;
return final;
```

The algorithm runs in time $\Theta(n)$.