# The algorithm

Iterative-Compute-Opt;

M[0] = 0;

S[0] = empty; # array that holds the current optimal solution (pairs made so far) at each step

for i = 1, 2, …, (n choose 2) # each cell of M represents a potential pair

if (vi + p(i)) > M[i-1]:

M[i] = vi + p(i);

S[i] = S[p(i)].add(i) # for solution at step i, add student i onto solution at step p(i)

else:

M[i] = M[i-1];

S[i] = S[i-1] # for solution at step i, keep the previous step’s solution

endif

endfor

return S[n choose 2] as the optimal set of all pairs

# Proof of Correctness

For purposes of comparison, let O be an optimal list of intervals and A be our solution. We need to prove |A|=|O|. First we will show inductively that our greedy algorithm solution A “stays ahead” of O and that it is doing better in a step-by-step fashion.

Notation:

* i1, … , ik is the list of partners’ availabilities in A in the order they were added to A.
* j1, … , jm is the list of partners’ availabilities in O.

We need to prove k=m. Assume that the availabilities in O, like A, are also ordered by increasing finish time.

## Base Case

Need to prove f(i1) ≤ f(j1) and f(i2) ≤ f(j2). This is true because our greedy algorithm chooses people with the earliest possible finish times.

## Inductive Case

∀ r>1, we need to prove f(ir+1) ≤ f(jr+1) assuming the inductive hypothesis f(ir) ≤ f(jr). We know that our greedy algorithm when attempting to choose a partner ir+1 for a student ir chooses the student with the earliest-ending overlapping availability; and that when it is trying to initiate a new pair on some student ir+1 (after a pair – the second student of which is ir – was just formed, or after a pairing attempt failed for ir, someone prior), it once again chooses the student with the earliest-ending availability. We thus see that our algorithm always stays ahead of any optimal solution. f(ir+1) ≤ f(jr+1).

## Proof of Optimality

We will prove it by contradiction. If A is not optimal, then an optimal list O must have more pairs, that is, we must have m>k. Applying what we just proved, we get that f(ik) ≤ f(jk). Since m>k, there must be a jk+1 and a jk+2 in O. These availabilities end after jk ends, and hence after ik ends. So after deleting all eligible partners and everyone whose availability doesn’t overlap with anyone else’s (i1, … , ik), the list of possible availabilities still contains jk+1 and jk+2. But the algorithm stops with availability ik, and it is only supposed to stop when R is empty – a contradiction.

# Runtime Analysis

Iterating through each element of R is O(n). Finding the first availability that overlaps with each element is also O(n). Thus, The entire algorithm is O(n \* n) = **O(n2)**.