# The Algorithm

Iterative-Compute-Opt

M[-1]= 0

M[0]= 0

for j = 1, 2, . . . , n

M[j]= max(vj + M[j-2], M[j − 1])

endfor

return M[n]

# Proof of Correctness

First, some terminology: vj means salary at shift j.

Observe that for an optimal solution O, shift n (the last shift) either belongs or doesn’t belong to O. If n ∈ O, then n-1 ∉ O because of labor laws. Moreover, if n ∈ O, then O *must* include an optimal solution to the problem consisting of shifts {1, … , n-2}. On the other hand, if n ∉ O, then O simply equals to the optimal solution to the problem consisting of shifts {1, … , n-1}. We summarize this in a formula that essentially says ∀ j, either j ∈ Oj, in which case Max\_Salary = vj + Max\_Salary(j-2), or j ∉ Oj, in which case Max\_Salary = Max\_Salary(j-1). So

Max\_Salary(j) = max(vj + Max\_Salary(j-2), Max\_Salary(j-1))

We will now prove by **strong induction** that the algorithm above returns the optimal answer.

## Base Case

We want to prove M[1] returns the maximum amount of money Alice can earn if there were only the first shift available to her. By definition,

M[1] = max(v1 + M[-1], M[0])

= max(v1 + 0, 0)

= max(v1, 0)

= v1

Which is correct because if she only had the first shift available, that’s the most she could earn.

## Inductive Case

We need to prove M[j]= Max\_Salary(j).

Since we’re using strong induction, we can assume that ∀ i < j, M[i] is the maximum salary Alice can earn if she only had the first i shifts available to her. Thus,

M[j]= max(vj + M[j-2], M[j − 1])

= max(vj + Max\_Salary[j-2], Max\_Salary[j − 1])

Which was our definition given above (before the base case was proved).

# Runtime Analysis

Iterating through the M array is O(n). Calculating vj + M[j-2] is O(1) because array lookup is O(1). Looking up M[j-1] is also O(1). Getting the max of the two is also O(1). In total, the algorithm is **O(n)**.