Tutorial 10 solutions

1. a. If *len* is odd, then the first player chooses a letter, and wants the letter that will maximizes his/her total. If *len* is even, then the second player chooses a letter, and (s)he wants to minimize the first player's total. Let us denote the k^{th} letter of the string by $S[k]$.

$$V[i, len] = \begin{cases} \max\{V[i+1, len-1] + val(S[i]), V[i, len-1] + val(S[i+len])\} & \text{if } len \text{ is odd.} \\ \min\{V[i+1, len-1], V[i, len-1]\} & \text{if } len \text{ is even.} \end{cases}$$

   b. The base cases are simple: if $len = 0$, then the first player has already finished playing, and can get 0 more points. So

$$V[1, 0] = V[2, 0] = V[3, 0] = \cdots = V[2n, 0] = 0$$

   c. We will keep the best moves for the first player in a separate table M. Here is an iterative solution (for a change).

```
Algorithm BestPlayer1Strategy(S)
  n ← length[S] / 2

  for i ← 1 to 2n do
    V[i,0] ← 0

  for len ← 1 to 2n-1 do
    for i ← 1 to 2n - len do
      if len is odd then
        choice1 ← val(S[i]) + V[i+1,len-1]
        choice2 ← val(S[i+len]) + V[i,len-1]
        if (choice1 > choice2)
          V[i,len] ← choice1
          M[i,len] ← "ChooseFirst"
        else
          V[i,len] ← choice2
          M[i,len] ← "ChooseLast"
      else
        V[i,len] ← min { V[i+1,len-1], V[i,len-1] }
```

   Then, when the first player is playing and letters in positions i to $i + len$ are all that are left, (s)he simply plays the move indicated by $M[i, len]$.

2. a. $$\begin{aligned} With[i] &= w_i + Without[i-1] \\ Without[i] &= \max\{ With[i-1], Without[i-1]\} \end{aligned}$$

b.

```
Algorithm LargestIndependentSetInAPath(V, W)
  With ← new table with len(V) elements, initialized to -1
  Without ← new table with len(V) elements, initialized to -1

  return BuildLargestIndependentSet(n)

//
// Assume this algorithm has access to V, W, With and Without
//
Algorithm LISIPHWith(n)
  if n = 0 then
    return 0

  if With[n] = -1 then
    With[n] ← w_n + LISIPHWithout(n-1)

  return With[n]

//
// Assume this algorithm has access to V, W, With and Without
//
Algorithm LISIPHWithout(n)
  if n = 0 then
    return 0

  if Without[n] = -1 then
    Without[n] ← max(LISIPHWith(n-1), LISIPHWithout(n-1))

  return Without[n]

//
// Assume this algorithm has access to V, W, With and Without
//
Algorithm BuildLargestIndependentSet(n)
  choice ← "either"
  while (n > 0) do
    if choice = "either" and LISIPHWith(n) > LISIPHWithout(n) then
      //
      // Use V[n] in the independent set
      //
      add V[n] to the output
```

```
    choice ← "without"
    n ← n - 1
  else
    //
    // Do not use V[n] in the independent set
    //
    choice ← "either"
    n ← n - 1
end while
```