Rod cutting summary

- The problem: Given a rod of length n and a table of prices $p[i]$ for $i = 1, 2, 3, ..., n$, determine the maximal revenue obtainable by cutting the rod in integer pieces and selling them.

- Notation and choice of subproblem: For an integer x, we denote by $maxrev(x)$ the maximal revenue obtainable by cutting up a rod of length x. To solve our problem we call $maxrev(n)$.

- For simplicity, we'll sssume the price array $p[]$ and the length of the rod n are global variables

- Recursive definition of $maxrev(x)$:

```
@returns the max revenue obtainable from a rod of length x, where x is an int
MAXREV(x)
if (x <= 0): return 0
maxopt = -∞
for i = 1; i <= x; i = i + 1
    // first cut of length i
    opt = p[i] + maxrev(x - i)
    if opt > maxopt : maxopt = opt
return maxopt
```

- Why correct? It tries *all* possibilities for first cut and recurses on the rest—which is correct because it has optimal substructure (why?)

- Dynamic programming, recursive (top-down) with memoization:

Create a table of size $n + 1$, where $table[i]$ will store (the result of) $maxrev(i)$. Initialize $table[i] = 0$ for all $i = 0..n$. Call $maxrevDP(n, table)$ and return the result.

```
@returns the max revenue obtainable from a rod of length x, where x is an int
MAXREVDP(x, table)
if (x <= 0): return 0
if table[x] ≠ 0: return table[x]
maxopt = -∞
for i = 1; i <= x; i = i + 1
    opt = p[i] + maxrevDP(x - i, table)
    if opt > maxopt : maxopt = opt
table[x] = maxopt
return maxopt
```

Running time for `maxrevDP`(n) : $\Theta(n^2)$

- Dynamic programming, iterative (bottom-up):

```
@returns the max revenue obtainable from a rod of length n
MAXREVDP_ITERATIVE()
create table[0..n] and initialize table[i] = 0 for all i = 0..n
for (x = 1; x ≤ n; x = x + 1)
    // find optimal revenue for length x
    for i = 1; i ≤ x; i = i + 1
        // first cut is of length i
        table[x] = max{table[x], p[i] + table[x − i]}
return table[n]
```

Running time for $maxrevDP\_iterative(n)$: $\Theta(n^2)$

- Computing full solution (without storing additional information while filling the table):

```
@param: table[0..n] as computed above, where table[i] stores the maxrev obtainable from
a rod of length i.
@return: prints the set of cuts corresponding to table[n]
FINDCUTS(table)
curLength = n
while (curLength > 1)
    for i = 1; i ≤ curLength; i = i + 1
        // is the value in table[currLength] achieved via a first cut of length i ?
        if table[curLength] = p[i] + table[curLength − i]
            print that a cut of length i was made
            curLength = curLength − i
```

Running time: $\Theta(n^2)$, no extra space

- Computing full solution (with storing additional information while filling the table):

In addition to $table[0...n]$ we use an array $firstcut[0..n]$ where $firstcut[i]$ will store the first cut in $maxrev(i)$. We can extend the `maxrevDP` (either recursive or iterative) to also fill in $firstcut[x]$: when determining that the maximum revenue for x is achieved with the first cut being of length i, we will set $firstcut[x] = i$.

```
@param: table[0..n] as computed above, where table[i] stores the maxrev obtainable from
a rod of length i.
@param: firstcut[0..n] where firstcut[i] stores the first cut in maxrev(i).
@return: prints the set of cuts corresponding to table[n]
FINDCUTS(table, firstCut)
curLength = n
while (curLength > 1)
    print that a cut of length firstCut[curLength] was made
    curLength = curLength − firstCut[curLength]
```

Running time: $\Theta(n)$, with $\Theta(n)$ extra space for $firstcut[]$