```
MEMOIZED-CUT-ROD(p, n)
let r[0 : n] be a new array        // will remember solution values in r
for i = 0 to n
    r[i] = -∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0                 // already have a solution for length n?
    return r[n]
if n == 0
    q = 0
else q = -∞
    for i = 1 to n          // i is the position of the first cut
        q = max {q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r)}
r[n] = q                    // remember the solution value for length n
return q
```

```
BOTTOM-UP-CUT-ROD(p, n)
1 let r[0 : n] be a new array        // will remember solution values in r
2 r[0] = 0
3 for j = 1 to n                     // for increasing rod length j
4     q = -∞
5     for i = 1 to j                 // i is the position of the first cut
6         q = max {q, p[i] + r[j - i]}
7     r[j] = q                       // remember the solution value for length j
8 return r[n]
```

size i is "smaller" than a subproblem of size j if $i < j$. Thus, the procedure solves subproblems of sizes $j = 0, 1, \ldots, n$, in that order.

Line 1 of BOTTOM-UP-CUT-ROD creates a new array $r[0:n]$ in which to save the results of the subproblems, and line 2 initializes $r[0]$ to 0, since a rod of length 0 earns no revenue. Lines 3–6 solve each subproblem of size j, for $j = 1, 2, \ldots, n$, in order of increasing size. The approach used to solve a problem of a particular size j is the same as that used by CUT-ROD, except that line 6 now directly references array entry $r[j-i]$ instead of making a recursive call to solve the subproblem of size $j-i$. Line 7 saves in $r[j]$ the solution to the subproblem of size j. Finally, line 8 returns $r[n]$, which equals the optimal value r_n.

The bottom-up and top-down versions have the same asymptotic running time. The running time of BOTTOM-UP-CUT-ROD is $\Theta(n^2)$, due to its doubly nested