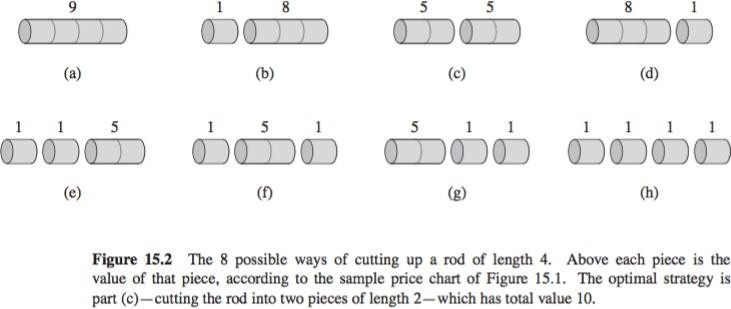
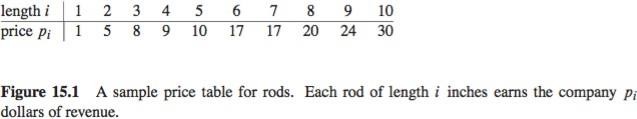
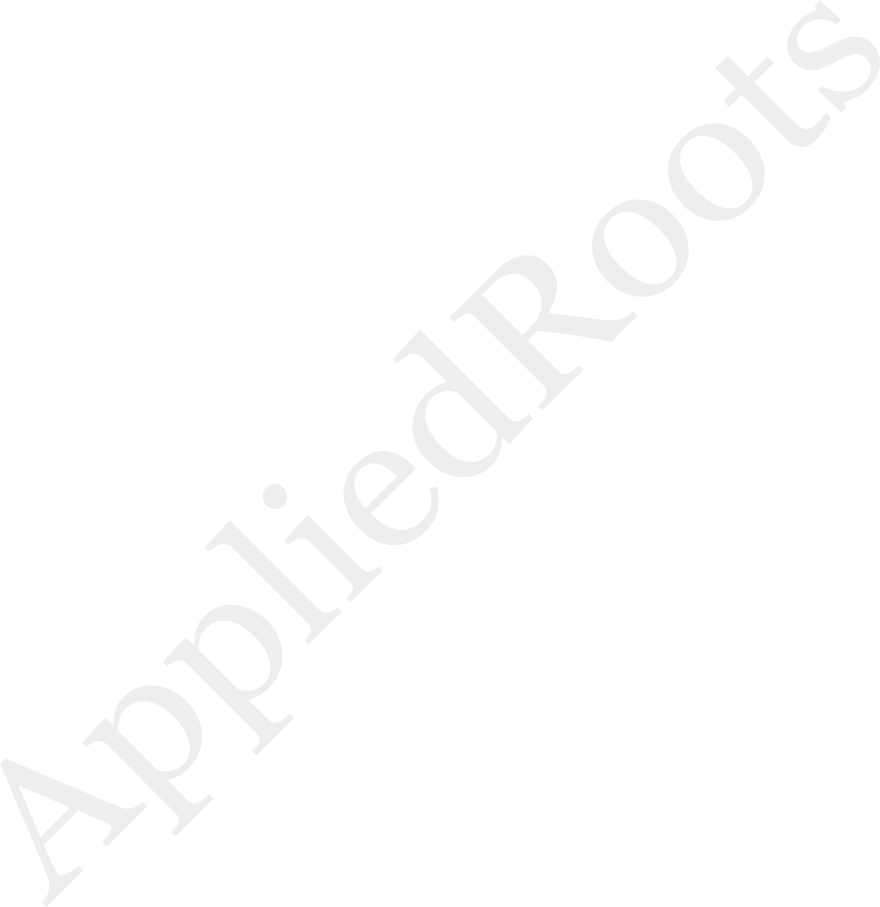
Dynamic programming is a problem solving method that is applicable to many different types of problems. I think it is best learned by example, so we will mostly do examples today.

1. Rod cutting

Suppose you have a rod of length *n*, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length *i* is worth *pi* dollars.



For example, if you have a rod of length 4, there are eight different ways to cut it, and the best strategy is cutting it into two pieces of length 2, which gives you 10 dollars.

**Exercise:** How many ways are there to cut up a rod of length *n*?

**Answer:** 2*n−*1, because there are *n −* 1 places where we can choose to make cuts, and at each place, we either make a cut or we do not make a cut.

Despite the exponentially large possibility space, we can use dynamic programming to write an algorithm that runs in Θ(*n*2).

* 1. **Basic approach**

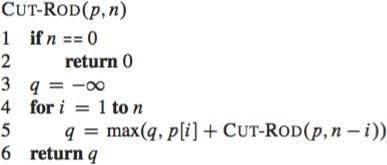
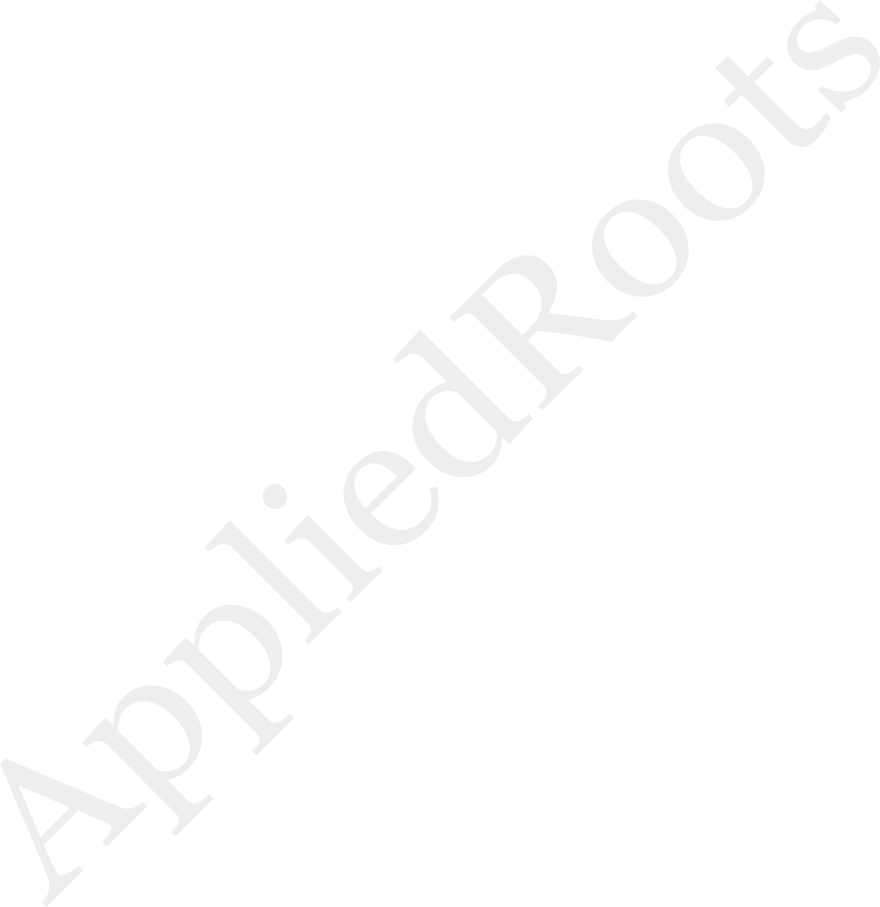
First we ask “what is the maximum amount of money we can get?” And later we can extend the algorithm to give us the actual rod decomposition that leads to that maximum value.

Let *ri* be the maximum amount of money you can get with a rod of size *i*. We can view the problem recursively as follows:

* + - First, cut a piece off the left end of the rod, and sell it.
    - Then, find the optimal way to cut the remainder of the rod.

Now we don’t know how large a piece we should cut off. So we try all possible cases. First we try cutting a piece of length 1, and combining it with the optimal way to cut a rod of length *n −* 1. Then we try cutting a piece of length 2, and combining it with the optimal way to cut a rod of length *n−* 2. We try all the possible lengths and then pick the best one.

We end up with



*rn* = max (*pi* + *r−n i*)

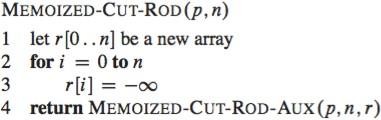
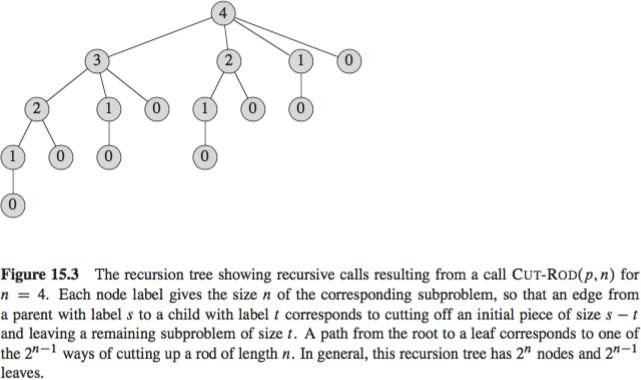
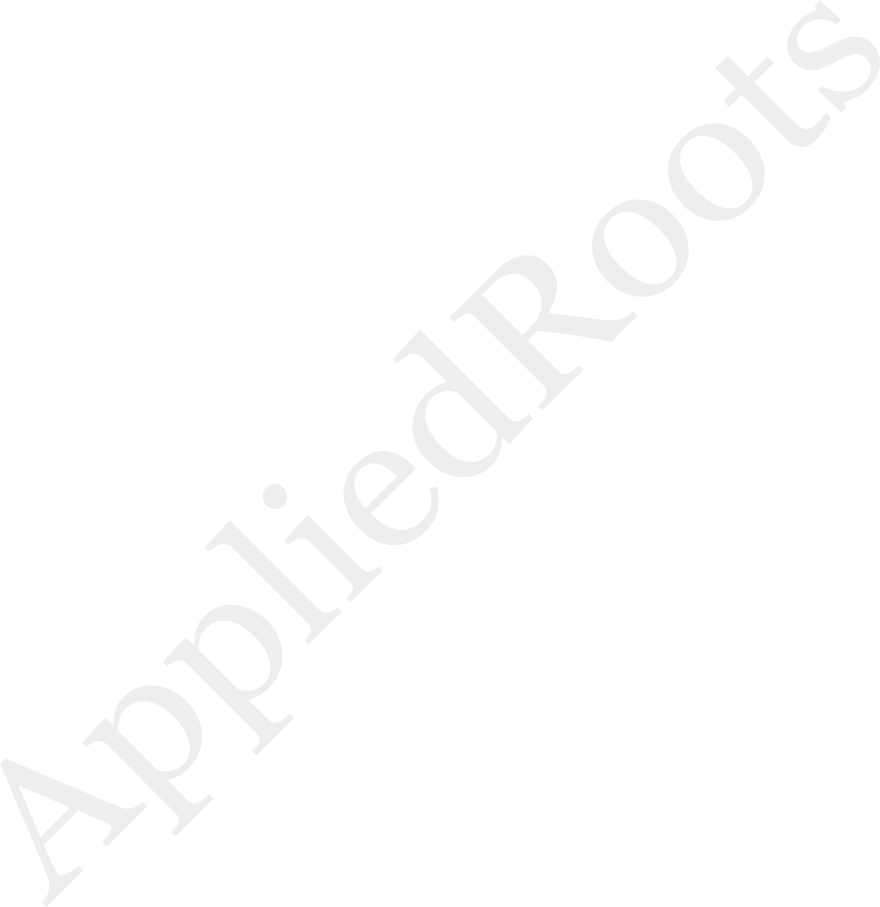
1*≤i≤n*

(Note that by allowing *i* to be *n*, we handle the case where the rod is not cut at all.)

# Naive algorithm

This formula immediately translates into a recursive algorithm.

However, the computation time isridiculous, because there are so many subproblems. Ifyou draw the recursion tree, you will see that we are actually doing a lot of extra work, because we are computing the same things over and over again. For example, in the computation for *n* = 4, we compute the optimal solution for *n* = 1 four times!

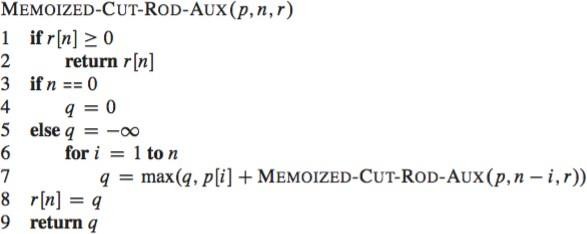
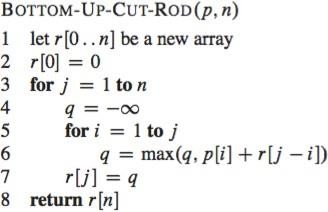
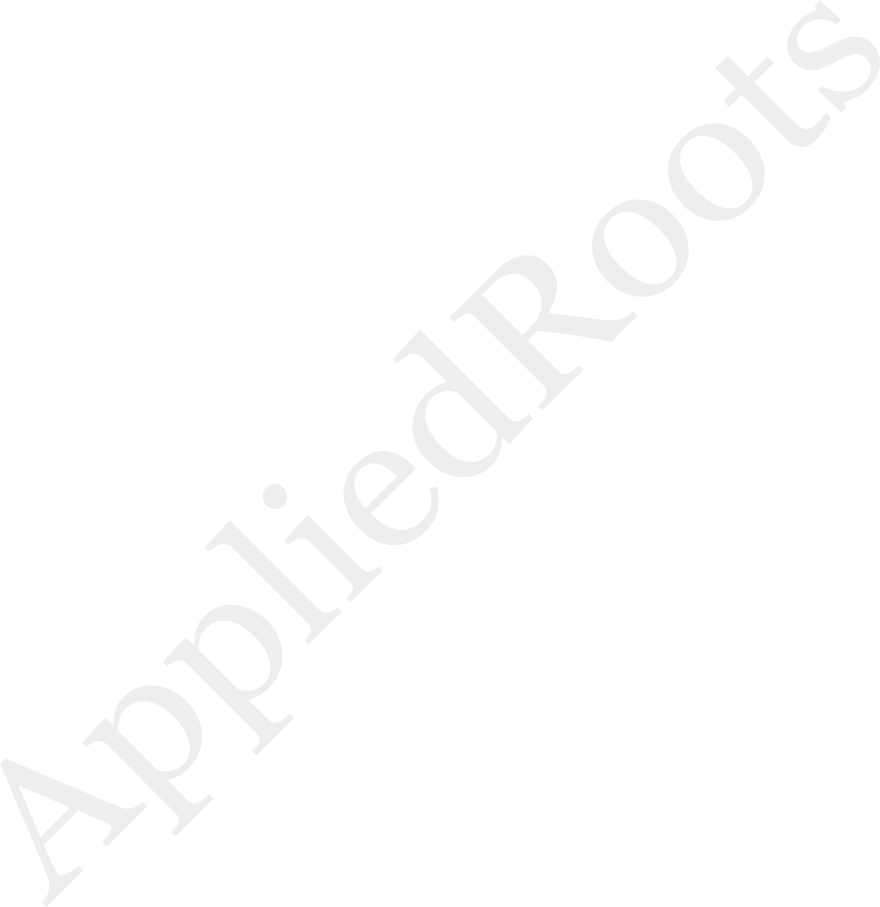


It is much better to compute it once, and then refer to it in future recursive calls.

# Memoization (top down approach)

One way we can do this is by writing the recursion as normal, but store the result of the recursive calls, andif weneed the result ina future recursive call, wecan use the precomputed value. The answer will be stored in r[n].

**Runtime:** Θ(*n*2). Each subproblem is solved exactly once, and to solve a subproblem of size *i*, we run through *i* iterations of the for loop. So the total number of iterations of the for loop, over all recursive calls, forms an arithmetic series, which produces Θ(*n*2) iterations in total.



# Bottom up approach

Here we proactively compute the solutions for smaller rods first, knowing that they will later be used to compute the solutions for larger rods. The answer will once again be stored in r[n].

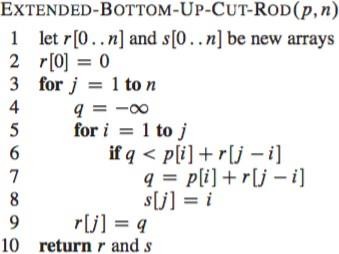
Often the bottom up approach is simpler to write, and has less overhead, because you don’t have to keep a recursive call stack. Most people will write the bottom up procedure when they implement a dynamic programming algorithm.

**Runtime:** Θ(*n*2), because of the double for loop.

# Reconstructing a solution

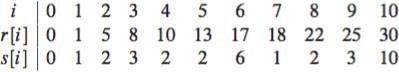
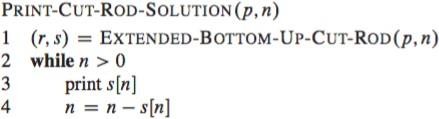
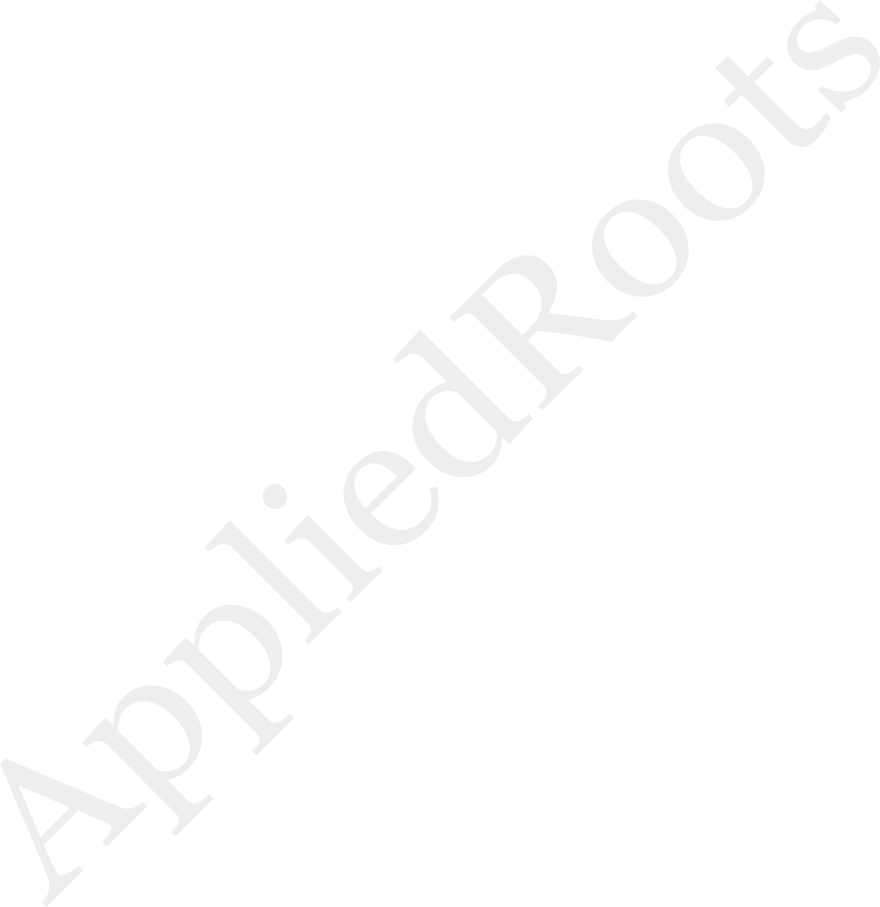
If we want to actually find the optimal way to split the rod, instead of just finding the maximum profit we can get, we can create another array *s*, and let *s*[*j*] = *i* if we determine

that the best thing to do when we have a rod of length *j* is to cut off a piece of length *i*.



Using these values *s*[*j*], we can reconstruct a rod decomposition as follows:

# Answer to example problem



In our example, the program produces this answer: