2. Rod cutting problem

We are given a steel rod of certain length. Also we are given a table of prices for rods of different lengths.

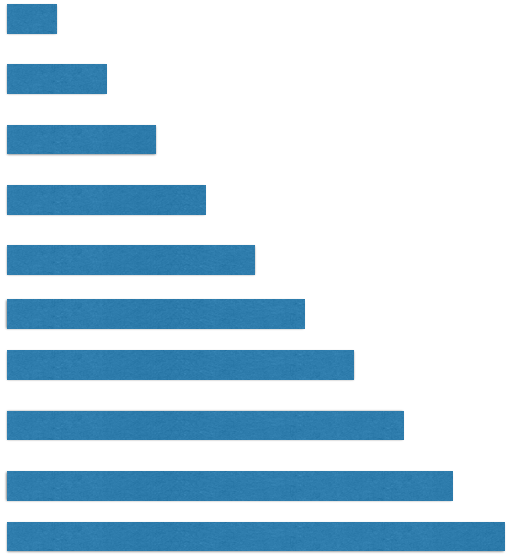
We have to maximize the profit by cutting the given rod into pieces as given in the table.

For example :

We are given rod of length l=6, and the profit table looks like this

L = 6

Length	Price
1	1
2	5
3	8
4	9
5	10
6	14
7	17
8	20
9	24
10	30



1. State

1. State

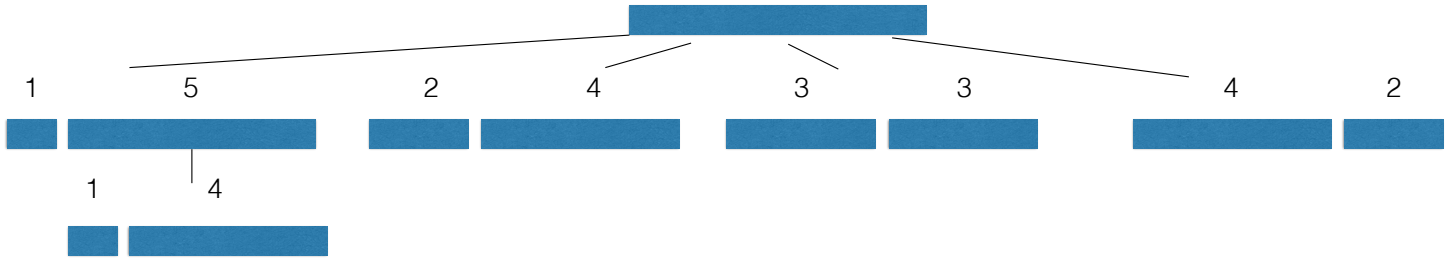
Parameters

l - length

Cost function

maxProfit(l,P) - Returns the maximum profit that can be obtained by cutting the rod of length l into peices.

Cutting the rod



2. Transitions

2. Transitions

maxProfit(l,prices)

Base case

L= 0 , return 0

Transitions

L = 6

P[0]+maxProfit(5,P), P[1]+maxProfit(4,P) …

Optimal choice

We are interested in maximizing the profit. So we have to choose the option which gives us maximum result.

Recurrence relation

maxProfit(L,P) = MAX(P[i]+maxProfit(L-i-1,P)) for i=0,1,2,…..L-1

3. Recursive solution

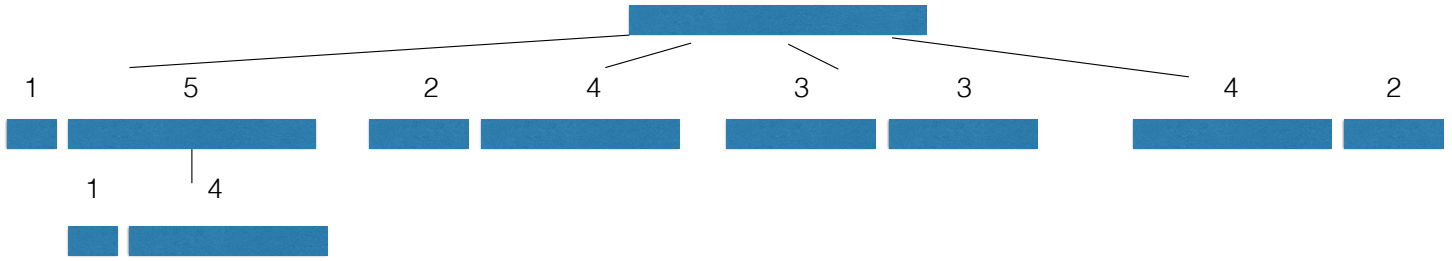
Java

```java
public static int maxProfit(int l,int[] p){
    if(l == 0){
        return 0;
    }
    int max = Integer.MIN_VALUE;
    for(int i=0;i<l;i++){
        max = Math.max(max,p[i]+maxProfit(l-i-1,p));
    }
    return max;
}
```

Python

```python
import sys
def max_profit(l,p):
    if l == 0:
        return 0
    max_p = -sys.maxsize
    for i in range(0, l):
        max_p = max(max_p, p[i] + max_profit(l-i-1,p))
    return max_p
```

Cutting the rod



4. Memoization

4. Memoization

Cache the results to the subproblems

Key = l , which is the input to the subproblem.

Value = maximum profit obtained by cutting the rod of given length

Default value will be -1

Java

```java
public static int maxProfitMemo(int l,int[] p,int[] cache){
    if(l == 0){
        return 0;
    }
    if(cache[l] != -1){
        return cache[l];
    }
    int max = Integer.MIN_VALUE;
    for(int i=0;i<l;i++){
        max = Math.max(max,p[i]+maxProfitMemo(l-i-1,p,cache));
    }
    cache[l]=max;
    return max;
}
```

Java

```java
public static int maxProfitMemo(int l,int[] prices){
    int[] cache = new int[l+1];
    Arrays.fill(cache,-1);
    return maxProfitMemo(l,prices,cache);
}
```

Python

```python
def max_profit_memo(l, p, cache):
    if l == 0:
        return 0
    if cache[l] != -1:
        return cache[l]
    max_p = -sys.maxsize
    for i in range(0, l):
        max_p = max(max_p, p[i] + max_profit(l - i - 1, p))
    cache[l] = max_p
    return max_p
```

5. Bottom up approach

5. Bottom up

Recurrence relation

maxProfit(L,P) = MAX(P[i]+maxProfit(l-i-1,P)) for i=0,1,2,…..L-1

dp[L] = MAX(p[i]+dp[L-i-1]) for i=0,1,2,….L-1

dp[0] = 0

dp[2] depends on the solution at dp[1]

dp[3] depends on the solution at dp[2] and dp[1]

We need to solve this for all l = 1,2,3, …….L,

Java

```java
public static int maxProfitDP(int L,int[] p){
    int[] dp = new int[L+1];
    for(int l=1;l<=L;l++){
        dp[l] = Integer.MIN_VALUE;
        for(int i=0;i<l;i++){
            dp[l] = Math.max(dp[l],p[i]+dp[l-i-1]);
        }
    }
    return dp[L];
}
```

Python

```python
def max_profit_dp(L, p):
    dp = [0 for _ in range(0, L + 1)]
    for l in range(1, L + 1):
        dp[l] = -sys.maxsize
        for i in range(0, l):
            dp[l] = max(dp[l], p[i] + dp[l - i - 1])
    return dp[L]
```

Time and space complexity

Recursive solution

Recurrence relation

maxProfit(L,P) = MAX(P[i]+maxProfit(l-i-1,P)) for i=0,1,2,…..L-1

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) = 2^N$$

$$T(0) = 1$$

Worst case time complexity = $O(2^N)$, Exponential

Space complexity = O(1)

2. Memoization and DP

There are two for loops.

Inner **for** loop , i=0,1,2,3,….l

Outer **for** loop , l=1,2,3,4…..L

$$\sum_{l=0}^{L}\sum_{i=0}^{l} 1 = \sum_{l=0}^{L} l = 1+2+3+4+\ldots\ldots..L = L(L+1)/2 = (L^2+L)/2 = O(L^2)$$

Time complexity $O(N^2)$, N = Length of the rod

Space complexity O(N) , N = Length of the rod

Reconstrcuting the solution

In previous example, if L = 8, then maximum profit is 21.

It can be acehived by cutting the rod into segments of sizes

2 , 3, 3

5+6+6 = 21

We record the decision along with recording the maximum profit in an array named cuts. For every subproblem we record what is the length of the cut which resulted in maximum profit.

Reconstruction

We start from the last i.e L, we use a variable named l

We check the cuts array at index l to get the length of the cut. We print it out, then we subtract that from l to get the remaining length.

We repeat this process until l becomes 0.

For example: L=8

8 = 2+**6 ,** cuts[8] = 2

6 = 3+**3 ,** cuts[6] = 3

3 = 3+0 , cuts[3] = 3 ,

Java

```java
public static int maxProfitDPReconstruct(int L,int[] p){
    int[] dp = new int[L+1];
    int[] cuts = new int[L+1];
    for(int l=1;l<=L;l++){
        dp[l] = Integer.MIN_VALUE;
        int cut = -1;
        for(int i=0;i<l;i++){
            if(p[i]+dp[l-i-1] > dp[l]) {
                dp[l] = p[i] + dp[l - i - 1];
                cut = i+1;
            }
        }
        cuts[l] = cut;
    }
```

Java

```
    int l = L;
    int cut = cuts[L];
    while (cut != 0) {
        System.out.print((cut) + " ");
        l = l - cut;
        cut = cuts[l];
    }
    System.out.println("");
    return dp[L];
}
```

Python

```python
def max_profit_dp_reconstruct(L, p):
    dp = [0 for _ in range(0, L + 1)]
    cuts = [i for i in range(0, L+1)]
    for l in range(1, L + 1):
        dp[l] = -sys.maxsize
        for i in range(0, l):
            if p[i] + dp[l - i - 1] > dp[l]:
                dp[l] = p[i] + dp[l - i - 1]
                cuts[l] = i+1

    l = L
    cut = cuts[L]
    while cut != 0:
        print(str(cut)+',',end='')
        l = l-cut
        cut = cuts[l]
    return dp[L]
```