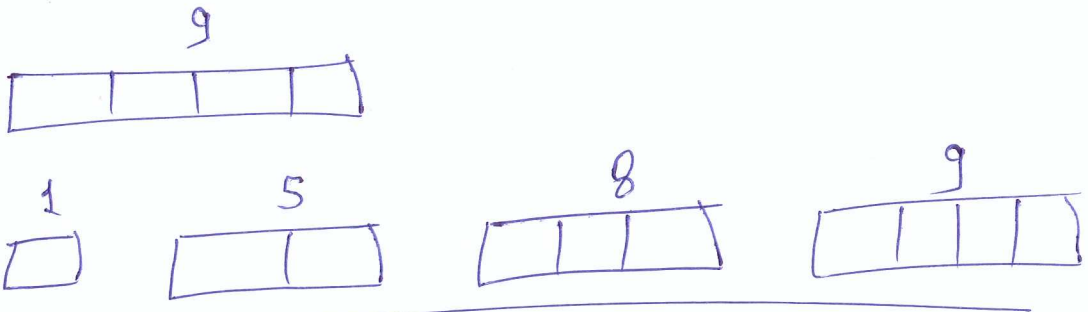
Rod Cutting Problem

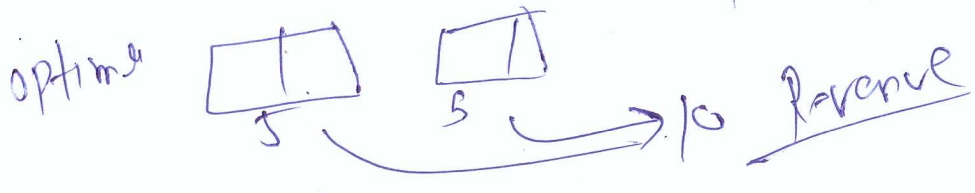
* Given a rod of length n
* Obtain the most beneficial way of cutting the rod which maximizes REVENUE

length (i)	1	2	3	———	n
price (p_i)	p_1	p_2	p_3		p_n

NOTE: If price p_n for a rod of length n is large enough, an optimal solution may require no cutting at all

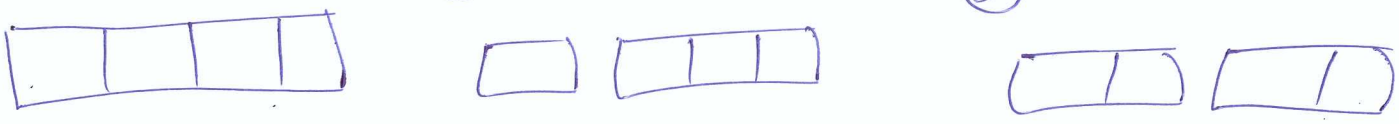


i	1	2	3	4
p_i	1	5	8	9



Different ways to cut the ROD

(1) (2) (3)



(4) (5)

(6) (7)

(8)

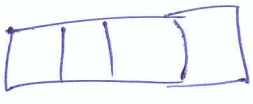
For length n → 2^{n-1}

(2) and (4) Same

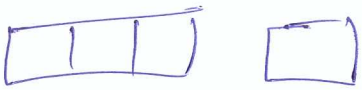
(5) (6) (7) Same

So actual

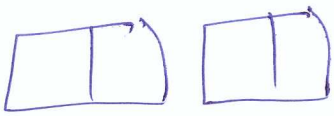
(1)



(2)



(3)



(4)

(5)

For n ?

It is equal to number of partition

For example

4 — { 4, 3+1, 2+2, 2+1+1, 1+1+1+1

5 — { 5, 4+1, 3+2, 3+1+1, 2+2+1, 2+1+1+1, 1+1+1+1+1

Number of ways to cut ROD of length n is (1913)

$$\frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

not exact but error rate decreases with increase in n

~~Recurrence~~

Let r_n be the revenue when a rod of length n is cut into pieces

$$r_n = \max \begin{cases} p_n \longrightarrow \text{means no cut} \\ r_1 + r_{n-1} \\ r_2 + r_{n-2} \\ \vdots \\ r_{n-1} + r_1 \end{cases} \quad \text{---------- (1)}$$

* Here to solve the original problem of size n, we are going to solve smaller problem of same type.

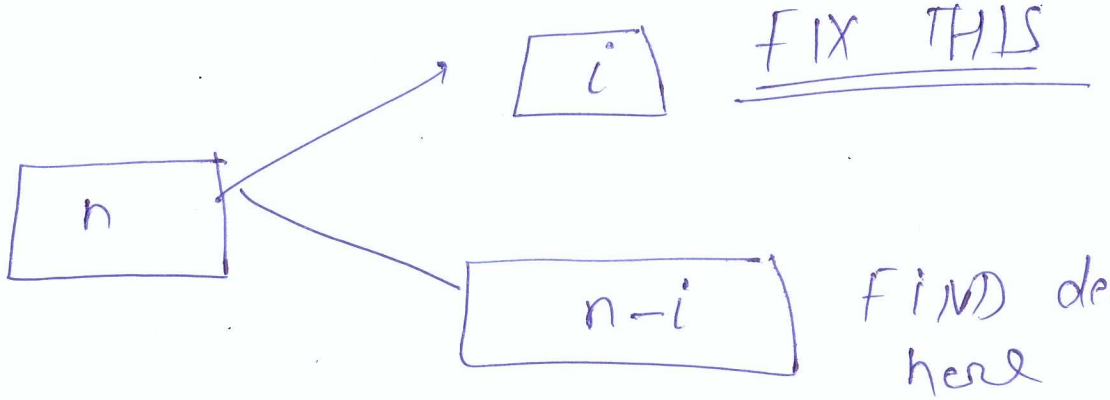
* The overall optimal solution ⟶ optimal solution to Related subproblem

OPTIMAL SUBSTRUCTURE

verify yourself

$$r_n = \max \begin{cases} p_n \\ r_1 + r_{n-1} \\ \vdots \\ r_{\lfloor n/2 \rfloor} + r_{\lceil n/2 \rceil} \end{cases} \quad \text{---------- (2)}$$

* Here we are ignoring rest half ($\approx$) as they will be same as above



FIX THIS

FIND decomposition here

Simpler version of Recurrence relation

$$r_n = \max \begin{cases} p_1 + r_{n-1} \\ p_2 + r_{n-2} \\ \vdots \\ p_n + r_0 \end{cases}$$

$r_0 = 0$ → slice of 0 length

$$= \max_{1 \le i \le n} (p_i + r_{n-i})$$

Algo

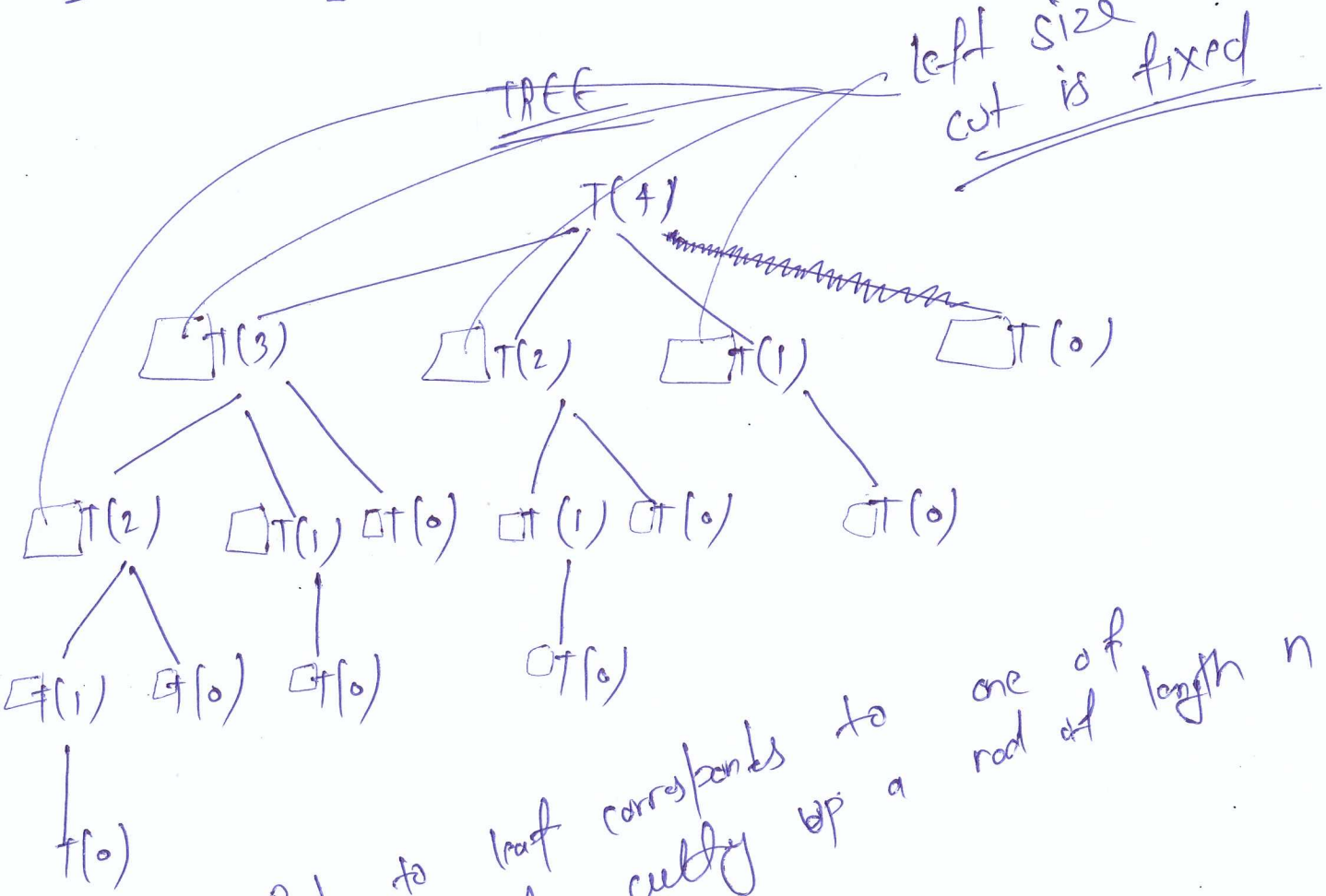
```
CUT-ROD (p, n) {          // An array that stores p1 ... pn
    if n = 0
        Return 0
    else
        r ← -∞
        for i ← 1 to n
            r ← Max (r, p_i + CUT-ROD(p, n-i))
        Return r
}
```

Using this eq^n

$$T(n) = \left[\sum_{i=0}^{n-1} T(i) \right] + 1$$

$T(i)$ corresponds to CUT-ROD$(n-i)$, $1 \leq i \leq n$

Base Case $T(0) = 1$



Path from Root to leaf corresponds to one of the 2^{n-1} ways of cutting up a rod of length n

↓ actually less but we have

T(4) ~~~~ T(0)

Single piece

Dynamic Programming approach runs in polynomial time when the

→ No. of distinct subproblems involved is polynomial in i/p size

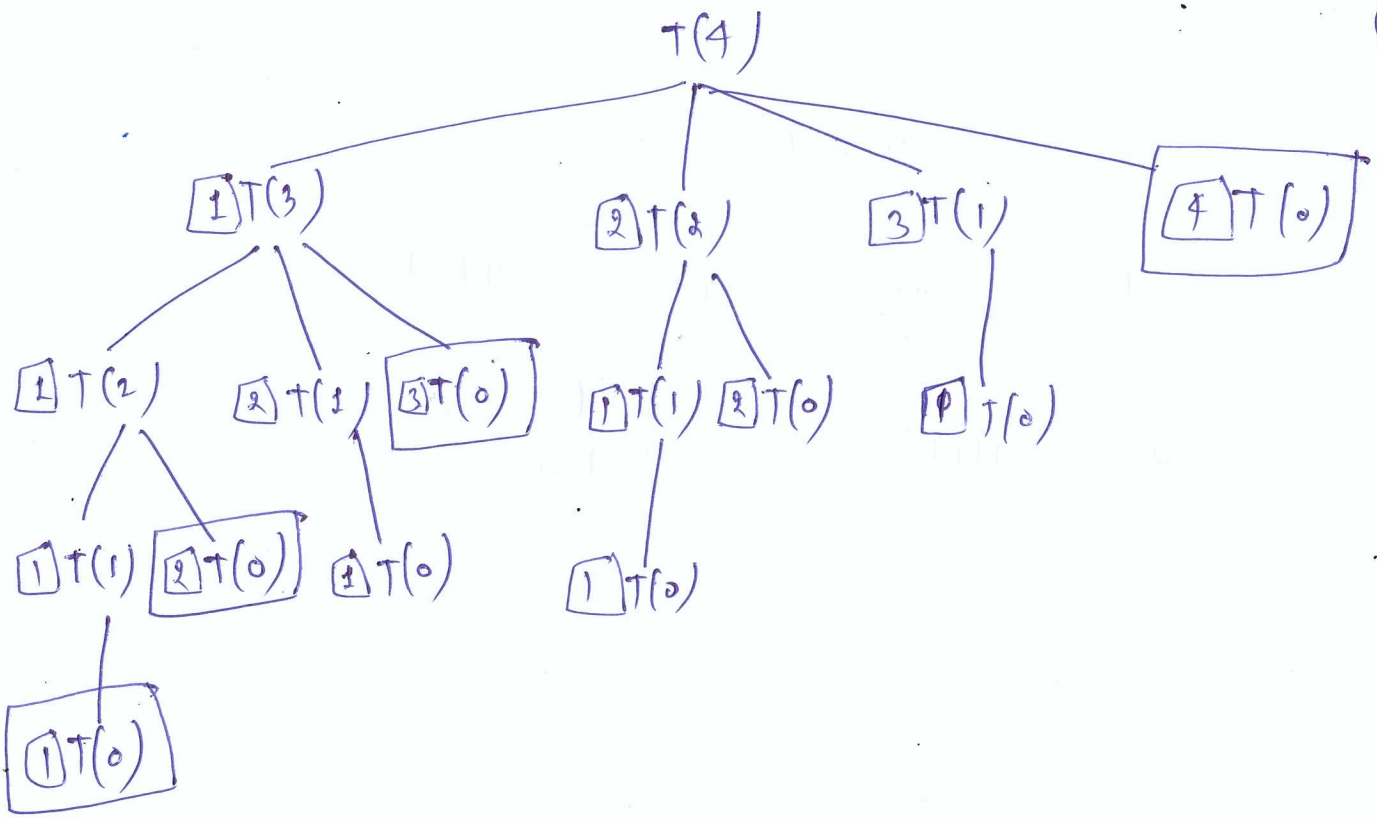
→ We can solve each subproblem in polynomial time

TOP-DOWN DP

```
Memoized-Cut-Rod (p, n) {
(1) r[0, ... n] ← An array to store the
                  answers to smaller subproblems
(2) for i ← 0 to n
(3)       r[i] ← -∞
(4) Return  AUX(p, n, r)

AUX(p, n, r) {
   if r[n] ≥ 0   Return r[n] // Means the subproblem is already solved

   if n = 0
        q ← 0
   else
        q ← -∞
        for i ← 1 to n
            q = Max (q, pi + AUX(p, n-i, r))
   r[n] ← q
   Return r[n]
```



No of subproblem

T(0)
T(1)
T(2)
T(3)
~~T(4)~~

But all these make pair with

p1 p2 —— p4

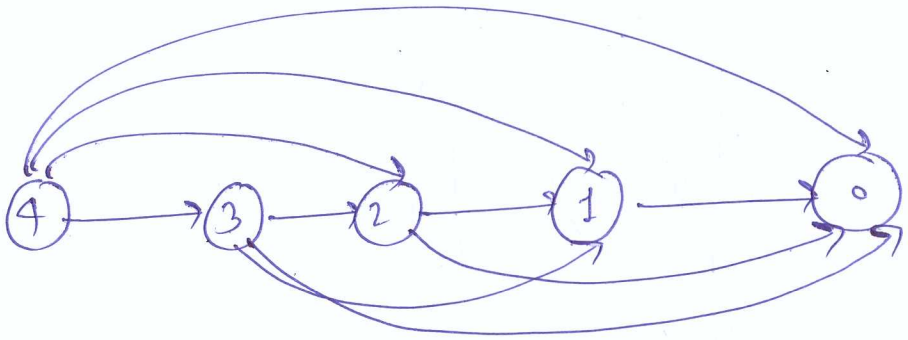
So total combinations are n^2

and time is $O(n^2)$

Bottom Up DP

```
Bottom_UP ( p, n ) {
    r[0, ... n] ← Array to store T(i)
                  0 ≤ i ≤ n
    r[0] ← 0
    for i ← 1 to n
    |     q ← -∞
    |     for j ← 1 to i
    |         q ← Max ( q, p_j + r[i - j] )
    |     r[i] ← q
    Return r[n]
```

Time $O(n^2)$ Easy in this case as compared to TOP-DOWN



Vertex Size of subproblem

(x, y) edge we need solution to subproblem y when solving subproblem x

Find Cutting

```
Bottom-UP(p,n){
    r[0..n]
    s[0..n]      > Arrays
    r[0] ← 0
    for i ← 1 to n
        q ← -∞
        for j ← 1 to i
            if q < pi + r[i-j]
                q ← pi + r[i-j]
                s[i] ← j
        r[j] ← q
    Return r and s
```

```
PRINT(p,n){
    (r,s) ← Bottom-UP(p,n)
    while n > 0
        Print s[n]
        n ← S - s[n]
```