Date : / /

Rod Cutting Problem

Given rod length (N) along with an array prices[] which represents ~~the price of pieces of cuts~~ the price you get in market for each cuts.

Ex:- N=5

	1	2	3	4	5
prices[ ] =	2	5	7	8	10

lengths are 1 based indexing

for length 1 the price in market is 2

for length 3 the price in market is 7

~~Identify~~ Maximize the total price you can get for length N while cutting the rod into pieces

Ex:- You can cut N=5 rod into 5 pieces

1 1 1 1 1 = 2+2+2+2+2 = 10

1 2 2 = 2+5+5 = 12

: and so on...

2 3 = 5+7 = 12

∴ Max = 12

Hint :- Collect rod lengths & to make N (solving from) instead of breaking & selling

Date : / /

Sol

Try to pick lengths & sum them up to make N
↳ in all possible ways

① $f(ind, N)$ ⇒ taking length(ind) & looking to form N

② Possibilities ⇒

```
int notTake = 0 + f(ind-1, N)
int take = INT_MIN;
rodLength = ind + 1; (since, lengths are 1 based index)
if(rodLength <= N){
    take = price[ind] + f(ind, N - rodLength)
return max(take, notTake);
```

Base Case

If single element.

price = N × price[0]

If there is only 1 element you cut them into N pieces to get Max.

Ex:- N=12 | 6

0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10 11 12 ⇒ 6x12

if(ind == 0) { return (N x price[0])

Date : / /

Recursion

```
int f(int ind, int N, int[] price){
  if (ind == 0) {return N*price[0];
}
int notake = 0+f(ind-1, N, price);
 int take = INT_MIN;
int rodlength = ind+1;
 if (rodlength <= N) {
     take = price[ind]+f(ind, N-rodlength, price);
}
return max(take, notake);
```

Date : / /

Memoization

```
static int f(int ind, int N, int[] price, int[][] dp){
  if(ind==0){
     return N*price[0];
  }
  if(dp[ind][N]!=-1)return dp[ind][N];
  int notTake=f(ind-1, N, price, dp);
  int take = Integer.MIN_VALUE;
  int rodLength=ind+1;
  if(rodLength <=N){
     take = price[ind]+f(ind, N-rodLength, price, dp);
  }
  return dp[ind][N] = Math.max(take, notTake);
}
```

Date : / /

Tabulation

```
int [][] dp = new int[n][n+1];
for(int len=0; len<=n; len++){
    dp[0][len] = len * price[0];
}
for(int ind=1; ind<n; ind++){
    for(int len=0; len<=n; len++){
        int nottake = dp[ind-1][len];
    int take = Integer.MIN_VALUE
    int rodlength = ind+1;
    if(rodlength <= len){
        take = price[ind] + dp[ind][len-rodlength];
    }
    dp[ind][len] = Math.max(take, nottake);
    }
}
dp[ind][len] = Math.max(take, nottake);
return dp[n-1][n];
```

Date : / /

Space Optimization

```
int[] prev = new int[n+1];
for(int N=0; N<=n; N++){
    prev[N] = N * price[0];
}
for(int ind=1; ind<n; ind++){
  for(int N=0; N<=n; N++){
      int notTake = 0 + prev[N];
      int take = INT_MIN;
    int rodLength = ind+1;
    if(rodLength <= N){
        take = price[ind] + prev[N - rodLength];
  }
  prev[N] = max(take, notTake);
}
}
return prev[n];
```