Dynamic Programming

Murat Osmanoglu

Dynamic Programming

- ‘programming’ here not refering software. The word itself older than computer. ‘programming’ means any tabular method to accomplish a task.

Dynamic Programming

- ‘programming’ here not refering software. The word itself older than computer. ‘programming’ means any tabular method to accomplish a task.
- introduced by Richard Bellman in 1949. He developed the method with Lester Ford to find the shortest path in a graph.

Dynamic Programming

- from "Eye of the Hurricane : an Autobiography" by Richard Bellman

"An interesting question is, 'Where did the name, dynamic programming, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. This, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities".

Dynamic Programming

- DP can be considered as brute force
  - search all posibilities but do it in a smart way to get the optimal solution

Dynamic Programming

- DP can be considered as brute force
  - search all posibilities but do it in a smart way to get the optimal solution

For what type of problems DP is useful?

Dynamic Programming

- DP can be considered as brute force

  search all posibilities but do it in a smart way to get the optimal solution

For what type of problems DP is useful?

the problems that can be broken into subproblems

Dynamic Programming

- DP can be considered as brute force

  search all posibilities but do it in a smart way to get the optimal solution

For what type of problems DP is useful?

the problems that can be broken into subproblems

Divide & Conquer **Dynamic Programming**

Dynamic Programming

- DP can be considered as brute force

  search all posibilities but do it in a smart way to get the optimal solution

For what type of problems DP is useful?

the problems that can be broken into subproblems

Divide & Conquer	**Dynamic Programming**
you deal with independent subproblems	

Dynamic Programming

- DP can be considered as brute force

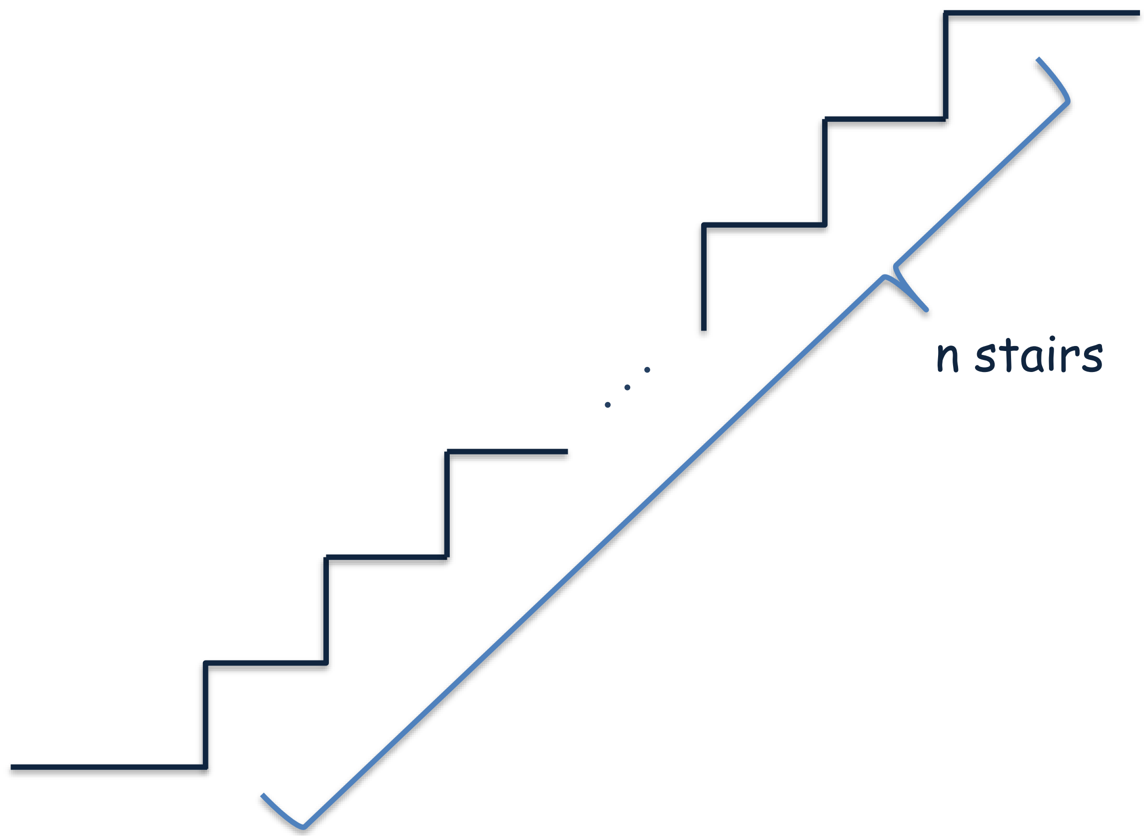
    search all posibilities but do it in a smart way to get the optimal solution

For what type of problems DP is useful?

the problems that can be broken into subproblems

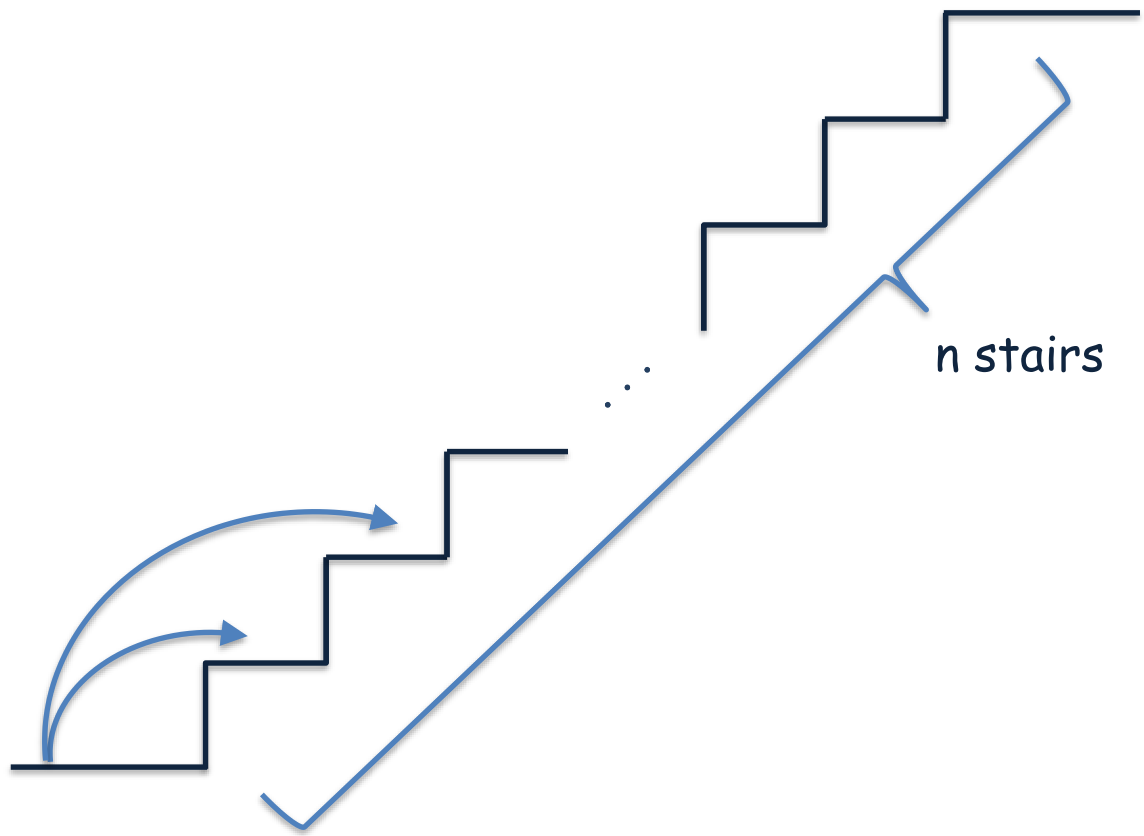
Divide & Conquer	**Dynamic Programming**
you deal with independent subproblems	you deal with overlapping subproblems

Stairs Climbing



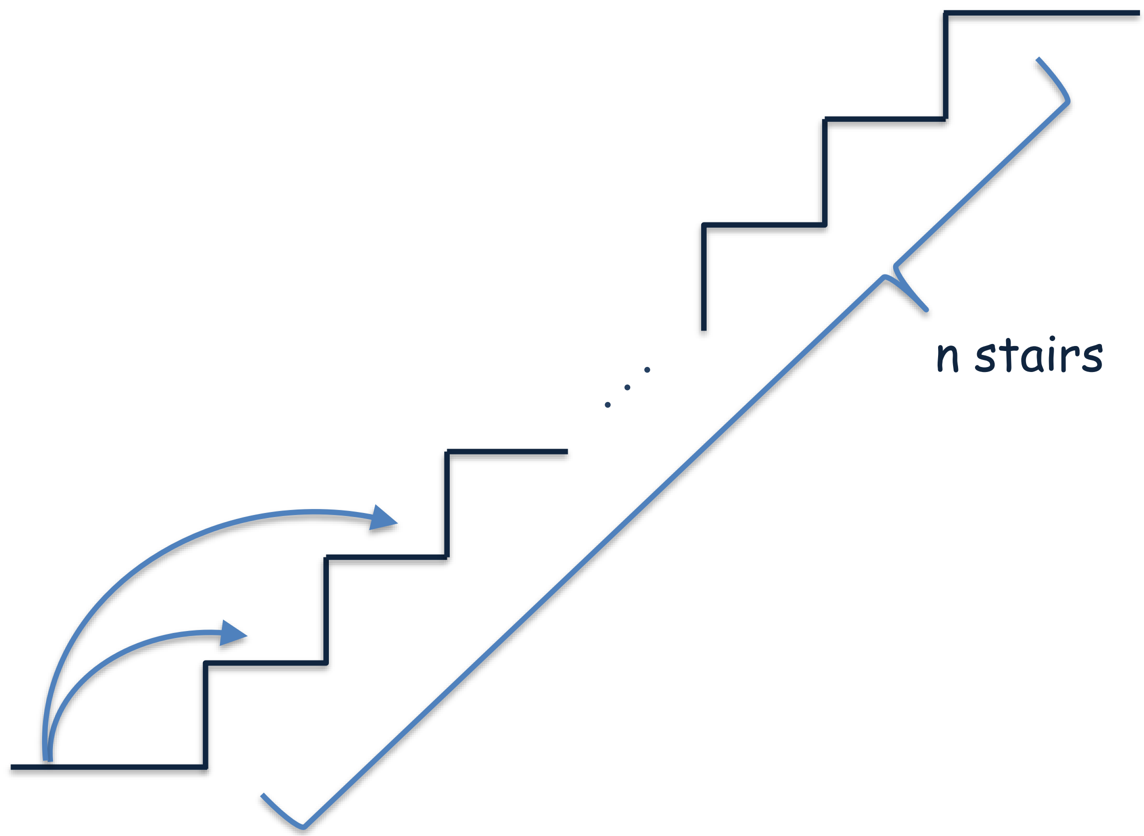
Stairs Climbing

You take one step or two steps at a time.



Stairs Climbing

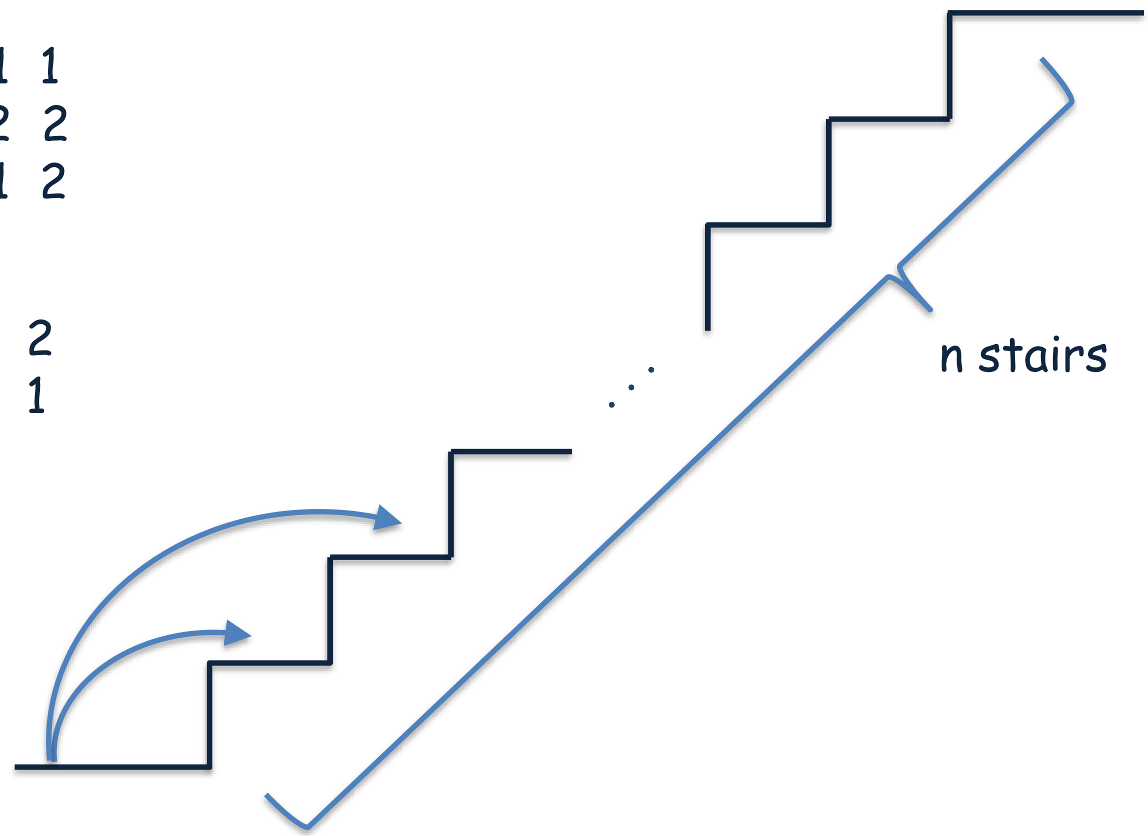
You take one step or two steps at a time.
How many possible ways to climb n stairs ?



Stairs Climbing

You take one step or two steps at a time.
How many possible ways to climb n stairs ?

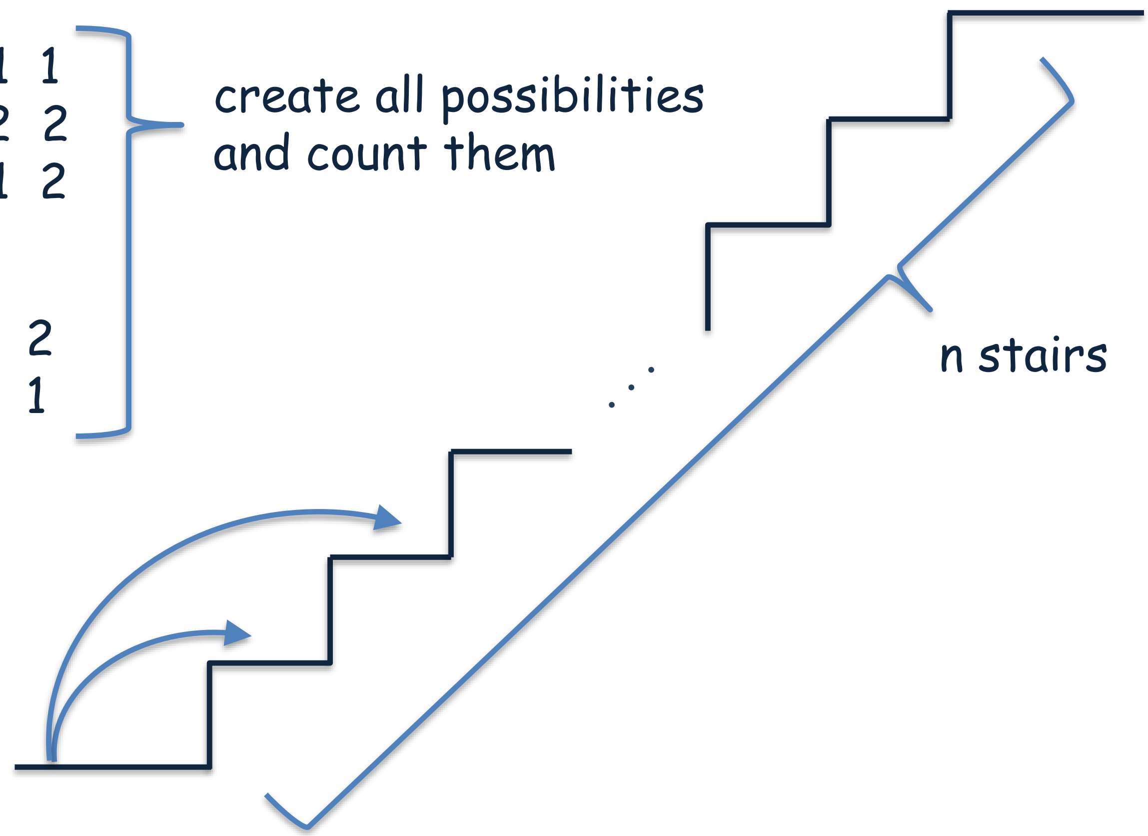
1 1 1 . . . 1 1
2 2 2 . . . 2 2
1 1 1 . . . 1 2
⋮
2 1 2 . . . 1 2
1 2 1 . . . 2 1



Stairs Climbing

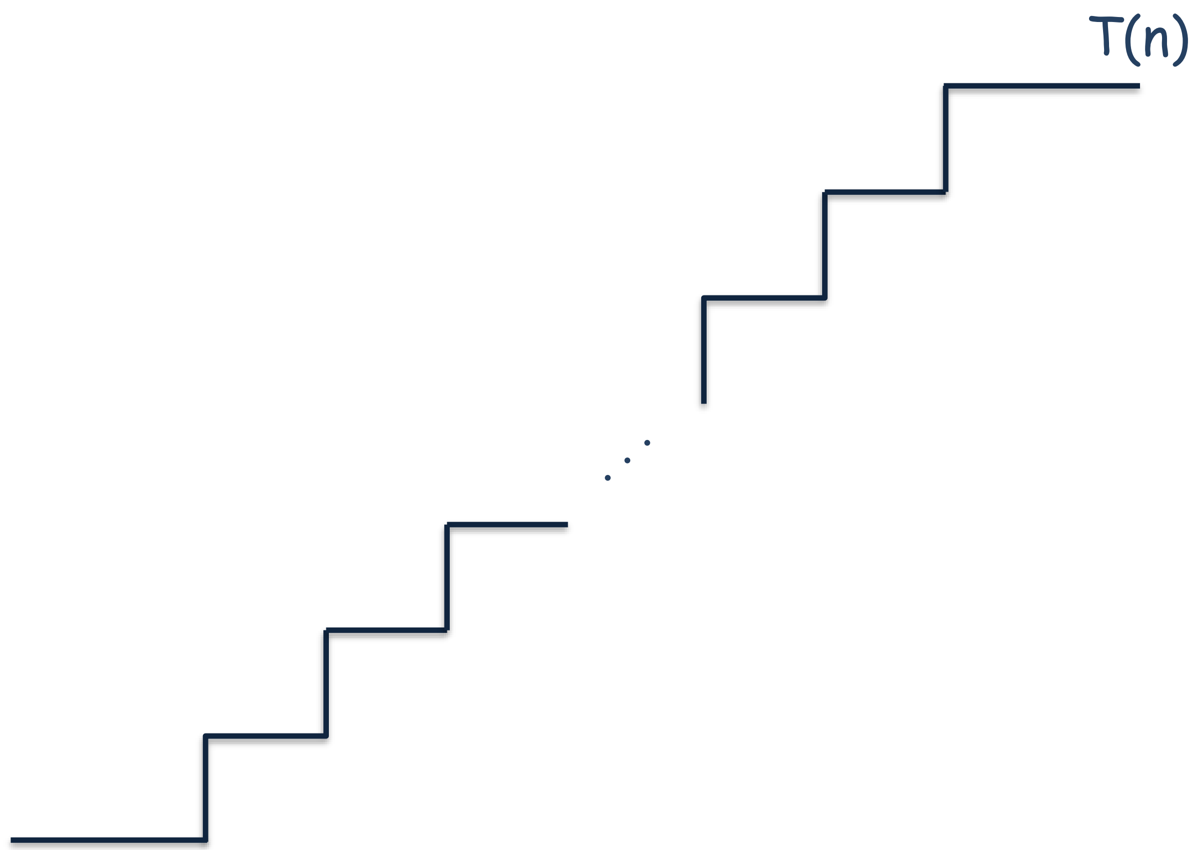
You take one step or two steps at a time.
How many possible ways to climb n stairs ?

$$\left.\begin{array}{l} 1\ 1\ 1\ \ldots\ 1\ 1 \\ 2\ 2\ 2\ \ldots\ 2\ 2 \\ 1\ 1\ 1\ \ldots\ 1\ 2 \\ \qquad\vdots \\ 2\ 1\ 2\ \ldots\ 1\ 2 \\ 1\ 2\ 1\ \ldots\ 2\ 1 \end{array}\right\}$$ create all possibilities and count them



Stairs Climbing

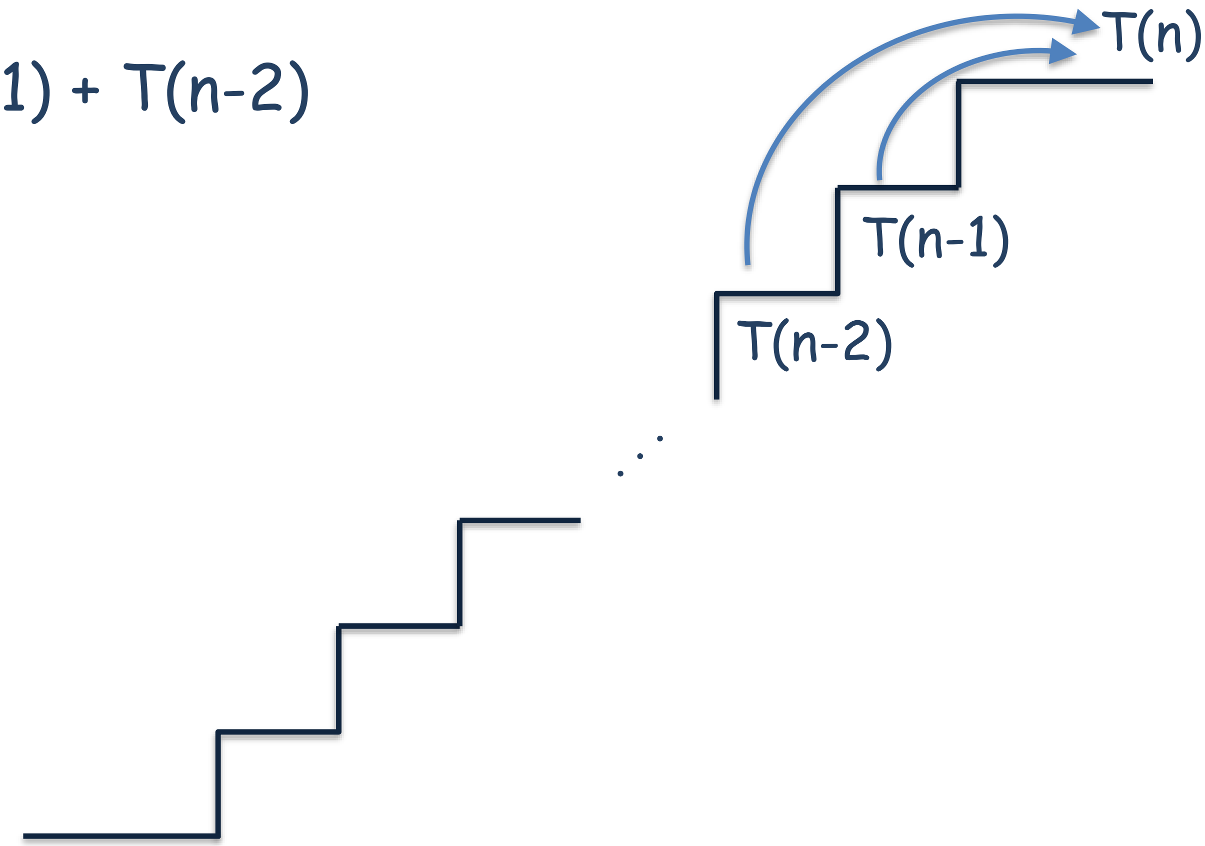
$T(n)$ = # of ways to climb n stairs



Stairs Climbing

T(n) = # of ways to climb n stairs

$T(n) = T(n-1) + T(n-2)$

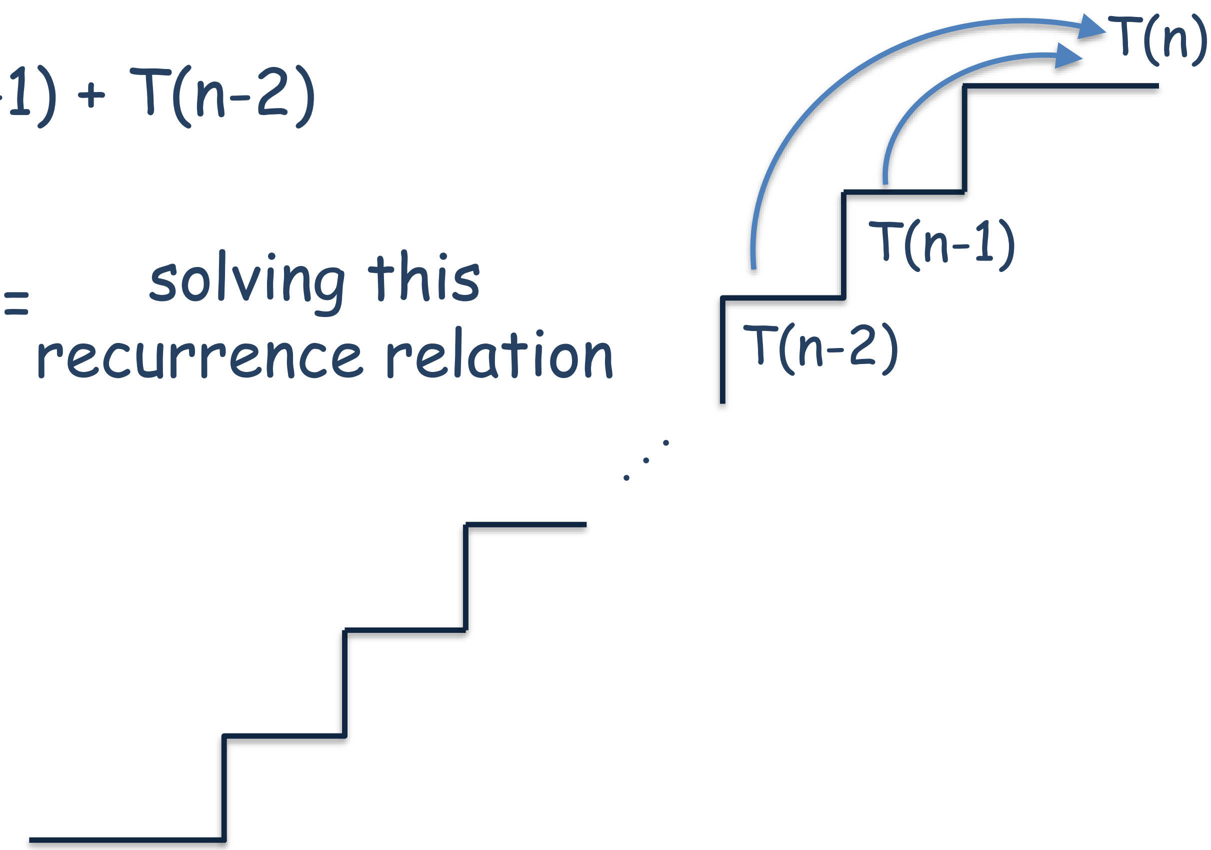


Stairs Climbing

T(n) = # of ways to climb n stairs

$T(n) = T(n-1) + T(n-2)$

finding # of different ways = solving this recurrence relation

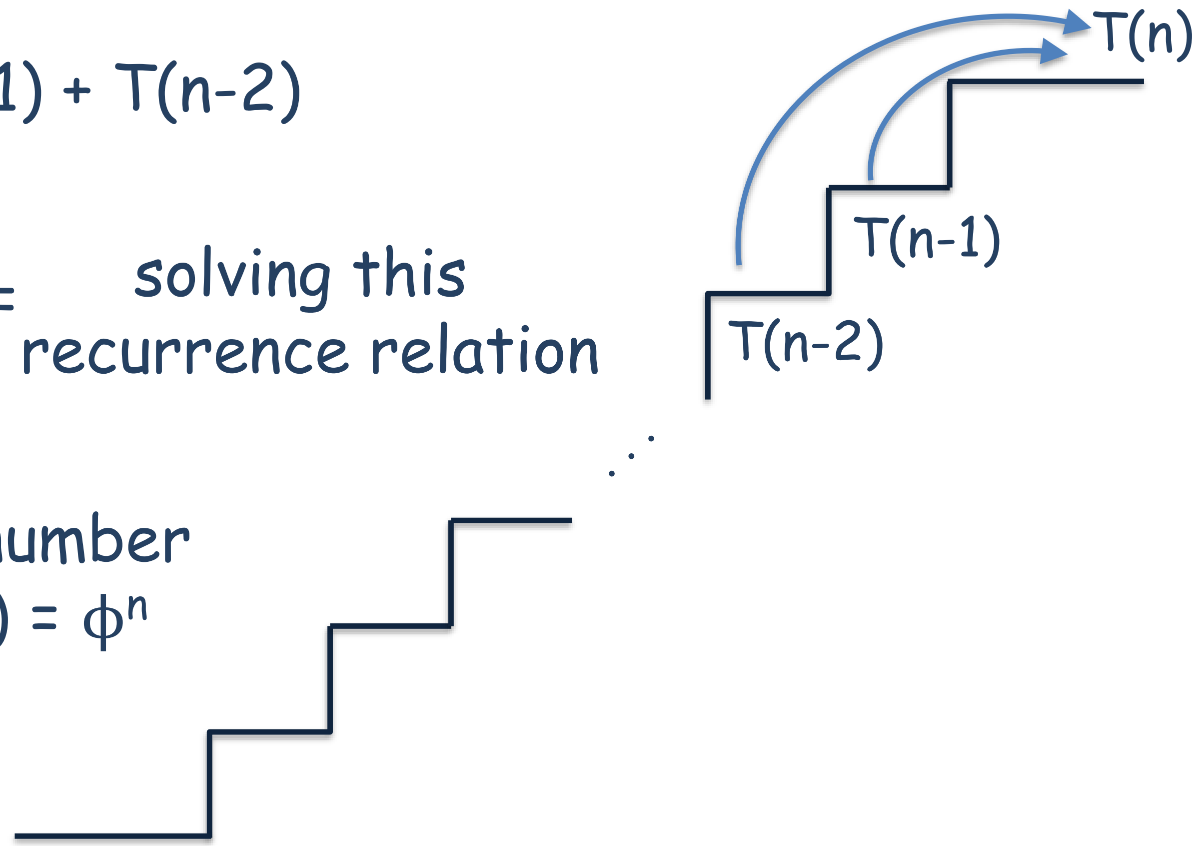


Stairs Climbing

$T(n)$ = # of ways to climb n stairs

$T(n) = T(n-1) + T(n-2)$

finding # of different ways = solving this recurrence relation



Fibonacci number
$T(n) = F(n) = \phi^n$

Stairs Climbing

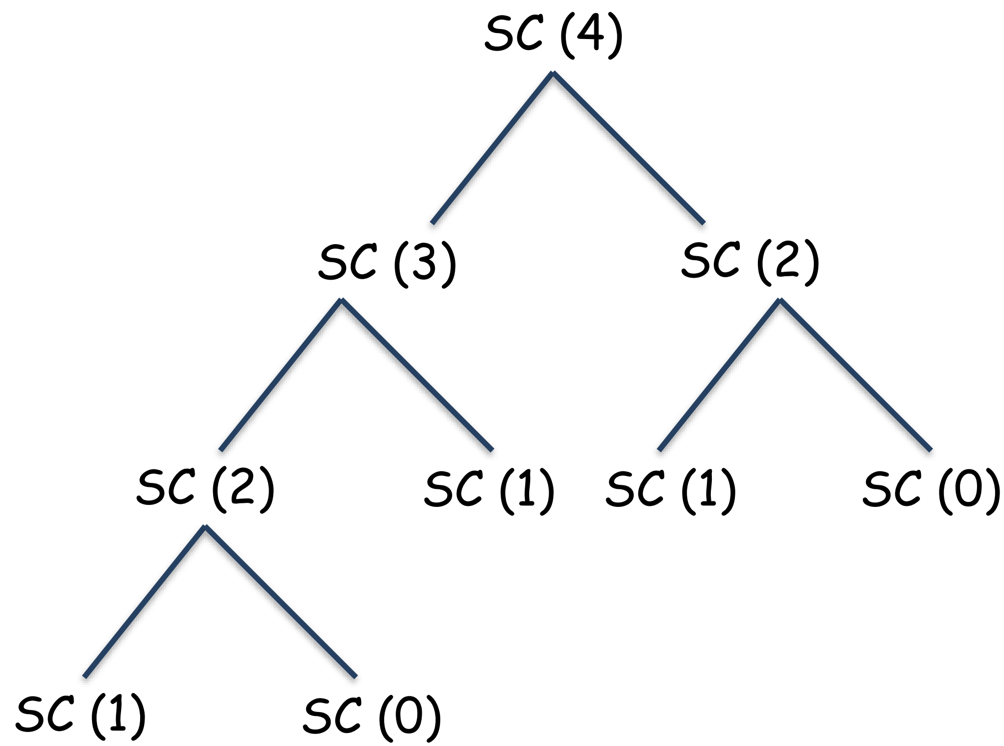
```
SC (n)

if n ≤ 1
   return 1
else
   return SC (n-1) + SC (n-2)
```

Stairs Climbing

SC (n)

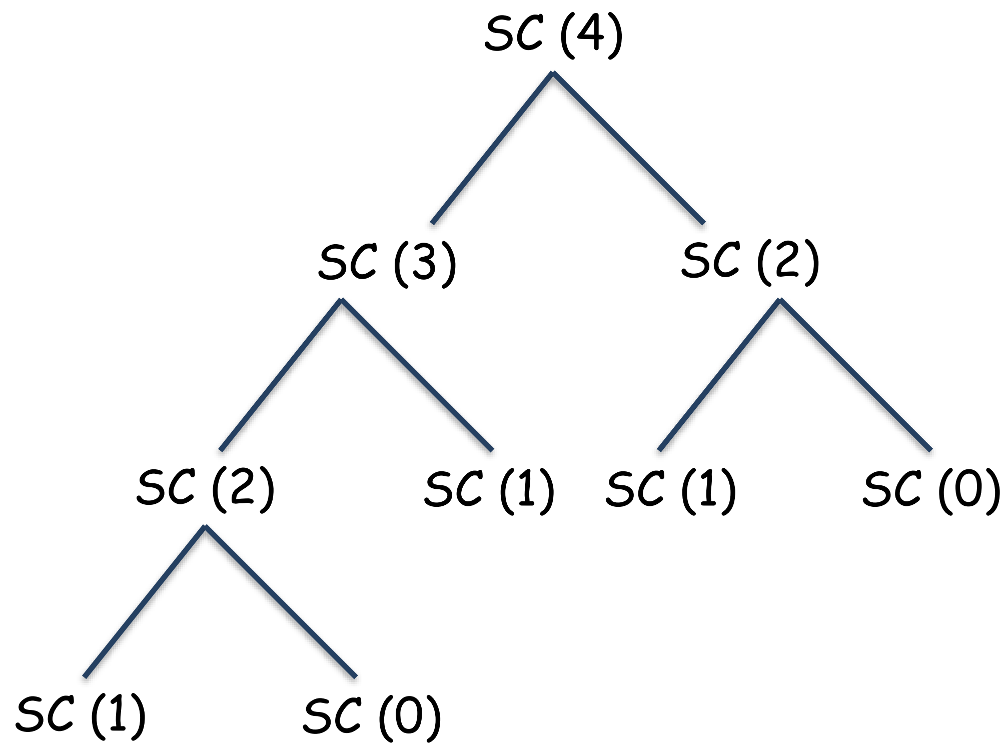
```
if n ≤ 1
    return 1
else
    return SC (n-1) + SC (n-2)
```



Stairs Climbing

SC (n)

if n ≤ 1
    return 1
else
    return SC (n-1) + SC (n-2)

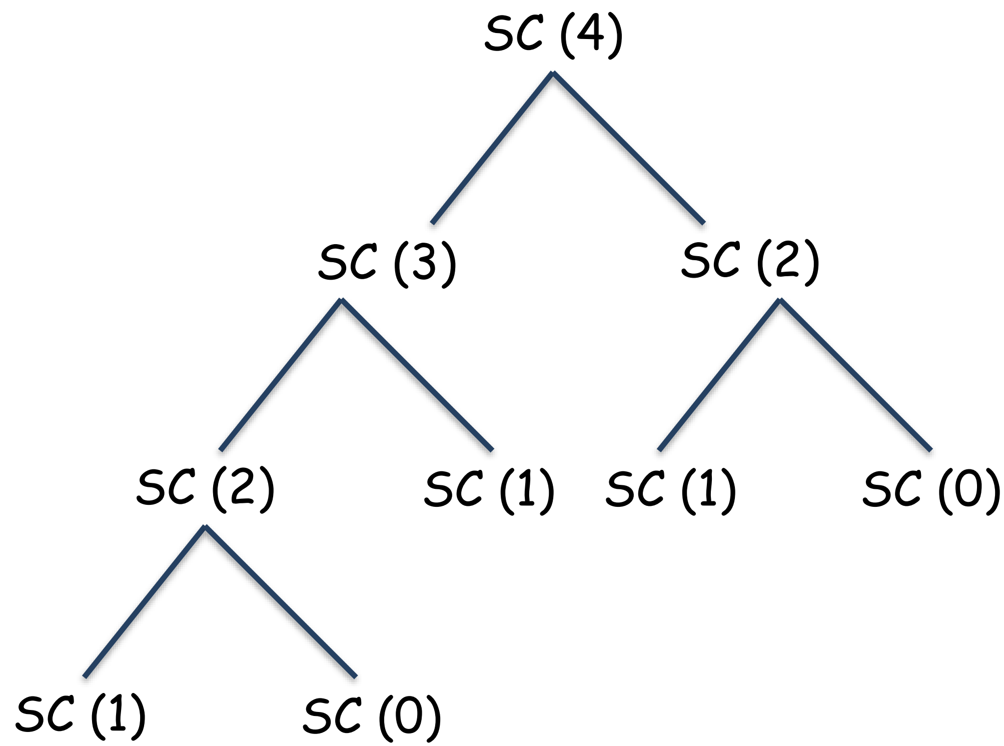


In order to calculate SC(4), the program makes 9 recursive calls;
5 for SC(3) + 3 for SC(2)

Stairs Climbing

SC (n)

if $n \leq 1$
    return 1
else
    return SC (n-1) + SC (n-2)



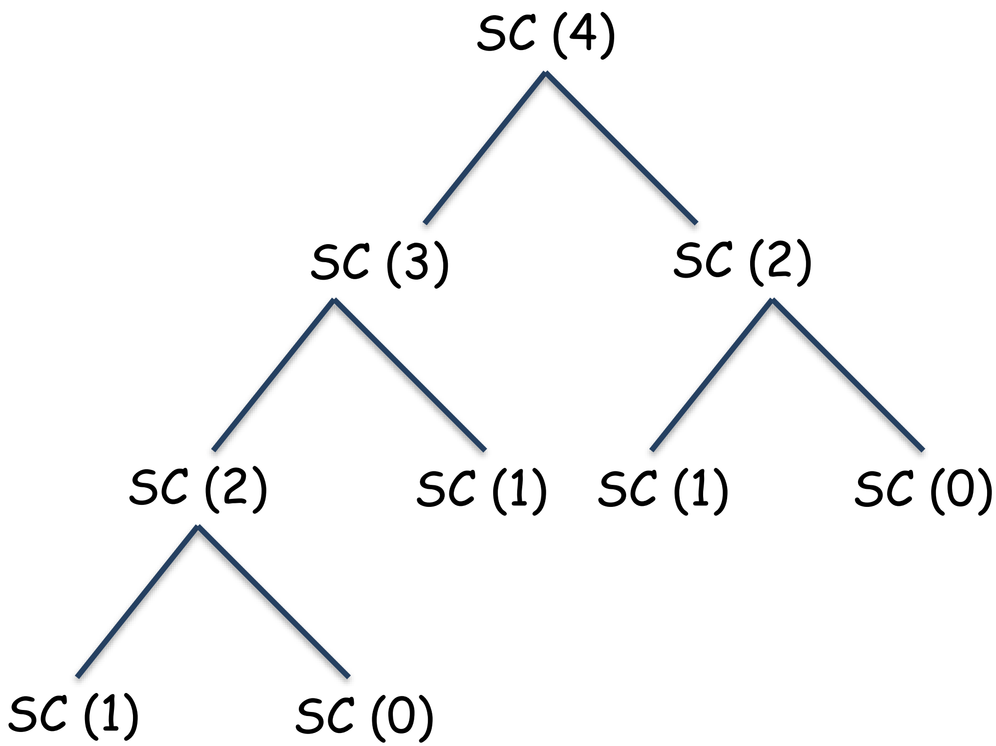
In order to calculate SC(4), the program makes 9 recursive calls;
5 for SC(3) + 3 for SC(2)

\# of calls for SC(n) = # of calls for SC(n-1) + # of calls for SC(n-2)

\# of calls for SC(n) = F(n) ≈ ϕ^n ; nth Fibonacci number

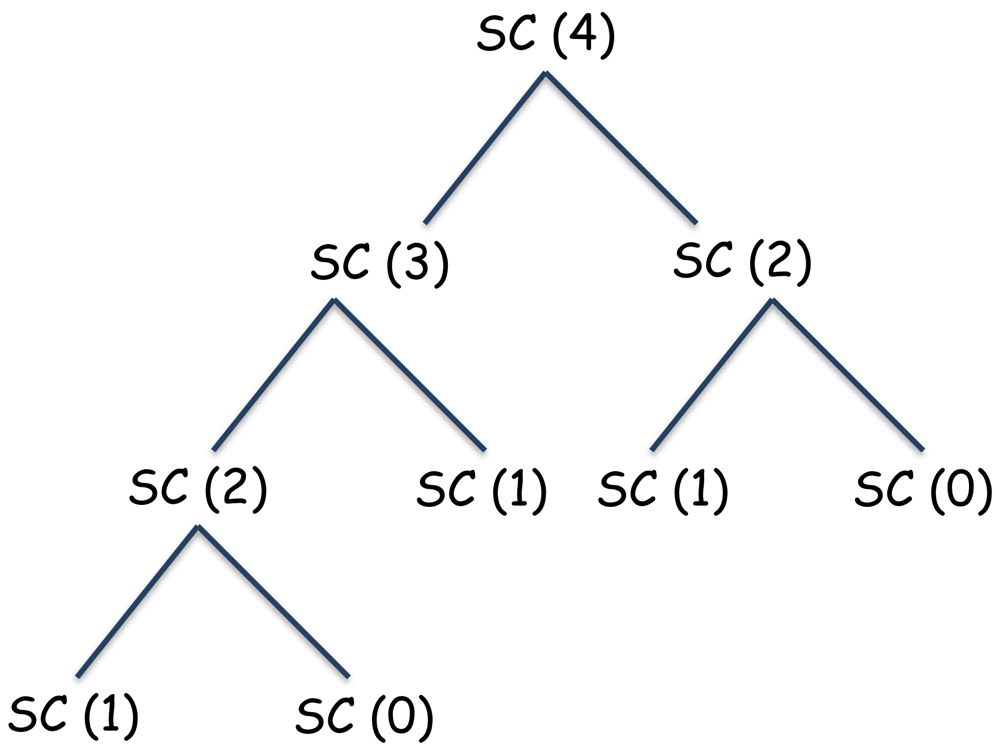
So the running time will be exponential $O(\phi^n)$

Stairs Climbing



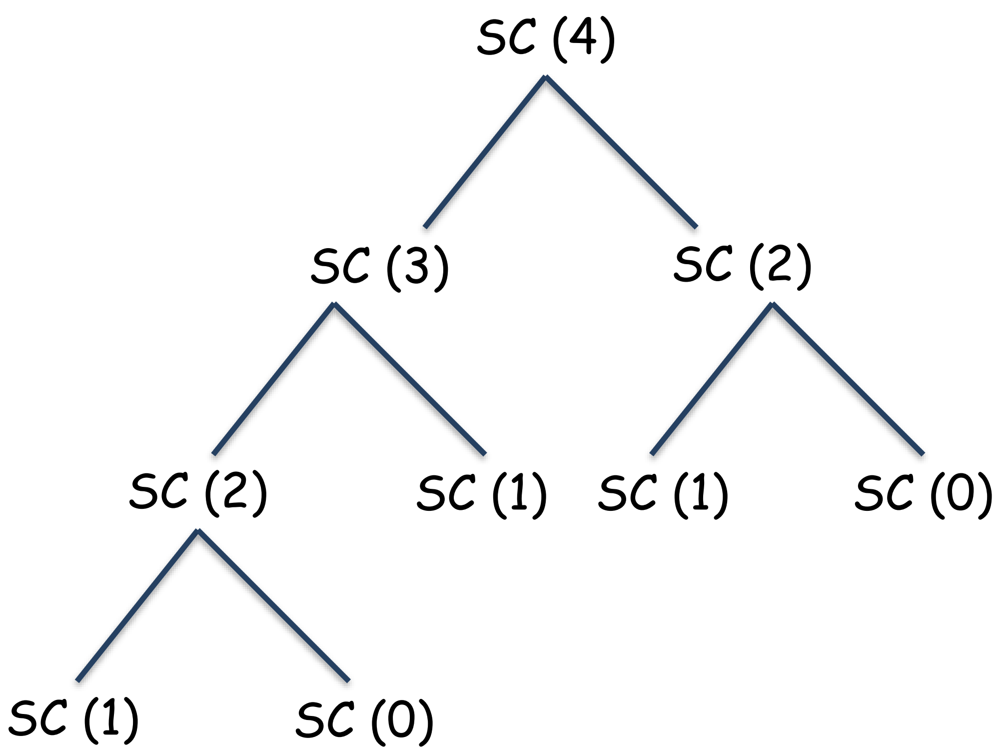
- We deal with the overlapping subproblems

Stairs Climbing



- We deal with the overlapping subproblems
- Whenever computing subproblems, keep them in a table to avoid recomputation, and refer them whenever they are needed

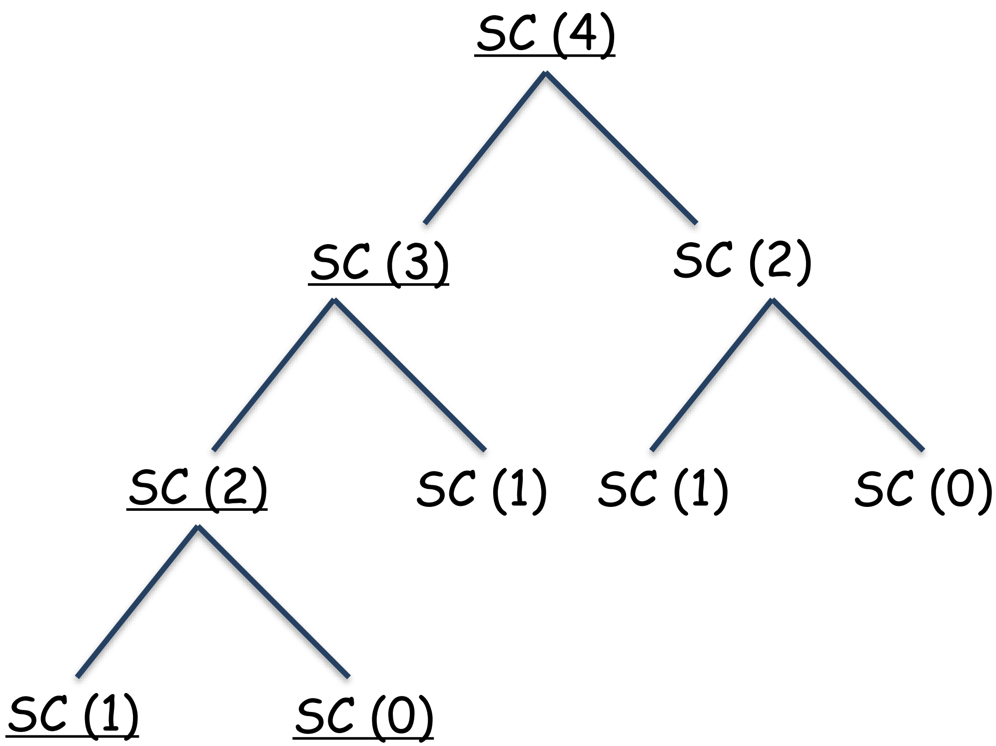
Stairs Climbing



- We deal with the overlapping subproblems
- Whenever computing subproblems, keep them in a table to avoid recomputation, and refer them whenever they are needed

MEMOIZATION

Stairs Climbing

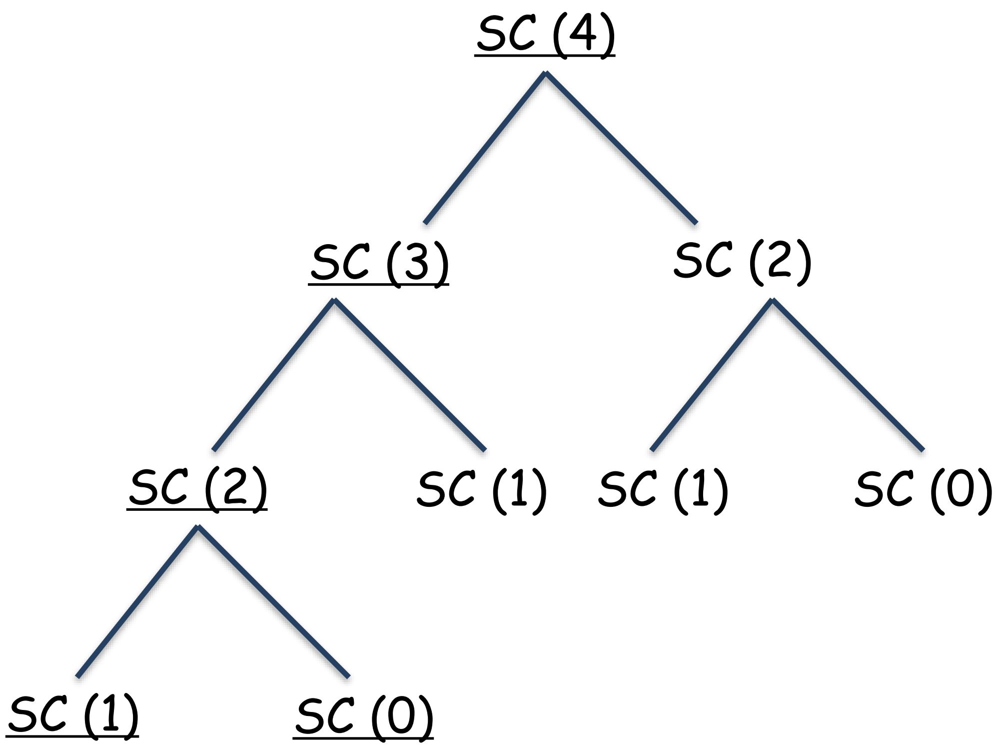



```
SC (n)
initialize a memory M
if n ≤ 1
    return 1
if M contains n
    return M[n]
else
    A = SC(n-1) + SC(n-2)
    M[n] = A
    return A
```

- We deal with the overlapping subproblems
- Whenever computing subproblems, keep them in a table to avoid recomputation, and refer them whenever they are needed

MEMOIZATION

Stairs Climbing



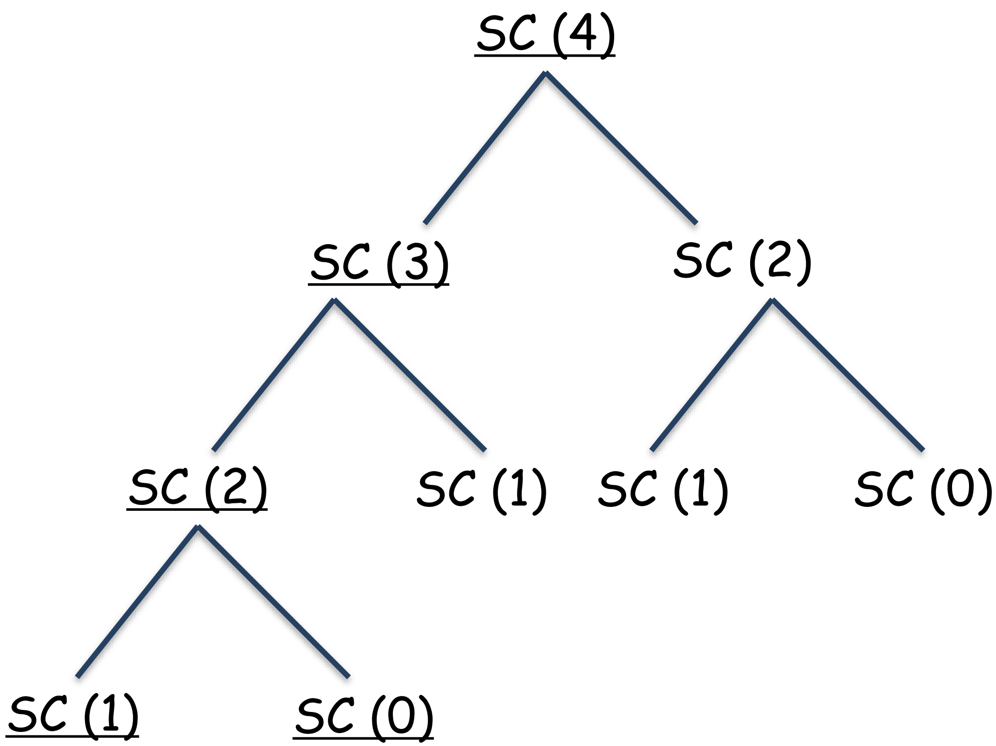

```
SC (n)
initialize a memory M
if n ≤ 1
    return 1
if M contains n
    return M[n]
else
    A = SC(n-1) + SC(n-2)
    M[n] = A
    return A
```

Top-Down

- We deal with the overlapping subproblems
- Whenever computing subproblems, keep them in a table to avoid recomputation, and refer them whenever they are needed

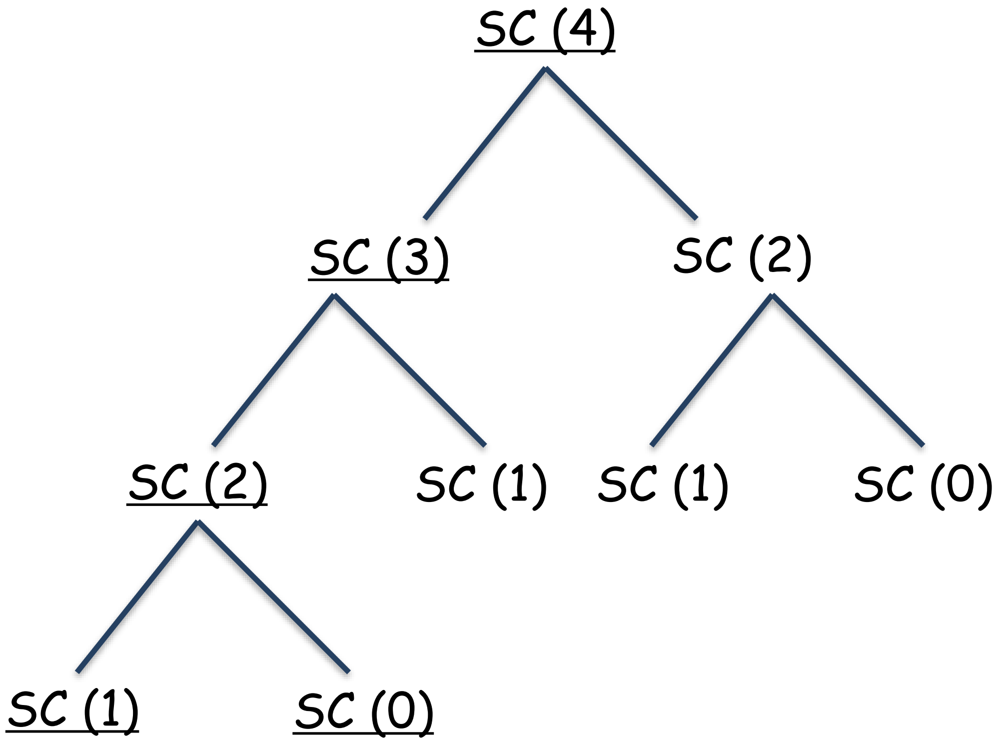
MEMOIZATION

Stairs Climbing



Can we come up with simpler program ?

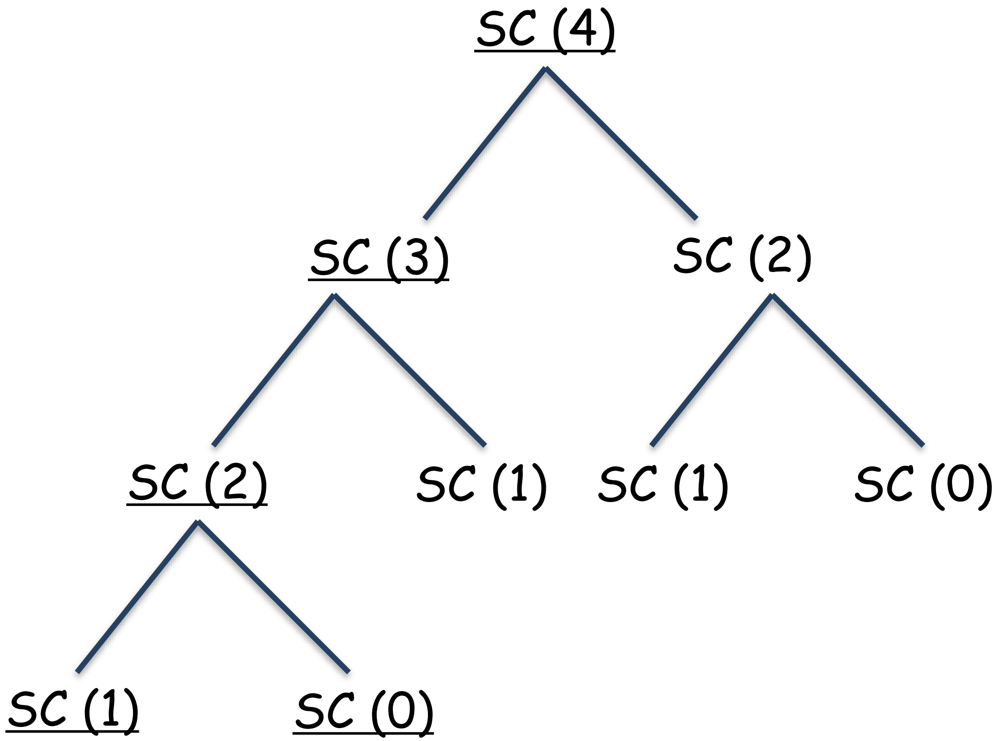
Stairs Climbing



Can we come up with simpler program ?

get rid of recursion
use a simple for loop

Stairs Climbing

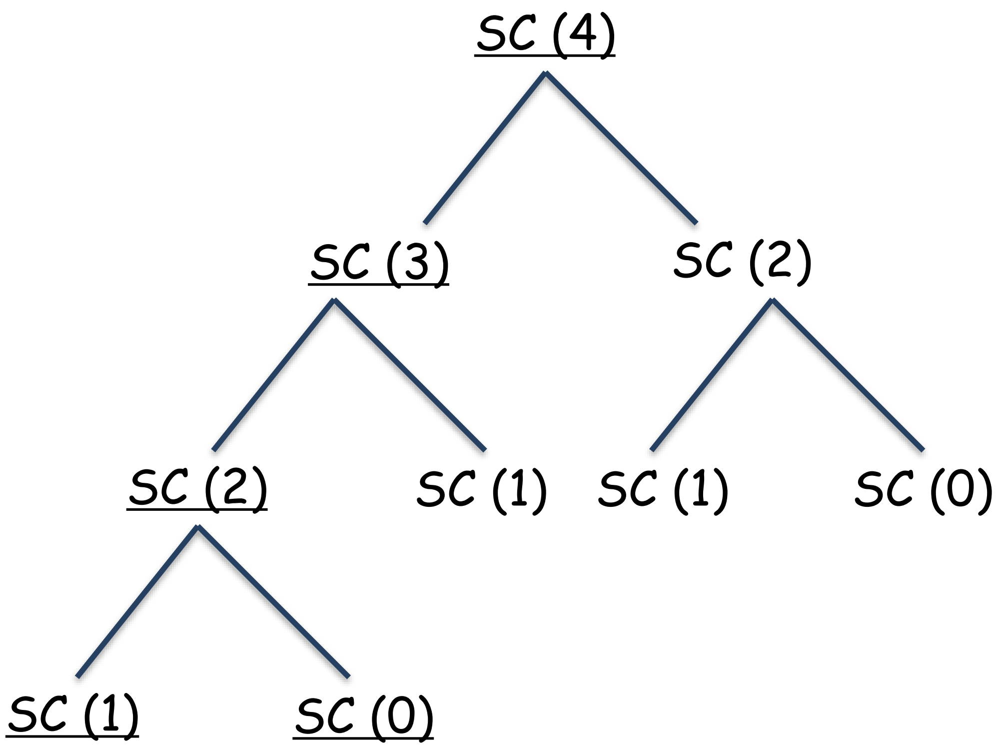



```
SC (n)
initialize a memory M
M[0] = 1
M[1] = 1
for ( i=2 to n)
     M[i] = M[i-1] + M[i-2]
return M[n]
```

Can we come up with simpler program ?

get rid of recursion
use a simple for loop

Stairs Climbing




```
SC (n)
initialize a memory M
M[0] = 1
M[1] = 1
for ( i=2 to n)
     M[i] = M[i-1] + M[i-2]
return M[n]
```

Bottom-Up

Can we come up with simpler program ?

get rid of recursion
use a simple for loop

Dynamic Programming

- analyze structure of the optimal solution and define subproblems that need to be solved in order to get the optimal solution

Dynamic Programming

- analyze structure of the optimal solution and define subproblems that need to be solved in order to get the optimal solution
- establish the relationship between the optimal solution and those subproblems (construct the recurrence relation)

Dynamic Programming

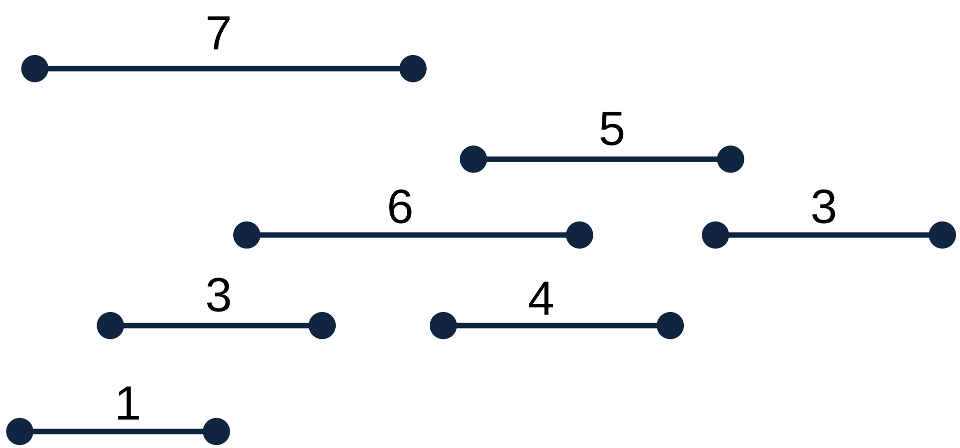
- analyze structure of the optimal solution and define subproblems that need to be solved in order to get the optimal solution
- establish the relationship between the optimal solution and those subproblems (construct the recurrence relation)
- compute the optimal values of subproblems, save them in a table (memoization), then compute the optimal values of larger subproblems, and eventually compute the optimal value of the original problem

Weighted Interval Scheduling

- given a set of intervals (I_1, I_2, ... , I_n)
- each interval I_i has a starting time s_i, a finishing time f_i, and a weight w_i
- your task is to find a subset of intervals (pairwise nonoverlapping) such that the total weight of intervals is maximized

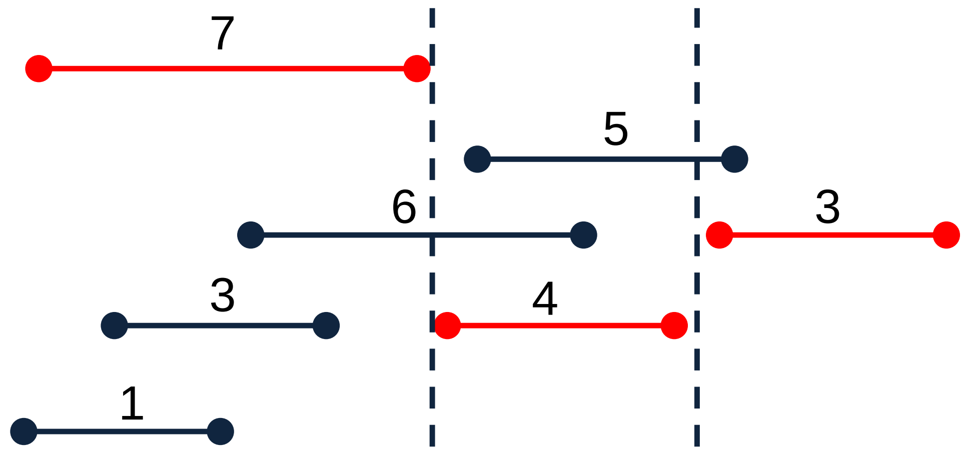
Weighted Interval Scheduling

- given a set of intervals ($I_1, I_2, \ldots, I_n$)
- each interval I_i has a starting time s_i, a finishing time f_i, and a weight w_i
- your task is to find a subset of intervals (pairwise nonoverlapping) such that the total weight of intervals is maximized



Weighted Interval Scheduling

- given a set of intervals ($I_1, I_2, \ldots, I_n$)
- each interval I_i has a starting time s_i, a finishing time f_i, and a weight w_i
- your task is to find a subset of intervals (pairwise nonoverlapping) such that the total weight of intervals is maximized



Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j

Weighted Interval Scheduling

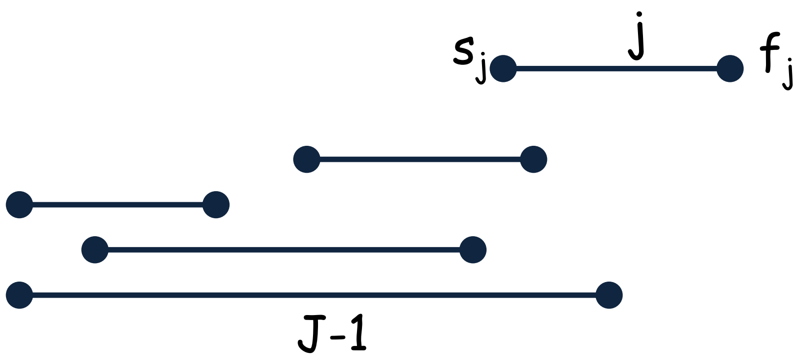
- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \le f_2 \le \ldots \le f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation

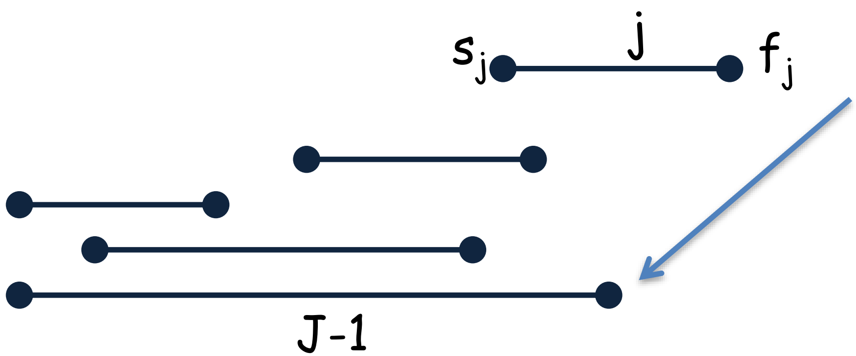


Two cases :

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation

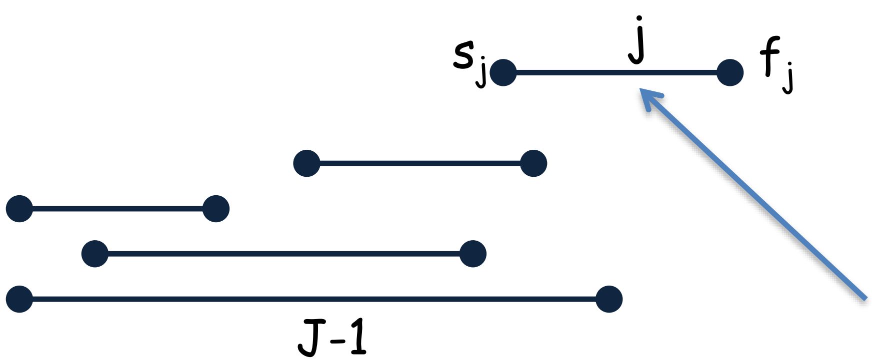


Two cases :
(1) either optimal solution does not include interval j, then continue with OPT (j-1)

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation



Two cases :

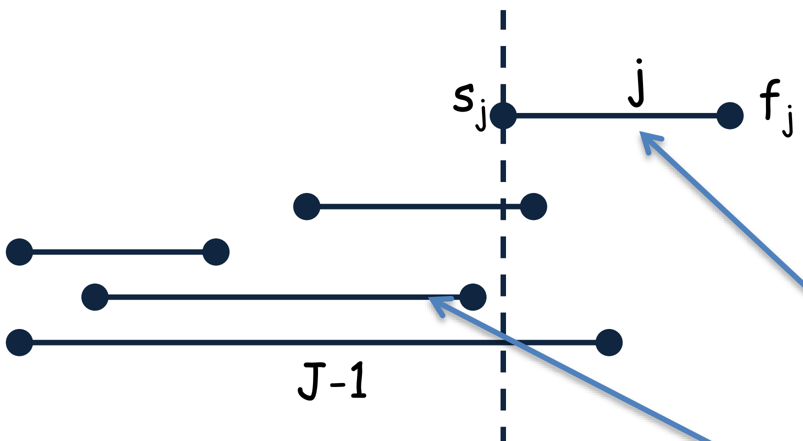
(1) either optimal solution does not include interval j, then continue with OPT (j-1)

(2) or optimal solution includes interval j, then continue with w_j +

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \le f_2 \le \ldots \le f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation



Two cases :

(1) either optimal solution does not include interval j, then continue with OPT (j-1)

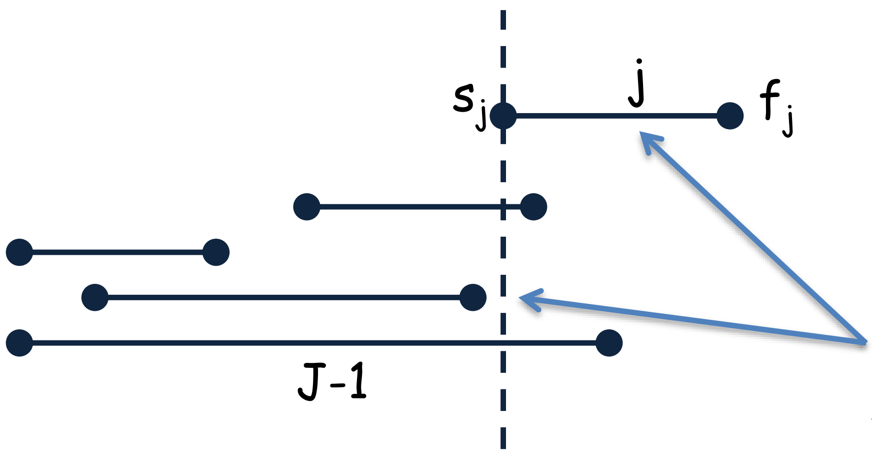
(2) or optimal solution includes interval j, then continue with w_j +

p(j) : largest index i such that $i < j$ and $f_i < s_j$

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for the first j intervals 1, … , j
- construct the recurrence relation



Two cases :

(1) either optimal solution does not include interval j, then continue with OPT (j-1)

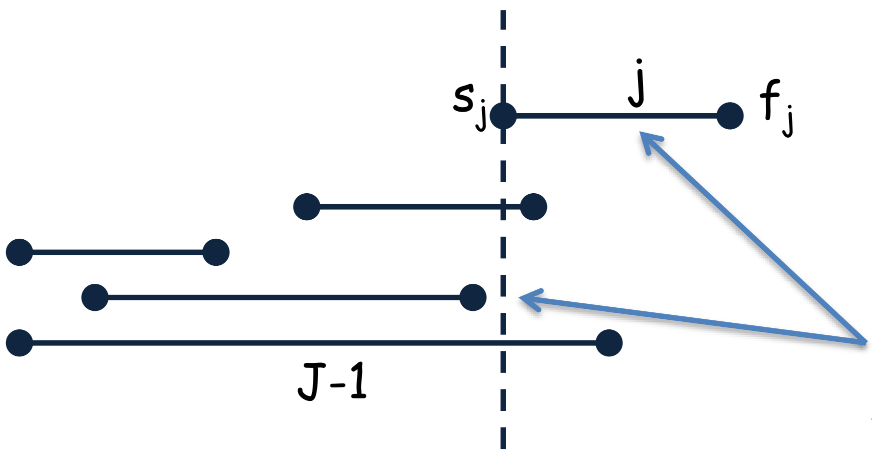
(2) or optimal solution includes interval j, then continue with w_j + OPT (p(j))

p(j) : largest index i such that $i < j$ and $f_i < s_j$

Weighted Interval Scheduling

- first sort all the intervals according to their finishing time : $i_1, i_2, \ldots, i_n$ such that $f_1 \leq f_2 \leq \ldots \leq f_n$
- define subproblems

  OPT (j) : value of the optimal solution for
  the first j intervals 1, … , j
- construct the recurrence relation



Two cases :
(1) either optimal solution does not include interval j, then continue with OPT (j-1)

(2) or optimal solution includes interval j, then continue with w_j + OPT (p(j))

$$OPT(j) = \max\{\, OPT(j-1),\ w_j + OPT(p(j)) \,\}$$

Weighted Interval Scheduling

OPT (n)

sort intervals according to finishing time

```
if n = 0
    return 0
else
    find p(n)
    if OPT (n-1) ≥ w_n + OPT(p(n))
        return OPT (n-1)
    else
        return w_n + OPT (p(n))
```

Weighted Interval Scheduling

OPT (n)
sort intervals according to finishing time
if n = 0
    return 0
else
    find p(n)
    if OPT (n-1) $\geq w_n$ + OPT(p(n))
        return OPT (n-1)
    else
        return w_n + OPT (p(n))

Similar to stairs climbing,
# of calls here also $F(n) \approx \phi^n$

Weighted Interval Scheduling

OPT (n)
sort intervals according to finishing time
if n = 0
    return 0
else
    find p(n)
    if OPT (n-1) $\geq w_n$ + OPT(p(n))
        return OPT (n-1)
    else
        return w_n + OPT (p(n))

Similar to stairs climbing,
# of calls here also $F(n) \approx \phi^n$

Do Memoization

Weighted Interval Scheduling

OPT (n)
sort intervals according to finishing time
if $n = 0$
    return 0
else
    find $p(n)$
    if $OPT(n-1) \geq w_n + OPT(p(n))$
        return $OPT(n-1)$
    else
        return $w_n + OPT(p(n))$

Similar to stairs climbing,
# of calls here also $F(n) \approx \phi^n$

Do Memoization

OPT (n)
sort intervals according to finishing time
initialize a memory M
compute $p(1), \ldots, p(n)$
$M[0] = 0$
for (i=1 to n)
    $M[i] = \max \{ w_i + M[p(i)], M[i-1] \}$

Weighted Interval Scheduling

OPT (n)
sort intervals according to finishing time
if n = 0
    return 0
else
    find p(n)
    if OPT (n-1) $\geq w_n$ + OPT(p(n))
        return OPT (n-1)
    else
        return w_n + OPT (p(n))

Top-Down

Similar to stairs climbing,
# of calls here also $F(n) \approx \phi^n$

Do Memoization

OPT (n)
sort intervals according to finishing time
initialize a memory M
compute p(1), ... , p(n)
M[0] = 0
for (i=1 to n)
    $M[i] = \max \{ w_i + M[p(i)], M[i-1] \}$

Bottom-Up

Longest Common Subsequence

- given two sequence x[1…m] and y[1…n], find a longest subsequence common to both of them

  (doesn't need to be unique)

x : A B C B D A B

y : B D C A B A

Longest Common Subsequence

- given two sequence x[1…m] and y[1…n], find a longest subsequence common to both of them

  (doesn't need to be unique)

$x: A\ \underline{B}\ \underline{C}\ B\ D\ \underline{A}\ \underline{B}$

$y: \underline{B}\ D\ \underline{C}\ \underline{A}\ \underline{B}\ A$

LCS(x,y) = BCAB

Longest Common Subsequence

- given two sequence x[1…m] and y[1…n], find a longest subsequence common to both of them

(doesn't need to be unique)

x : A <u>B</u> <u>C</u> B D <u>A</u> <u>B</u>

y : <u>B</u> D <u>C</u> <u>A</u> <u>B</u> A

LCS(x,y) = BCAB

these subsets don't need to be continuous

Longest Common Subsequence

Brute-Force

Longest Common Subsequence

Brute-Force

- check every subsequence of $x[1\ldots m]$ whether it is a subsequence of $y[1\ldots n]$

Longest Common Subsequence

Brute-Force

- check every subsequence of $x[1\ldots m]$ whether it is a subsequence of $y[1\ldots n]$
- each check takes $O(n)$ time

Longest Common Subsequence

Brute-Force

- check every subsequence of $x[1\ldots m]$ whether it is a subsequence of $y[1\ldots n]$
- each check takes $O(n)$ time
- 2^m subsequence of x (each bit-vector defines a subsequence).

Longest Common Subsequence

Brute-Force

- check every subsequence of x[1…m] whether it is a subsequence of y[1…n]
- each check takes $O(n)$ time
- 2^m subsequence of x (each bit-vector defines a subsequence).
- total running time will be $O(2^m.n)$

Longest Common Subsequence

Simplified Version

- rather than directly calculating LCS(x,y), calculate the length of LCS(x,y) ($c[i,j] = |LCS(x,y)|$)

Longest Common Subsequence

Simplified Version

- rather than directly calculating LCS(x,y), calculate the length of LCS(x,y) ($c[i,j] = |LCS(x,y)|$)
- define subproblems

Longest Common Subsequence

Simplified Version

- rather than directly calculating LCS(x,y), calculate the length of LCS(x,y) ($c[i,j] = |LCS(x,y)|$)

- define subproblems

consider the prefix x[1…i] of x and the prefix y[1…j] of y

Longest Common Subsequence

Simplified Version

- rather than directly calculating LCS(x,y), calculate the length of LCS(x,y) ($c[i,j] = |LCS(x,y)|$)

- define subproblems

  consider the prefix $x[1\ldots i]$ of x and the prefix $y[1\ldots j]$ of y

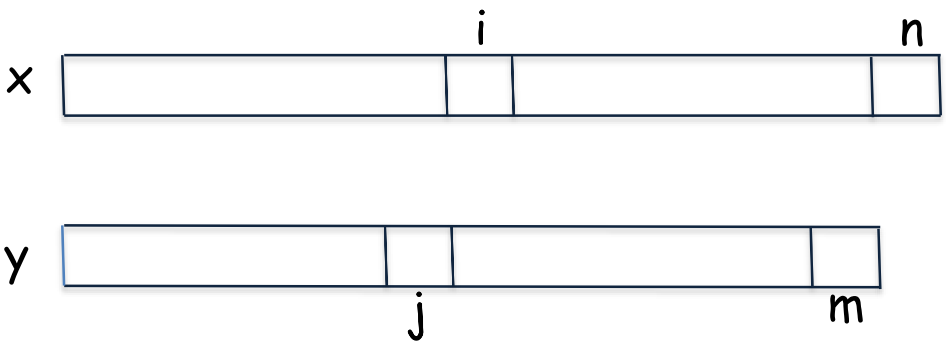
  $c[i,j] = |LCS(x[1\ldots i], y[1\ldots j])|$: length of the longest common subsequence of the prefixes $x[1\ldots i]$ and $y[1\ldots j]$

Longest Common Subsequence

- construct recurrence relation

Longest Common Subsequence

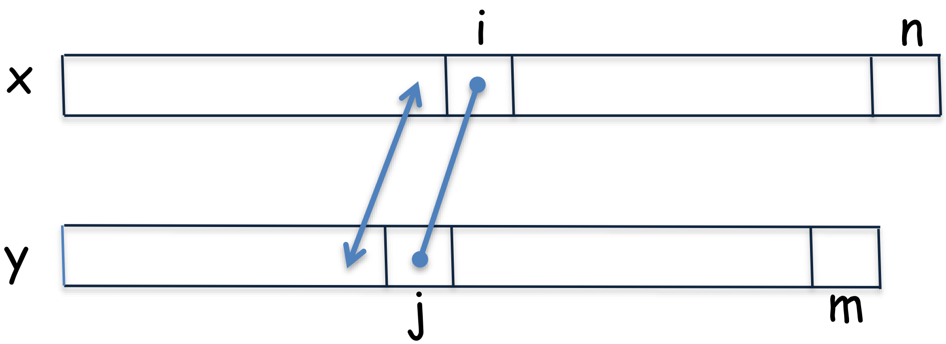
- construct recurrence relation



Two cases :

Longest Common Subsequence

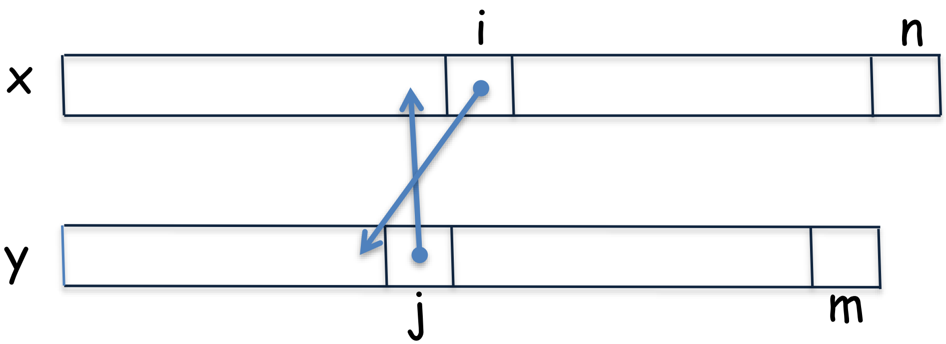
- construct recurrence relation



Two cases :
(1) if x[i] = y[j], then continue with
c[i-1,j-1] + 1

Longest Common Subsequence

- construct recurrence relation

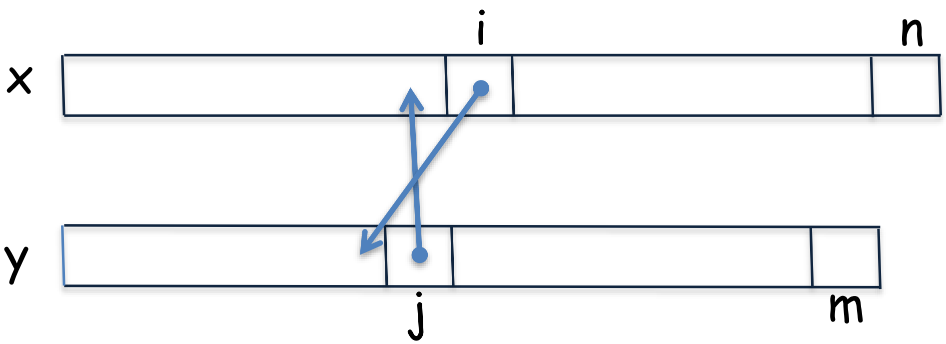


Two cases :
(1) if x[i] = y[j], then continue with
c[i-1,j-1] + 1

(2) otherwise continue with
max { c[i,j-1], c[i-1,j] }

Longest Common Subsequence

- construct recurrence relation



Two cases :

(1) if x[i] = y[j], then continue with
c[i-1,j-1] + 1

(2) otherwise continue with
max { c[i,j-1], c[i-1,j] }

$$c[i,j] = \begin{cases} c[i-1,j-1] + 1 & \text{if } x[i] = y[j] \\ \max \{ c[i,j-1], c[i-1,j] \} & \text{otherwise} \end{cases}$$

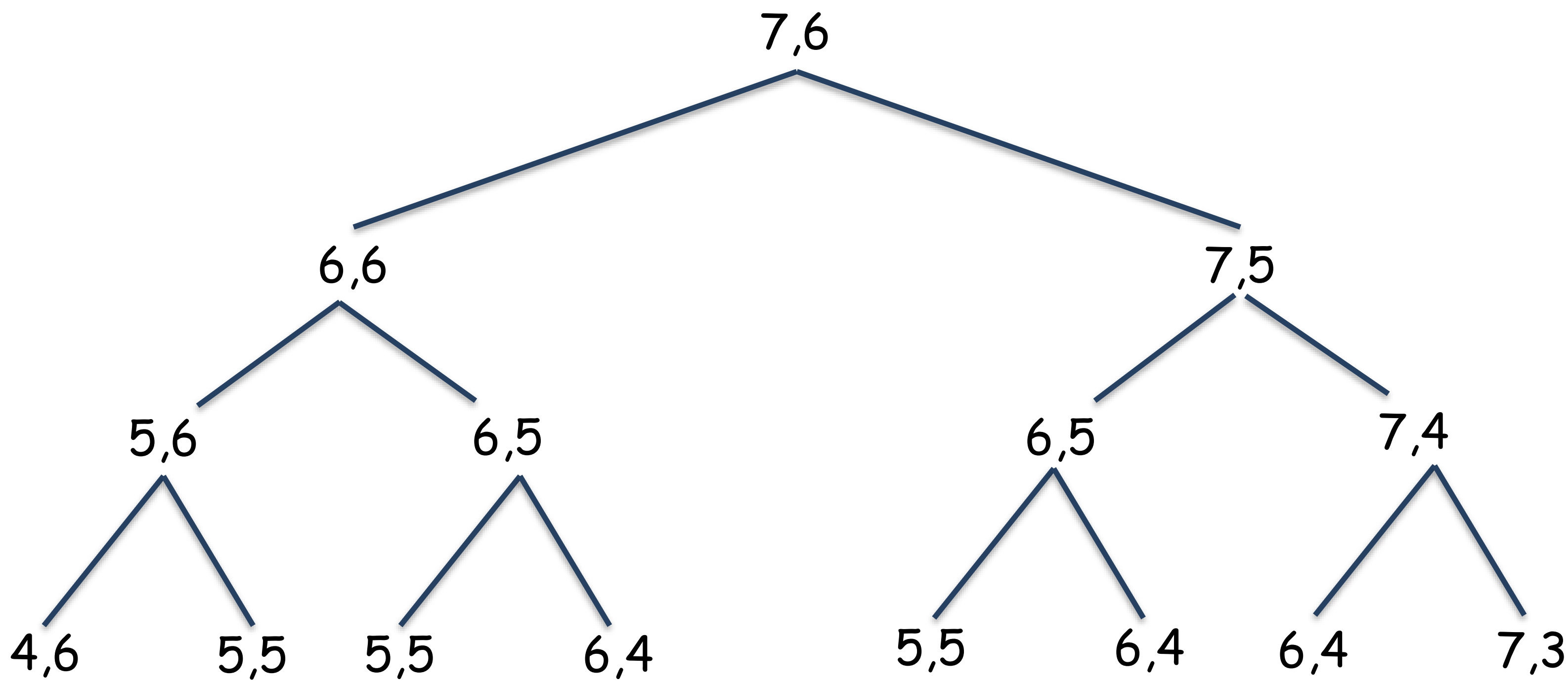
Longest Common Subsequence

LCS (x,y,n,m)

```
if i = 0 and j = 0
    return 0
if x[n] = y[m]
    c[n,m] = LCS(x,y,n-1,m-1) + 1
else
    c[n,m] = max { LCS(x,y,n-1,m), LCS(x,y,n,m-1) }
return c[n,m]
```

Longest Common Subsequence

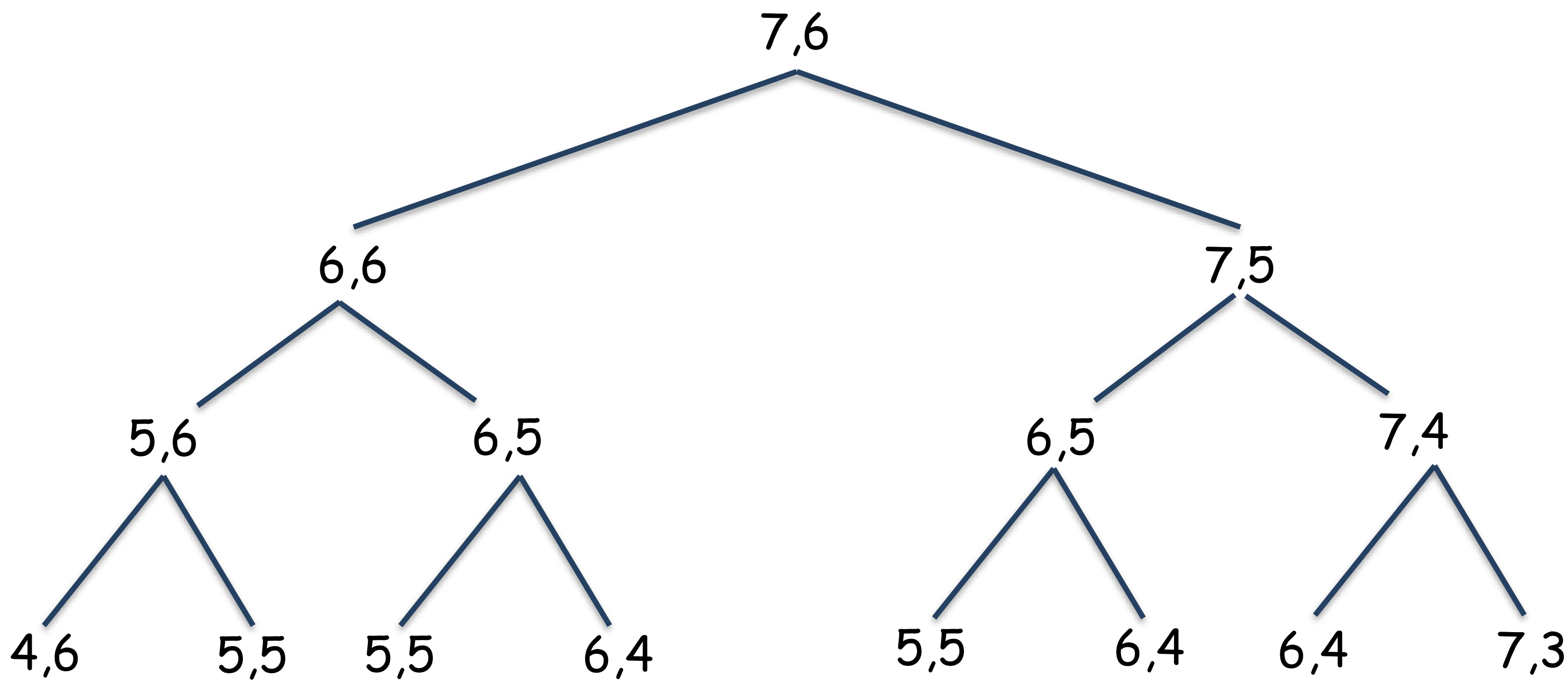
Let's check it for m = 6, n = 7



The height of the tree m + n,

Longest Common Subsequence

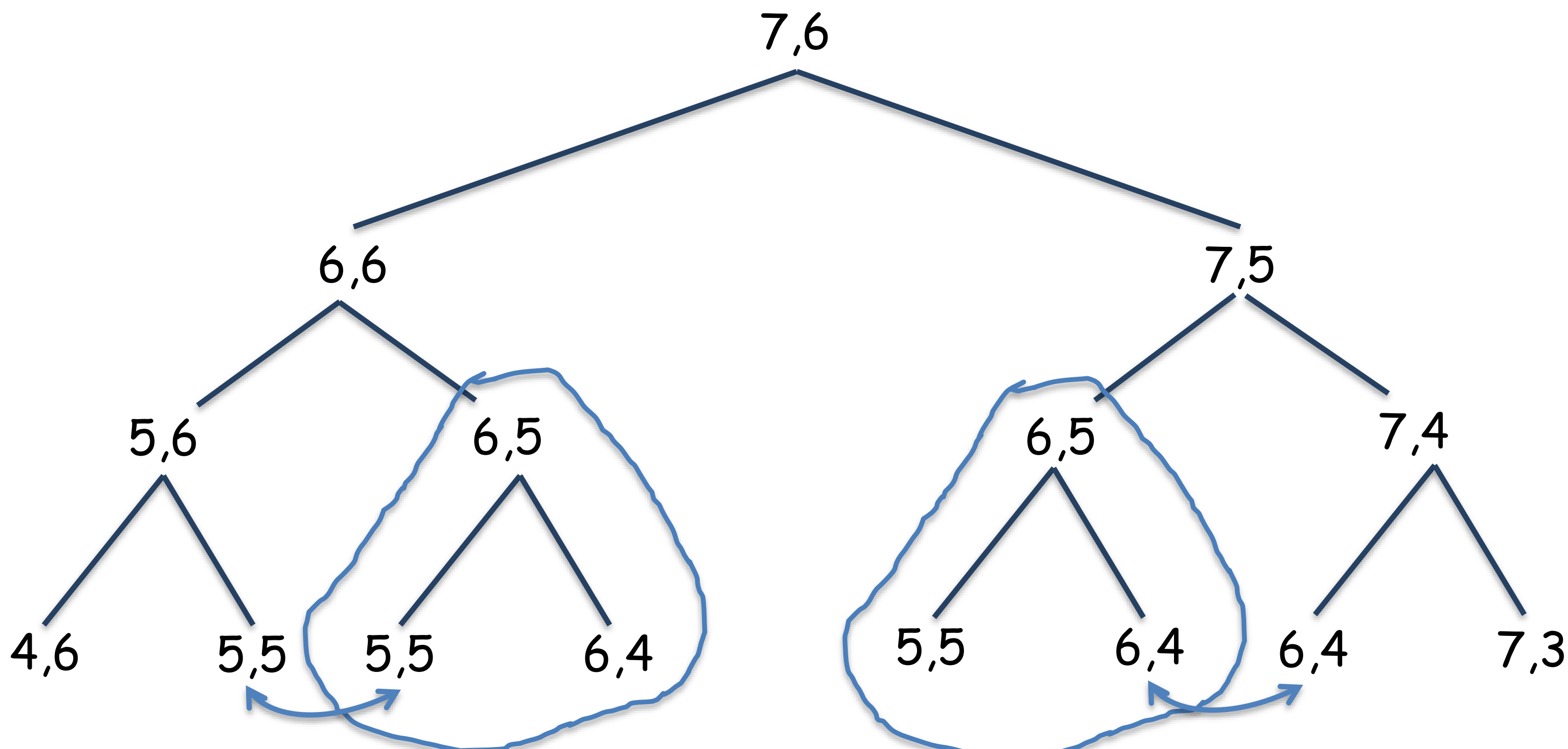
Let's check it for m = 6, n = 7



The height of the tree m + n,
So the running time will be $O(2^{m+n})$

Longest Common Subsequence

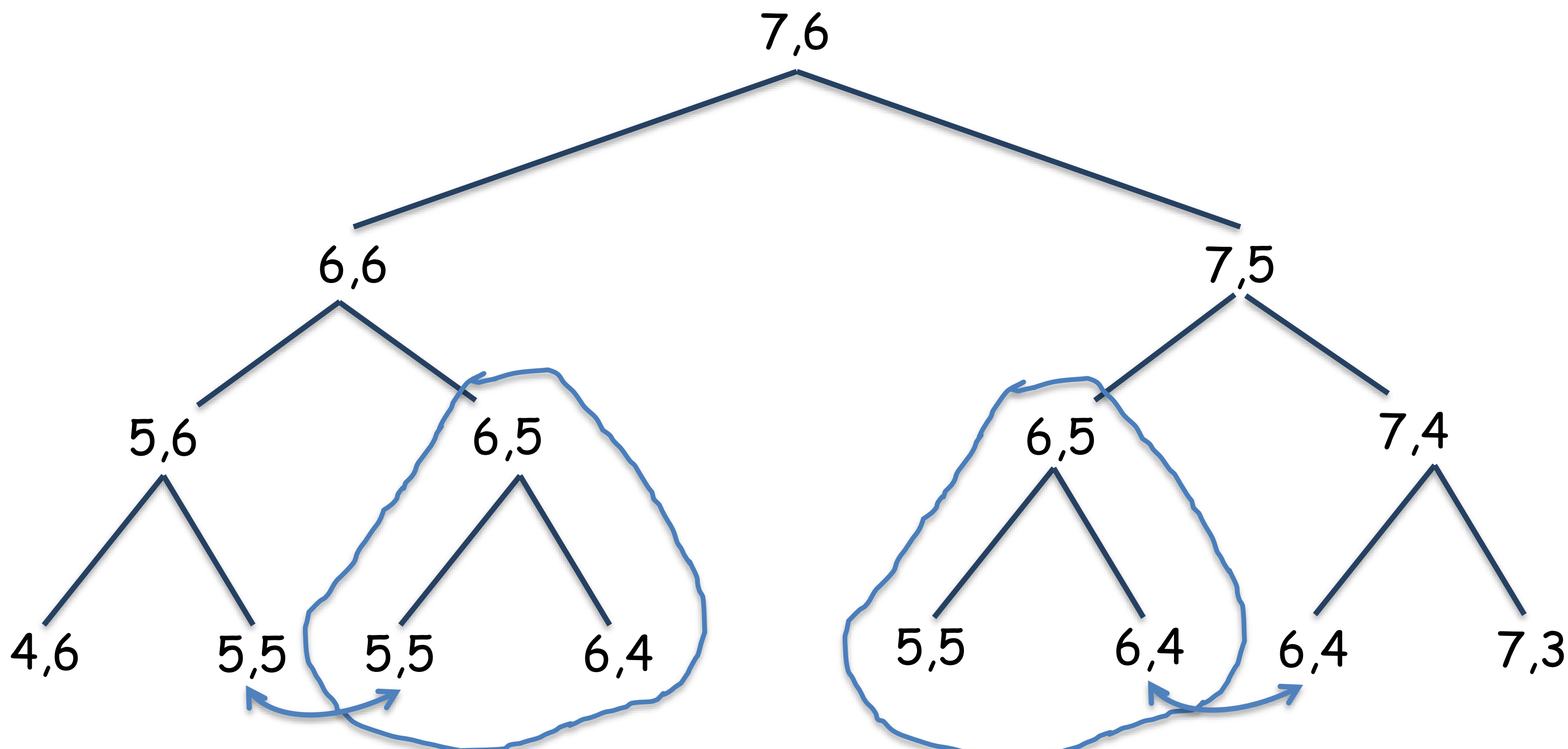
Let's check it for m = 6, n = 7



The height of the tree m + n,
So the running time will be $O(2^{m+n})$

Longest Common Subsequence

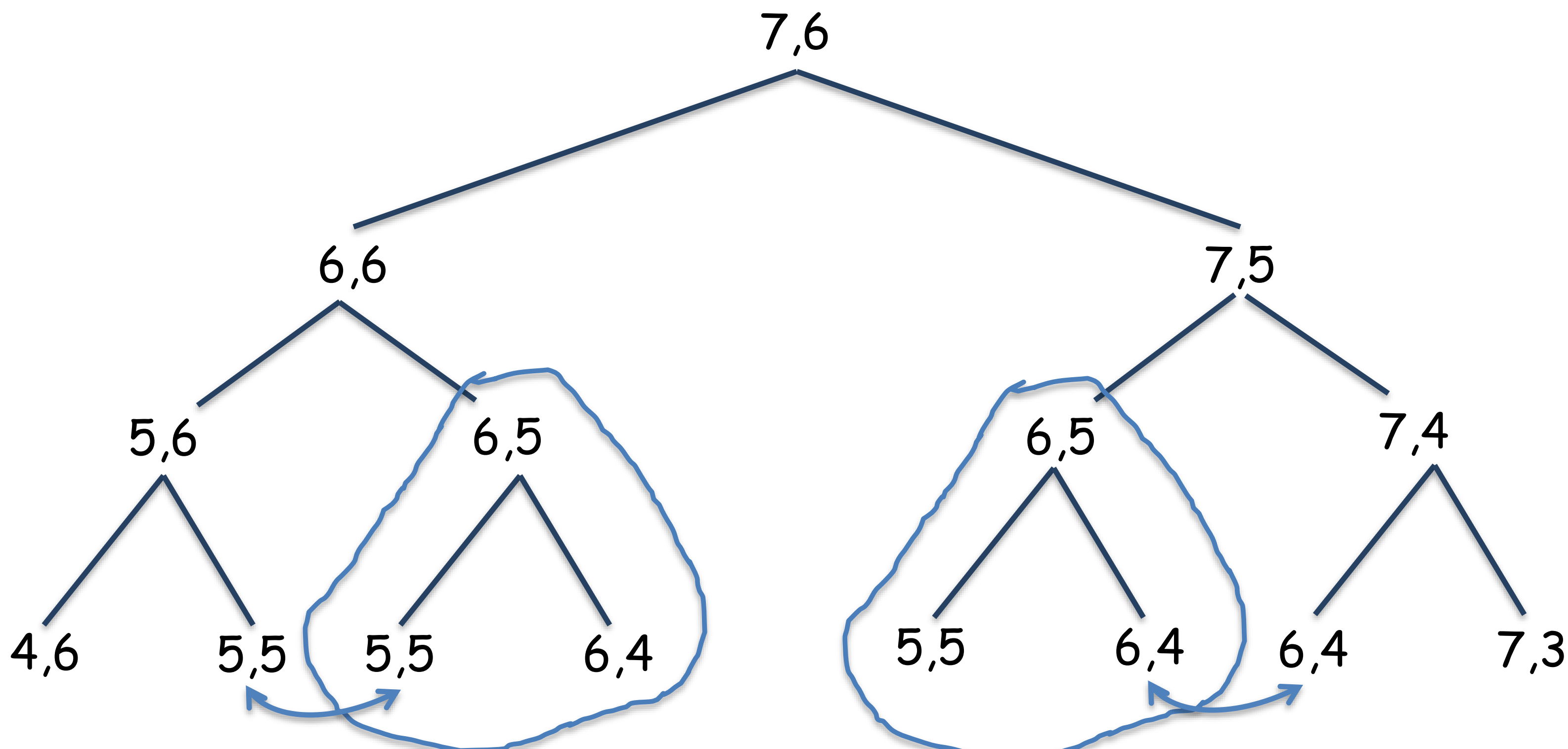
Let's check it for m = 6, n = 7



The height of the tree m + n,
So the running time will be $O(2^{m+n})$
How many different subproblems are there?

Longest Common Subsequence

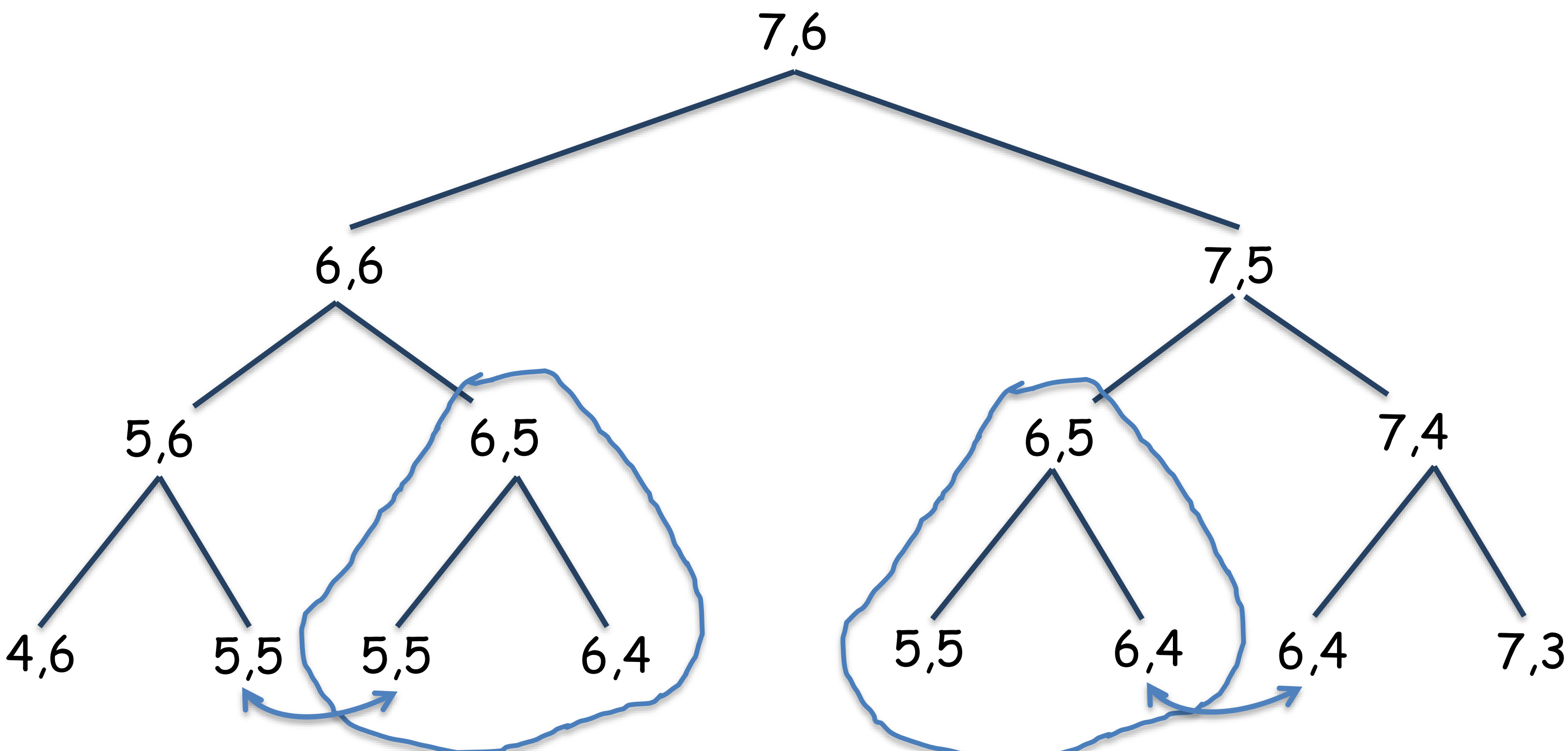
Let's check it for m = 6, n = 7



The height of the tree m + n,
So the running time will be $O(2^{m+n})$
How many different subproblems are there? (m.n)

Longest Common Subsequence

Let's check it for m = 6, n = 7



The height of the tree m + n,

So the running time will be $O(2^{m+n})$

How many different subproblems are there? (m.n)

Do Memoization

Longest Common Subsequence

LCS (x,y,n,m) (with Memoization)

```
initialize a memory M
M[0,0] = 0
if M[n,m] = null
      if x[n] = y[m]
            M[n,m] = LCS(x,y,n-1,m-1) + 1
      else
            M[n,m] = max { LCS(x,y,n-1,m), LCS(x,y,n,m-1) }
return M[n,m]
```

Longest Common Subsequence

LCS (x,y,n,m) (with Memoization)

running time : O(m.n)
space : O(m.n)

```
initialize a memory M
M[0,0] = 0
if M[n,m] = null
    if x[n] = y[m]
        M[n,m] = LCS(x,y,n-1,m-1) + 1
    else
        M[n,m] = max { LCS(x,y,n-1,m), LCS(x,y,n,m-1) }
return M[n,m]
```

Longest Common Subsequence

LCS (x,y,n,m) (with Memoization)

```
initialize a memory M
M[0,0] = 0
if M[n,m] = null
      if x[n] = y[m]
            M[n,m] = LCS(x,y,n-1,m-1) + 1
      else
            M[n,m] = max { LCS(x,y,n-1,m), LCS(x,y,n,m-1) }
return M[n,m]
```

running time : $O(m.n)$
space : $O(m.n)$

Top-Down

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
   for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
   for j=1 to m
      if x[i] = y[j]
         M[i,j] = M[i-1,j-1] + 1
      else
         M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

		A	B	C	B	A
	0	0	0	0	0	0
B	0					
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 1

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0					
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
      if x[i] = y[j]
         M[i,j] = M[i-1,j-1] + 1
      else
         M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

$j = 1$

$i = 1$

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0				
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
  for j=1 to m
     if x[i] = y[j]
        M[i,j] = M[i-1,j-1] + 1
     else
        M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 2

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0				
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
   for j=1 to m
        if x[i] = y[j]
             M[i,j] = M[i-1,j-1] + 1
        else
             M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 2

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1			
D	0					
C	0					
A	0					

Longest Common Subsequence

j = 3

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
   for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1			
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
      if x[i] = y[j]
         M[i,j] = M[i-1,j-1] + 1
      else
         M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 3

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1		
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 4

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1		
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 4

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
    for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 5

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
    for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 5

i = 1

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
    for j=1 to m
         if x[i] = y[j]
             M[i,j] = M[i-1,j-1] + 1
         else
             M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 1

i = 2

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0					
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
    for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 1

i = 2

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0				
C	0					
A	0					

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
     M[i,0] = 0
for i=1 to m
     M[0,i] = 0
for i=1 to n
    for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 5

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

Longest Common Subsequence

j = 5

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
      if x[i] = y[j]
         M[i,j] = M[i-1,j-1] + 1
      else
         M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Longest Common Subsequence

j = 5

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

A

We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Longest Common Subsequence

j = 5

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
      if x[i] = y[j]
         M[i,j] = M[i-1,j-1] + 1
      else
         M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

A

We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Longest Common Subsequence

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

j = 5

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

C A

We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Longest Common Subsequence

j = 5

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
    for j=1 to m
        if x[i] = y[j]
            M[i,j] = M[i-1,j-1] + 1
        else
            M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 4

			A	B	C	B	A
	0	0	0	0	0	0	
B	0	0	1	1	1	1	
D	0	0	1	1	1	1	
C	0	0	1	2	2	2	
A	0	1	1	2	2	3	

C A

We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Longest Common Subsequence

j = 5

Bottom-up

```
initialize a memory M
for i=0 to n
    M[i,0] = 0
for i=1 to m
    M[0,i] = 0
for i=1 to n
   for j=1 to m
       if x[i] = y[j]
          M[i,j] = M[i-1,j-1] + 1
       else
          M[i,j] = max { M[i-1,j], M[i,j-1] }
return M[n,m]
```

i = 4

		A	B	C	B	A
	0	0	0	0	0	0
B	0	0	1	1	1	1
D	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	3

B C A

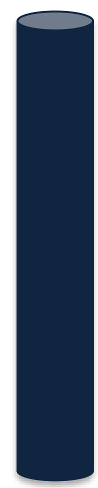
We can reconstruct LCS by tracing backwards
Whenever we have a diagonal, we have a match!

Rod Cutting

- given a rod of length n with the prices $p_1,\ldots,p_n$ where p_i is the price of a rod of length i, find an optimal way of cutting the given rod that maximizes the profit

Rod Cutting

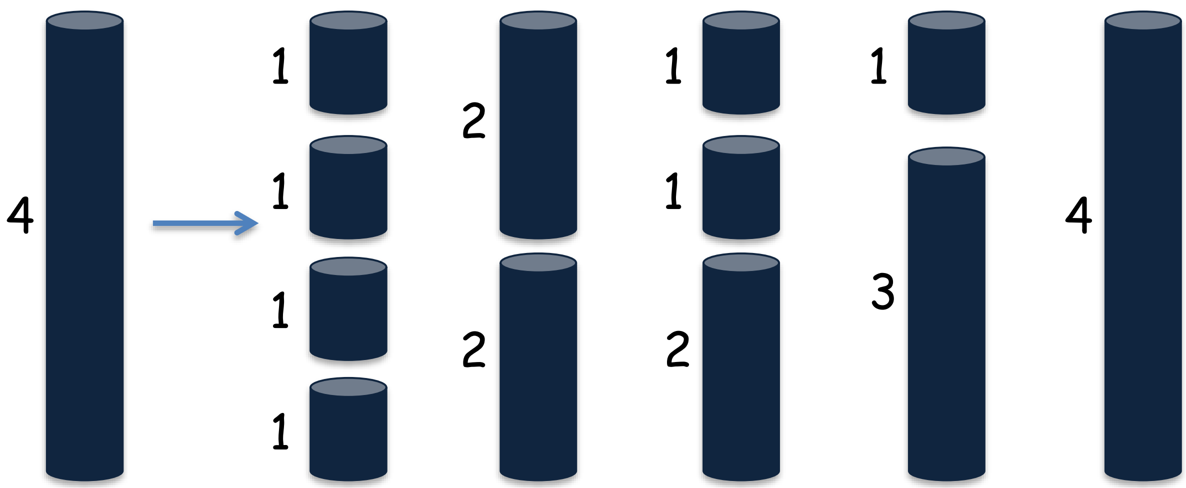
- given a rod of length n with the prices $p_1,\dots,p_n$ where p_i is the price of a rod of length i, find an optimal way of cutting the given rod that maximizes the profit



length	1	2	3	4
price	1	5	8	9

Rod Cutting

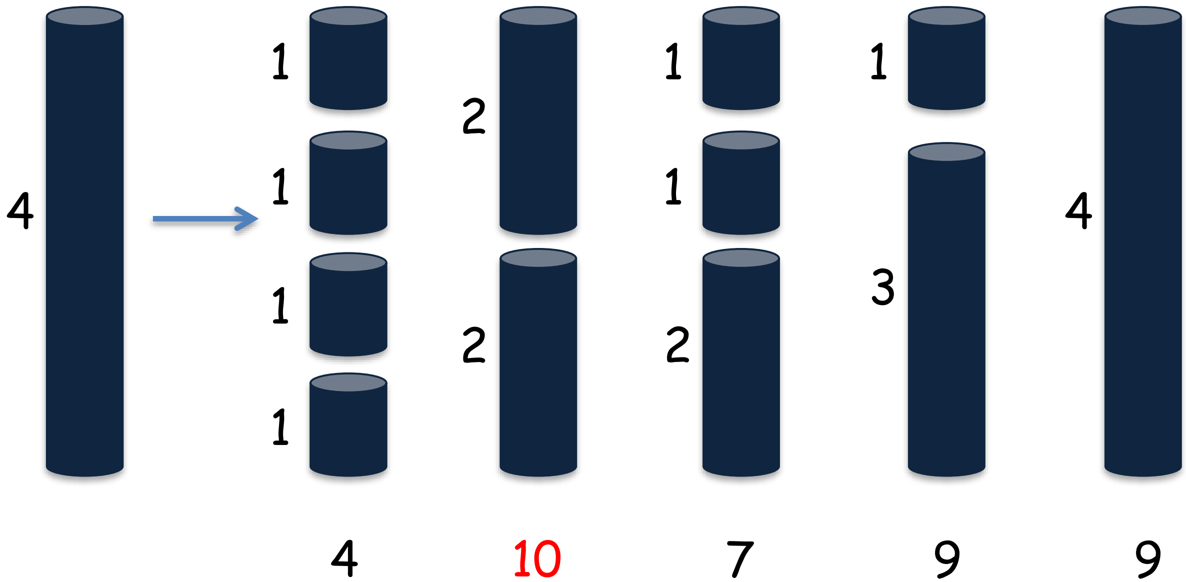
- given a rod of length n with the prices $p_1,\ldots,p_n$ where p_i is the price of a rod of length i, find an optimal way of cutting the given rod that maximizes the profit



length	1	2	3	4
price	1	5	8	9

Rod Cutting

- given a rod of length n with the prices $p_1,\ldots,p_n$ where p_i is the price of a rod of length i, find an optimal way of cutting the given rod that maximizes the profit



length	1	2	3	4
price	1	5	8	9

Rod Cutting

- define subproblems

Rod Cutting

- define subproblems

c(i) : max profit for the first length i part

Rod Cutting

- define subproblems

  $c(i)$: max profit for the first length i part

- construct recurrence relation

Rod Cutting

- define subproblems

c(i) : max profit for the first length i part

- construct recurrence relation

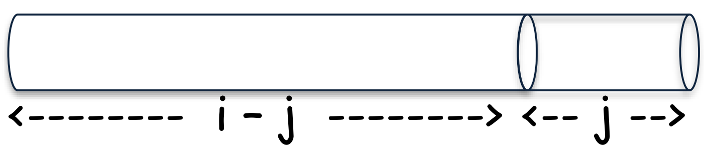
c(i) =

Rod Cutting

- define subproblems

c(i) : max profit for the first length i part

- construct recurrence relation



- focus on last cutting

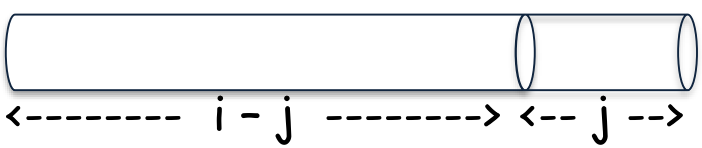
c(i) =

Rod Cutting

- define subproblems

c(i) : max profit for the first length i part

- construct recurrence relation



- focus on last cutting
- find best j maximizing the profit

$$c(i) = \max \{ p_j +$$

Rod Cutting

- define subproblems

  c(i) : max profit for the first length i part

- construct recurrence relation

<--------- i – j --------> <-- j -->

$$c(i) = \max \{ p_j + c(i-j) \}$$

- focus on last cutting
- find best j maximizing the profit
- recursively continue on the remaining part

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

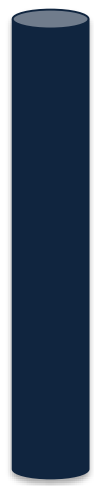
```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

Rod Cutting

```
input : ( n ; p_1 , p_2 , ... , p_n )

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



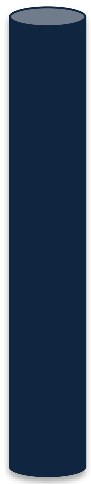
	1	2	3	4	5
p_i	1	5	8	9	10
M[i]					

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[1] = - ∞

j = 1

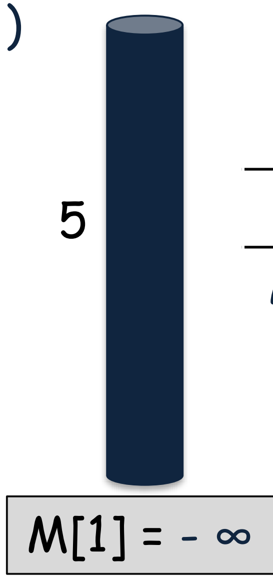
	1	2	3	4	5
p_i	1	5	8	9	10
M[i]					

i = 1

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```



j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]					

i = 1

$q = M[0] + p_1$

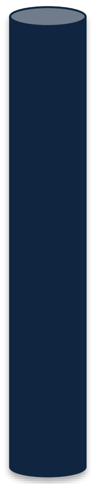
$q = 1$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[1] = 1

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

i = 1

$q = M[0] + p_1$

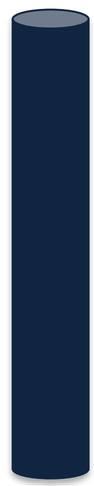
$q = 1$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[2] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

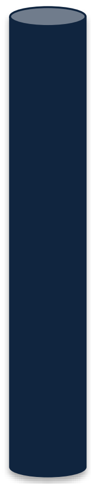
i = 2

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[2] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

i = 2

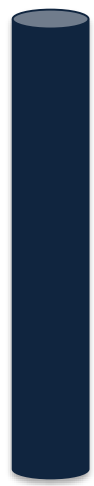
$q = M[1] + p_1$
$q = 2$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i-j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



M[2] = 2

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

i = 2

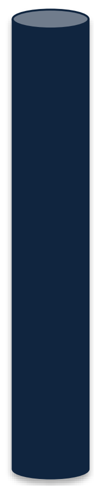
$q = M[1] + p_1$
$q = 2$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[2] = 2

$j = 2$

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

$i = 2$

$q = M[1] + p_1$
$q = 2$

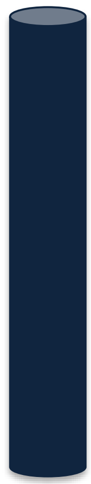
$q = M[0] + p_2$
$q = 5$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[2] = 5

j = 2

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1				

i = 2

q = M[1] + p_1
q = 2

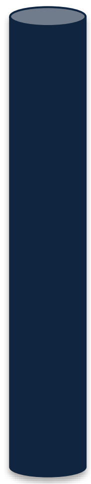
q = M[0] + p_2
q = 5

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[2] = 5

j = 2

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

i = 2

q = M[1] + p_1
q = 2

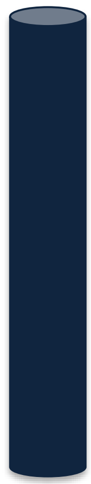
q = M[0] + p_2
q = 5

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[3] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

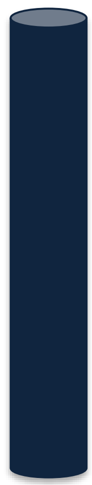
i = 3

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i-j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



M[3] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

i = 3

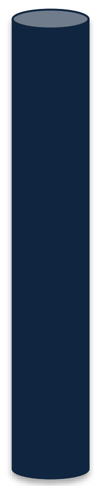
$q = M[2] + p_1$
$q = 6$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i-j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



$\boxed{M[3] = 6}$

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

i = 3

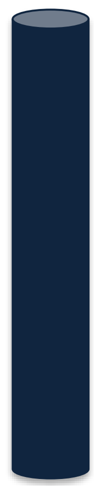
$q = M[2] + p_1$
$q = 6$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
$M[0] = 0$
for i=1 to n
    $M[i] = -\infty$
    for j=1 to i
        $q = M[i-j] + p_j$
        if $q > M[i]$
            $M[i] = q$
return $M[n]$

5



$\boxed{M[3] = 6}$

$j = 2$

	1	2	3	4	5
p_i	1	5	8	9	10
$M[i]$	1	5 2			

$i = 3$

$q = M[2] + p_1$
$q = 6$

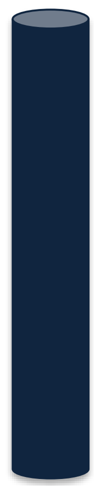
$q = M[1] + p_2$
$q = 6$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[3] = 6

j = 3

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

i = 3

$q = M[2] + p_1$
$q = 6$

$q = M[1] + p_2$
$q = 6$

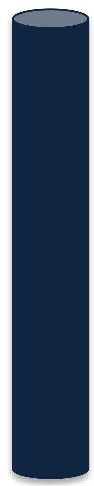
$q = M[0] + p_3$
$q = 8$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i\text{-}j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



$\boxed{M[3] = 8}$

j = 3

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2			

i = 3

$q = M[2] + p_1$
$q = 6$

$q = M[1] + p_2$
$q = 6$

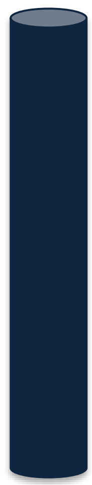
$q = M[0] + p_3$
$q = 8$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i-j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



M[3] = 8

j = 3

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2	8 3		

i = 3

$q = M[2] + p_1$
$q = 6$

$q = M[1] + p_2$
$q = 6$

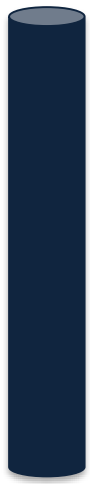
$q = M[0] + p_3$
$q = 8$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[4] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 [2]	8 [3]		

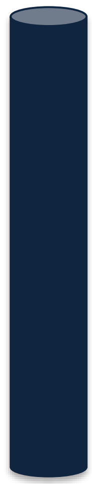
i = 4

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[4] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 [2]	8 [3]		

i = 4

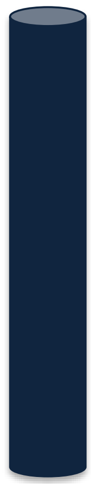
q = M[3] + p_1
q = 9

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



$M[4] = 9$

$j = 1$

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 ^{2}	8 ^{3}		

$i = 4$

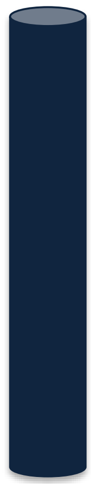
$q = M[3] + p_1$
$q = 9$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



$M[4] = 9$

j = 2

	1	2	3	4	5
p_i	1	5	8	9	10
$M[i]$	1	5 [2]	8 [3]		

i = 4

$q = M[3] + p_1$
$q = 9$

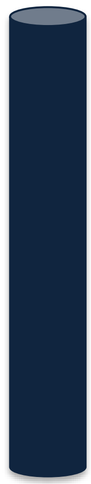
$q = M[2] + p_2$
$q = 10$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[4] = 10

j = 2

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 ^{2}	8 ^{3}		

i = 4

q = M[3] + p_1
q = 9

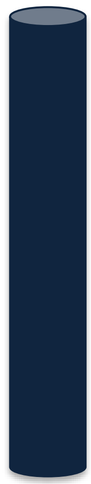
q = M[2] + p_2
q = 10

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



$\boxed{M[4] = 10}$

j = 3

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2	8 3		

i = 4

$q = M[3] + p_1$
$q = 9$

$q = M[1] + p_3$
$q = 9$

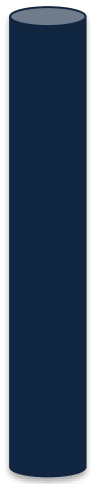
$q = M[2] + p_2$
$q = 10$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[4] = 10

j = 4

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 [2]	8 [3]		

i = 4

$q = M[3] + p_1$
$q = 9$

$q = M[1] + p_3$
$q = 9$

$q = M[2] + p_2$
$q = 10$

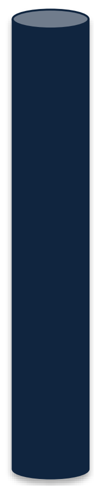
$q = M[0] + p_4$
$q = 9$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[4] = 10

j = 4

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2	8 3	10 2	

i = 4

$q = M[3] + p_1$
$q = 9$

$q = M[1] + p_3$
$q = 9$

$q = M[2] + p_2$
$q = 10$

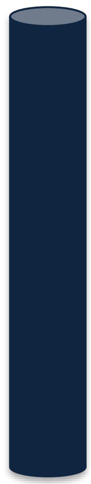
$q = M[0] + p_4$
$q = 9$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[5] = - ∞

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 [2]	8 [3]	10 [2]	

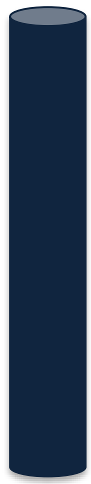
i = 5

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



$M[5] = -\infty$

$j = 1$

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5^{2}	8^{3}	10^{2}	

$i = 5$

$q = M[4] + p_1$

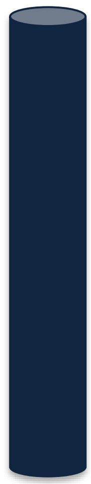
$q = 11$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[5] = 11

j = 1

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 2	8 3	10 2	

i = 5

$q = M[4] + p_1$

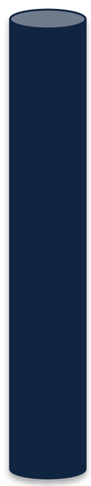
$q = 11$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]

5



M[5] = 11

j = 2

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5^{2}	8^{3}	10^{2}	

i = 5

q = M[4] + p_1
q = 11

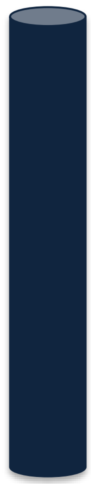
q = M[3] + p_2
q = 13

Rod Cutting

```
input : ( n ; p1 , p2 , ... , pn )

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + pj
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[5] = 13

$j = 2$

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5^{2}	8^{3}	10^{2}	

$i = 5$

$q = M[4] + p_1$
$q = 11$

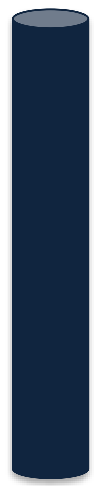
$q = M[3] + p_2$
$q = 13$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        $q = M[i-j] + p_j$
        if q > M[i]
            M[i] = q
return M[n]

5



$\boxed{M[5] = 13}$

j = 3

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 [2]	8 [3]	10 [2]	

i = 5

$q = M[4] + p_1$
$q = 11$

$q = M[3] + p_2$
$q = 13$

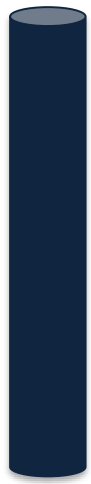
$q = M[2] + p_3$
$q = 13$

Rod Cutting

input : (n ; p_1 , p_2 , ... , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

5



$\boxed{M[5] = 13}$

j = 4

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5^{2}	8^{3}	10^{2}	

i = 5

$q = M[4] + p_1$
$q = 11$

$q = M[3] + p_2$
$q = 13$

$q = M[2] + p_3$
$q = 13$

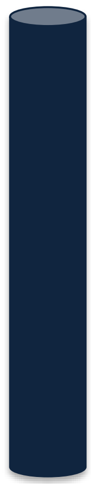
$q = M[1] + p_4$
$q = 10$

Rod Cutting

input : (n ; p_1 , p_2 , … , p_n)

initialize a memory M
$M[0] = 0$
for i=1 to n
    $M[i] = -\infty$
    for j=1 to i
        $q = M[i-j] + p_j$
        if $q > M[i]$
            $M[i] = q$
return $M[n]$

5



$M[5] = 13$

$j = 5$

	1	2	3	4	5
p_i	1	5	8	9	10
$M[i]$	1	5^{2}	8^{3}	10^{2}	

$i = 5$

$q = M[4] + p_1$
$q = 11$

$q = M[3] + p_2$
$q = 13$

$q = M[2] + p_3$
$q = 13$

$q = M[1] + p_4$
$q = 10$

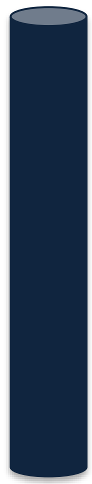
$q = M[0] + p_5$
$q = 10$

Rod Cutting

```
input : ( n ; p1 , p2 , ... , pn )

initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + pj
        if q > M[i]
            M[i] = q
return M[n]
```

5



M[5] = 13

j = 5

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 ^{2}	8 ^{3}	10 ^{2}	13 ^{2}

i = 5

$q = M[4] + p_1$
$q = 11$

$q = M[3] + p_2$
$q = 13$

$q = M[2] + p_3$
$q = 13$

$q = M[1] + p_4$
$q = 10$

$q = M[0] + p_5$
$q = 10$

Rod Cutting

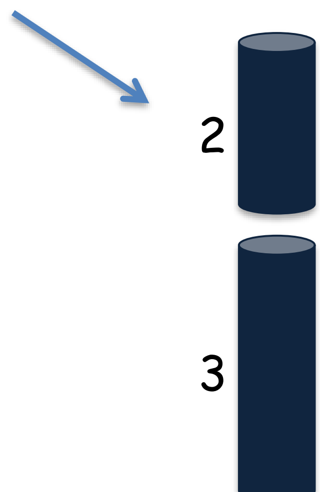
input : (n ; p_1 , p_2 , … , p_n)

```
initialize a memory M
M[0] = 0
for i=1 to n
    M[i] = - ∞
    for j=1 to i
        q = M[i-j] + p_j
        if q > M[i]
            M[i] = q
return M[n]
```

j = 5

	1	2	3	4	5
p_i	1	5	8	9	10
M[i]	1	5 ^{2}	8 ^{3}	10 ^{2}	13 ^{2}

i = 5



Matrix Chain Multiplication

- given bunch of matrices $A_0,\ldots,A_{n-1}$, find an optimal parenthesization that minimizes the cost of multiplication

Matrix Chain Multiplication

- given bunch of matrices $A_0,\ldots,A_{n-1}$, find an optimal parenthesization that minimizes the cost of multiplication

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} . \begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mk} \end{pmatrix}$$

$$c_{ij} =$$

Matrix Chain Multiplication

- given bunch of matrices $A_0,\ldots,A_{n-1}$, find an optimal parenthesization that minimizes the cost of multiplication

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \boxed{a_{i1} \quad a_{i2} \quad \ldots \quad a_{in}} \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} . \begin{pmatrix} b_{11} & \boxed{\begin{matrix} b_{1j} \\ b_{2j} \\ . \\ . \\ . \\ b_{nj} \end{matrix}} & b_{1k} \\ \vdots & & \vdots \\ b_{n1} & & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mk} \end{pmatrix}$$

$$c_{ij} =$$

Matrix Chain Multiplication

- given bunch of matrices $A_0,\ldots,A_{n-1}$, find an optimal parenthesization that minimizes the cost of multiplication

$$\begin{pmatrix} a_{11} & \cdots & & a_{1n} \\ a_{i1} & a_{i2} & \ldots & a_{in} \\ a_{m1} & \cdots & & a_{mn} \end{pmatrix} . \begin{pmatrix} b_{11} & b_{1j} & b_{1k} \\ \vdots & b_{2j} & \vdots \\ & \vdots & \\ b_{n1} & b_{nj} & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mk} \end{pmatrix}$$

$$c_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + \cdots + a_{in}.b_{nj}$$

Matrix Chain Multiplication

- given bunch of matrices $A_0,\dots,A_{n-1}$, find an optimal parenthesization that minimizes the cost of multiplication

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} . \begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mk} \end{pmatrix}$$

$$c_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + \cdots + a_{in}.b_{nj}$$

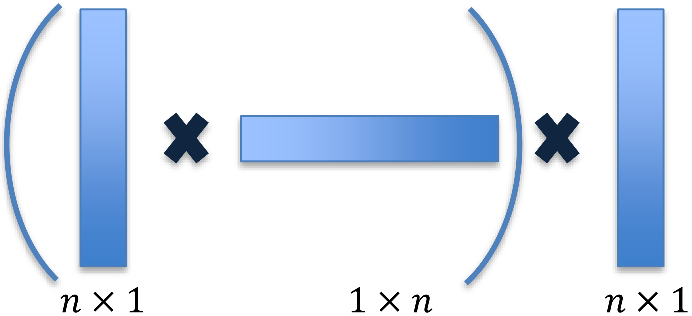
- for each c_{ij}, n multiplications and n-1 additions; O(n) operations
- m.k entries in C, thus total O(m.n.k) operations (simply m.n.k) operations.

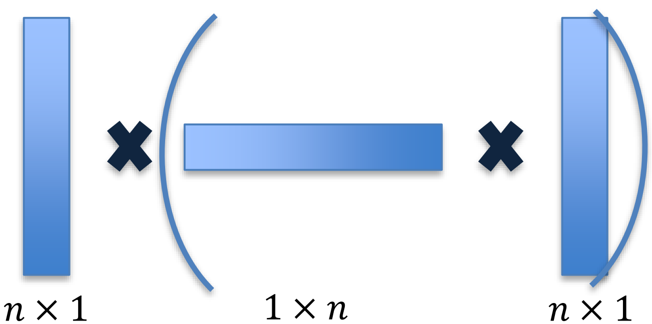
Matrix Chain Multiplication

$n \times 1$ $1 \times n$ $n \times 1$

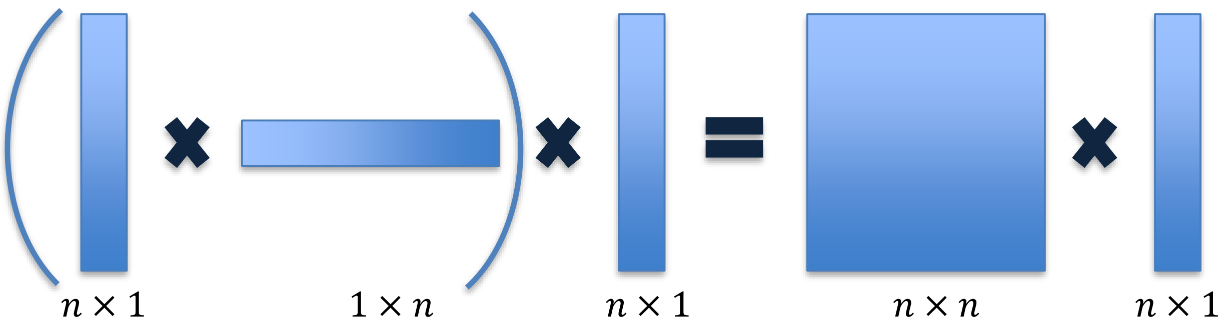
$n \times 1$ $1 \times n$ $n \times 1$

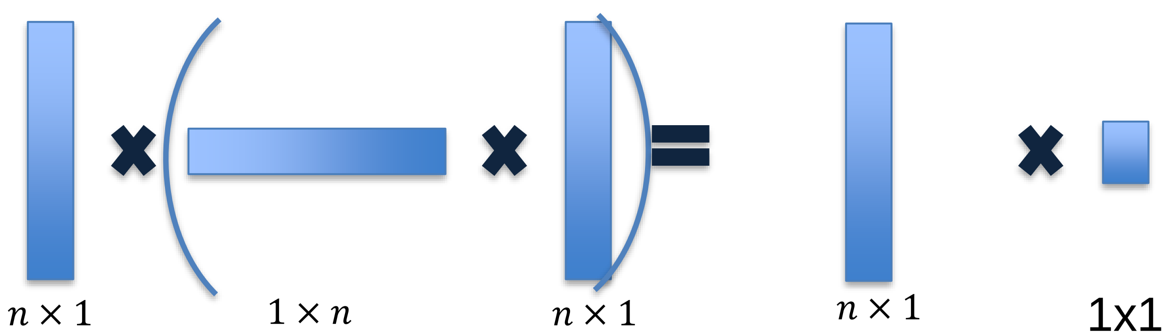
Matrix Chain Multiplication



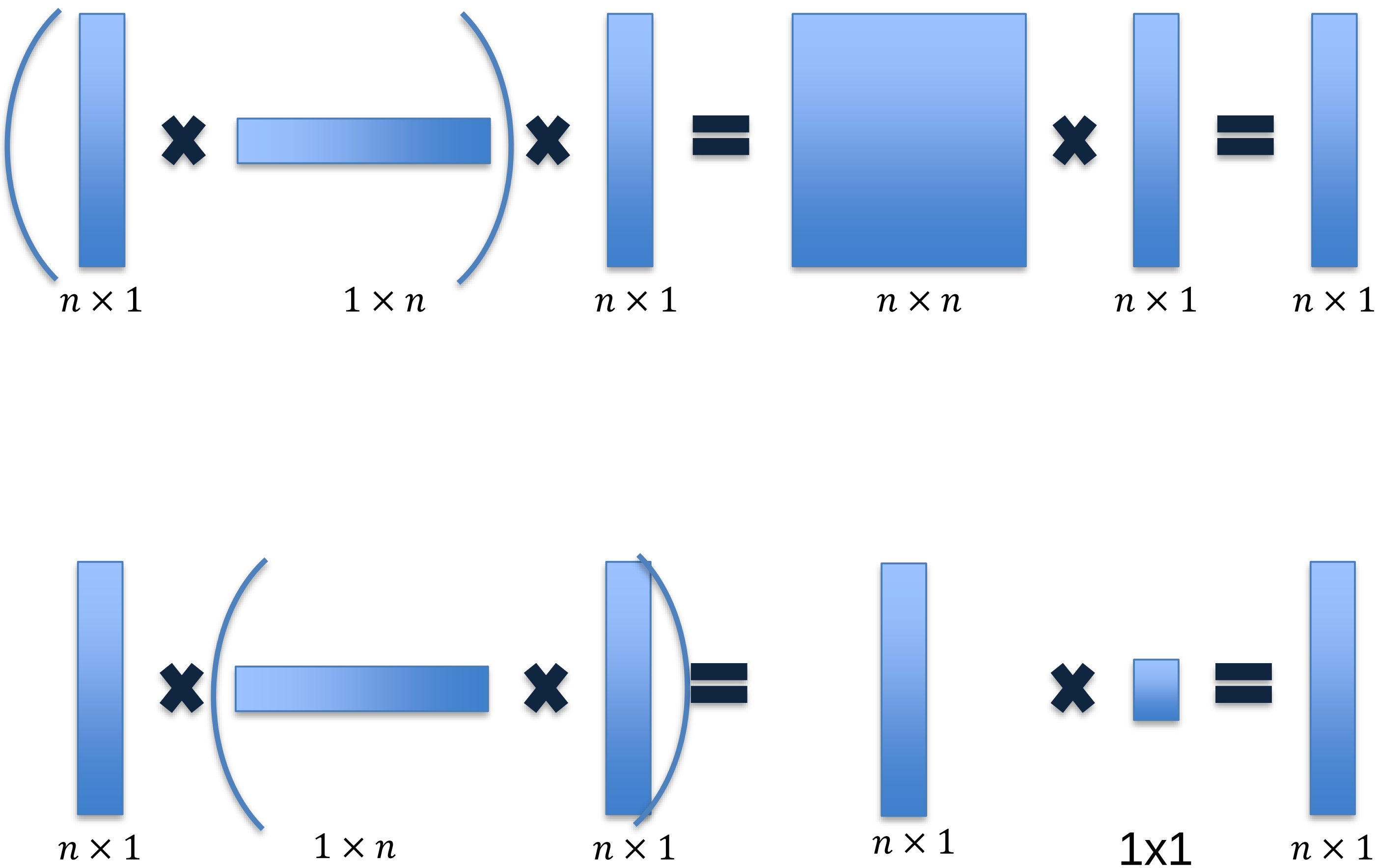


Matrix Chain Multiplication

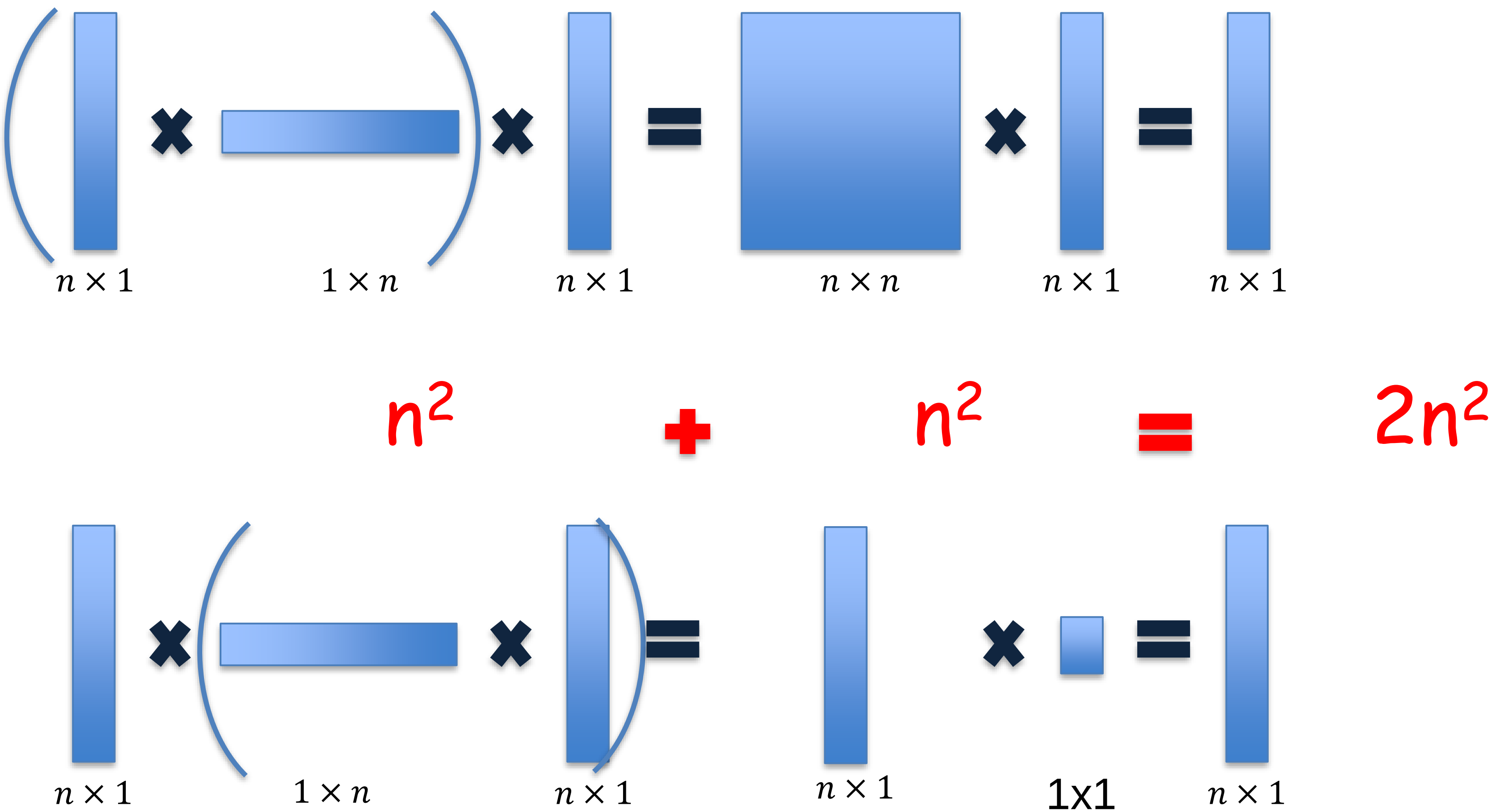




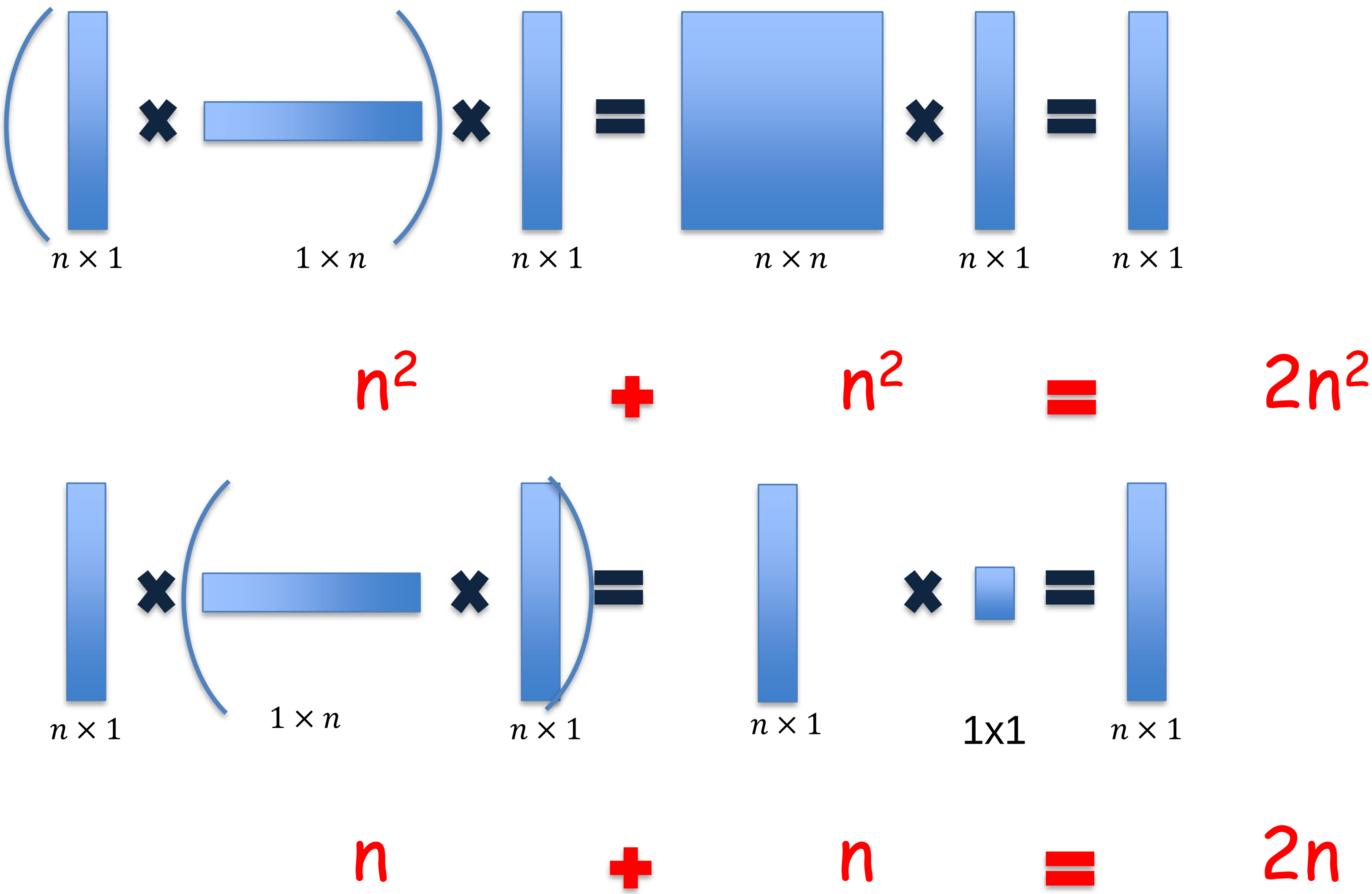
Matrix Chain Multiplication



Matrix Chain Multiplication



Matrix Chain Multiplication



Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization

total cost

Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization A_1 A_2 A_3 A_4

total cost

Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization $A_1 (A_2 (A_3 A_4)))$

total cost

Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization $A_1 (A_2 (A_3 A_4)))$

total cost cde +

Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization $A_1\,(A_2\ (A_3\,A_4))) = A_1\,(A_2\,B)$

total cost cde + bce +

Matrix Chain Multiplication

A_1	A_2	A_3	A_4
axb	bxc	cxd	dxe

paranthesization $A_1 (A_2\ (A_3\ A_4))) = A_1 (A_2\ B) = A_1\ C$

total cost cde + bce + abe

Matrix Chain Multiplication

- define subproblems

Matrix Chain Multiplication

- define subproblems

OPT(i,j) : optimal parenthesization of $A_i,\ldots,A_j$

Matrix Chain Multiplication

- define subproblems

  OPT(i,j) : optimal parenthesization of $A_i,\ldots,A_j$

- construct recurrence relation

Matrix Chain Multiplication

- define subproblems

$OPT(i,j)$: optimal parenthesization of $A_i,\ldots,A_j$

- construct recurrence relation

$$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$$

- focus on last move
  (last parenthesization)

Matrix Chain Multiplication

- define subproblems

$OPT(i,j)$: optimal parenthesization of $A_i, \ldots, A_{j-1}$

- construct recurrence relation

$$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$$

- focus on last move (last parenthesization)
- find best k minimizing the cost

Matrix Chain Multiplication

- define subproblems

$OPT(i,j)$: optimal parenthesization of $A_i,\dots,A_{j-1}$

- construct recurrence relation

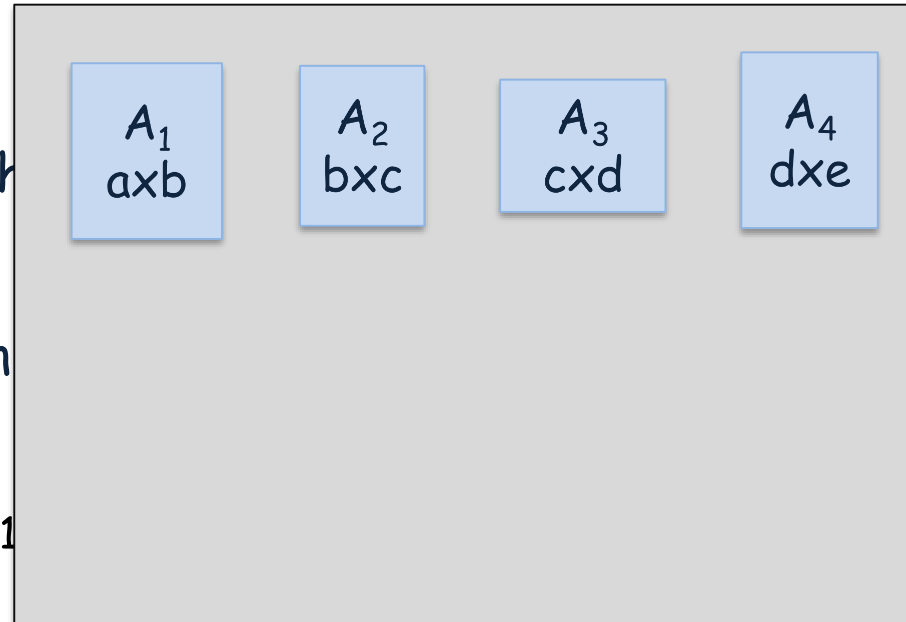
$$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$$

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

Matrix Chain Multiplication

- define subproblems

  OPT(i,j) : optimal parenth



- construct recurrence relation

  $(A_i \,.\,.\,.\, A_k) (A_{k+1}$

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

OPT(i,j) =min { OPT(i,k) + OPT(k+1,j) +
cost of $(A_i \ldots A_k) (A_{k+1} \ldots A_j)$ }

Matrix Chain Multiplication

- define subproblems

OPT(i,j) : optimal parenth

- construct recurrence relation

$$(A_i \ldots A_k) (A_{k+1}$$

A_1 axb	A_2 bxc	A_3 cxd	A_4 dxe

- dimensions of $(A_1A_2A_3A_4)$ will be axe

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

$$OPT(i,j) = \min \{ OPT(i,k) + OPT(k+1,j) + \underline{\text{cost of } (A_i \ldots A_k)(A_{k+1} \ldots A_j)} \}$$

Matrix Chain Multiplication

- define subproblems

OPT(i,j) : optimal parenth

- construct recurrence relation

$(A_i \ . \ . \ . \ A_k \ ) \ (\ A_{k+1}$

A_1 axb	A_2 bxc	A_3 cxd	A_4 dxe

- dimensions of $(A_1A_2A_3A_4)$ will be axe
- dimension of each matrix A_i : $p_i x p_{i+1}$
- dimensions of $(A_i \ldots A_k)$: $p_i x p_{k+1}$

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

OPT(i,j) =min { OPT(i,k) + OPT(k+1,j) +
cost of $(A_i \ldots A_k)$ $(A_{k+1} \ldots A_j)$ }

Matrix Chain Multiplication

- define subproblems

OPT(i,j) : optimal parenth

- construct recurrence relation

$(A_i \ldots A_k) (A_{k+1}$

A_1 axb

A_2 bxc

A_3 cxd

A_4 dxe

- dimensions of $(A_1A_2A_3A_4)$ will be axe
- dimension of each matrix A_i : $p_i x p_{i+1}$
- dimensions of $(A_i \ldots A_k)$: $p_i x p_{k+1}$
- cost of $(A_i \ldots A_k)$ $(A_{k+1} \ldots A_j)$: $p_i x p_{k+1} x p_{j+1}$

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

OPT(i,j) =min { OPT(i,k) + OPT(k+1,j) + cost of $(A_i \ldots A_k)$ $(A_{k+1} \ldots A_j)$ }

Matrix Chain Multiplication

- define subproblems

  OPT(i,j) : optimal parenthesization of $A_i,\ldots,A_{j-1}$

- construct recurrence relation

$$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$$

- focus on last move (last parenthesization)
- find best k minimizing the cost
- recursively continue on left and right

$$OPT(i,j) = \min \{ OPT(i,k) + OPT(k+1,j) + \text{cost of } (A_i \ldots A_k)(A_{k+1} \ldots A_j) \}$$

OPT(i,i) = 0 for all i

Matrix Chain Multiplication

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3, \ldots$, $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
   for i=1 to n-l+1
       j = i + l – 1
       M[i,j] = ∞
       for k=i to j-1
           q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
           if q < M[i,j]
               M[i,j] = q
return M[1,n]
```

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \dots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \dots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
   for i=1 to n-l+1
       j = i + l – 1
       M[i,j] = ∞
       for k=i to j-1
           q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
           if q < M[i,j]
               M[i,j] = q
return M[1,n]
```

	1	2	3	4	5
1					
2					
3					
4					
5					

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5
4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3, \ldots$, $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l – 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
   for i=1 to n-l+1
      j = i + l – 1
      M[i,j] = ∞
      for k=i to j-1
         q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
         if q < M[i,j]
            M[i,j] = q
return M[1,n]
```

l = 2

j = 2

i = 1

	1	2	3	4	5
1	0	[1]60			
2		0			
3			0		
4				0	
5					0

<u>k = 1</u>

$OPT(1,2) = OPT(1,1) + OPT(2,2) + p_1\, p_2\, p_3$

$OPT(1,2) = 60$

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 2

j = 3

i = 2

	1	2	3	4	5
1	0	[1] 60			
2		0	[2] 90		
3			0		
4				0	
5					0

k = 2

$OPT(2,3) = OPT(2,2) + OPT(3,3) + p_2\, p_3\, p_4$

$OPT(2,3) = 90$

Matrix Chain Multiplication

A_1	A_2	A_3	A_4	A_5
4x5	5x3	3x6	6x2	2x3

input : $A_1, A_2, \dots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \dots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 2

j = 4

i = 3

	1	2	3	4	5
1	0	^{1}60			
2		0	^{2}90		
3			0	^{3}36	
4				0	
5					0

<u>k = 3</u>

$OPT(3,4) = OPT(3,3) + OPT(4,4) + p_3\, p_4\, p_5$

$OPT(3,4) = 36$

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l – 1
        M[i,j] = ∞
        for k=i to j-1
            $q = M[i,k] + M[k+1,j] + p_{i-1}\, p_k\, p_j$
            if q < M[i,j]
                M[i,j] = q
return M[1,n]

l = 2

j = 5

i = 4

	1	2	3	4	5
1	0	[1] 60			
2		0	[2] 90		
3			0	[3] 36	
4				0	[4] 36
5					0

k = 4

$OPT(4,5) = OPT(4,4) + OPT(5,5) + p_4\, p_5\, p_6$
$OPT(4,5) = 36$

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 3

j = 3

i = 1

	1	2	3	4	5
1	0	[1] 60			
2		0	[2] 90		
3			0	[3] 36	
4				0	[4] 36
5					0

<u>k = 1</u>

$OPT(1,3) = OPT(1,1) + OPT(2,3) + p_1\, p_2\, p_4$

$OPT(1,3) = 90 + 120 = 210$

Matrix Chain Multiplication

A_1	,	A_2	,	A_3	,	A_4	,	A_5
4x5	,	5x3	,	3x6	,	6x2	,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l – 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 3

j = 3

i = 1

	1	2	3	4	5
1	0	[1] 60			
2		0	[2] 90		
3			0	[3] 36	
4				0	[4] 36
5					0

k = 1

$OPT(1,3) = OPT(1,1) + OPT(2,3) + p_1\, p_2\, p_4$
$OPT(1,3) = 90 + 120 = 210$

k = 2

$OPT(1,3) = OPT(1,2) + OPT(3,3) + p_1\, p_3\, p_4$
$OPT(1,3) = 60 + 72 = 132$

Matrix Chain Multiplication

A_1	A_2	A_3	A_4	A_5
4x5	5x3	3x6	6x2	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
   for i=1 to n-l+1
       j = i + l - 1
       M[i,j] = ∞
       for k=i to j-1
           q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
           if q < M[i,j]
               M[i,j] = q
return M[1,n]
```

l = 3

j = 3

i = 1

	1	2	3	4	5
1	0	[1] 60	[2] 132		
2		0	[2] 90		
3			0	[3] 36	
4				0	[4] 36
5					0

k = 1

$OPT(1,3) = OPT(1,1) + OPT(2,3) + p_1\, p_2\, p_4$

$OPT(1,3) = 90 + 120 = 210$

k = 2

$OPT(1,3) = OPT(1,2) + OPT(3,3) + p_1\, p_3\, p_4$

$OPT(1,3) = 60 + 72 = 132$

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5
4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3$, …, $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l – 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	^{1}60	^{2}132	^{1}106	^{4}130
2		0	^{2}90	^{2}66	^{4}96
3			0	^{3}36	^{4}54
4				0	^{4}36
5					0

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5
4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \dots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3$, ..., $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	[1] 60	[2] 132	[1] 106	[4] 130
2		0	[2] 90	[2] 66	[4] 96
3			0	[3] 36	[4] 54
4				0	[4] 36
5					0

$A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5$

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5
4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3, \ldots,$ $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	^{1}60	^{2}132	^{1}106	^{4}130
2		0	^{2}90	^{2}66	^{4}96
3			0	^{3}36	^{4}54
4				0	^{4}36
5					0

A_1 . A_2 . A_3 . A_4 . A_5

k = 4 ➔ OPT(1,4) and OPT(5,5)

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5
4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2$, $p_2 \times p_3, \ldots$, $p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
  for i=1 to n-l+1
     j = i + l - 1
     M[i,j] = ∞
     for k=i to j-1
        q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
        if q < M[i,j]
           M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	^{1}60	^{2}132	^{1}106	^{4}130
2		0	^{2}90	^{2}66	^{4}96
3			0	^{3}36	^{4}54
4				0	^{4}36
5					0

$(A_1 \ . \ A_2 \ . \ A_3 \ . \ A_4). \ A_5$

k = 4 ➔ OPT(1,4) and OPT(5,5)

Matrix Chain Multiplication

A_1	A_2	A_3	A_4	A_5
4x5	5x3	3x6	6x2	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	^{1}60	^{2}132	^{1}106	^{4}130
2		0	^{2}90	^{2}66	^{4}96
3			0	^{3}36	^{4}54
4				0	^{4}36
5					0

$(A_1 \ . \ A_2 \ . \ A_3 \ . \ A_4).\ A_5$

k = 4 ➔ OPT(1,4) and OPT(5,5)

k = 1 ➔ OPT(1,1) and OPT(2,4)

Matrix Chain Multiplication

A_1 ,	A_2 ,	A_3 ,	A_4 ,	A_5
4x5 ,	5x3 ,	3x6 ,	6x2 ,	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l - 1
        M[i,j] = ∞
        for k=i to j-1
            q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
            if q < M[i,j]
                M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	^{1}60	^{2}132	^{1}106	^{4}130
2		0	^{2}90	^{2}66	^{4}96
3			0	^{3}36	^{4}54
4				0	^{4}36
5					0

$(A_1 \;.\; (\; A_2 \;.\; A_3 \;.\; A_4 \;)).\; A_5$

k = 4 ➔ OPT(1,4) and OPT(5,5)
k = 1 ➔ OPT(1,1) and OPT(2,4)

Matrix Chain Multiplication

A_1 , A_2 , A_3 , A_4 , A_5

4x5 , 5x3 , 3x6 , 6x2 , 2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

```
initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
   for i=1 to n-l+1
      j = i + l – 1
      M[i,j] = ∞
      for k=i to j-1
         q = M[i,k] + M[k+1,j] + p_{i-1} p_k p_j
         if q < M[i,j]
            M[i,j] = q
return M[1,n]
```

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	$^{1}60$	$^{2}132$	$^{1}106$	$^{4}130$
2		0	$^{2}90$	$^{2}66$	$^{4}96$
3			0	$^{3}36$	$^{4}54$
4				0	$^{4}36$
5					0

$(A_1 \;.\; (\; A_2 \;.\; A_3 \;.\; A_4\;)).\; A_5$

k = 4 ➔ OPT(1,4) and OPT(5,5)

k = 1 ➔ OPT(1,1) and OPT(2,4)

k = 2 ➔ OPT(2,2) and OPT(3,4)

Matrix Chain Multiplication

A_1	A_2	A_3	A_4	A_5
4x5	5x3	3x6	6x2	2x3

input : $A_1, A_2, \ldots, A_n$ with
$p_1 \times p_2,\ p_2 \times p_3, \ldots,\ p_n \times p_{n+1}$

initialize a memory M
for i=1 to n
    M[i,i] = 0
for l=2 to n
    for i=1 to n-l+1
        j = i + l – 1
        M[i,j] = ∞
        for k=i to j-1
            $q = M[i,k] + M[k+1,j] + p_{i-1}\, p_k\, p_j$
            if q < M[i,j]
                M[i,j] = q
return M[1,n]

l = 5

j = 5

i = 1

	1	2	3	4	5
1	0	$^{1}60$	$^{2}132$	$^{1}106$	$^{4}130$
2		0	$^{2}90$	$^{2}66$	$^{4}96$
3			0	$^{3}36$	$^{4}54$
4				0	$^{4}36$
5					0

$(A_1 . (A_2 . (A_3 . A_4))). A_5$

k = 4 ➔ OPT(1,4) and OPT(5,5)
k = 1 ➔ OPT(1,1) and OPT(2,4)
k = 2 ➔ OPT(2,2) and OPT(3,4)

Subset Sum

- given a set of M positive integers $A=\{a_1,a_2,\ldots,a_M\}$, a predefined number N, find out whether it is possible to find a subset of A such that the sum of the elements in this subset is N

Subset Sum

- given a set of M positive integers $A=\{a_1,a_2,\ldots,a_M\}$, a predefined number N, find out whether it is possible to find a subset of A such that the sum of the elements in this subset is N

$A = \{ 1, 4, 12, 20, 9 \}$ and $N = 14$

Subset Sum

- given a set of M positive integers $A=\{a_1,a_2,\dots,a_M\}$, a predefined number N, find out whether it is possible to find a subset of A such that the sum of the elements in this subset is N

$A = \{ 1, 4, 12, 20, 9 \}$ and $N = 14$

for the subset $\{ 1, 4, 9 \}$, the sum is $1 + 4 + 9 = 14$

Subset Sum

- given a set of M positive integers $A=\{a_1,a_2,\ldots,a_M\}$, a predefined number N, find out whether it is possible to find a subset of A such that the sum of the elements in this subset is N

A = { 1, 4, 12, 20, 9 } and N = 14

for the subset { 1, 4, 9 }, the sum is 1 + 4 + 9 = 14

your program should output true for this input.

Subset Sum

- define subproblems

Subset Sum

- define subproblems

  OPT(i,j) : it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

Subset Sum

- define subproblems

OPT(i,j) : it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

Subset Sum

- define subproblems

OPT(i,j) : it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

(1)

(2)

Subset Sum

- define subproblems

$OPT(i,j)$: it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

(1) if the subset contains a_i, then continue with

$$OPT(i - 1, j - a_i)$$

(2)

Subset Sum

- define subproblems

  $OPT(i,j)$: it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

(1) if the subset contains a_i, then continue with

$$OPT(i - 1, j - a_i)$$

(2) if it doesn't, then continue with

$$OPT(i - 1, j)$$

Subset Sum

- define subproblems

OPT(i,j) : it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

(1) if the subset contains a_i, then continue with

OPT(i - 1, j – a_i)

(2) if it doesn't, then continue with

OPT(i – 1, j)

combine them with 'OR'

Subset Sum

- define subproblems

  $OPT(i,j)$: it is possible to find a subset of $\{a_1,\ldots,a_i\}$ such that the sum is j

- construct recurrence relation

Two cases

(1) if the subset contains a_i, then continue with

$OPT(i - 1, j - a_i)$

(2) if it doesn't, then continue with

$OPT(i - 1, j)$

combine them with 'OR'

Base cases

$OPT(m,0)$ = TRUE for all m

$OPT(0,N)$ = FALSE for all N

Partition into Lines

- given a sequence of N words w_1, w_2, …, w_N where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

(you also have to put one space between each pair of words to separate them)

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

Partition into Lines

- given a sequence of N words w_1, w_2, …, w_N where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer
music
discrete
linear

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer	18 – 8
music	18 – 5
discrete	18 – 8
linear	18 – 6

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer	18 – 8
music	18 – 5
discrete	18 – 8
linear	18 – 6

100 + 169 + 100 + 144 = 513

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer	18 – 8	computer music
music	18 – 5	discrete linear
discrete	18 – 8	
linear	18 – 6	

100 + 169 + 100 + 144 = 513

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer	18 – 8
music	18 – 5
discrete	18 – 8
linear	18 – 6

computer music	18 – 14
discrete linear	18 – 15

100 + 169 + 100 + 144 = 513

Partition into Lines

- given a sequence of N words $w_1, w_2, \ldots, w_N$ where each w_i contains c_i characters.
- you insert line breaks that partition these words into lines such that the total number of characters in each line is at most L

  (you also have to put one space between each pair of words to separate them)
- the slack of a line containing c characters is defined to be L – c
- Your task is to find a partition such that the sum of the squares of the slacks of all lines is minimized.

{computer, music, discrete, linear}, {8, 5, 8, 6}, L = 18

computer	18 – 8
music	18 – 5
discrete	18 – 8
linear	18 – 6

100 + 169 + 100 + 144 = 513

computer music	18 – 14
discrete linear	18 – 15

16 + 9 = 25

Partition into Lines

- define subproblems

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

Partition into Lines

- define subproblems

OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

$w_1\ w_2\ .\ .\ .\ w_{i-1}\ w_i\ .\ .\ .\ w_j$

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

$w_1\, w_2\, .\,.\,.\; w_{i-1}\; w_i\, .\,.\,.\; w_j \longrightarrow$

$w_1\, w_2\, .\,.\,.\; w_{i-1}$

$w_i\, .\,.\,.\; w_j$

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

$w_1\, w_2\, \ldots\, w_{i-1}\, w_i\, \ldots\, w_j \longrightarrow$ $w_1\, w_2\, \ldots\, w_{i-1}$
$w_i\, \ldots\, w_j$

$$OPT(j) = \min\{OPT(i-1) + S[i,j]^2\} \text{ where } S[i,j] \geq 0$$

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

$w_1\, w_2\, .\, .\, .\, w_{i-1}\, w_i\, .\, .\, .\, w_j \rightarrow$ $w_1\, w_2\, .\, .\, .\, w_{i-1}$

$w_i\, .\, .\, .\, w_j$

$$OPT(j) = \min \{OPT(i-1) + S[i,j]^2\} \text{ where } S[i,j] \geq 0$$

$$S[i,j] = L - (j - i) - \Sigma\, c_i$$

# of spaces

Partition into Lines

- define subproblems

  OPT(j) : the cost of the optimal partition for the first j words

- construct recurrence relation

$w_1\, w_2 \,.\,.\,.\, w_{i-1}\, w_i \,.\,.\,.\, w_j \rightarrow$ $w_1\, w_2 \,.\,.\,.\, w_{i-1}$
$w_i \,.\,.\,.\, w_j$

$$OPT(j) = \min\{OPT(i-1) + S[i,j]^2\} \text{ where } S[i,j] \geq 0$$

$$S[i,j] = L - (j - i) - \Sigma\, c_i$$

# of spaces

<u>Base case</u>

$OPT(0) = 0$

$S[i,i] = L - c_i$ for all i

$S[i,j] = S[i,j-1] - (c_i + 1)$

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

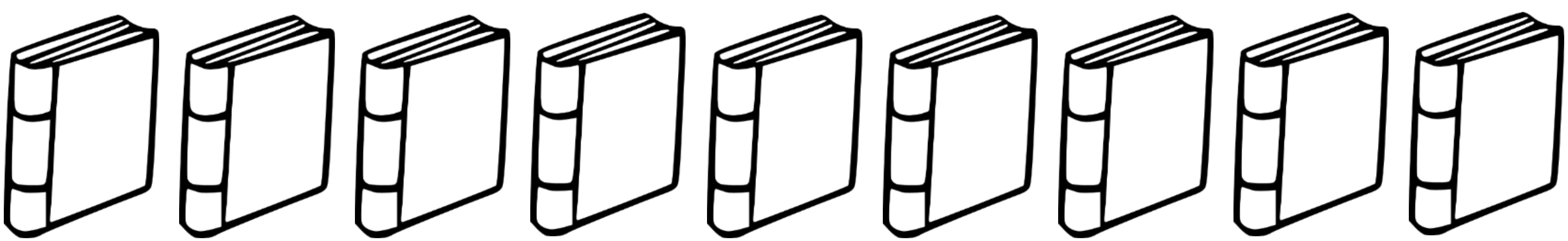
What will be the fairest way to divide the book among k workers?

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

What will be the fairest way to divide the book among k workers?

(the total number of pages each worker gets will be close to each other)



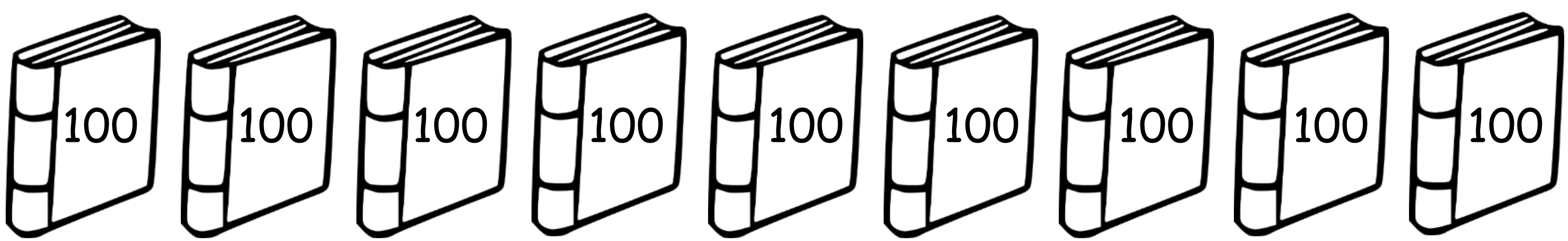
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

What will be the fairest way to divide the book among k workers?

(the total number of pages each worker gets will be close to each other)



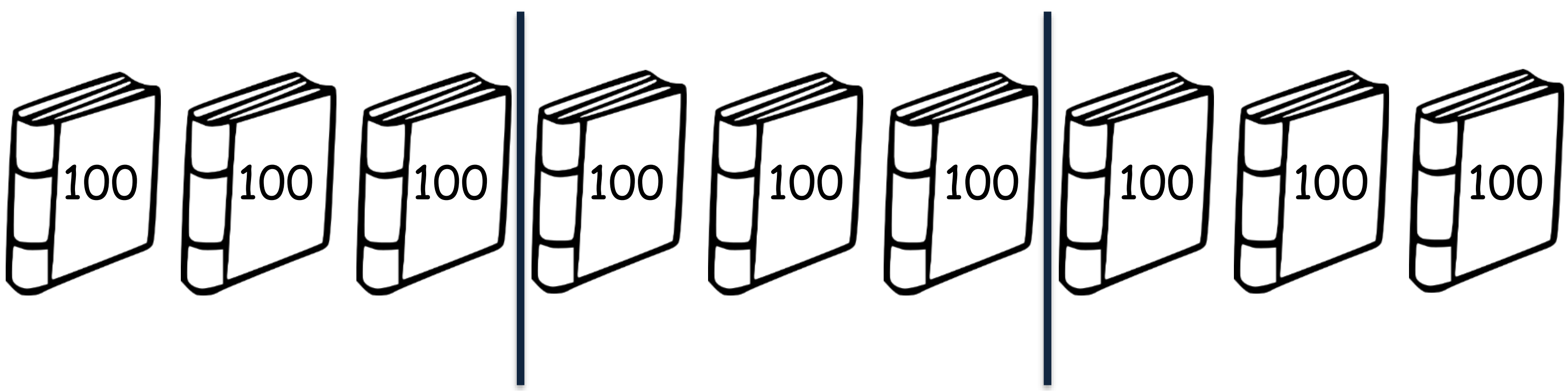
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

<u>What will be the fairest way to divide the book among k workers?</u>

(the total number of pages each worker gets will be close to each other)



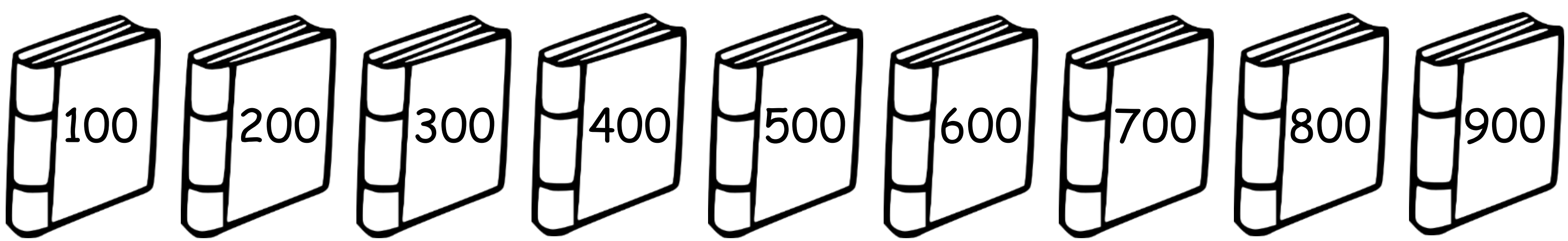
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

What will be the fairest way to divide the book among k workers?

(the total number of pages each worker gets will be close to each other)



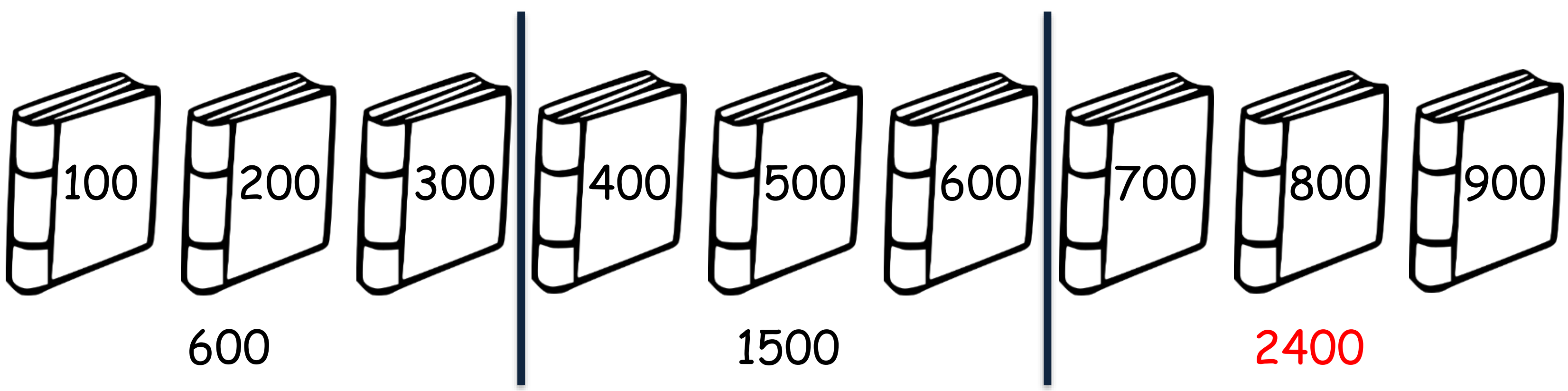
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

<u>What will be the fairest way to divide the book among k workers?</u>

(the total number of pages each worker gets will be close to each other)



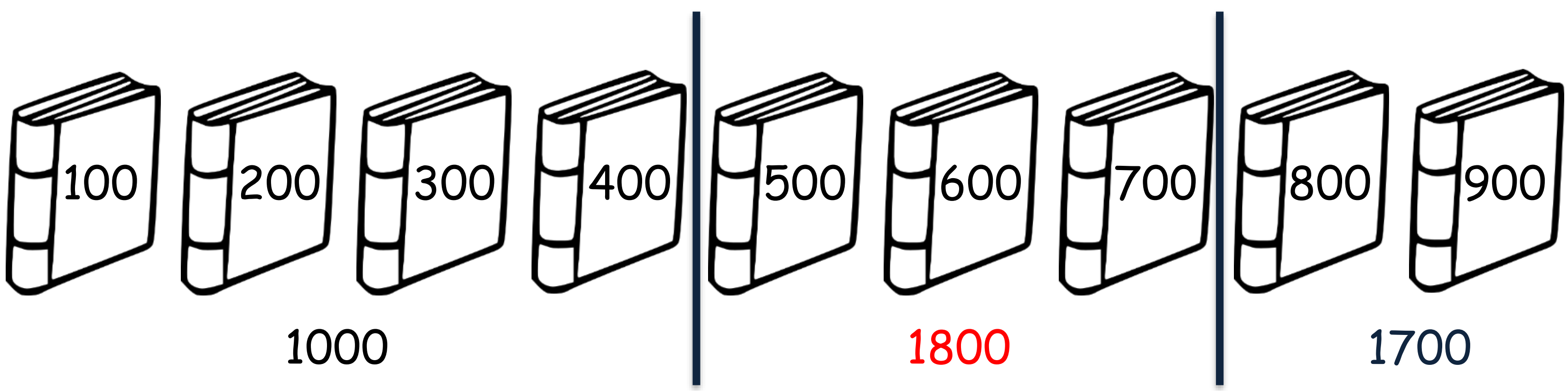
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

<u>What will be the fairest way to divide the book among k workers?</u>

(the total number of pages each worker gets will be close to each other)



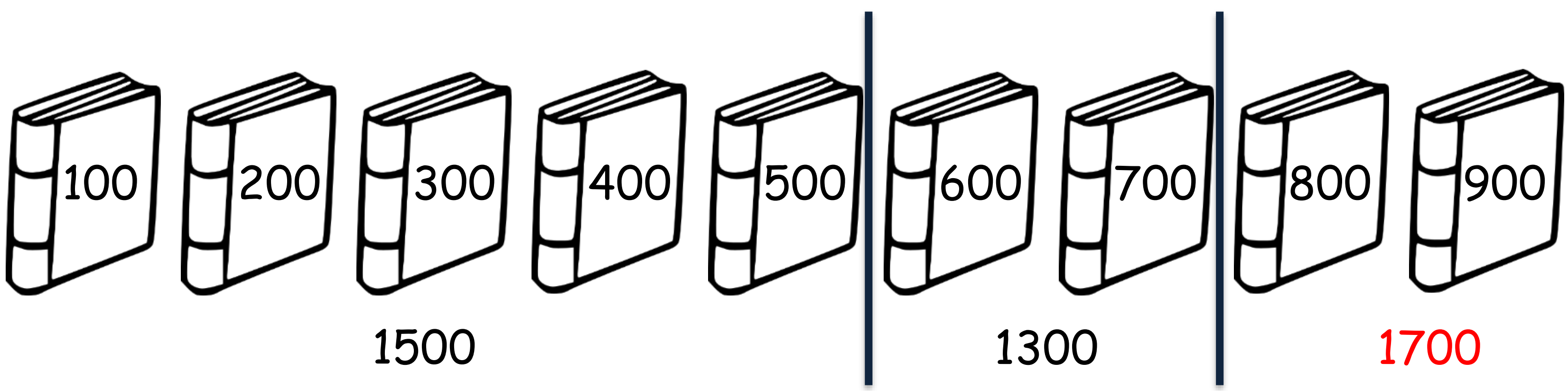
9 books and 3 workers

Dividing the Books

- Suppose you given a shelf of books, and your job is to divide them among k workers so that they scan the books to find some codes. You can divide the shelf into k regions and assign each region to a worker. (The books ordered according to the number of pages)

<u>What will be the fairest way to divide the book among k workers?</u>

(the total number of pages each worker gets will be close to each other)



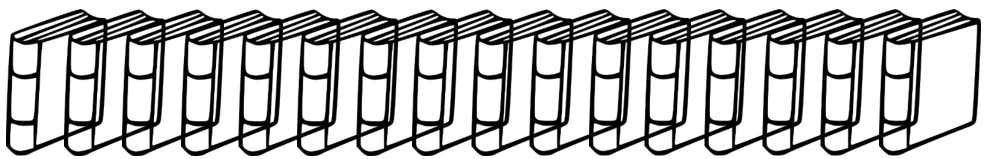
9 books and 3 workers

Dividing the Books

n books ($p_1, p_2, \ldots, p_n$) and k workers

- construct recurrence relation

n books

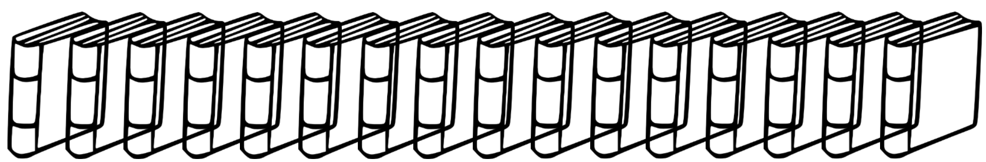


$$M[n,k] =$$

Dividing the Books

n books ($p_1, p_2, \ldots, p_n$) and k workers

- construct recurrence relation



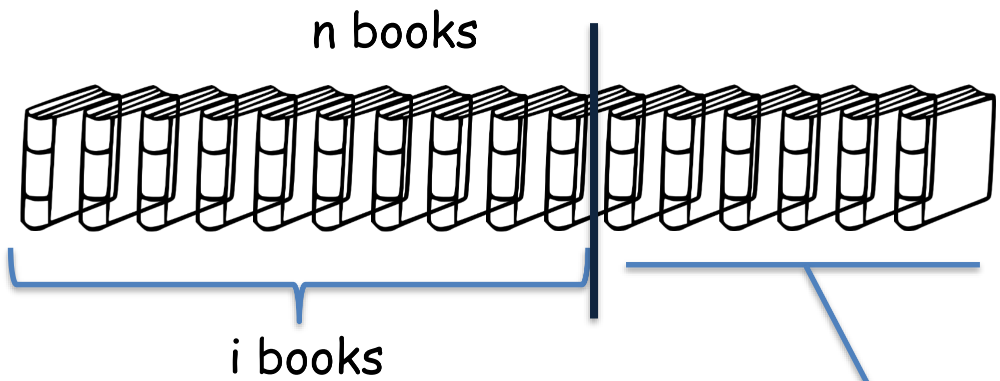
$$M[n, k] =$$

optimum number of pages
of the largest share

Dividing the Books

n books ($p_1, p_2, \ldots, p_n$) and k workers

- construct recurrence relation



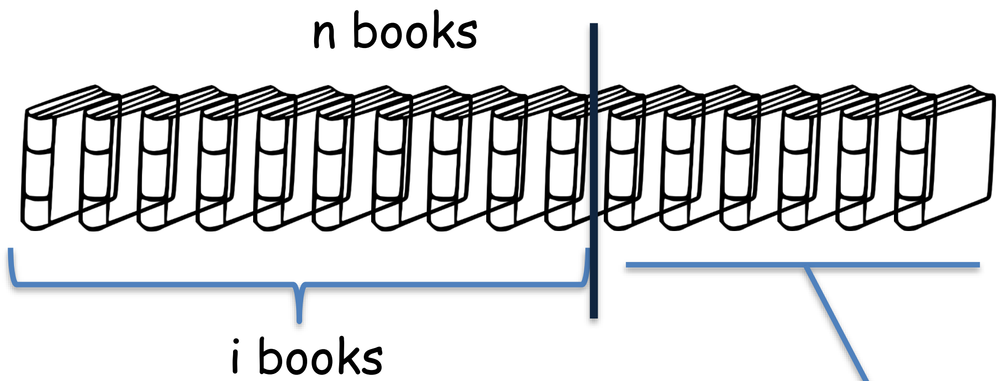
$$M[n,k] = \max(\qquad , \sum_{j=i+1}^{n} p_j)$$

optimum number of pages
of the largest share

Dividing the Books

n books ($p_1, p_2, \ldots, p_n$) and k workers

- construct recurrence relation



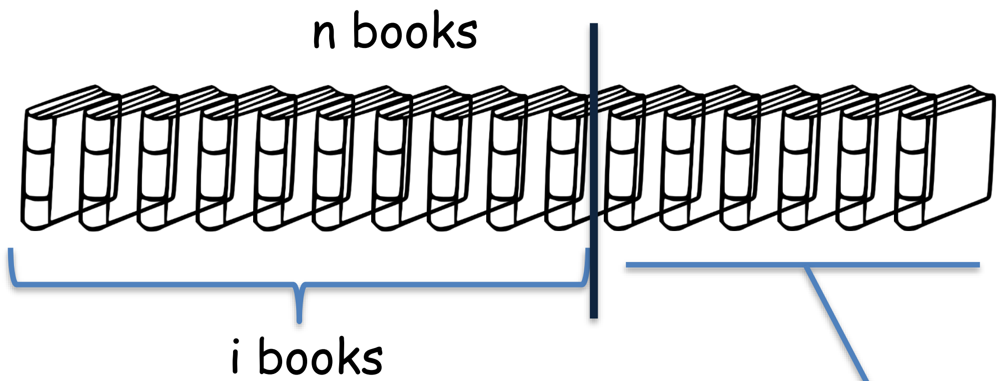
$$M[n,k] = \quad \max\left(M[i,k-1], \sum_{j=i+1}^{n} p_j\right)$$

optimum number of pages
of the largest share

Dividing the Books

n books $(p_1, p_2, \ldots, p_n)$ ***and k workers***

- construct recurrence relation



Base cases

$$M[1, k] = p_1$$

$$M[n, 1] = \sum p_i$$

$$M[n, k] = min_{i=1}^{n} \max(M[i, k-1], \sum_{j=i+1}^{n} p_j)$$

optimum number of pages
of the largest share

Dynamic Programming

- analyze structure of the optimal solution and define subproblems that need to be solved in order to get the optimal solution
- establish the relationship between the optimal solution and those subproblems (construct the recurrence relation)
- compute the optimal values of subproblems, save them in a table (memoization), then compute the optimal values of larger subproblems, and eventually compute the optimal value of the original problem