Lecture 23

Introduction to Dynamic Programming

Source: Introduction to Algorithms, CLRS

Introduction to Dynamic Programming

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems,

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems, when subproblems share subsubproblem.

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems, when subproblems share subsubproblem.

Idea:

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems, when subproblems share subsubproblem.

Idea: Instead of solving a subproblem repeatedly,

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems, when subproblems share subsubproblem.

Idea: Instead of solving a subproblem repeatedly, solve it once and store the result in an array.

Introduction to Dynamic Programming

Dynamic programming solves problem by combining the solutions to subproblems, when subproblems share subsubproblem.

Idea: Instead of solving a subproblem **repeatedly**, solve it **once** and **store** the result in an array.

Let's learn DP through an example!

Rod Cutting

Rod Cutting

Rod-Cutting:

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1 : n]$, where $p[i] \geq 0$ is the price of a rod of

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example:

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1 : n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4$,

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:

Profit earned:

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod: (4)

Profit earned: 9

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:	(4)	(1,3)
Profit earned:	9	9

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:	(4)	(1,3)	(2,2)
Profit earned:	9	9	10

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:	(4)	(1,3)	(2,2)	(1,1,2)
Profit earned:	9	9	10	7

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4$, $p =$

1	5	8	9

Possible cuttings for a length 4 rod:	(4)	(1,3)	(2,2)	(1,1,2)	(1,1,1,1)
Profit earned:	9	9	10	7	4

Rod Cutting

Rod-Cutting:

Input: A rod of length n inches and an array $p[1:n]$, where $p[i] \geq 0$ is the price of a rod of i inches long rod.

Output: Maximum profit obtainable by cutting (or not) the rod and selling the pieces.

Example: $n = 4,\ p =$

1	5	8	9

Possible cuttings for a length 4 rod:	(4)	(1,3)	(2,2)	(1,1,2)	(1,1,1,1)
Profit earned:	9	9	10	7	4

Maximum profit

Brute Force Solution

Brute Force Solution

Brute force approach:

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.
- Output the maximum profit after calculating profits for all the cuttings.

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.
- Output the maximum profit after calculating profits for all the cuttings.

Time Complexity:

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.
- Output the maximum profit after calculating profits for all the cuttings.

Time Complexity: You can either cut or not cut at every ith inch.

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.
- Output the maximum profit after calculating profits for all the cuttings.

Time Complexity: You can either cut or not cut at every ith inch. Generating all cuttings this

Brute Force Solution

Brute force approach:

- Find all possible cuttings of the length n rod.
- Output the maximum profit after calculating profits for all the cuttings.

Time Complexity: You can either cut or not cut at every ith inch. Generating all cuttings this way can lead to $O(2^n)$ time.

Towards a Better Algorithm

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n$ =

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = p[i] +$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = p[i] + profit_{n-i}$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = \underbrace{p[i]} + profit_{n-i}$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = \underbrace{p[i]} + profit_{n-i}$

Price of the first cut

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = \underbrace{p[i]} + \underbrace{profit_{n-i}}$

Price of the first cut

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is i, then $profit_n = \underbrace{p[i]} + \underbrace{profit_{n-i}}$

Price of the first cut

For the remaining $(n - i)$ length rod,
we cannot get more than $profit_{n-i}$.

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

If length of the first cut in optimal cutting is 3, then $profit_n = p[3] + profit_{n-3}$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

If length of the first cut in optimal cutting is 3, then $profit_n = p[3] + profit_{n-3}$

$\vdots$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

If length of the first cut in optimal cutting is 3, then $profit_n = p[3] + profit_{n-3}$

$\vdots$

If length of the first cut in optimal cutting is n (i.e. no cut is made), then $profit_n = p[n]$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

If length of the first cut in optimal cutting is 3, then $profit_n = p[3] + profit_{n-3}$

$\vdots$

If length of the first cut in optimal cutting is n (i.e. no cut is made), then $profit_n = p[n]$

Towards a Better Algorithm

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

If length of the first cut in optimal cutting is 1, then $profit_n = p[1] + profit_{n-1}$

If length of the first cut in optimal cutting is 2, then $profit_n = p[2] + profit_{n-2}$

If length of the first cut in optimal cutting is 3, then $profit_n = p[3] + profit_{n-3}$

$\vdots$

If length of the first cut in optimal cutting is n (i.e. no cut is made), then $profit_n = p[n]$

$profit_n$ is maximum of these

A Recursive Solution

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

RC(n, p)**:**

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

RC(n, p)**:**

1. **if** $n == 1$

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

RC(n, p)**:**

```
if n == 1
    return p[1]
```

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

RC(n, p)**:**

if $n == 1$

  return $p[1]$

$profit = p[n]$

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

```
RC(n, p):
    if n == 1
        return p[1]
    profit = p[n]
    for i = 1 to n - 1
```

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

```
RC(n, p):
    if n == 1
        return p[1]
    profit = p[n]
    for i = 1 to n − 1
```

i is the length of the first cut

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

```
RC(n, p):
    if n == 1
        return p[1]
    profit = p[n]
    for i = 1 to n − 1
        profit = Max(profit, p[i] + RC(n − i, p))
```

i is the length of the first cut

A Recursive Solution

Let $profit_n$ = maximum profit obtainable from an n inches rod. Then,

- $profit_n = p[1]$, if $n = 1$
- $profit_n = max(p[n],\ p[1] + profit_{n-1},\ p[2] + profit_{n-2},\ \ldots,\ p[n-1] + profit_1)$, if $n > 1$

RC(n, p):

if $n == 1$
  return $p[1]$
$profit = p[n]$
for $i = 1$ **to** $n - 1$ ← i is the length of the first cut
  $profit = \textbf{Max}(profit, p[i] + \textbf{RC}(n - i, p))$
return $profit$

All Good?

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time.

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

Recursive tree of RC:

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

Recursive tree of RC:

$\mathbf{RC}(5, p)$

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

Recursive tree of RC:

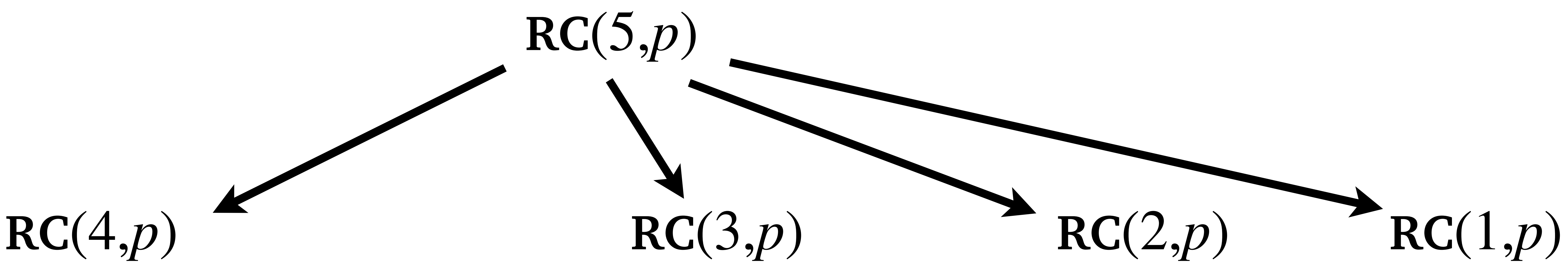
$\mathbf{RC}(5, p)$

```
RC(n, p):
    if n == 1
        return p[1]
    profit = p[n]
    for i = 1 to n − 1
        profit = Max(profit, p[i] + RC(n − i, p))
    return profit
```

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

Recursive tree of RC:

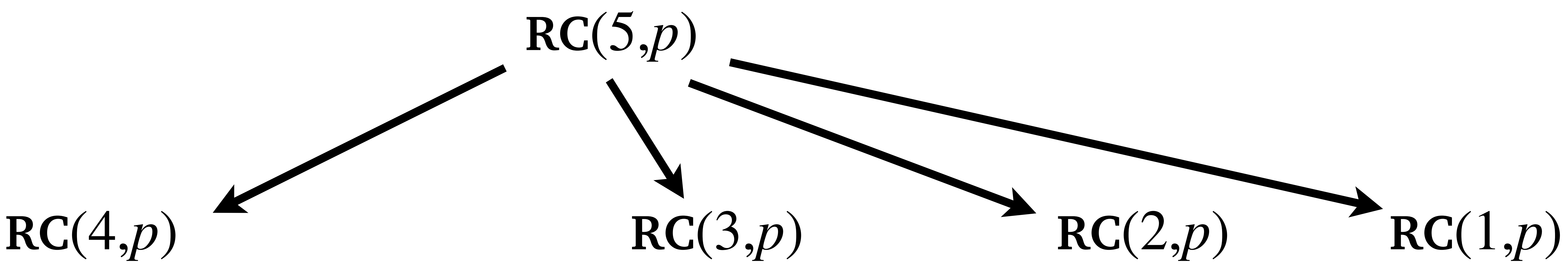



```
RC(n, p):
    if n == 1
        return p[1]
    profit = p[n]
    for i = 1 to n − 1
        profit = Max(profit, p[i] + RC(n − i, p))
    return profit
```

All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

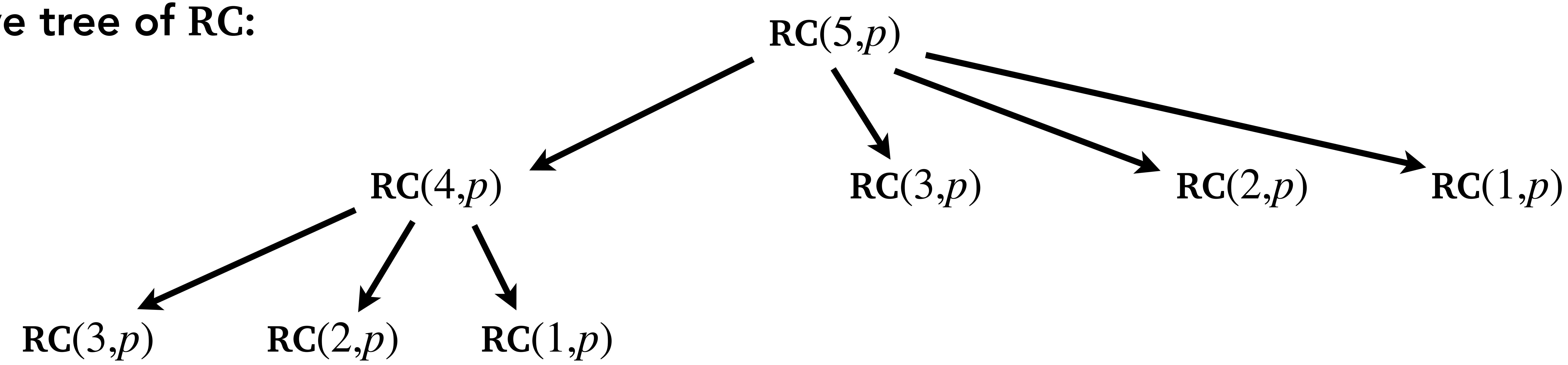
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

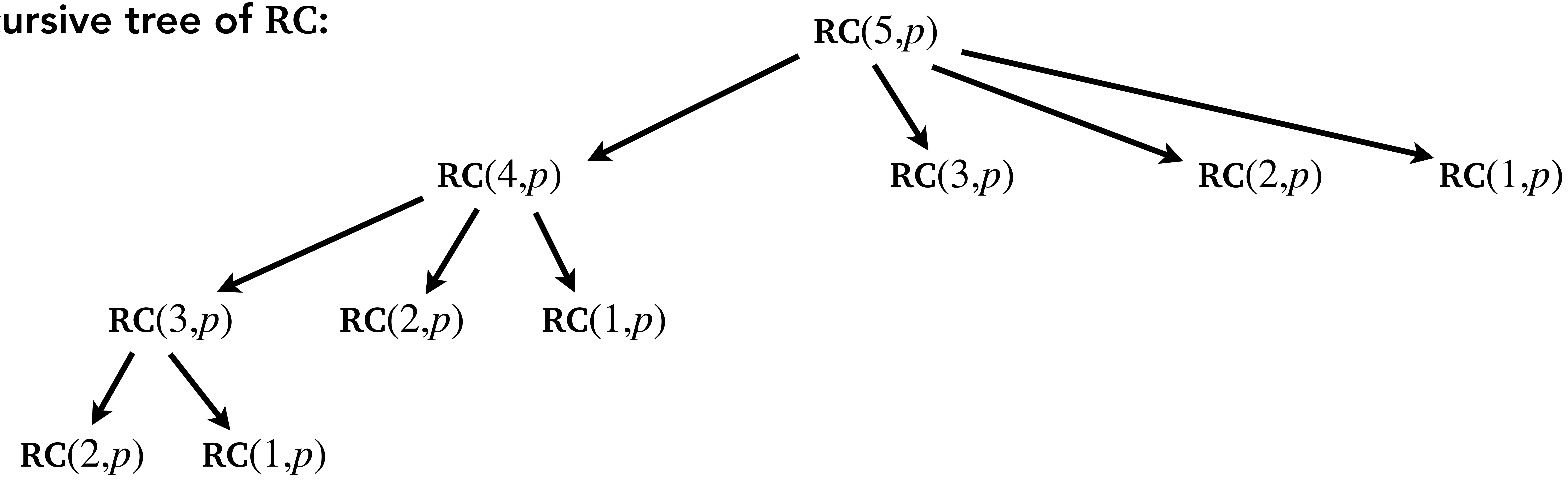
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

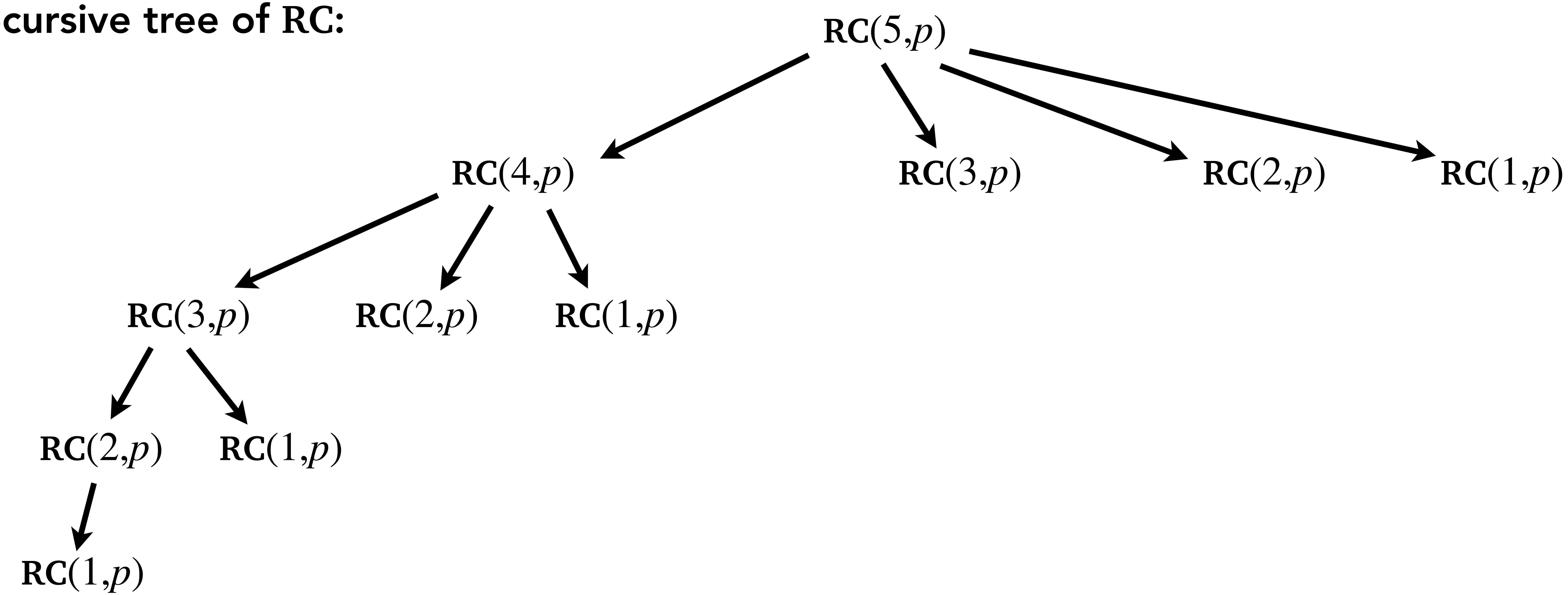
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

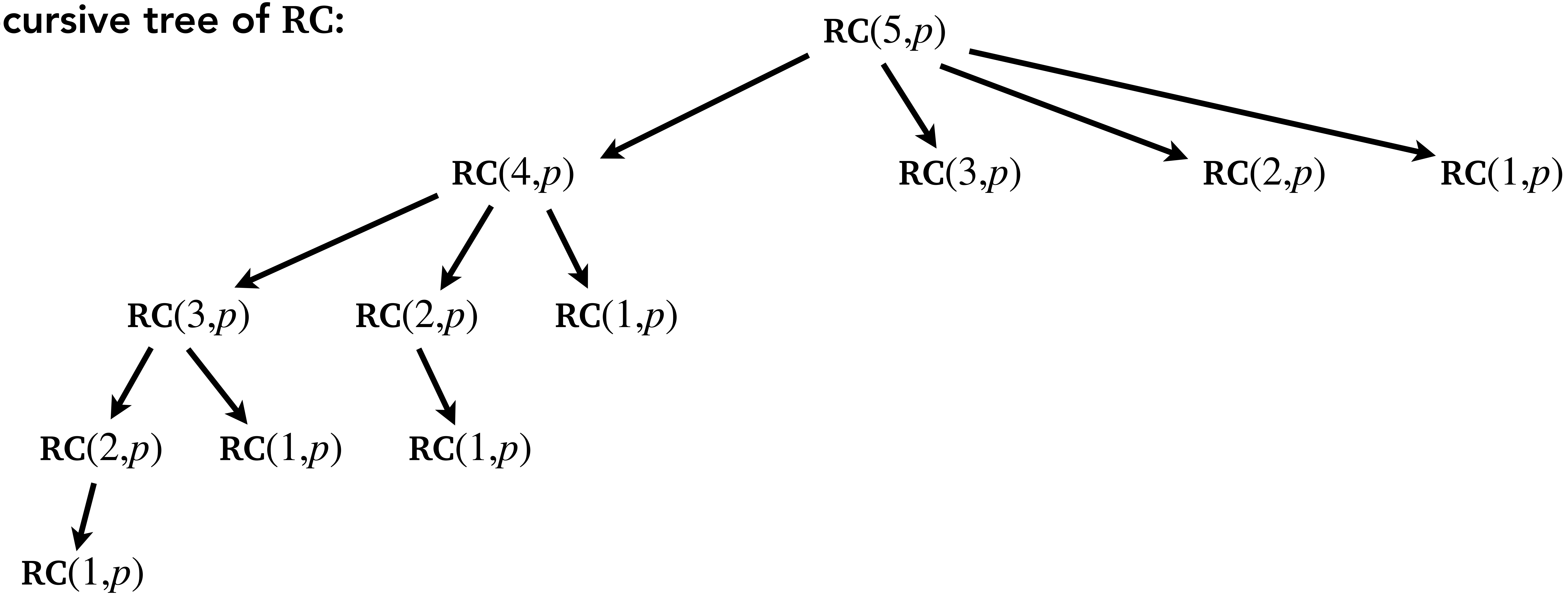
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

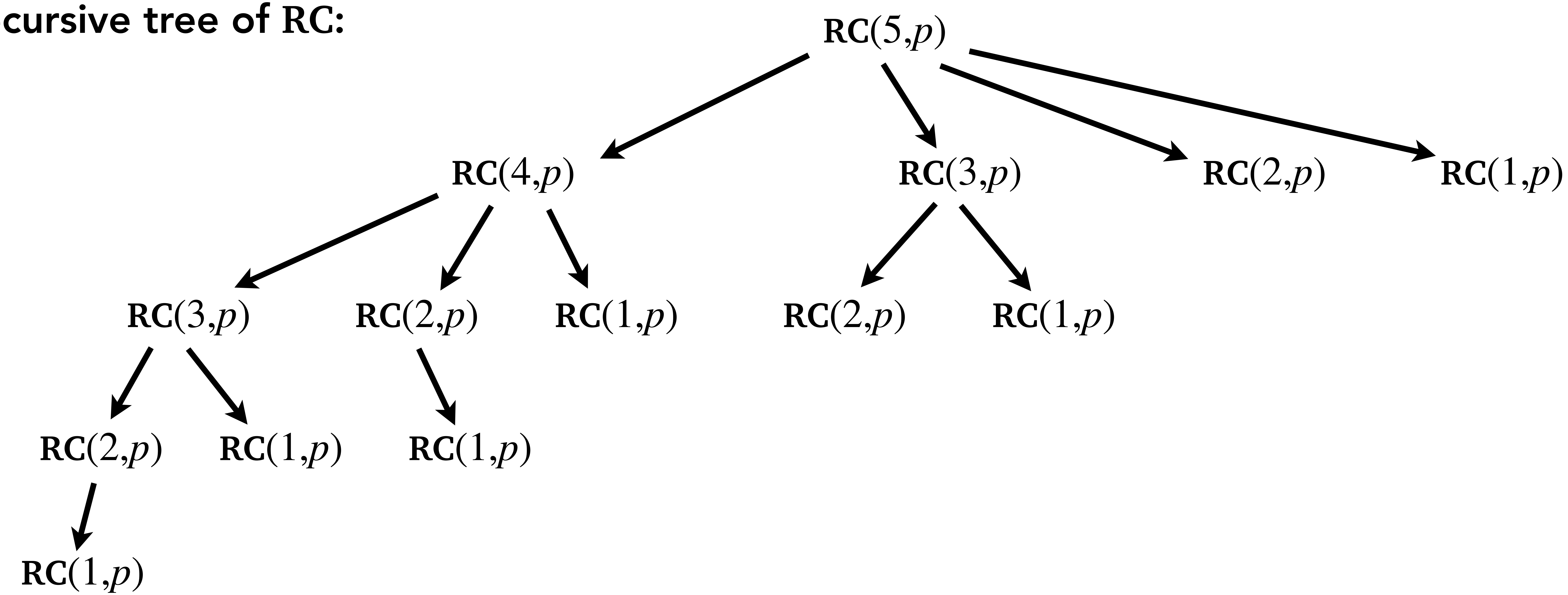
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

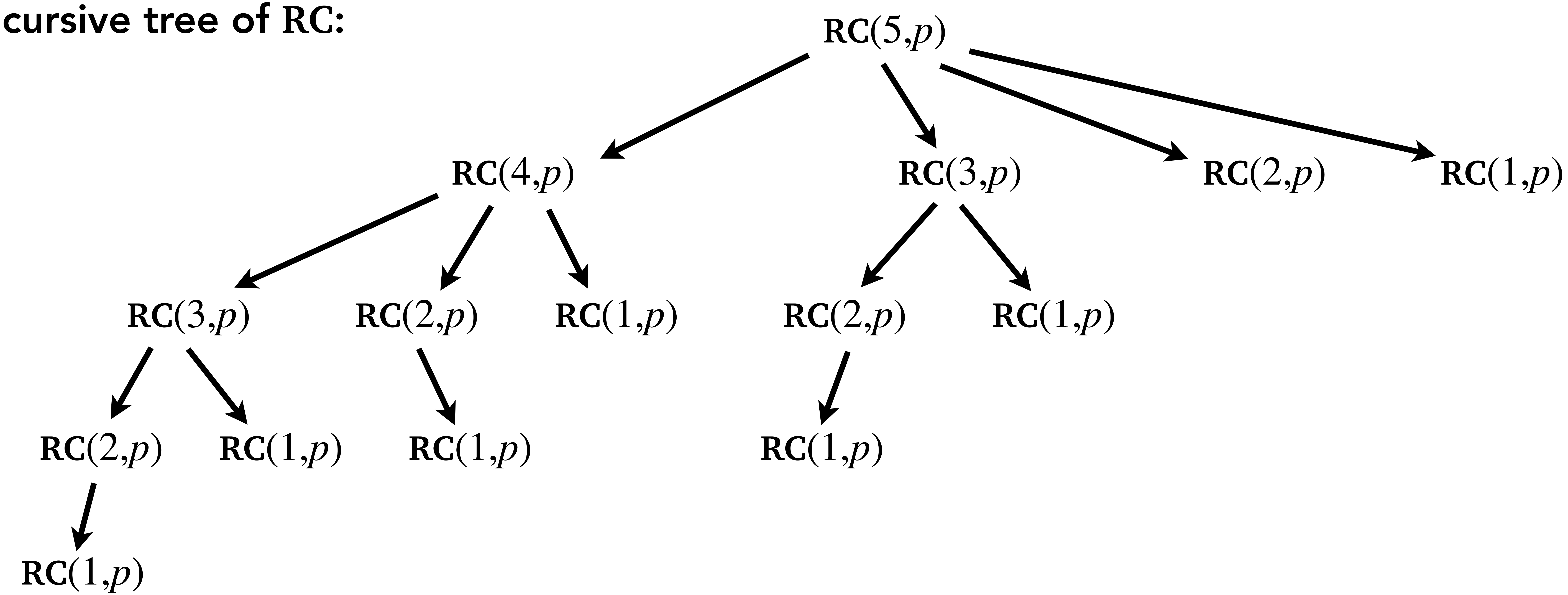
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

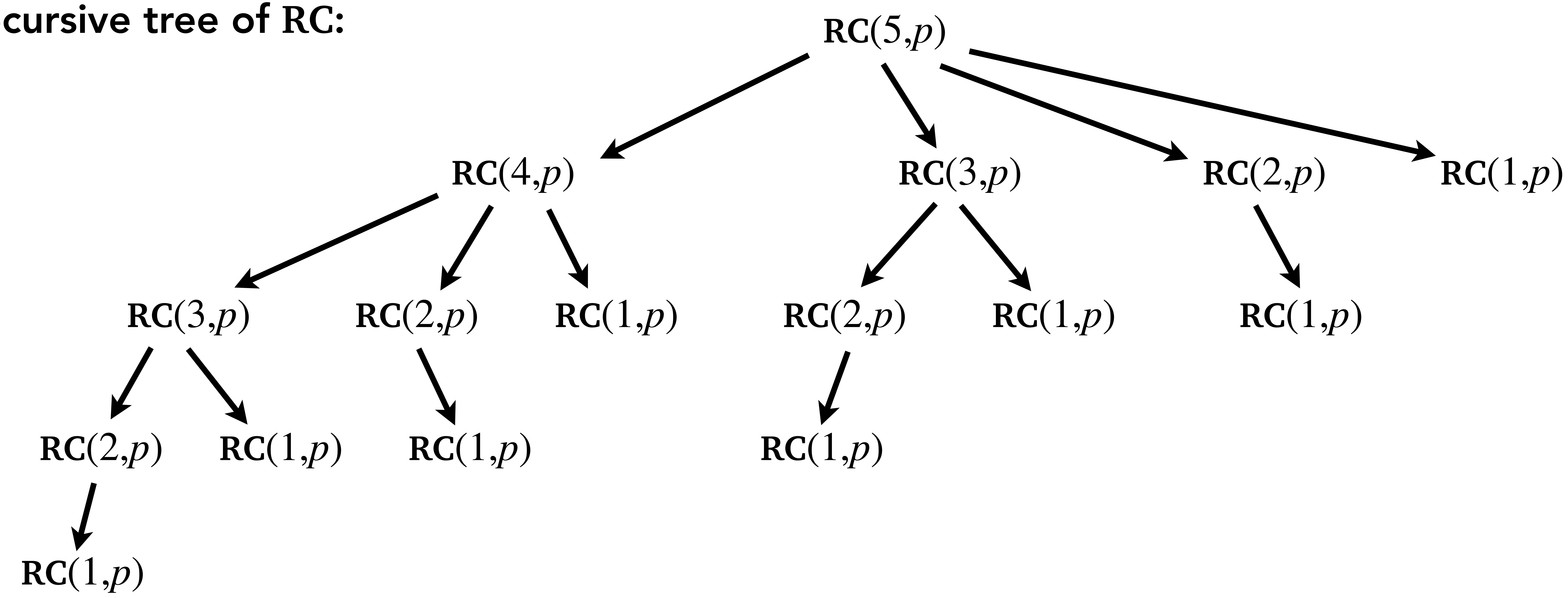
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

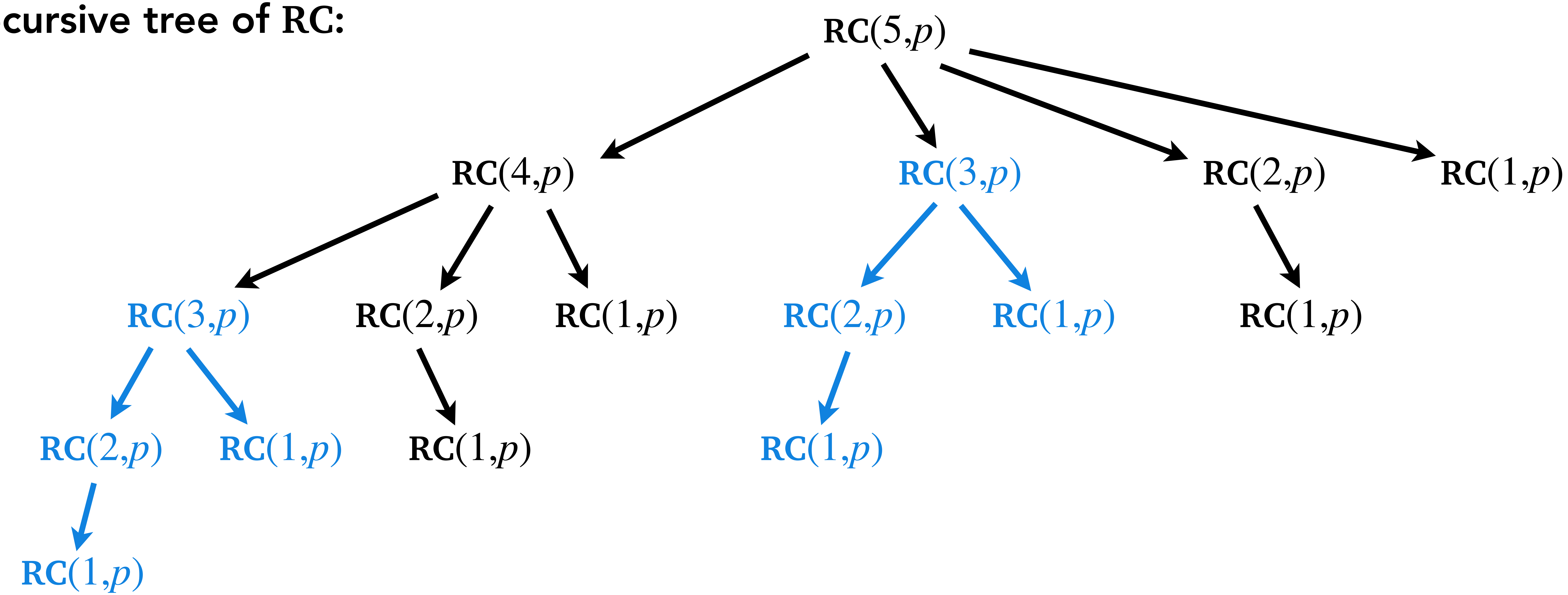
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

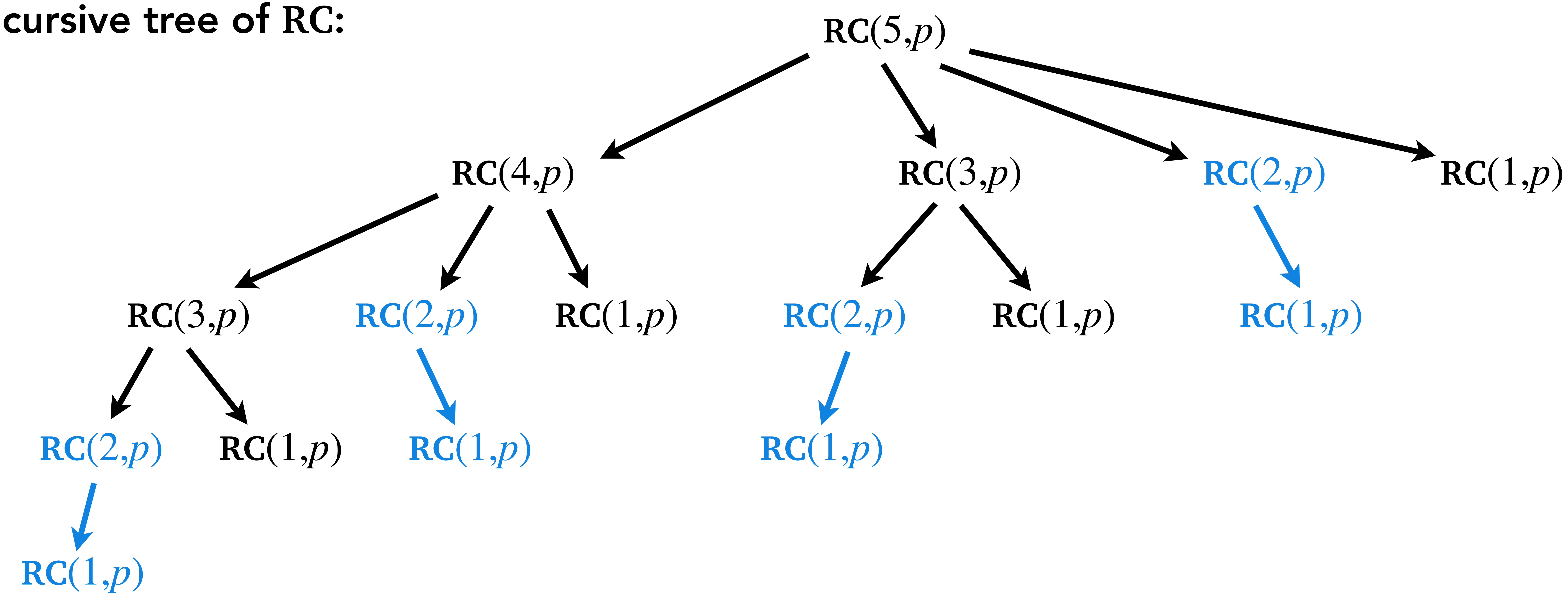
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

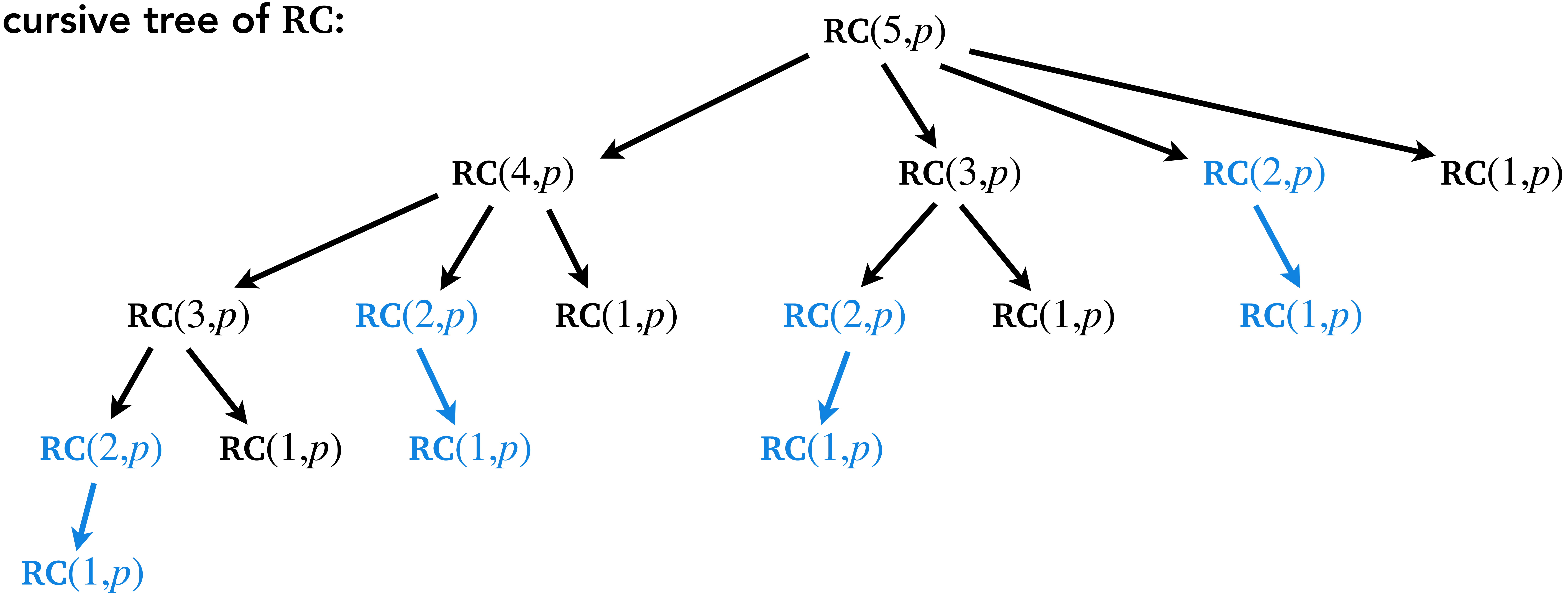
Recursive tree of RC:



All Good?

Even this recursive algorithm **RC** is slow and takes $O(2^n)$ time. (Why $O(2^n)$?)

Recursive tree of RC:



Observation: **RC**(3,*p*) and **RC**(2,*p*) is getting computed from scratch multiple times.

Dynamic Programming Enters

Dynamic Programming Enters

Idea:

Dynamic Programming Enters

Idea: 1) Define an array $profit[1 : n]$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use

Dynamic Programming Enters

Idea: 1) Define an array $\mathit{profit}[1:n]$

2) Store the maximum profit obtainable from an i length rod in $\mathit{profit}[i]$ for future use after calculating it the first time.

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$$profit[1:n] = \{p[1], -1, \ldots, -1\}$$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

$p[i] = -1$ means profit for i length rod is not computed yet

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

RC(n, p)**:**

$p[i] = -1$ means profit for i length rod is not computed yet

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \dots, -1\}$

$p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$ ← $p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$ ← Profit of n length rod is already computed

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$ ← $p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$ ← Profit of n length rod is already computed
2. **return** $profit[n]$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$ ← $p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$ ← Profit of n length rod is already computed
2. **return** $profit[n]$
3. $profit[n] = p[n]$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$ ← $p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$ ← Profit of n length rod is already computed
2. **return** $profit[n]$
3. $profit[n] = p[n]$
4. **for** $i = 1$ **to** $n - 1$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

```
RC(n, p):
    if profit[n] ≠ −1
        return profit[n]
    profit[n] = p[n]
    for i = 1 to n − 1
```

$p[i] = -1$ means profit for i length rod is not computed yet

Profit of n length rod is already computed

i is the length of the first cut

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$ ← $p[i] = -1$ means profit for i length rod is not computed yet

RC(n, p)**:**

1. **if** $profit[n] \neq -1$ ← Profit of n length rod is already computed
2. **return** $profit[n]$
3. $profit[n] = p[n]$
4. **for** $i = 1$ **to** $n - 1$ ← i is the length of the first cut
5. $profit[n] = \textbf{Max}(profit[n], p[i] + \textbf{RC}(n - i, p))$

Dynamic Programming Enters

Idea: 1) Define an array $profit[1:n]$

2) Store the maximum profit obtainable from an i length rod in $profit[i]$ for future use after calculating it the first time.

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

RC(n, p)**:**

```
if profit[n] ≠ -1
    return profit[n]
profit[n] = p[n]
for i = 1 to n - 1
    profit[n] = Max(profit[n], p[i] + RC(n - i, p))
return profit[n]
```

$p[i] = -1$ means profit for i length rod is not computed yet

Profit of n length rod is already computed

i is the length of the first cut

Time Complexity of RC

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

RC(n, p)**:**

if $profit[n] \neq -1$

  return $profit[n]$

$profit[n] = p[n]$

for $i = 1$ **to** $n - 1$

  $profit[n] = \textbf{Max}(profit[n], p[i] + \textbf{RC}(n - i, p))$

return $profit[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\textbf{RC}(n, p)$

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

$\textbf{RC}(n, p)$**:**

if $profit[n] \neq -1$
    return $profit[n]$
$profit[n] = p[n]$
for $i = 1$ **to** $n - 1$
    $profit[n] = \textbf{Max}(profit[n], p[i] + \textbf{RC}(n - i, p))$
return $profit[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$profit[1:n] \;= \{p[1], -1, \ldots, -1\}$

$\mathbf{RC}(n, p)$**:**

if $profit[n] \neq -1$

    return $profit[n]$

$profit[n] = p[n]$

for $i = 1$ **to** $n - 1$

    $profit[n] = \mathbf{Max}(profit[n], p[i] + \mathbf{RC}(n - i, p))$

return $profit[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$\mathit{profit}[1:n] = \{p[1], -1, \ldots, -1\}$

$\mathbf{RC}(n, p)$**:**

if $\mathit{profit}[n] \neq -1$

  return $\mathit{profit}[n]$

$\mathit{profit}[n] = p[n]$

for $i = 1$ **to** $n - 1$

  $\mathit{profit}[n] = \mathbf{Max}(\mathit{profit}[n], p[i] + \mathbf{RC}(n - i, p))$

return $\mathit{profit}[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = c' + \Theta(n) + T(n-1)$$

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

$\mathbf{RC}(n, p)$:

```
if profit[n] ≠ −1
    return profit[n]
profit[n] = p[n]
for i = 1 to n − 1
    profit[n] = Max(profit[n], p[i] + RC(n − i, p))
return profit[n]
```

Time Complexity of RC

Let $\color{red}{T(n)}$ denote the runtime of $\textbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = \underbrace{c'}_{} + \Theta(n) + T(n-1)$$

Cost of line 1,2,3, & 6

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

RC(n, p)**:**

1. **if** $profit[n] \neq -1$
2.   **return** $profit[n]$
3. $profit[n] = p[n]$
4. **for** $i = 1$ **to** $n - 1$
5. $profit[n] = \textbf{Max}(profit[n], p[i] + \textbf{RC}(n - i, p))$
6. **return** $profit[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

Cost of $\mathbf{RC}(n-1, p)$

$$T(1) = c$$

$$T(n) = \underbrace{c'}_{\text{Cost of line 1,2,3, \& 6}} + \Theta(n) + \overbrace{T(n-1)}^{\text{Cost of } \mathbf{RC}(n-1,p)}$$

Cost of line 1,2,3, & 6

$profit[1:n] = \{p[1], -1, \ldots, -1\}$

$\mathbf{RC}(n, p)$:

1. **if** $profit[n] \neq -1$
2. **return** $profit[n]$
3. $profit[n] = p[n]$
4. **for** $i = 1$ **to** $n - 1$
5. $profit[n] = \mathbf{Max}(profit[n], p[i] + \mathbf{RC}(n-i, p))$
6. **return** $profit[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Call to $\mathbf{RC}(n-1, p)$ computes $\mathit{profit}[1:(n-1)]$

Then,

$$T(1) = c$$

$$T(n) = \underbrace{c'}_{\text{Cost of line 1,2,3, \& 6}} + \Theta(n) + \overbrace{T(n-1)}^{\text{Cost of } \mathbf{RC}(n-1,p)}$$

Cost of $\mathbf{RC}(n-1, p)$

Cost of line 1,2,3, & 6

$\mathit{profit}[1:n] = \{p[1], -1, \ldots, -1\}$

RC(n, p):

1. **if** $\mathit{profit}[n] \neq -1$
2.   **return** $\mathit{profit}[n]$
3. $\mathit{profit}[n] = p[n]$
4. **for** $i = 1$ **to** $n-1$
5. $\mathit{profit}[n] = \mathbf{Max}(\mathit{profit}[n], p[i] + \mathbf{RC}(n-i, p))$
6. **return** $\mathit{profit}[n]$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Call to $\mathbf{RC}(n-1, p)$ computes $profit[1:(n-1)]$

Then,

$$T(1) = c$$

$$T(n) = \underbrace{c'}_{} + \underbrace{\Theta(n)}_{} + \overbrace{T(n-1)}^{}$$

Cost of $\mathbf{RC}(n-1, p)$

Cost of line 1,2,3, & 6

Cost of for loop from $i = 2$ to $n-1$ as $\mathbf{RC}(n-i, p)$ will return immediately.

```
profit[1 : n] = {p[1], -1, ..., -1}
RC(n, p):
1.  if profit[n] ≠ -1
2.      return profit[n]
3.  profit[n] = p[n]
4.  for i = 1 to n - 1
5.      profit[n] = Max(profit[n], p[i] + RC(n - i, p))
6.  return profit[n]
```

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = c' + \Theta(n) + T(n-1)$$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = c' + \Theta(n) + T(n-1)$$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\textbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = c' + \Theta(n) + T(n-1)$$

$$= c' + \Theta(n) + c' + \Theta(n-1) + T(n-2)$$

Time Complexity of $\mathbf{RC}$

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$T(n) = c' + \Theta(n) + T(n-1)$$

$$= c' + \Theta(n) + c' + \Theta(n-1) + T(n-2)$$

Time Complexity of **RC**

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$\begin{aligned} T(n) &= c' + \Theta(n) + T(n-1) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + T(n-2) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + c' + \Theta(n-2) + T(n-3) \end{aligned}$$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$\begin{aligned} T(n) &= c' + \Theta(n) + T(n-1) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + T(n-2) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + c' + \Theta(n-2) + T(n-3) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + c' + \Theta(n-2) + c' + \Theta(n-3) + \ldots + c \end{aligned}$$

Time Complexity of RC

Let $T(n)$ denote the runtime of $\mathbf{RC}(n, p)$

Then,

$$T(1) = c$$

$$\begin{aligned} T(n) &= c' + \Theta(n) + T(n-1) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + T(n-2) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + c' + \Theta(n-2) + T(n-3) \\ &= c' + \Theta(n) + c' + \Theta(n-1) + c' + \Theta(n-2) + c' + \Theta(n-3) + \ldots + c \\ &= \Theta(n^2) \end{aligned}$$