Dynamic Programming Part 2

CSCI 232

Quiz 3 Logistics

Taken via D2L. You are not timed, but you have only one attempt
Opens 6:00 AM on Thursday, closes 11:59 PM on Thursday

I will be traveling on Thursday, but you can still message/email me if you have questions

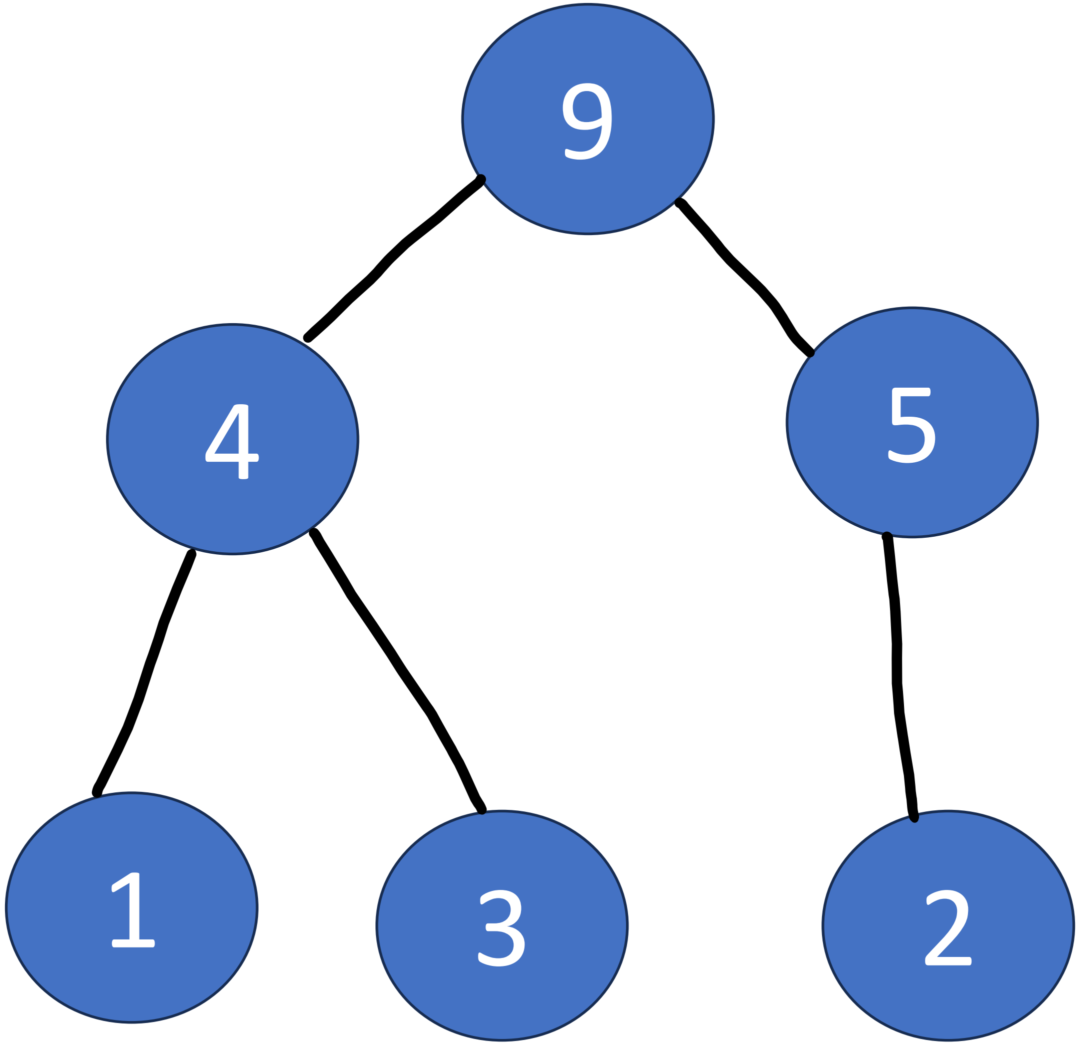
10-15 Questions

- Short answer, multiple choice, true or false

Quiz Content

- Basic Graphs
- MST
- Kruskal’s Algorithm, Prims Algorithm
- Dijkstra's Algorithm
- Divide and Conquer, Closest Pair of points algorithm
- Greedy Algorithms
- Dynamic Programming
- Heaps

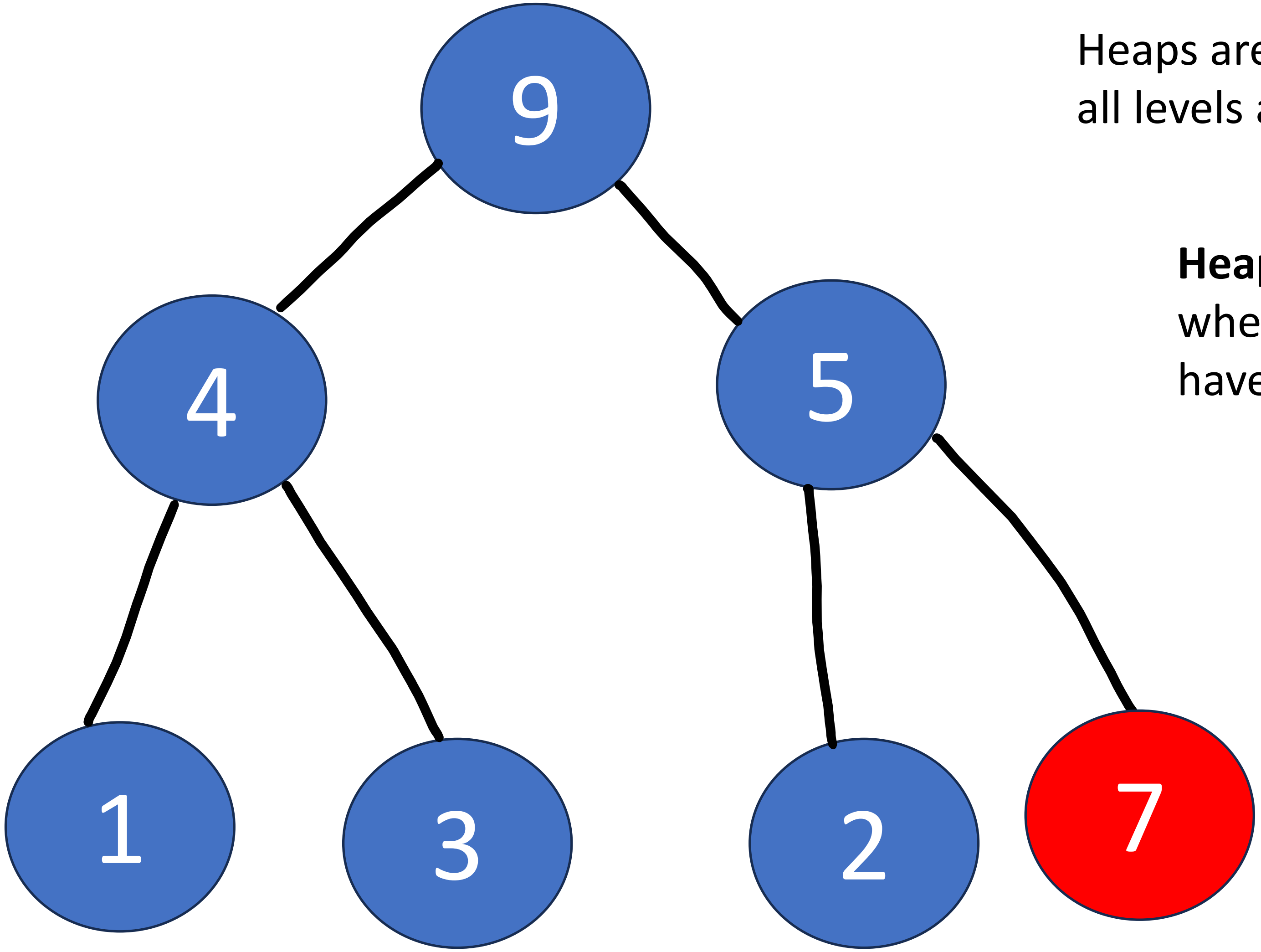
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

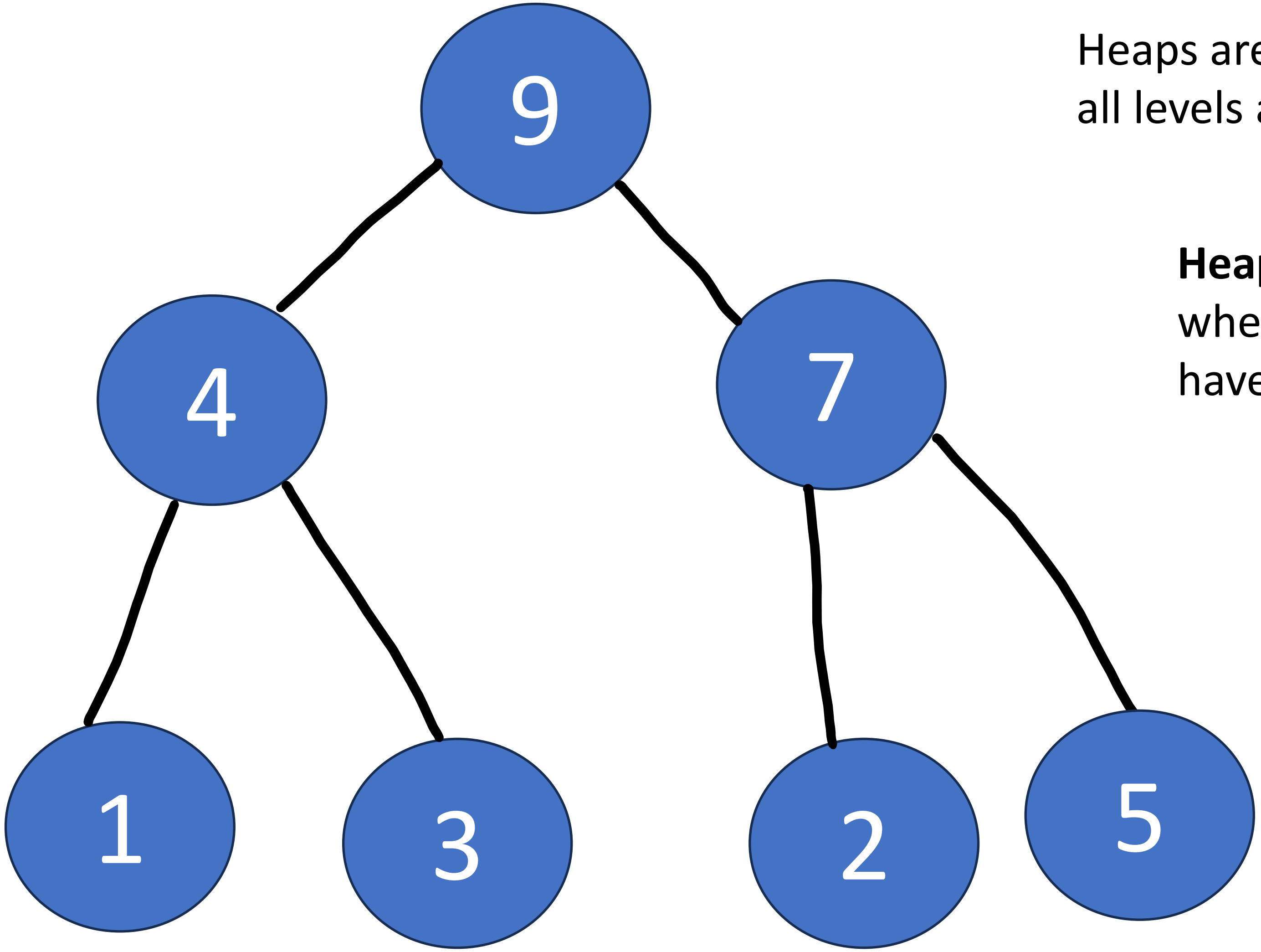
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

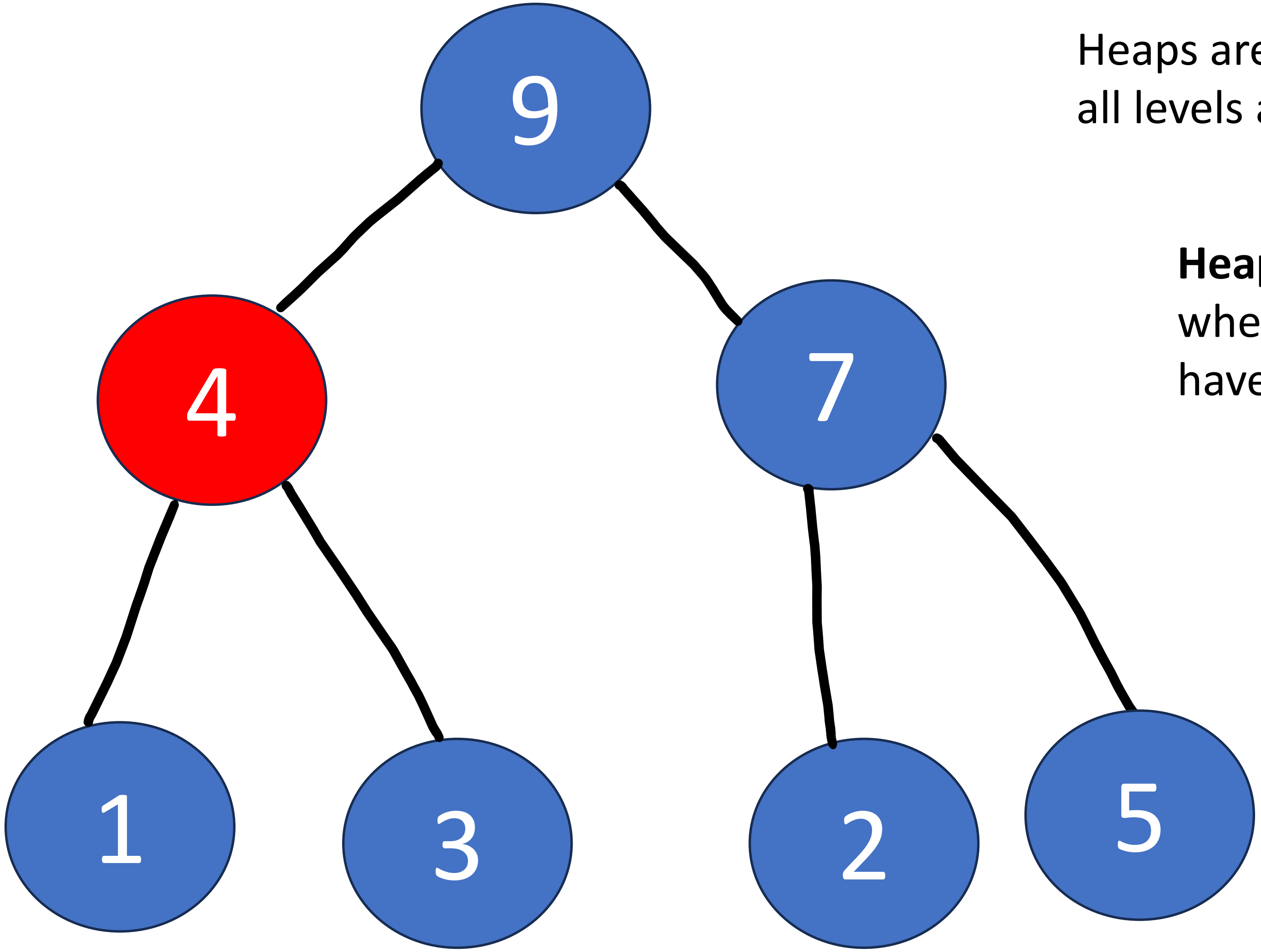
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

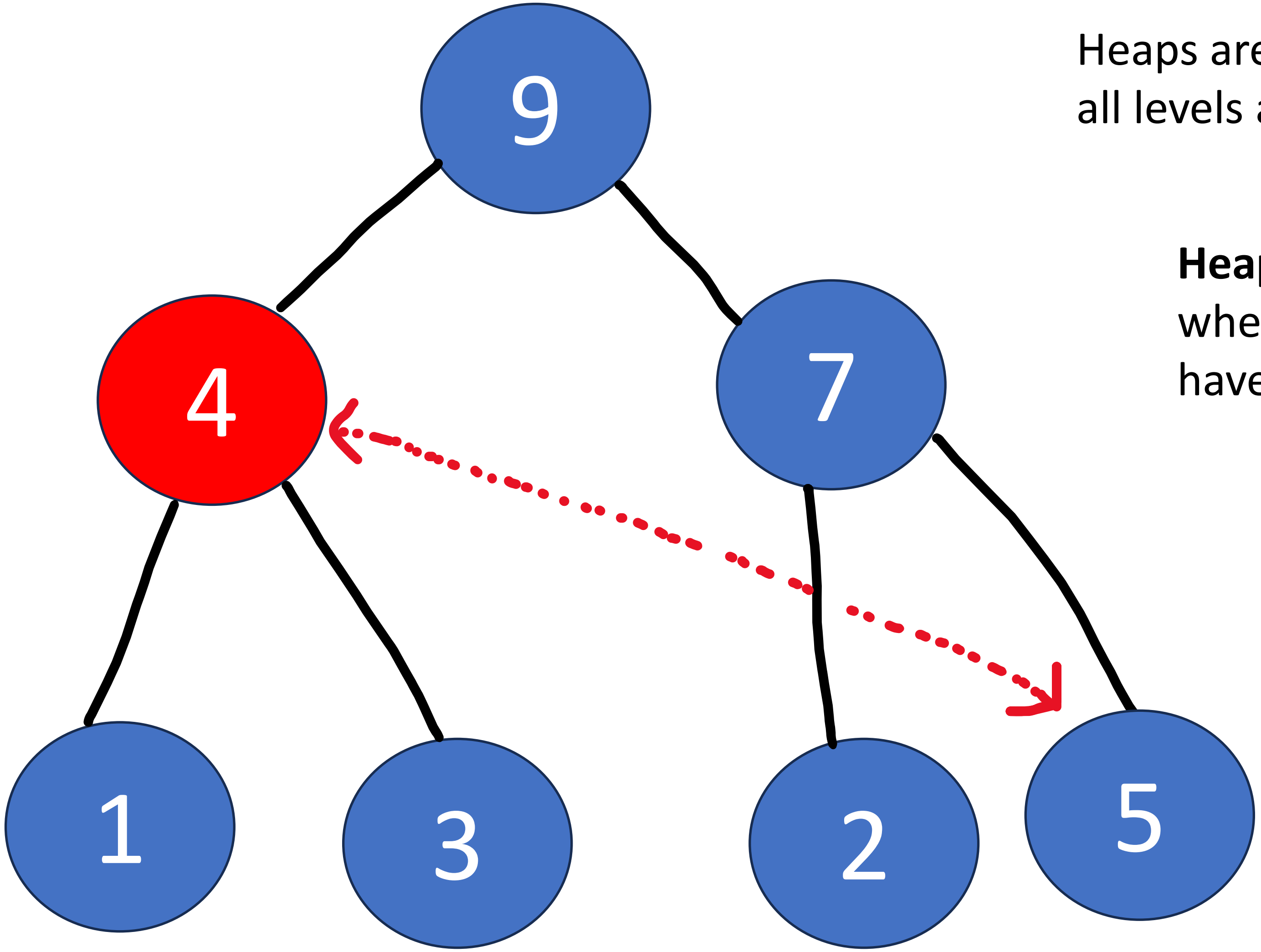
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

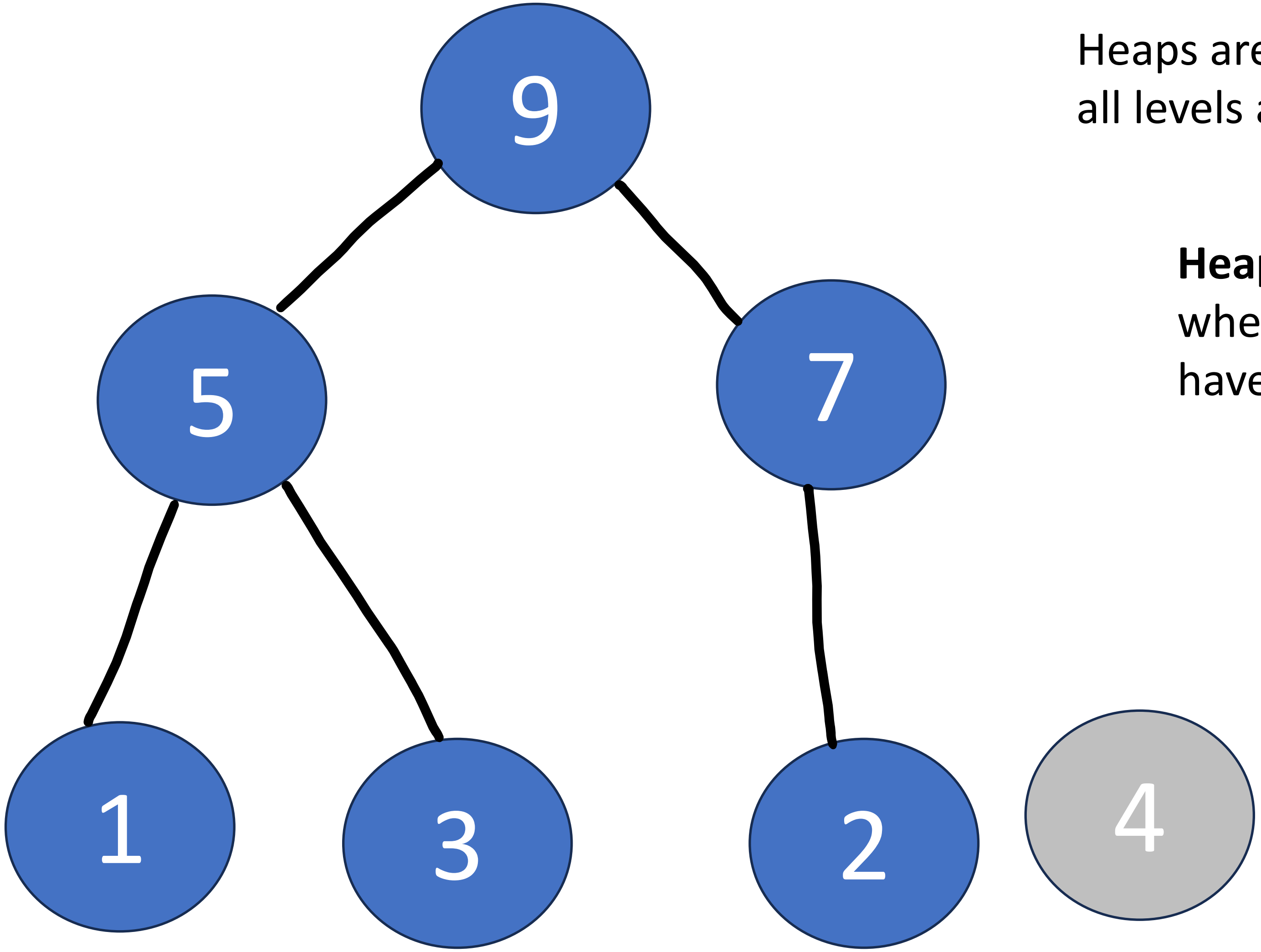
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

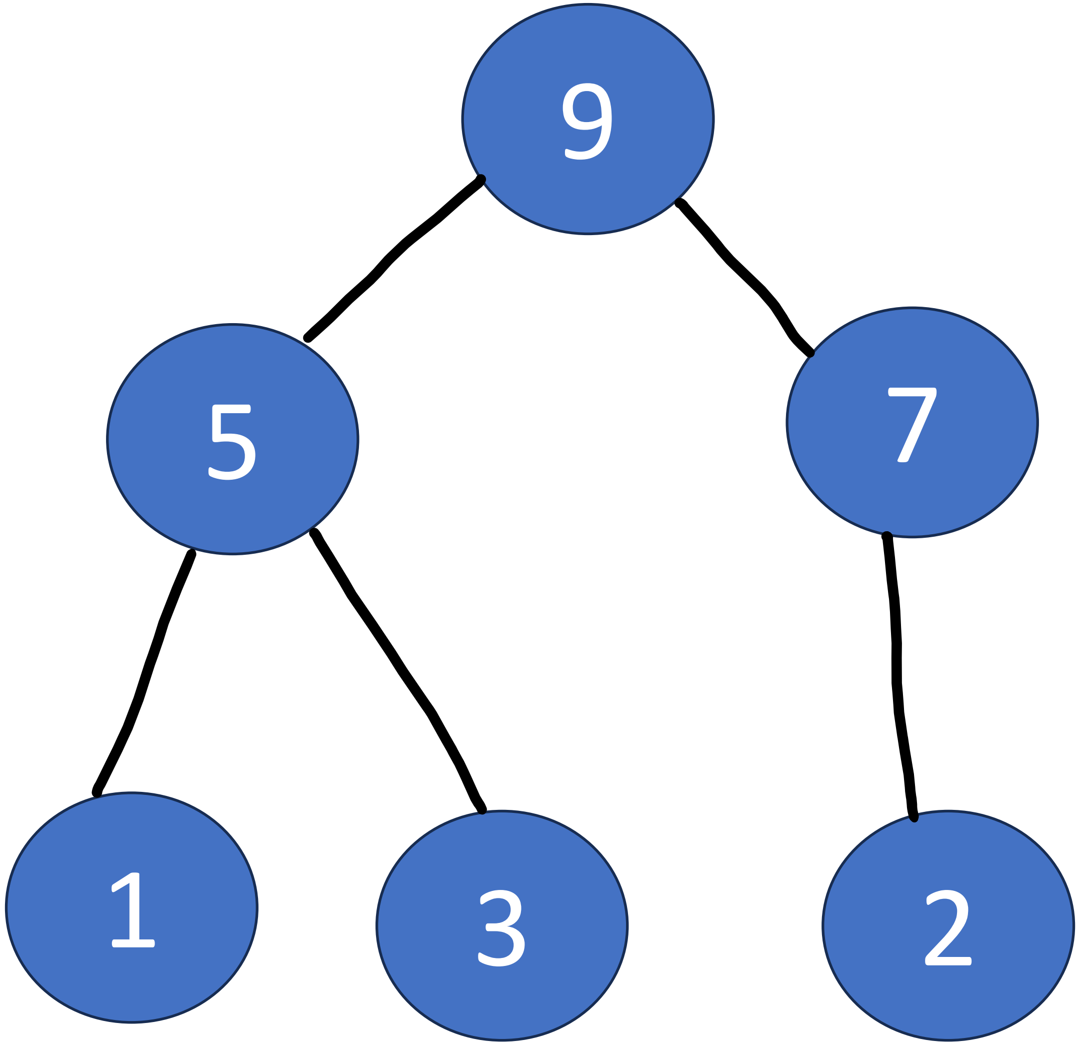
A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*

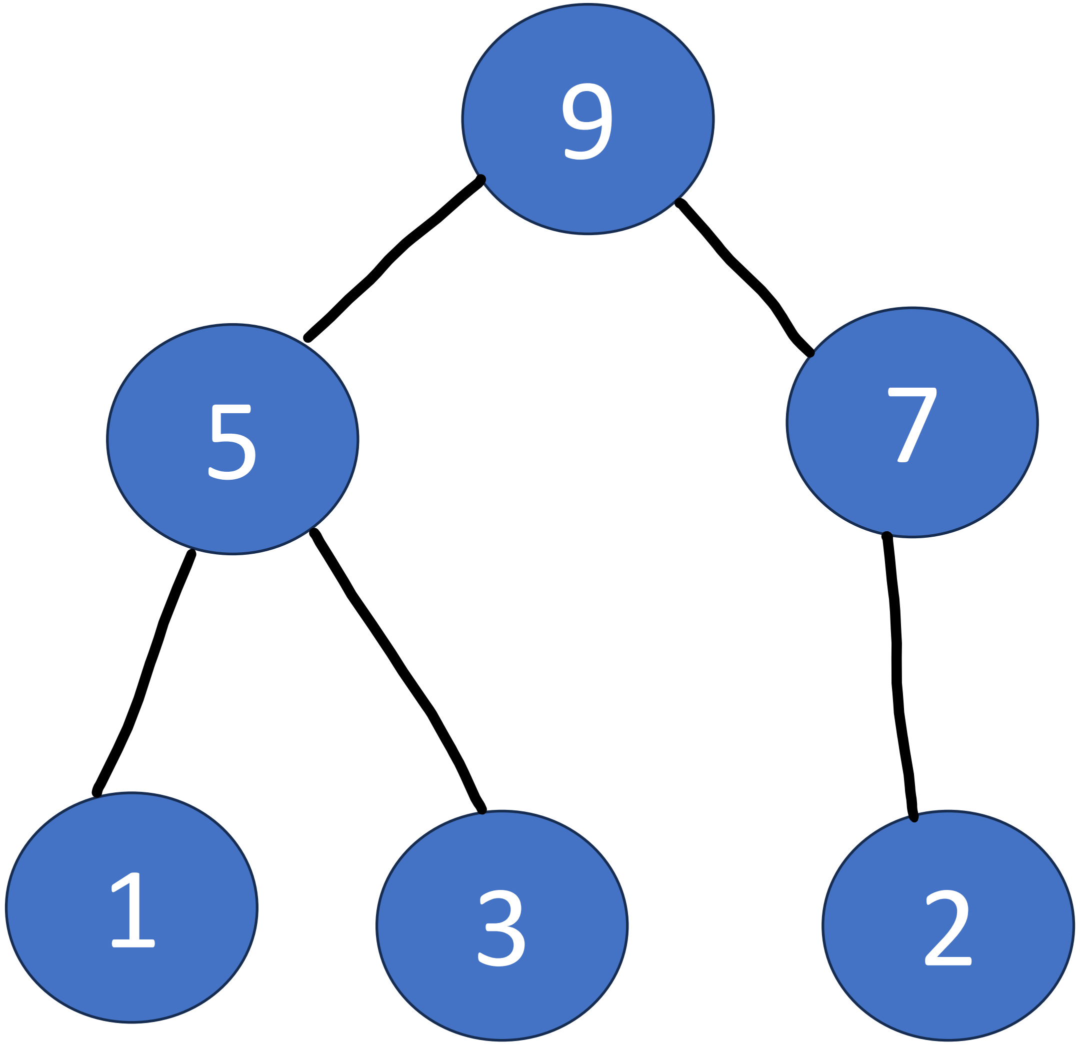


Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

Whenever we remove or add something, we will need to re-heapify the tree

A **Heap** is a special type of tree, where the parent nodes are greater than both the child nodes. The root will always be the largest element *(Max heap)*



Heaps are always a **complete** binary tree, which means all levels are filled except for the lowest one

Heapify is the process for organizing the heap whenever we add or remove nodes to ensure we still have a Heap

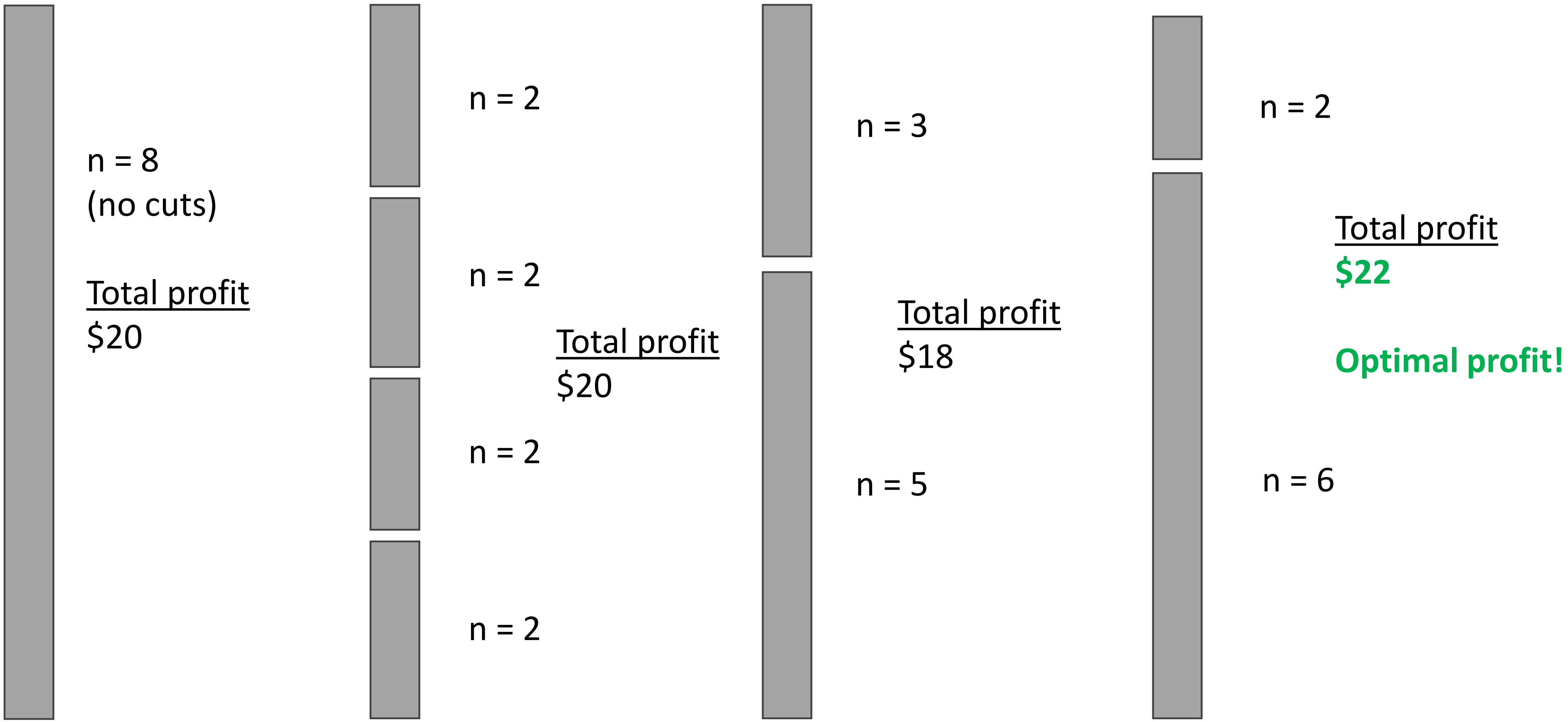
Whenever we remove or add something, we will need to re-heapify the tree

This data structure can be helpful when you frequently need to interact with the highest/lowest element in a tree → O(1) time for getting min/max

Rod Cutting

Given a rod of length n inches, and an array of prices that includes prices of all pieces of size smaller than n, determine the maximum value obtainable by cutting up the road and selling the pieces.

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



n = 8
(no cuts)

Total profit
$20

n = 2

n = 2

n = 2

n = 2

Total profit
$20

n = 3

n = 5

Total profit
$18

n = 2

n = 6

Total profit
$22

Optimal profit!

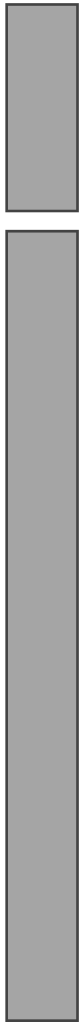
Rod Cutting

Given a rod of length n inches, and an array of prices that includes prices of all pieces of size smaller than n, determine the maximum value obtainable by cutting up the road and selling the pieces.

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

Optimal Substructure

Our solution for a rod length of n=8, has the optimal solution for rod length of n = 6, and n = 2



Total profit
$22

Optimal profit!

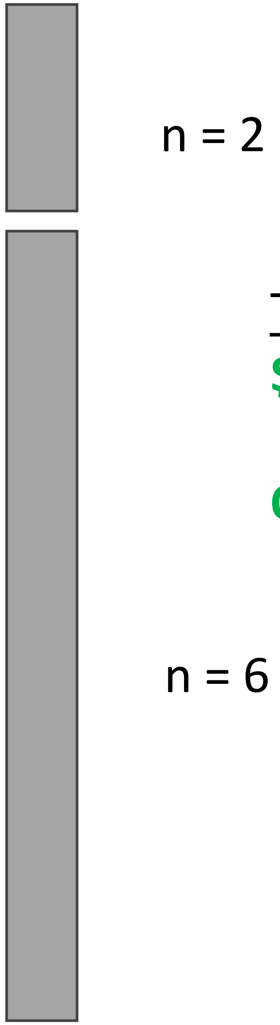
Rod Cutting

Given a rod of length n inches, and an array of prices that includes prices of all pieces of size smaller than n, determine the maximum value obtainable by cutting up the road and selling the pieces.

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

General Approach:

Compute all possible ways to cut the rod using dynamic programming, and return which one had the highest profit



Total profit

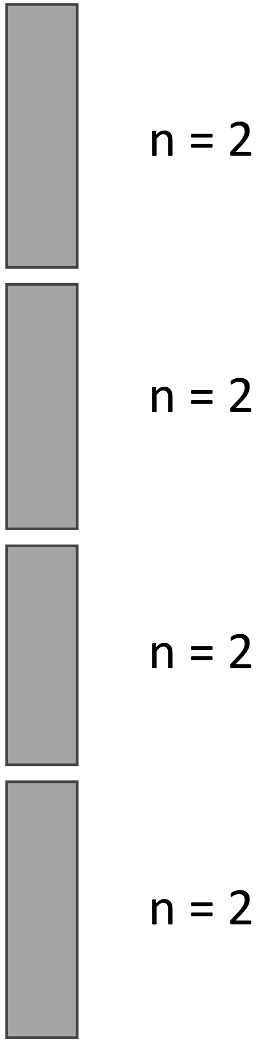
$22

Optimal profit!

Rod Cutting

Given a rod of length n inches, and an array of prices that includes prices of all pieces of size smaller than n, determine the maximum value obtainable by cutting up the road and selling the pieces.

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



Total profit
$20

Overlapping subproblems

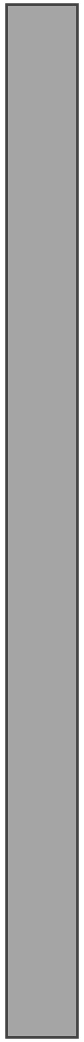
We will compute the optimal way to cut a rod of length n=2 many times. We will use memoization to make sure we don't compute problems that we have already solved.

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index

n = 8



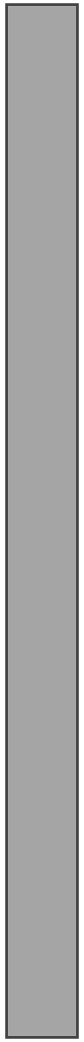
Technically, out algorithm will consider making a cut of length 8 first, but we will skip over this part to avoid confusion

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index

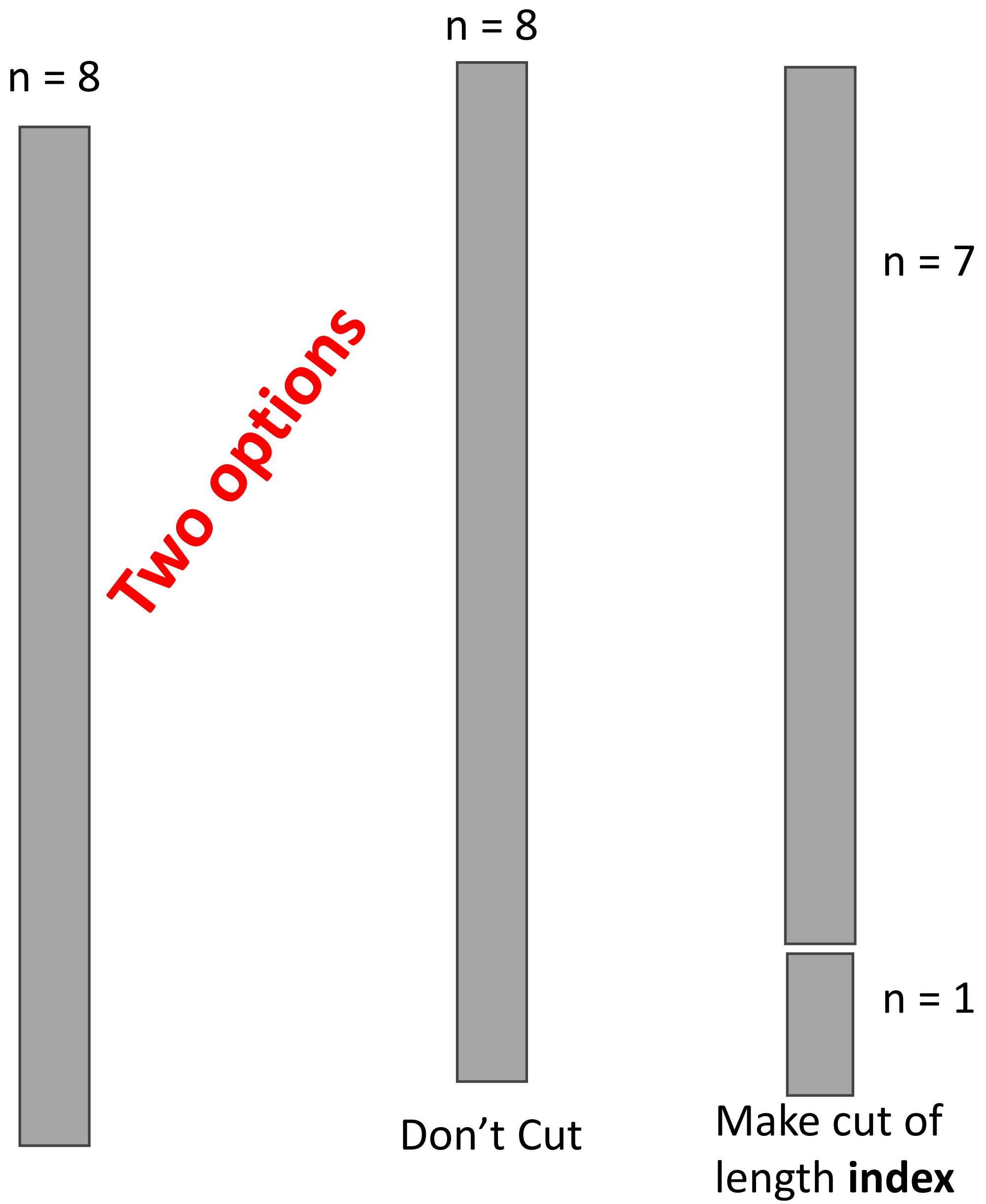
n = 8



Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



n = 8

Two options

n = 8

Don’t Cut

n = 7

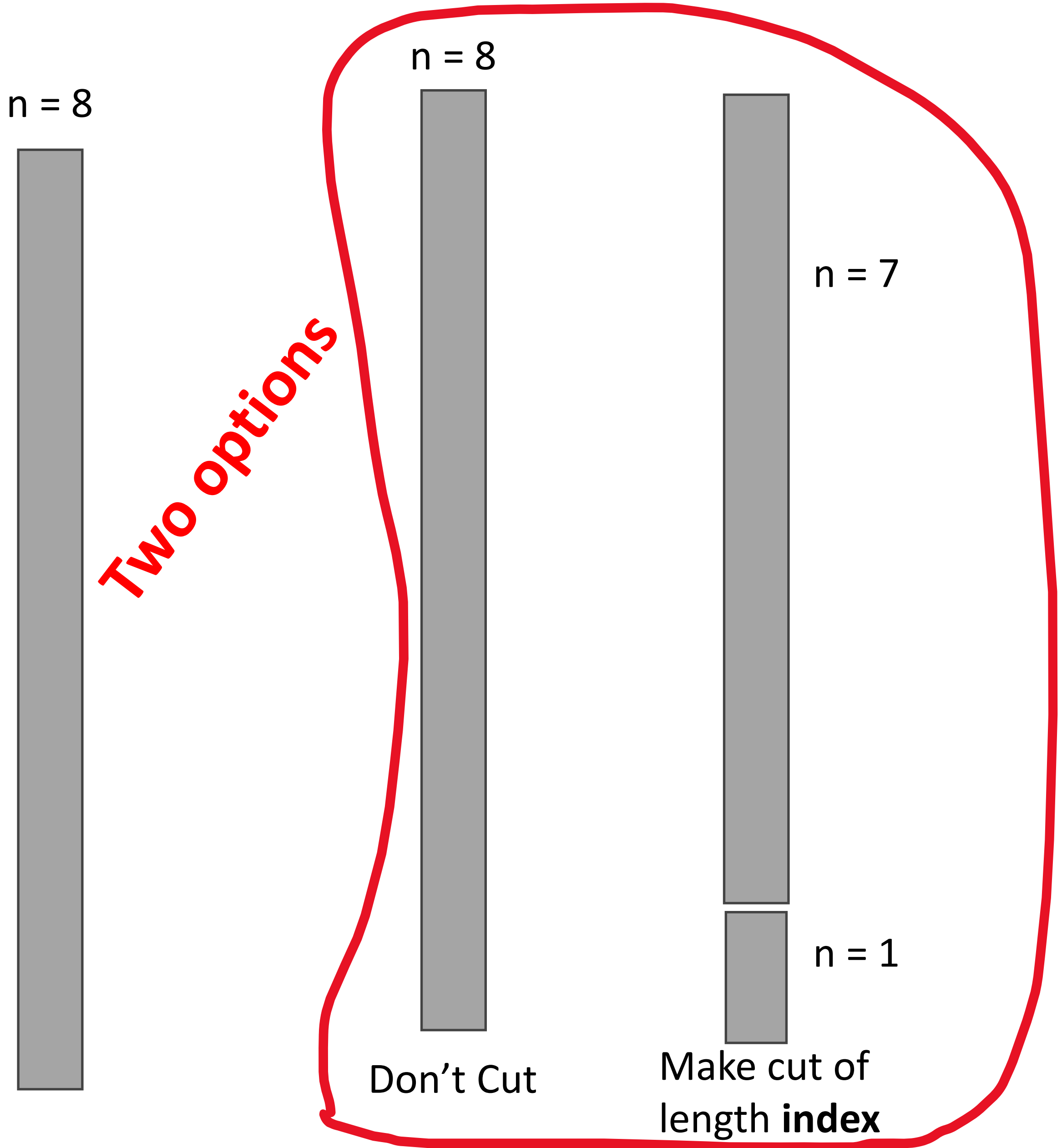
n = 1

Make cut of length **index**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



n = 8

Two options

n = 8

Don't Cut

n = 7

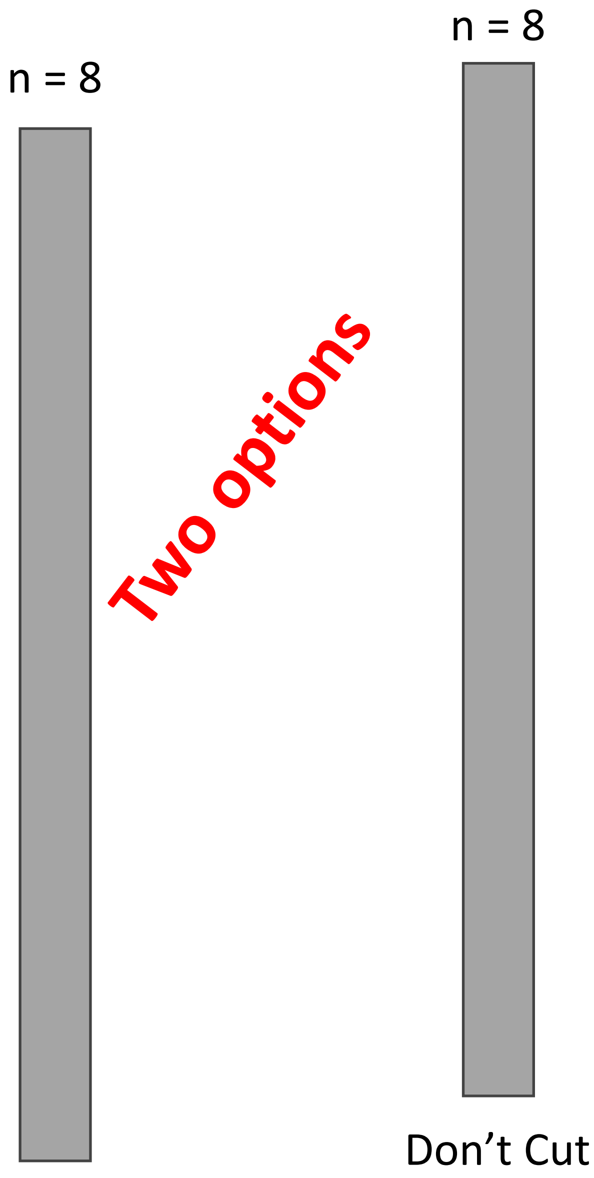
n = 1

Make cut of length **index**

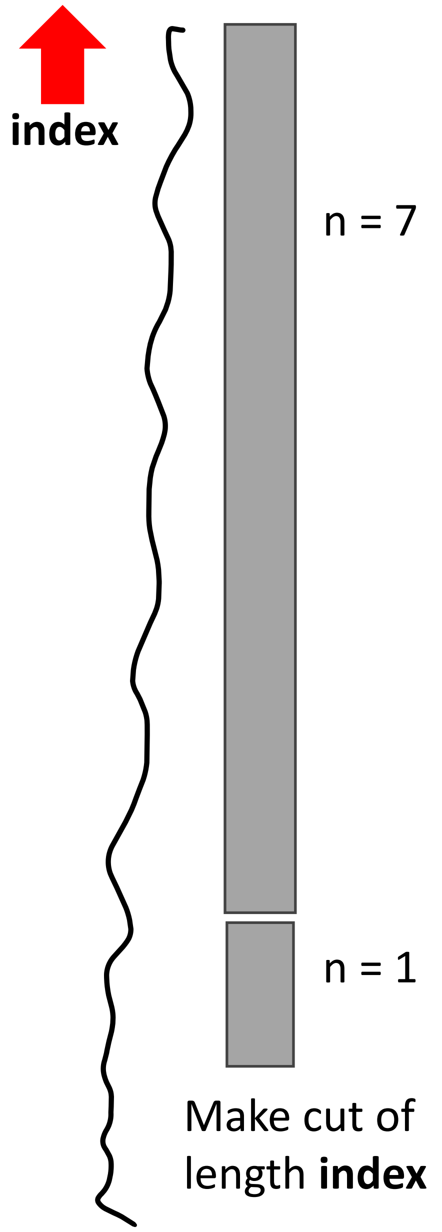
We want to select the option that yield the highest profit

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



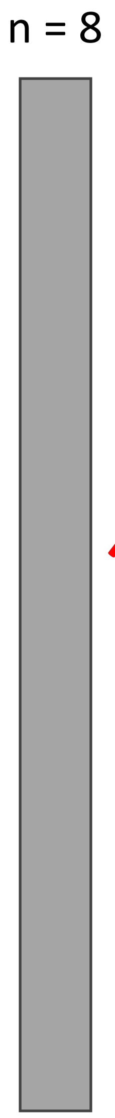
Now we recurse, and check a new cut value



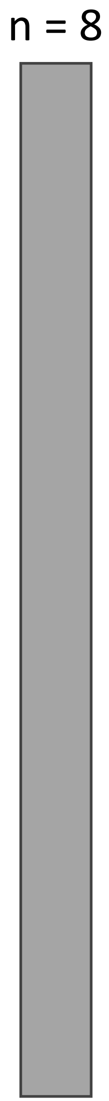
Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



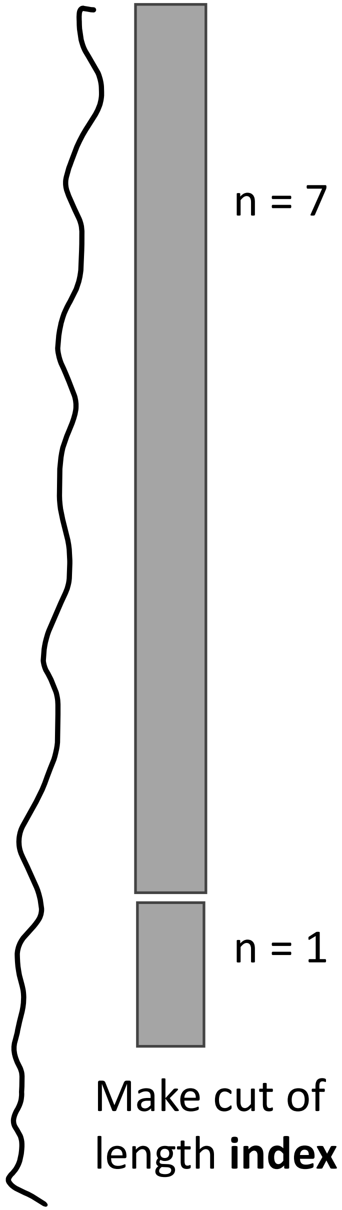
Two options



Don’t Cut

Now we recurse, and check a new cut value

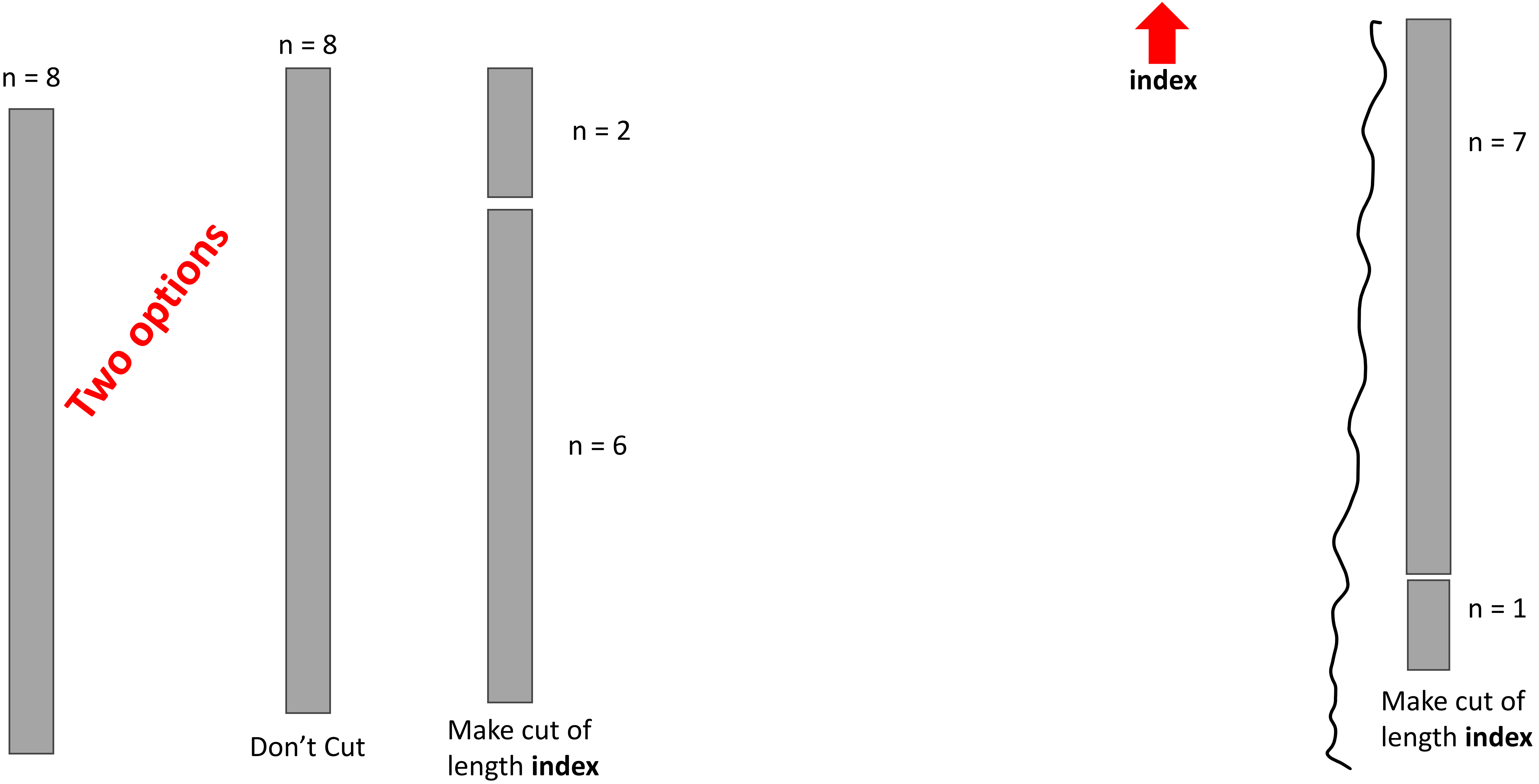
(index – 1)



Make cut of length **index**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 8

Don't Cut

n = 2

n = 6

Make cut of length **index**

n = 7

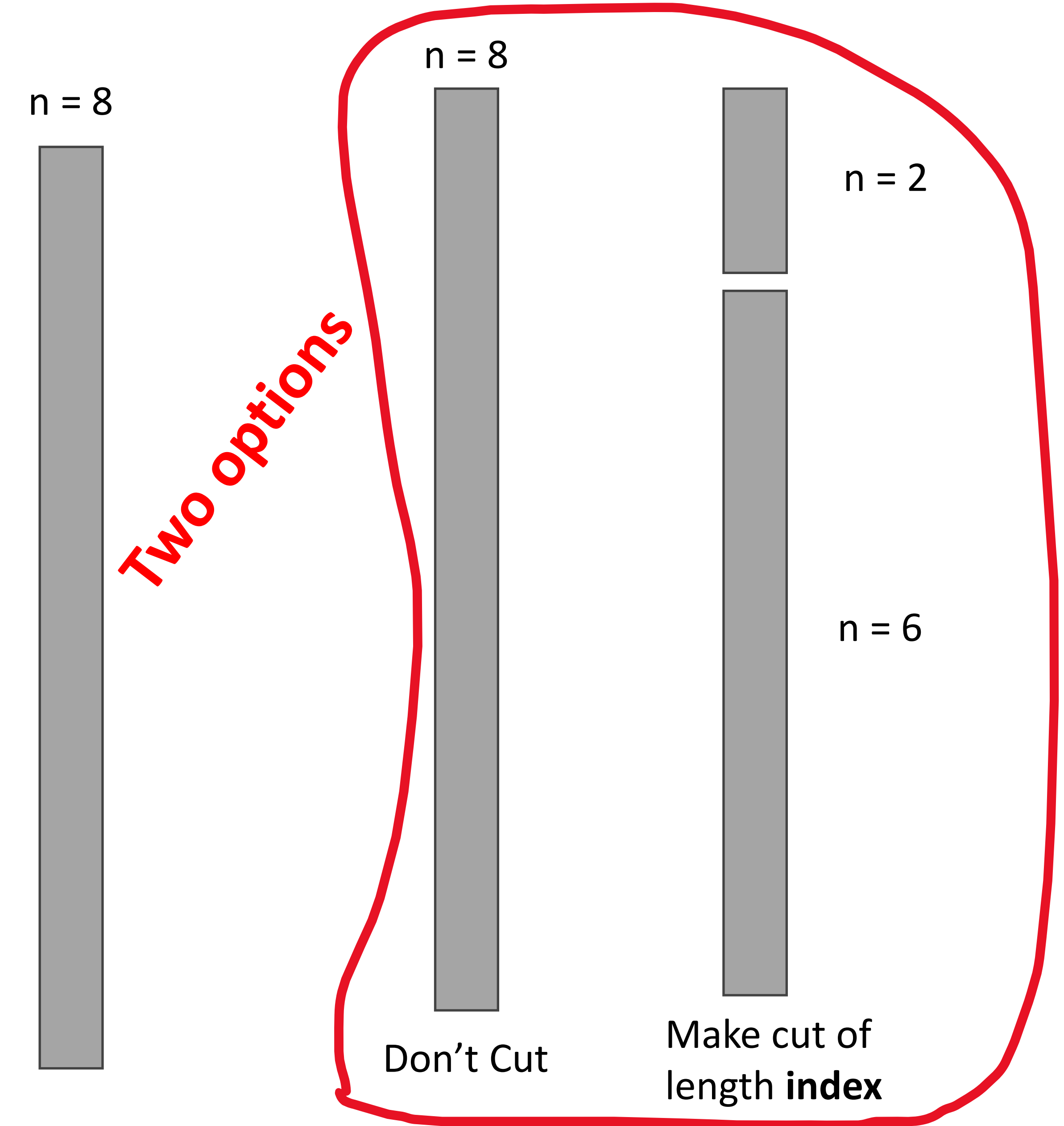
n = 1

Make cut of length **index**

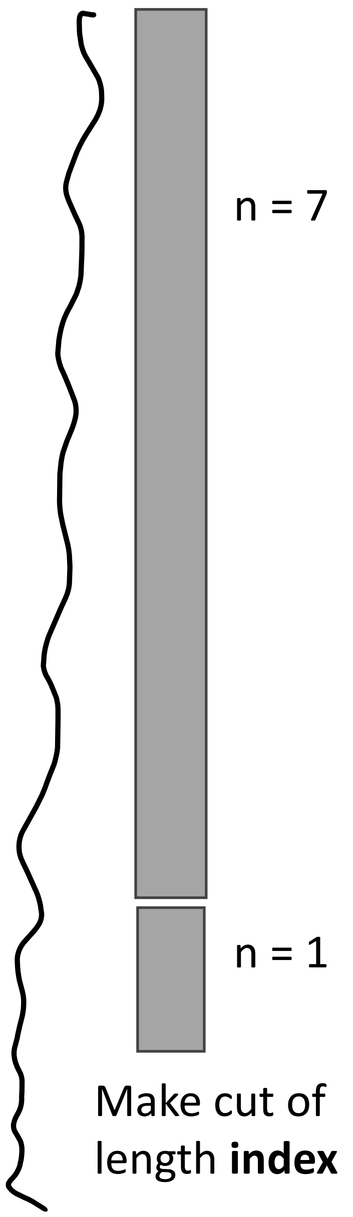
Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index

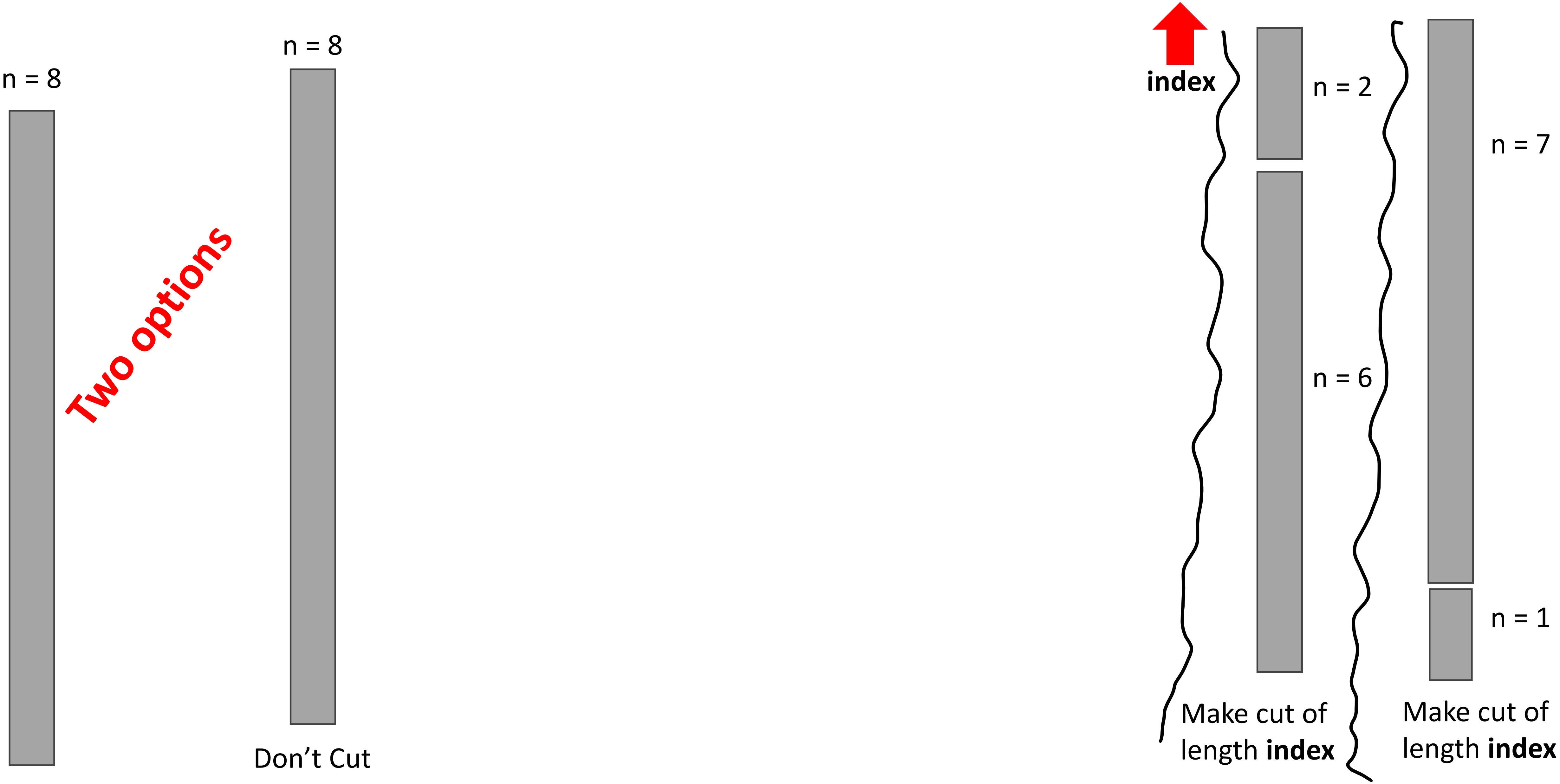


We want to select the option that yield the highest profit



Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 8

Don't Cut

n = 2

n = 6

Make cut of length **index**

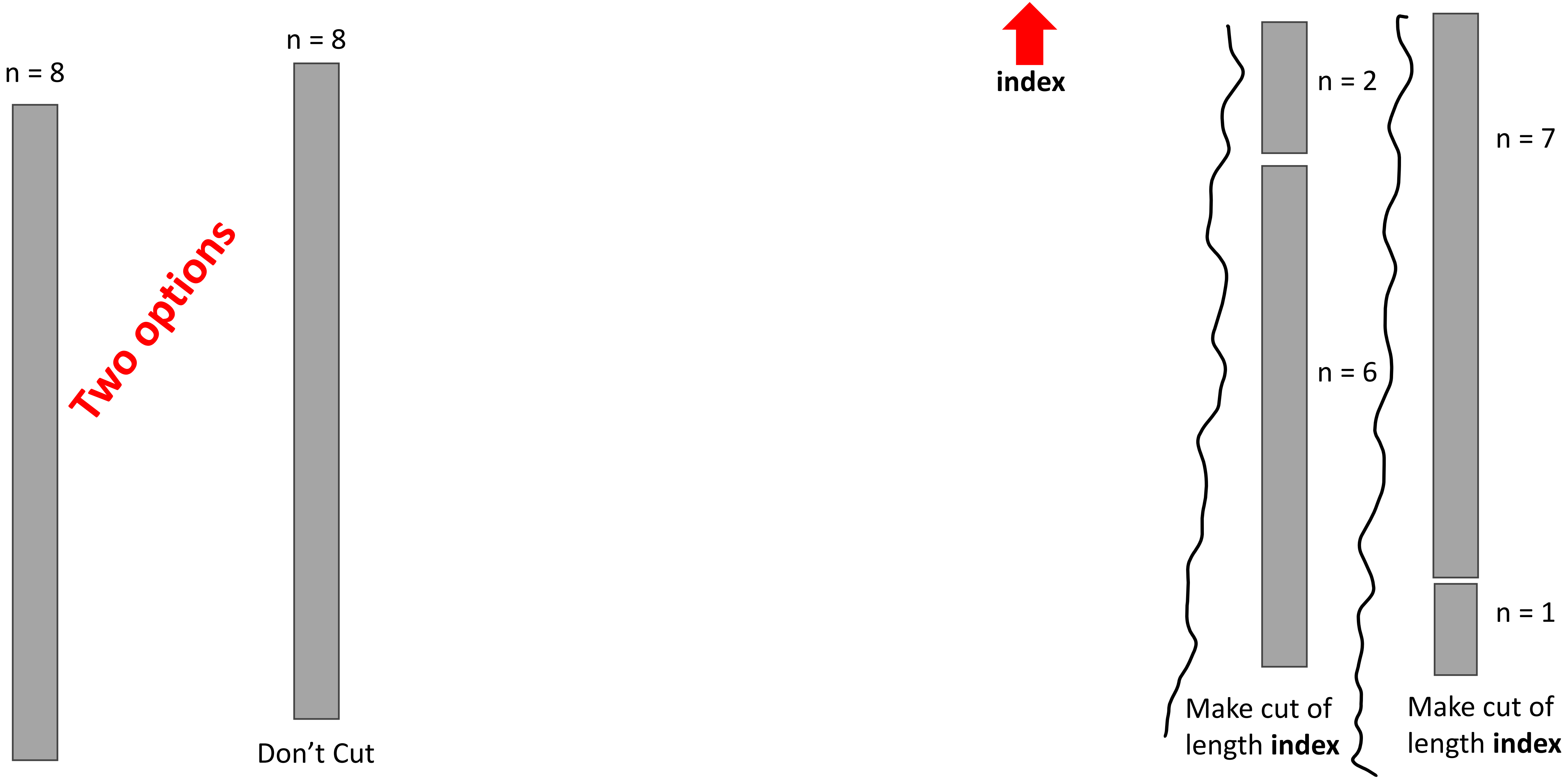
n = 7

n = 1

Make cut of length **index**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



n = 8

Two options

n = 8

Don't Cut

index

n = 2

n = 6

Make cut of length **index**

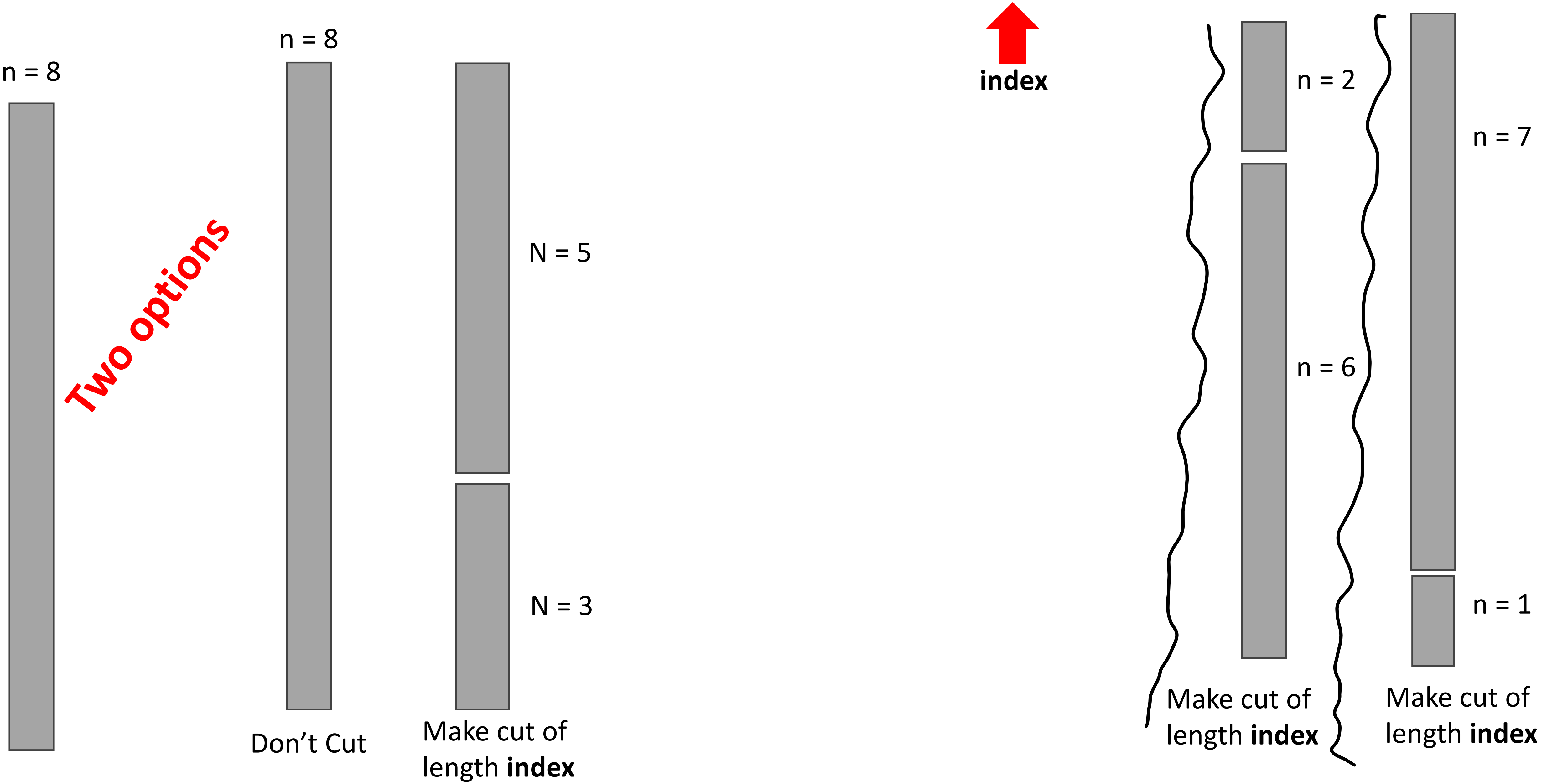
n = 7

n = 1

Make cut of length **index**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 8

Don't Cut

N = 5

N = 3

Make cut of length **index**

n = 2

n = 6

Make cut of length **index**

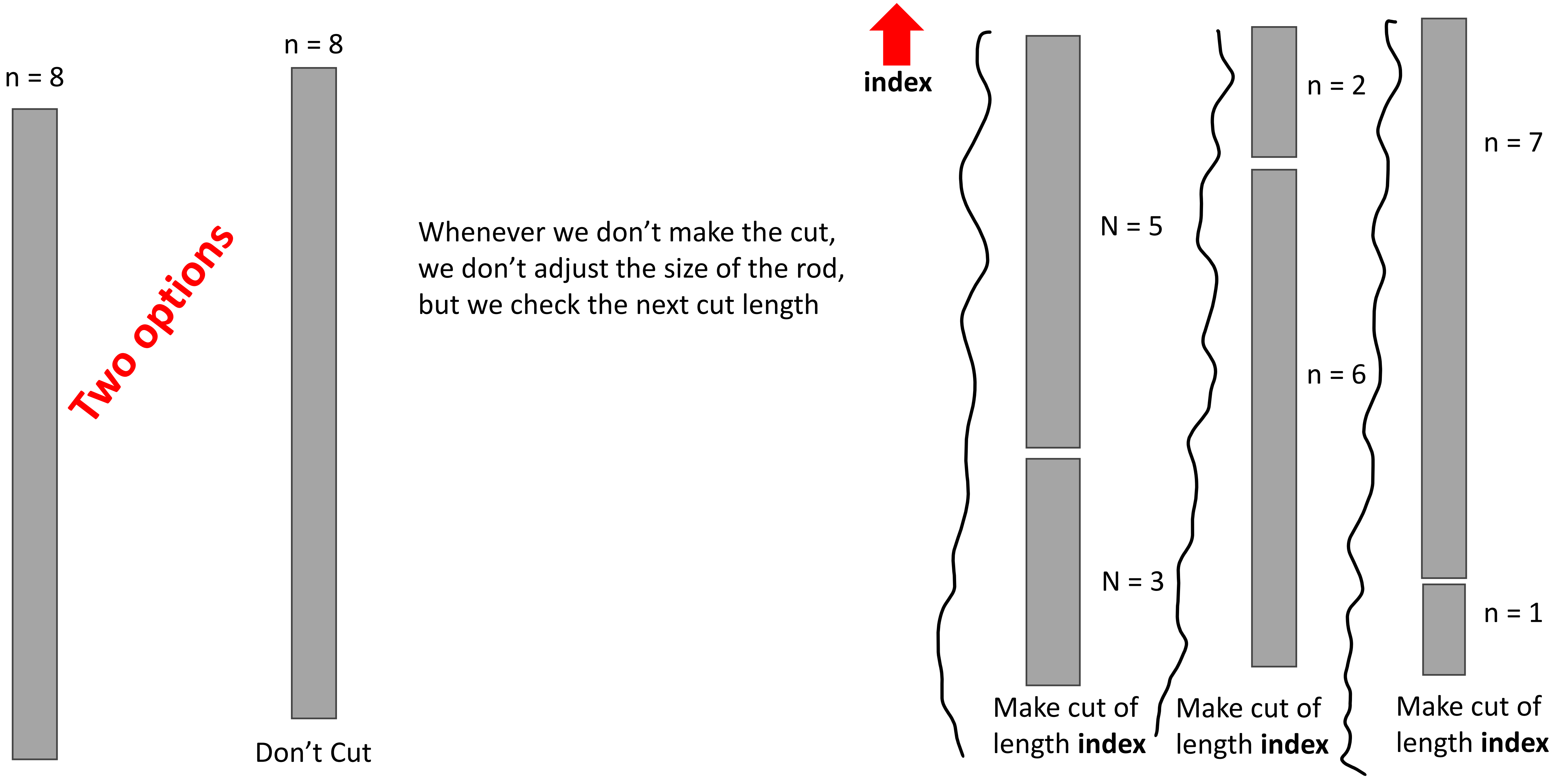
n = 7

n = 1

Make cut of length **index**

Rod Cutting

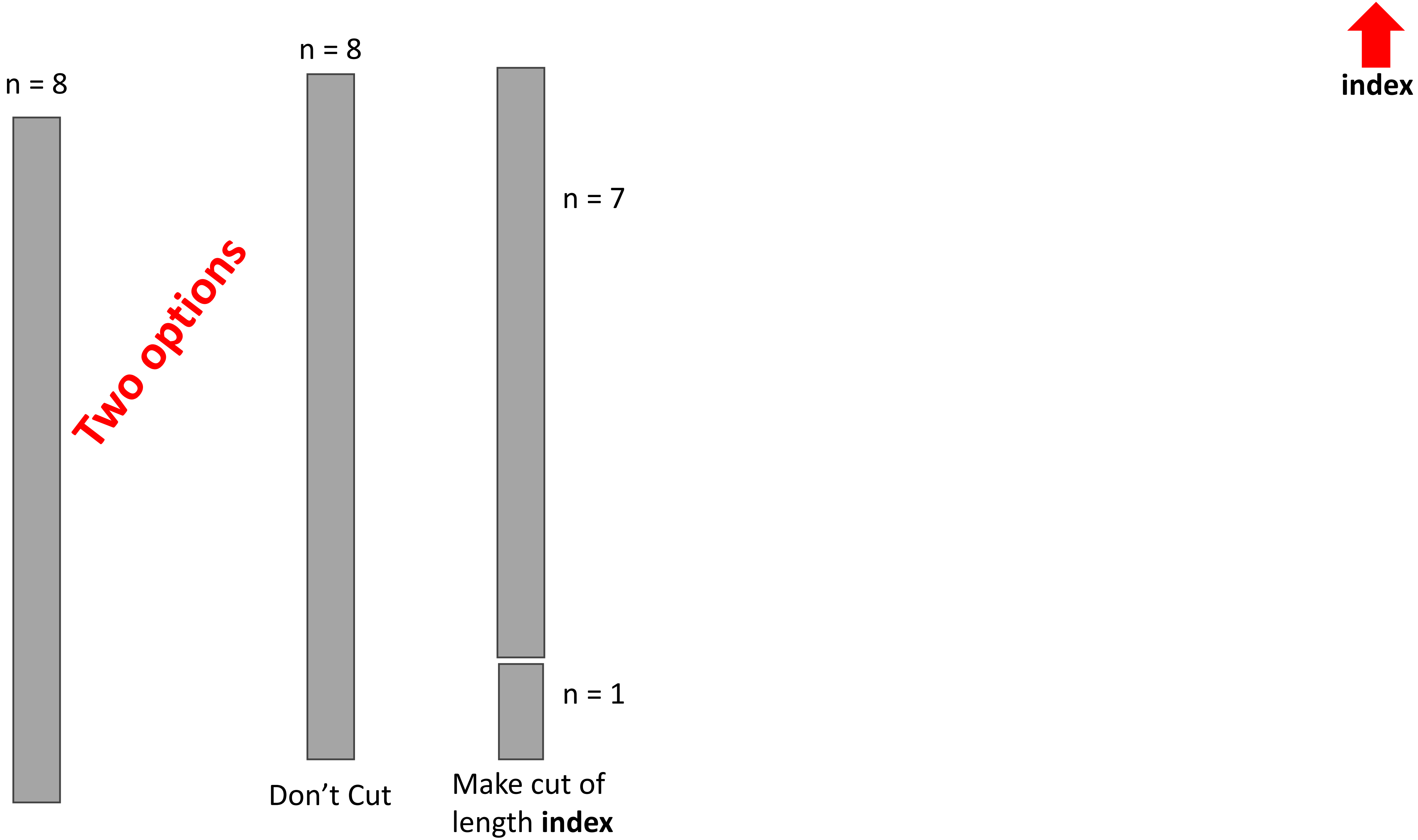
	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20


n = 8
Two options
n = 8
Don't Cut
index
N = 5
N = 3
Make cut of length **index**
n = 2
n = 6
Make cut of length **index**
n = 7
n = 1
Make cut of length **index**

Whenever we don't make the cut, we don't adjust the size of the rod, but we check the next cut length

Rod Cutting

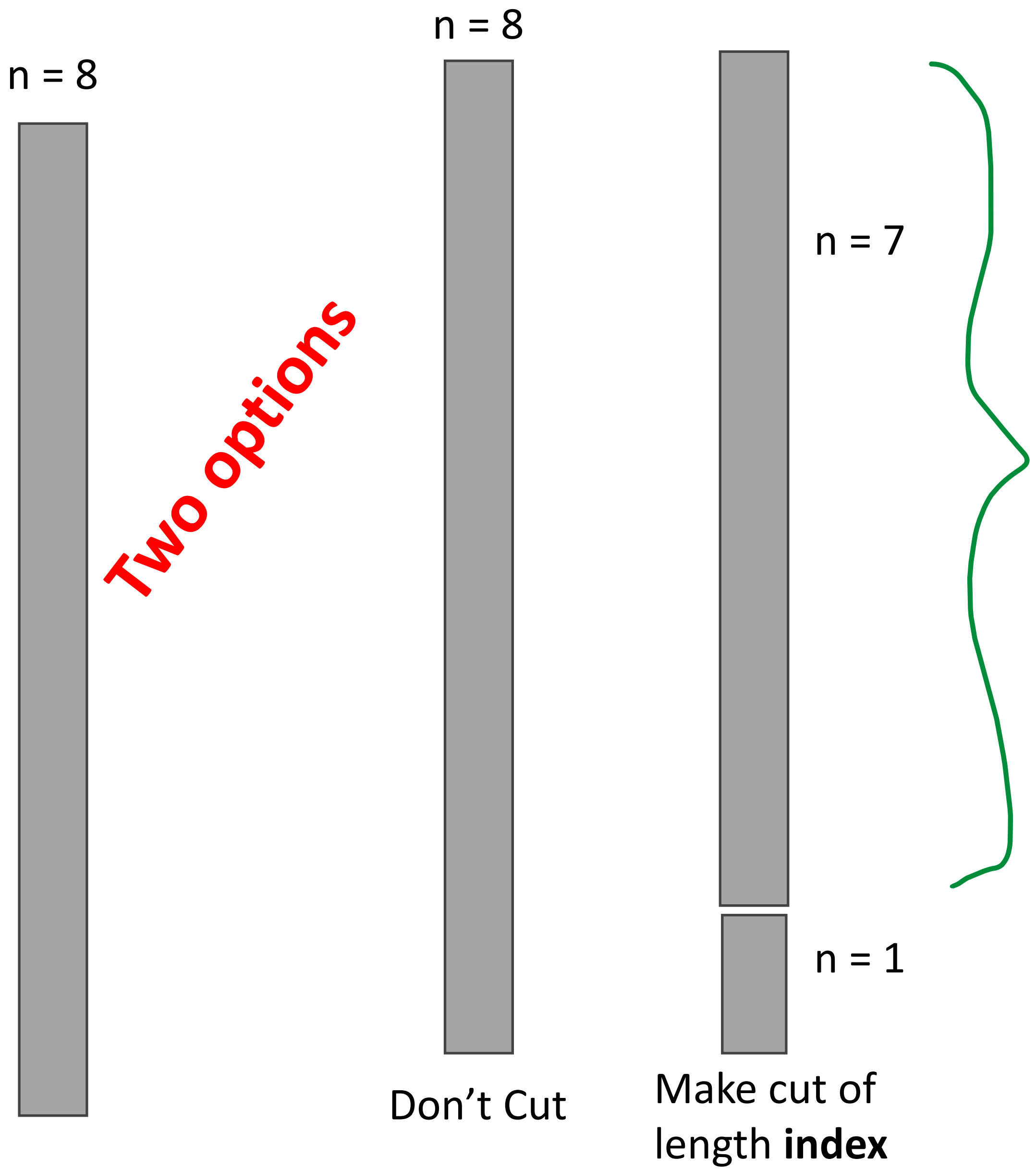
	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



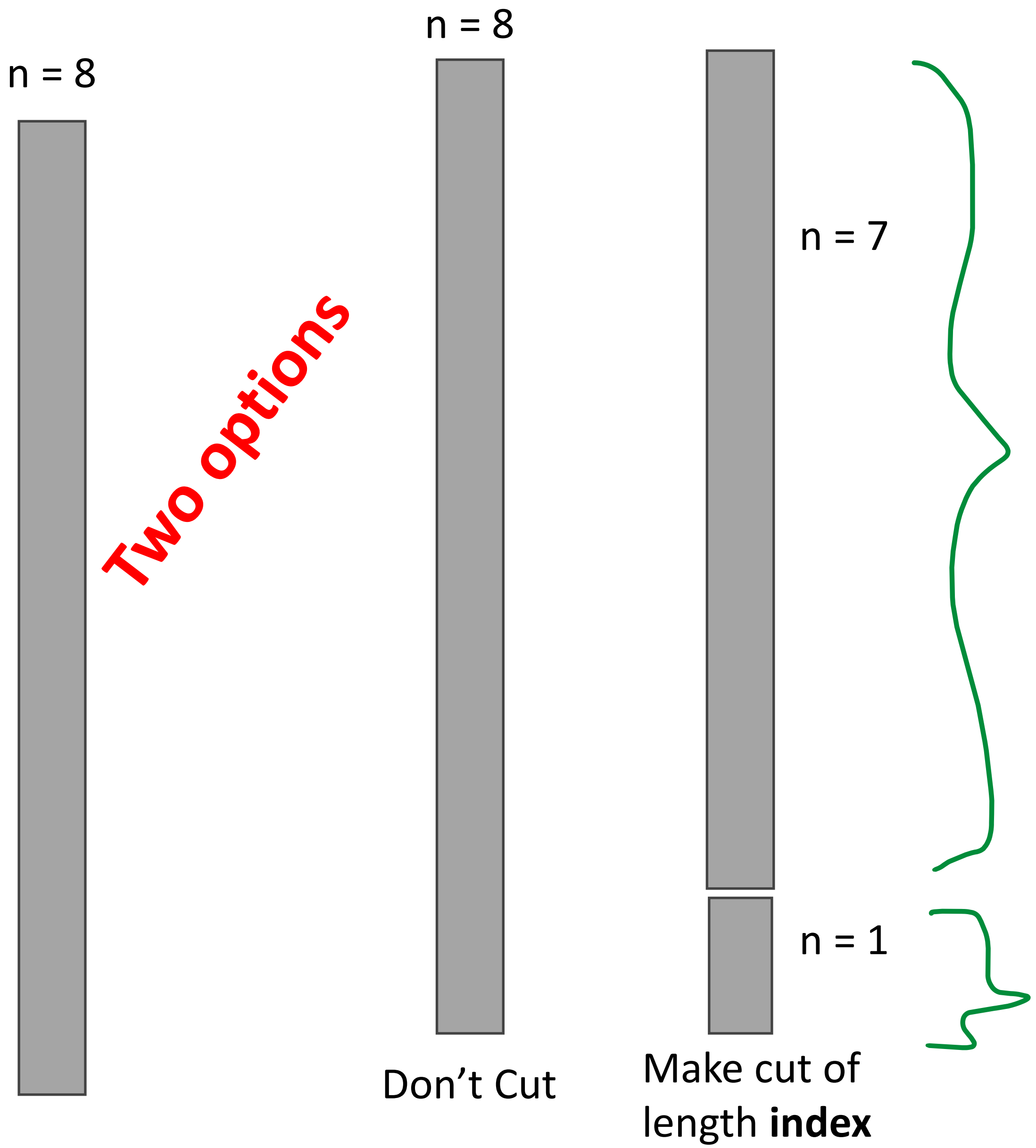
We made a cut of length index, so lets figure out how much that piece is worth!

```
prices[index]
```

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



We made a cut of length index, so lets figure out how much that piece is worth!

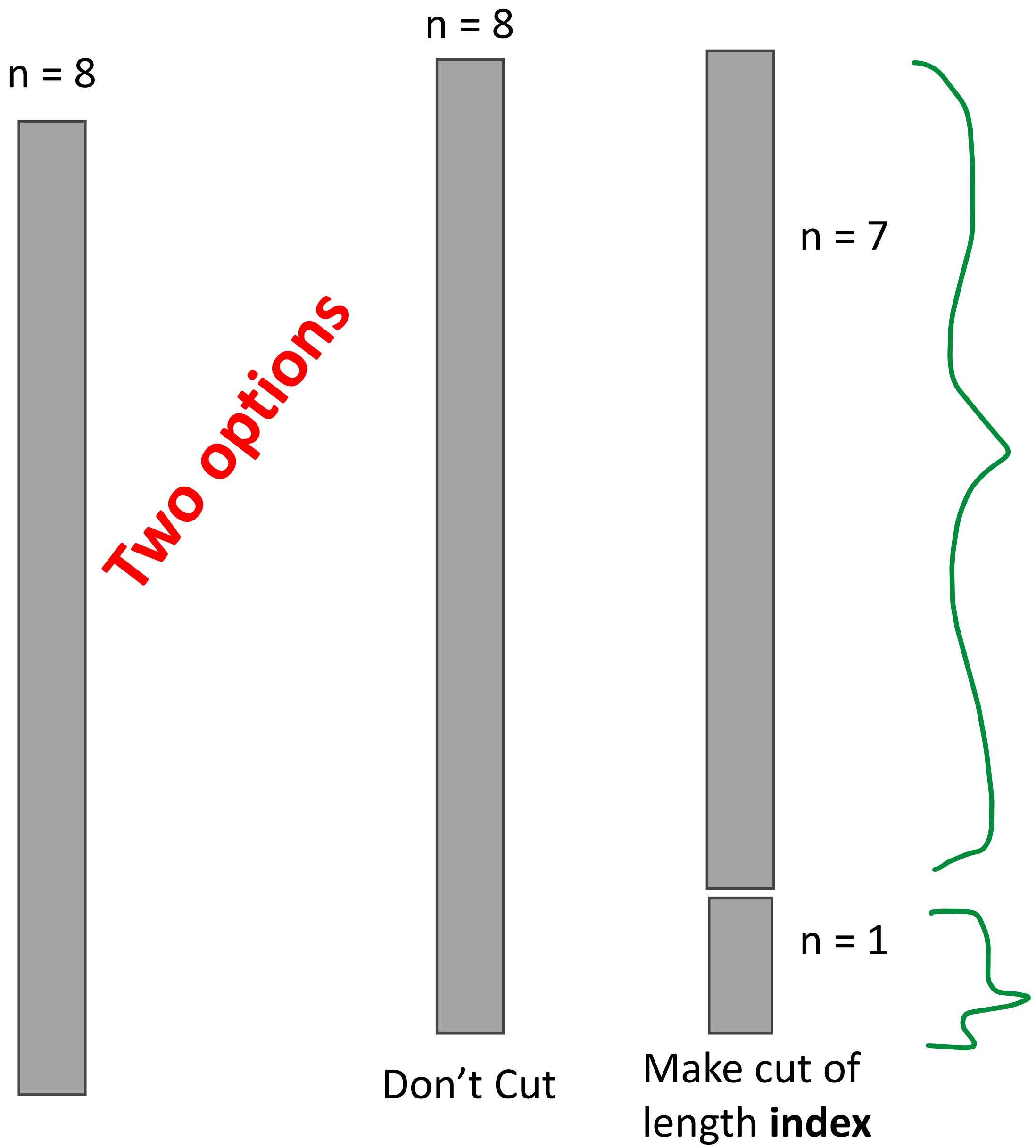
```
prices[index]
```

We have 1 inch of rod left, so we need to now figure out the optimal way to cut this

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



We made a cut of length index, so lets figure out how much that piece is worth!

```
prices[index]
```

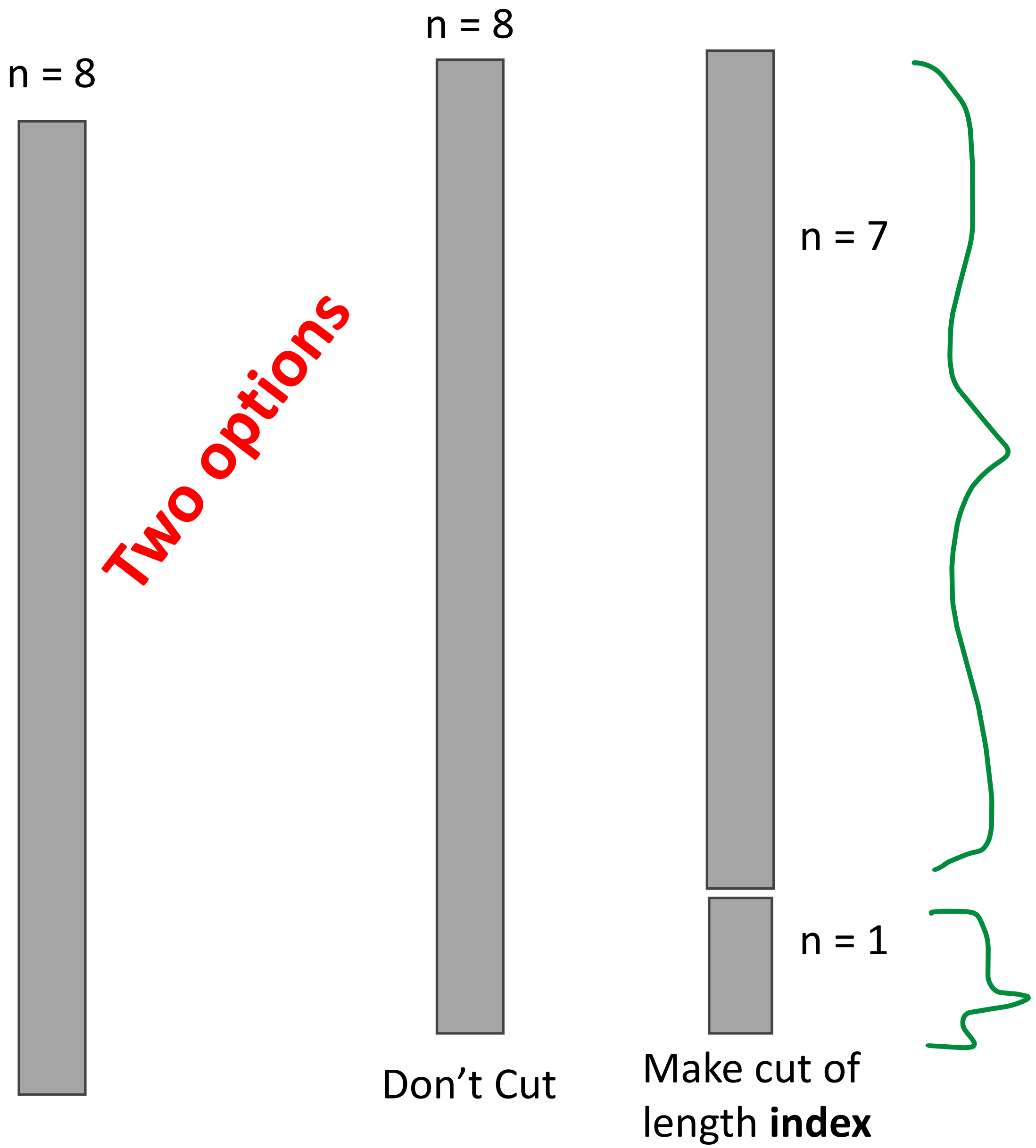
Length of cut made = (index + 1)

We have 1 inch of rod left, so we need to now figure out the optimal way to cut this --Recurse!

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



We made a cut of length index, so lets figure out how much that piece is worth!

`prices[index]`

Length of cut made = (index + 1)

New subproblem = `n- length_of_cut`

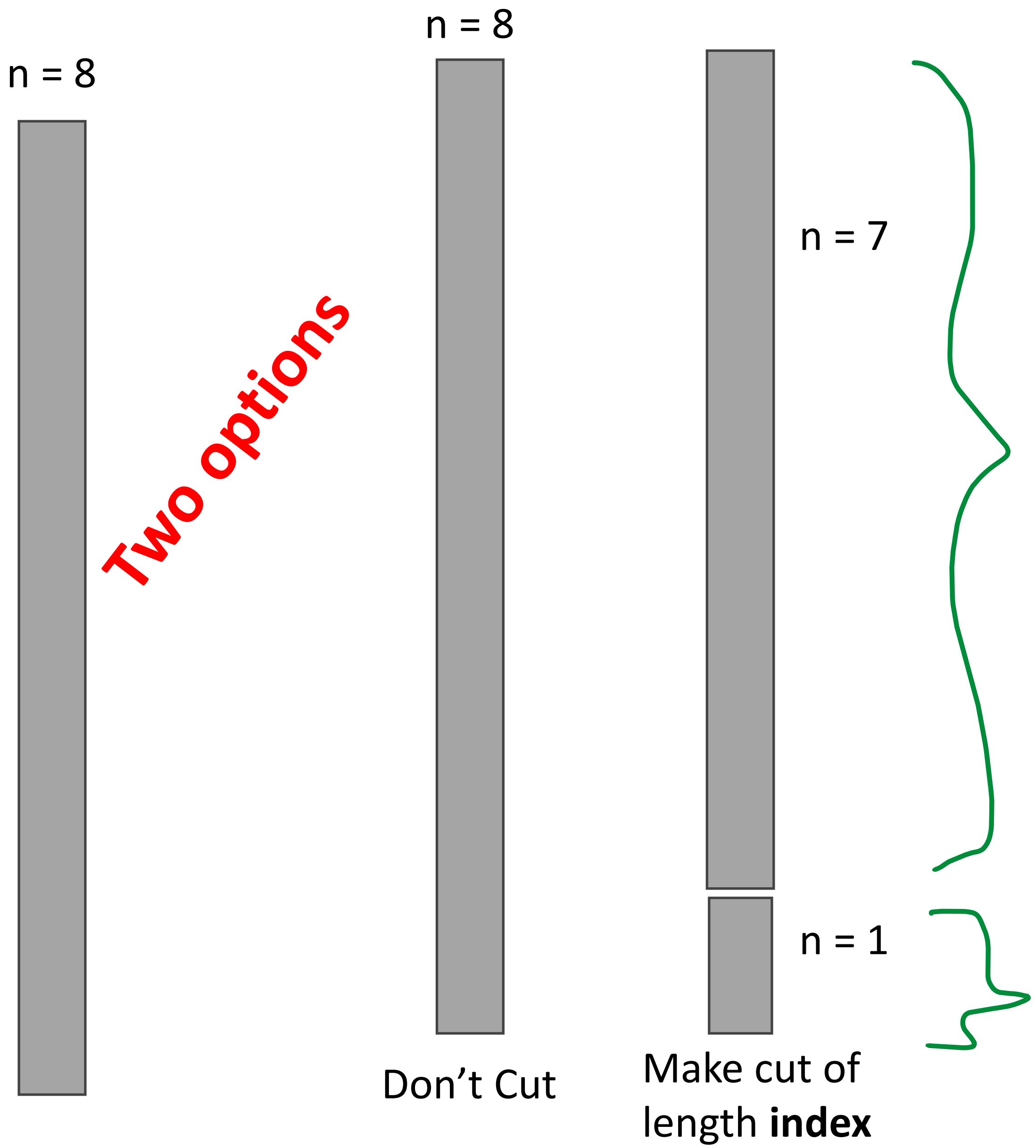
We have 1 inch of rod left, so we need to now figure out the optimal way to cut this

--Recurse!

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



We made a cut of length index, so lets figure out how much that piece is worth!

`prices[index]`

Length of cut made = (index + 1)

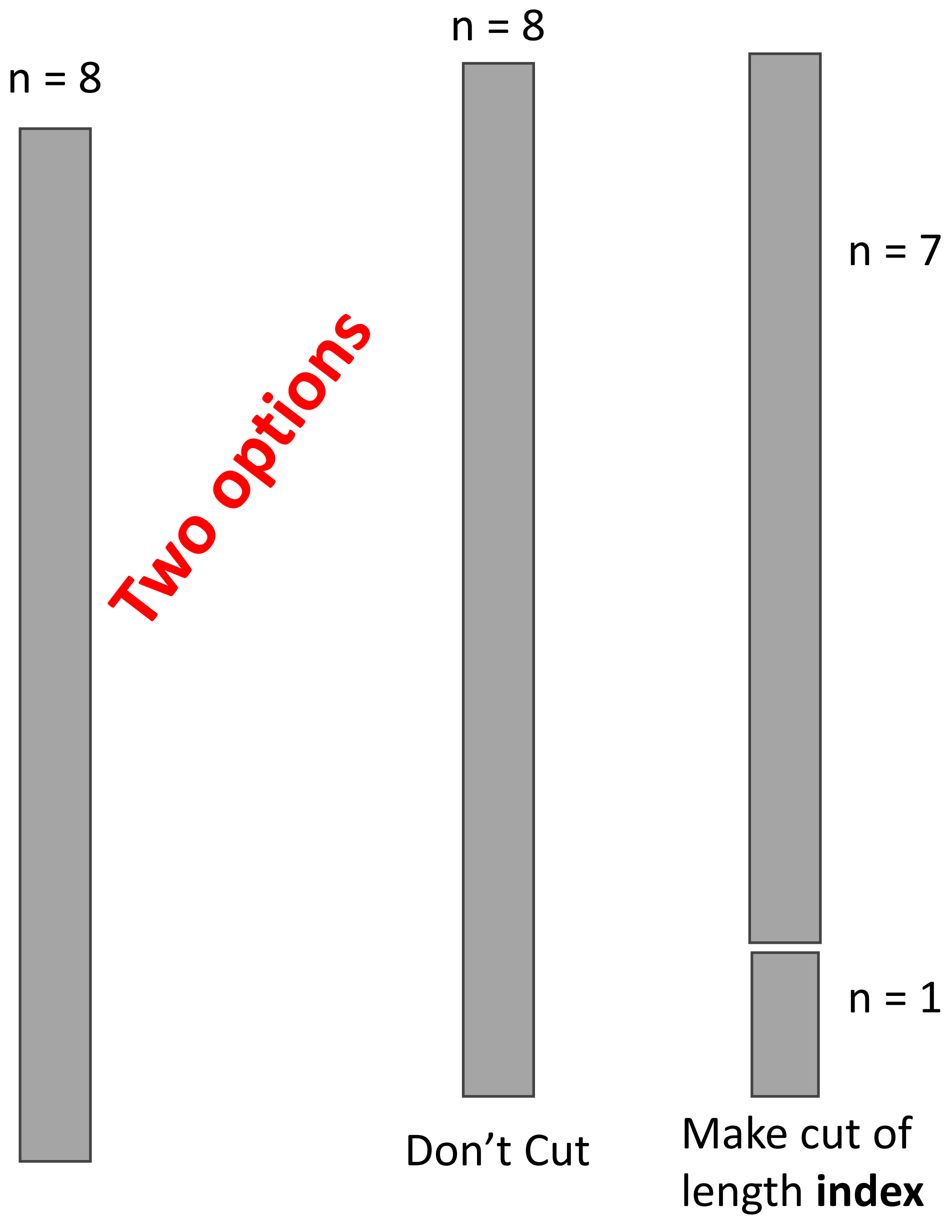
New subproblem = `n- length_of_cut`

We have 1 inch of rod left, so we need to now figure out the optimal way to cut this --Recurse!

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



n = 8

Two options

n = 8

Don't Cut

n = 7

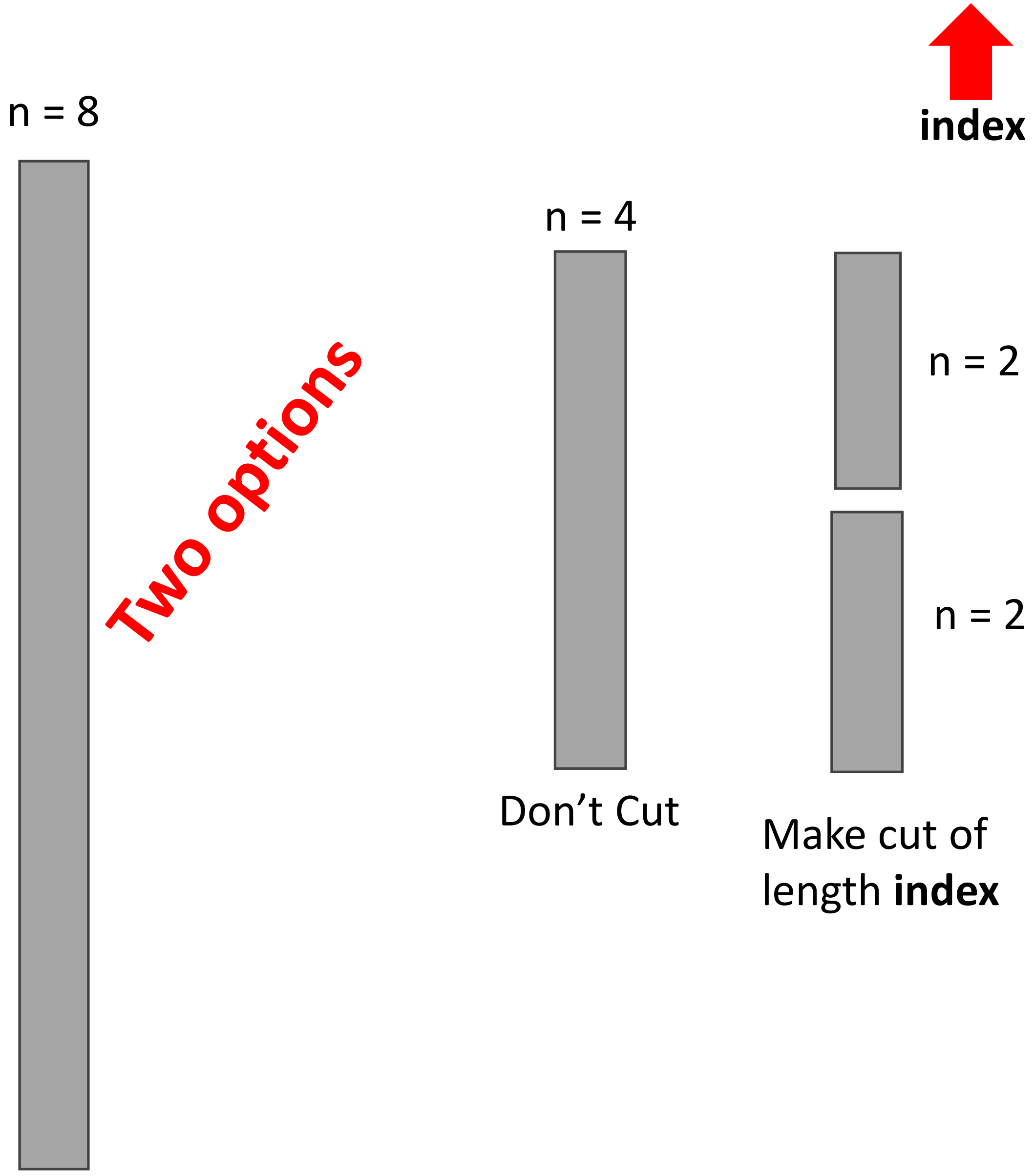
n = 1

Make cut of length **index**

Whenever we make the cut, we adjust the size of the rod, but keep the same index

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 4

Don’t Cut

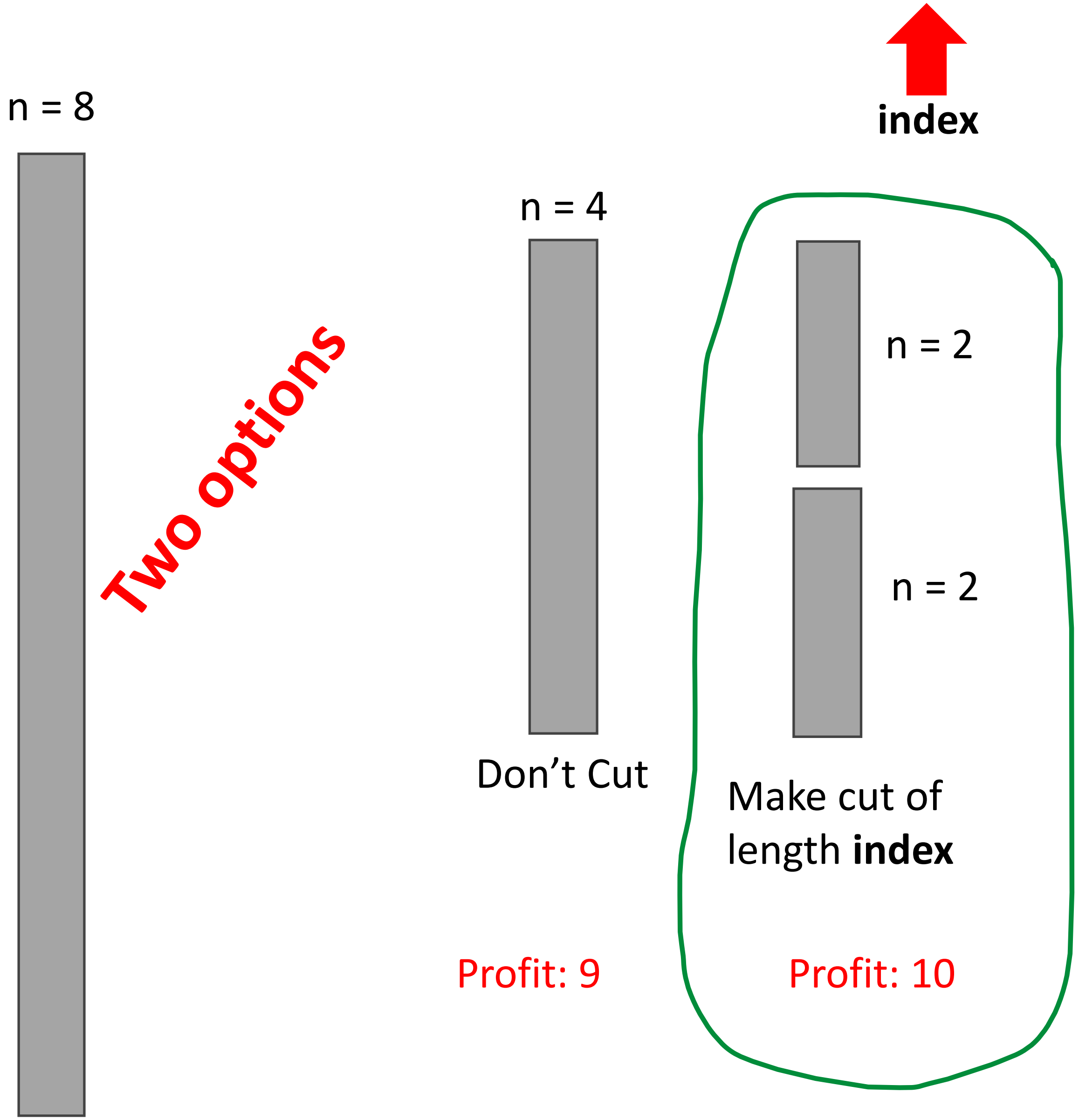
n = 2

n = 2

Make cut of length **index**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 4

Don't Cut

Profit: 9

n = 2

n = 2

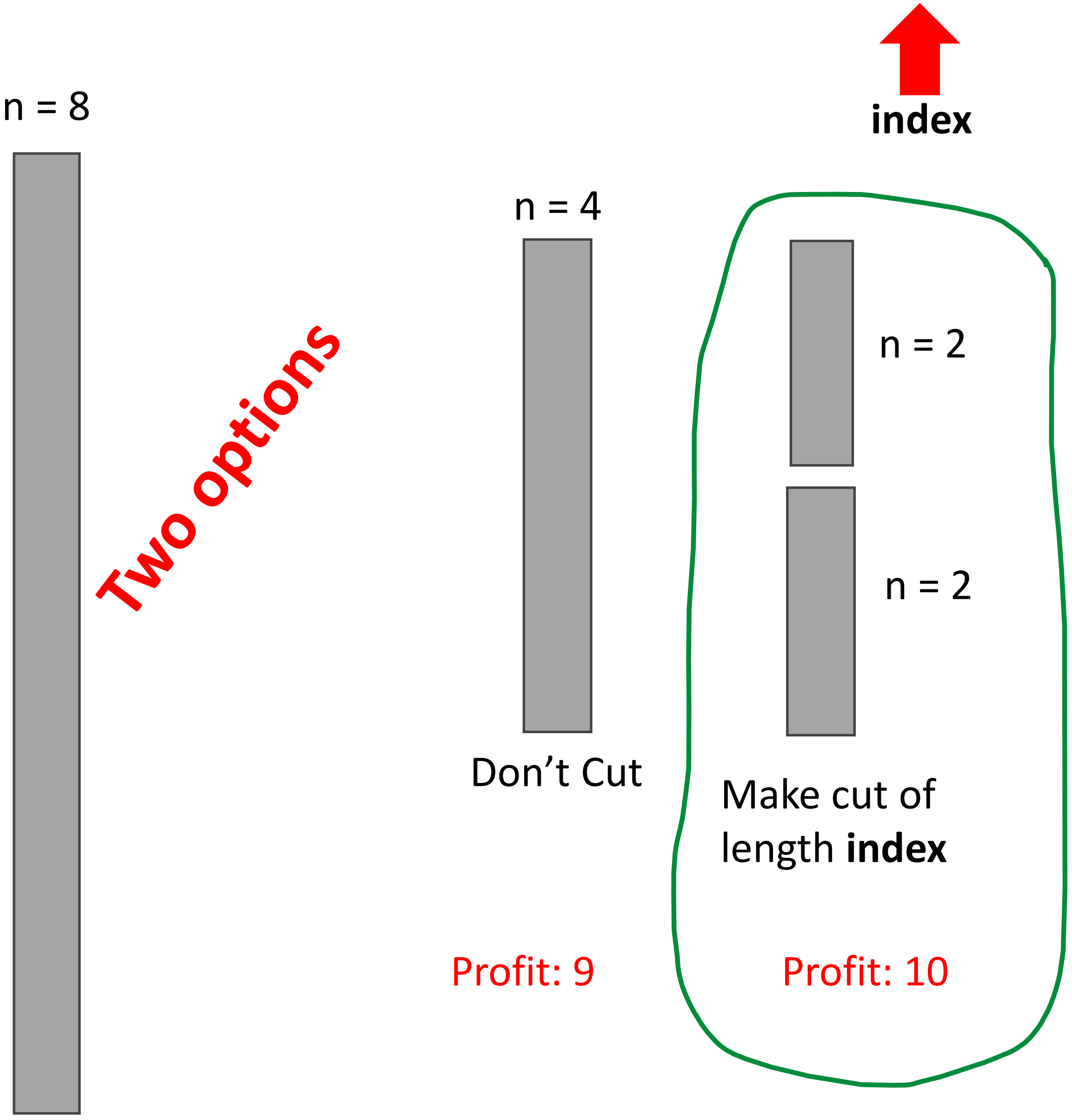
Make cut of length **index**

Profit: 10

Given a rod of length 4 and a potential cut value of length 2, the optimal solution is to **make the cut**

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 4

Don't Cut

Profit: 9

n = 2

n = 2

Make cut of length **index**

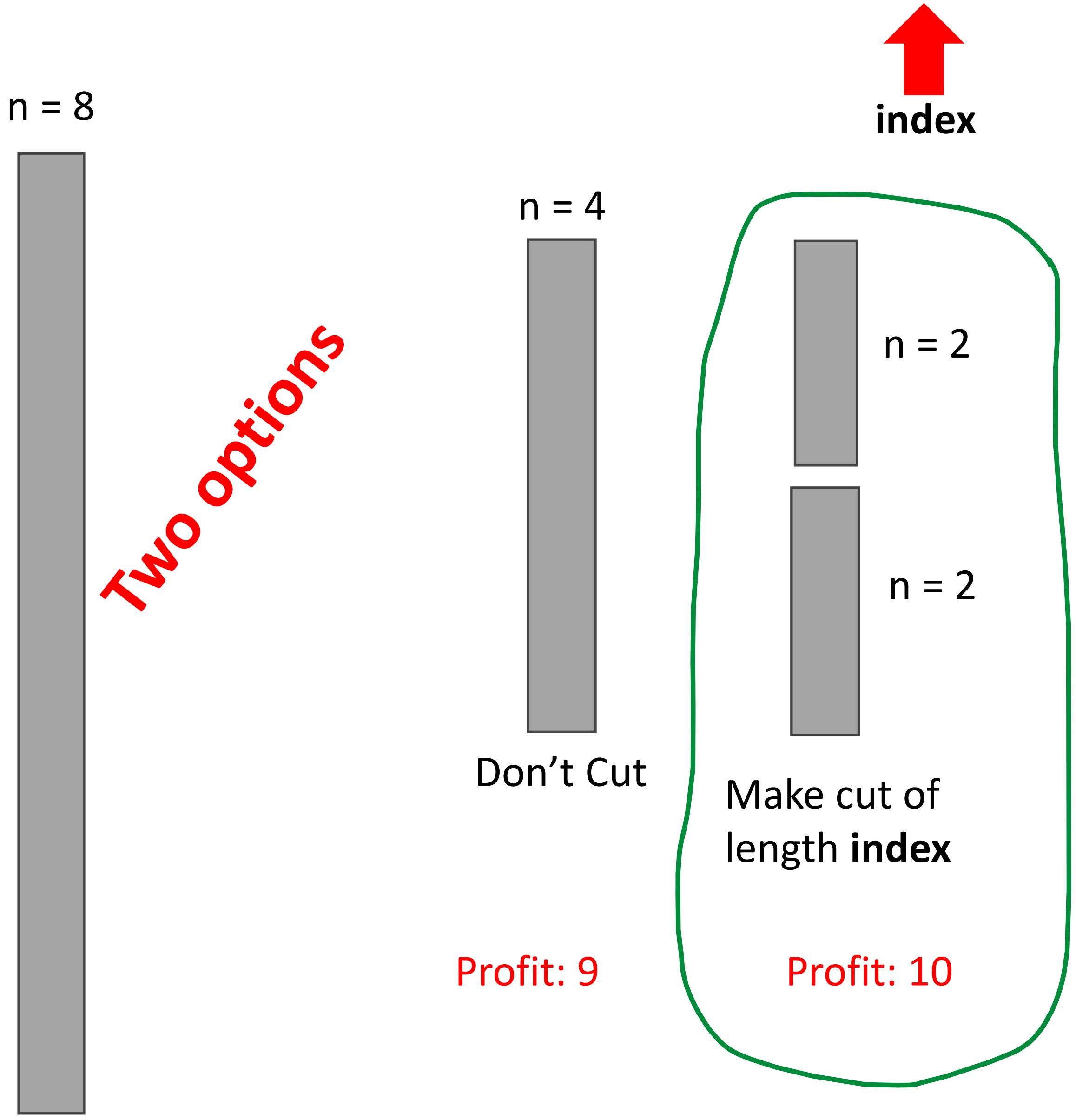
Profit: 10

Given a rod of length 4 and a potential cut value of length 2, the optimal solution is to **make the cut**

If we ever encounter this same subproblem again, we want to make sure we don't recompute it

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20



index

n = 8

Two options

n = 4

Don't Cut

Profit: 9

n = 2

n = 2

Make cut of length **index**

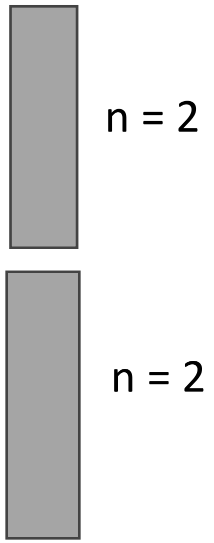
Profit: 10

We need to put this solution (10) into our memorization table

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

↑
index



Make cut of length **index**

Profit: 10

Rod Length

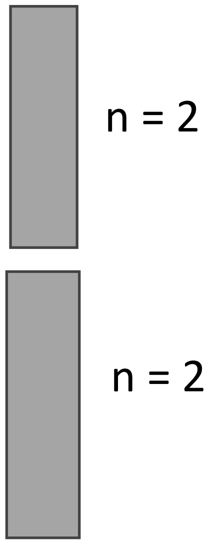
Cut Length

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



Make cut of length **index**

Profit: 10

Rod Length

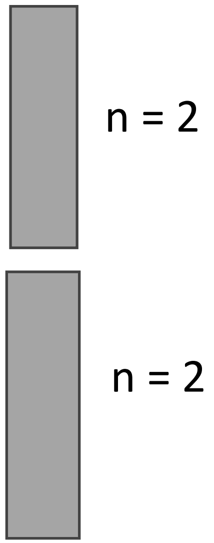
Cut Length

	1	2	3	4	5	6	7	8
1								
2				🟥				
3								
4								
5								
6								
7								
8								

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



Make cut of length **index**

Profit: 10

`dp[index][n] = 10`

Rod Length

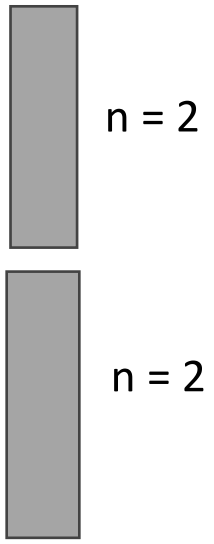
Cut Length

	1	2	3	4	5	6	7	8
1								
2				10				
3								
4								
5								
6								
7								
8								

Rod Cutting

	0	1	2	3	4	5	6	7
Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

index



Make cut of length **index**

Profit: 10

```
dp[index][n] = 10
```

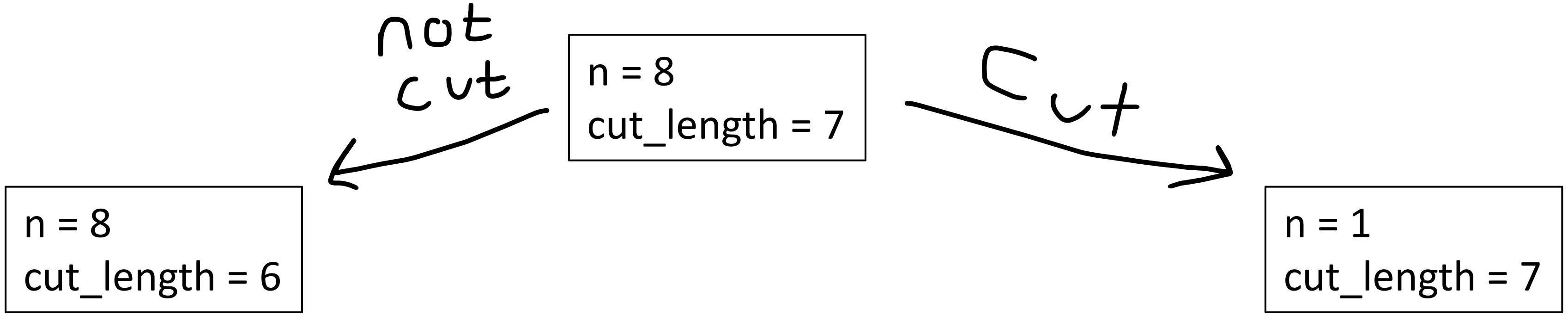
Rod Length

Cut Length

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1								
2				10				
3								
4								
5								
6								
7								
8								

Whenever we solve a subproblem, remember to place it inside of our memoization table

Rod Cutting

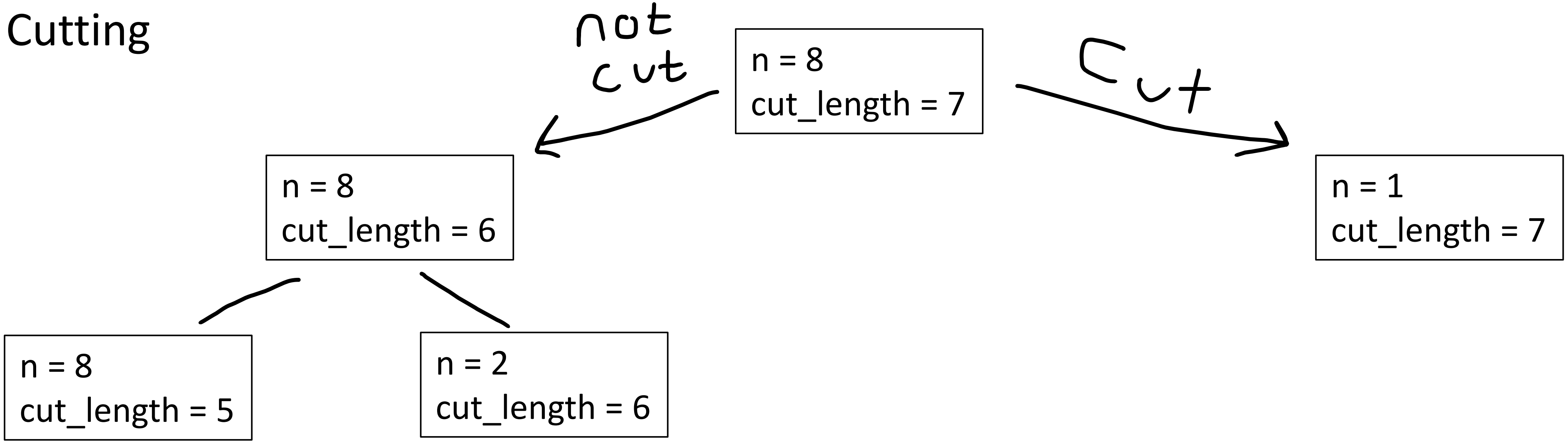


n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

Rod Cutting



n = 8
cut_length = 7

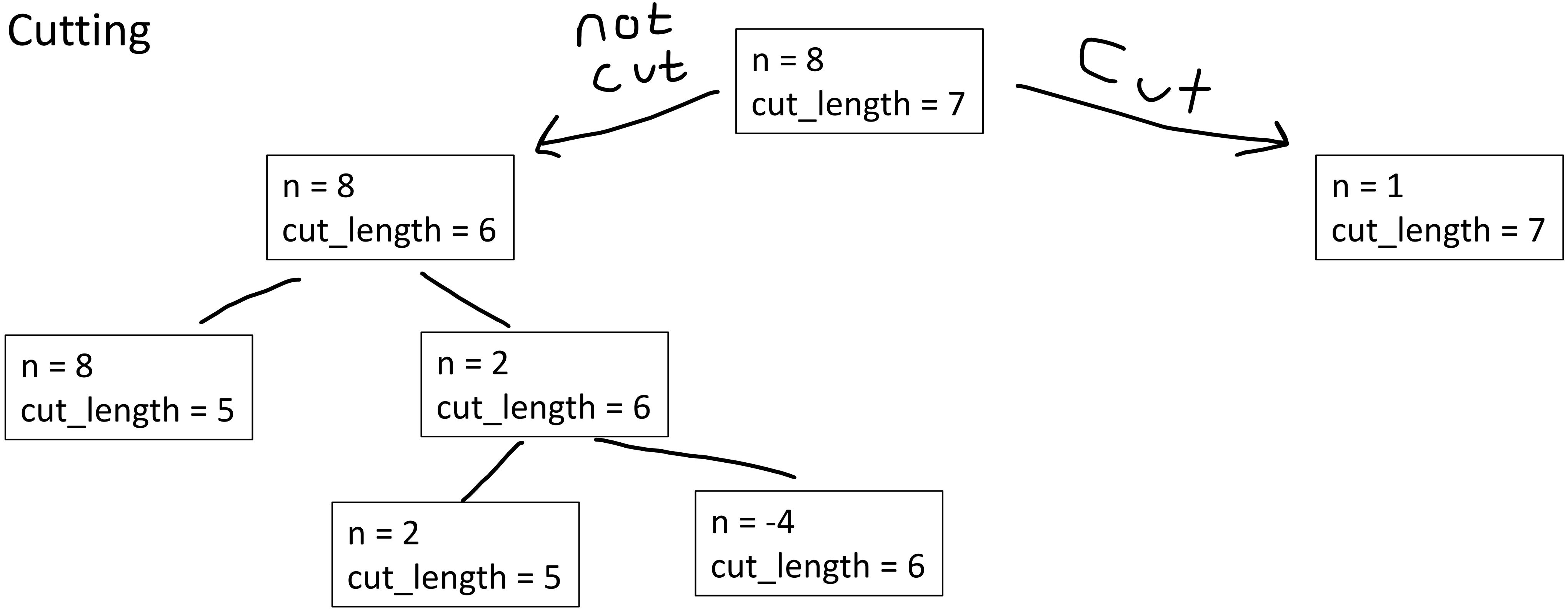
n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

n = 2
cut_length = 6

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

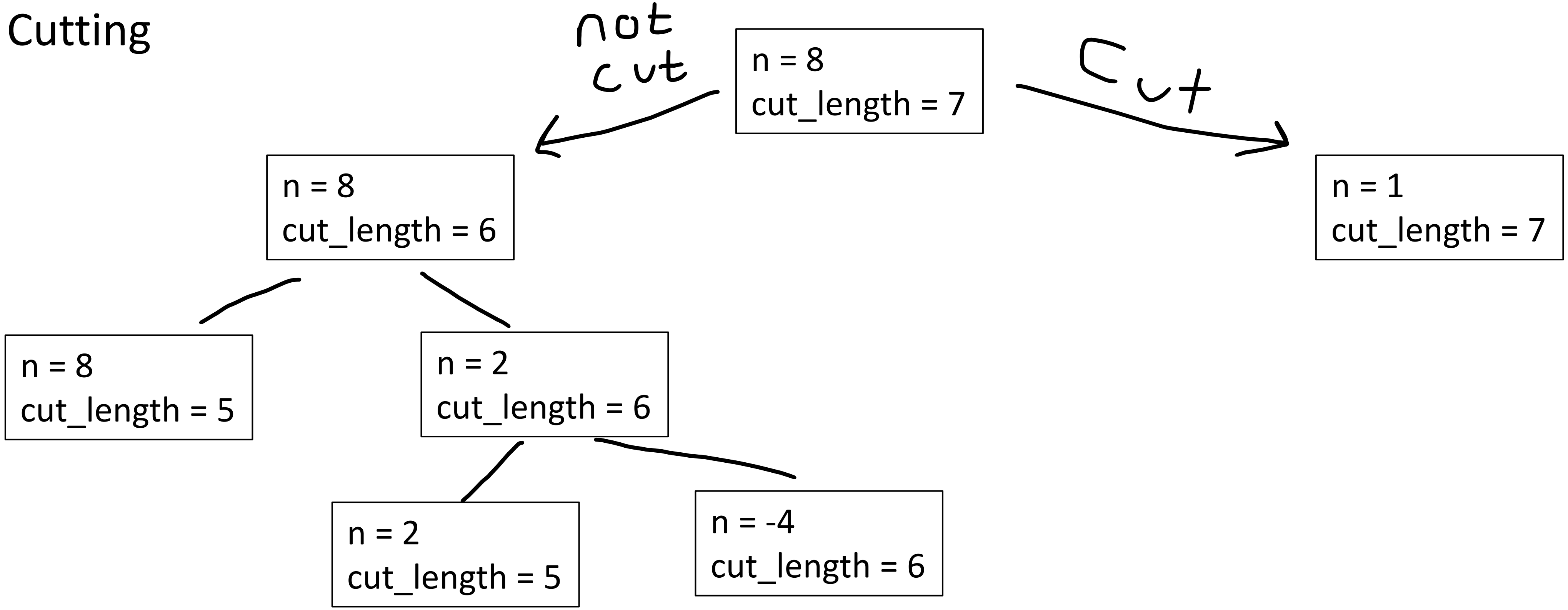
n = 8
cut_length = 5

n = 2
cut_length = 6

n = 2
cut_length = 5

n = -4
cut_length = 6

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

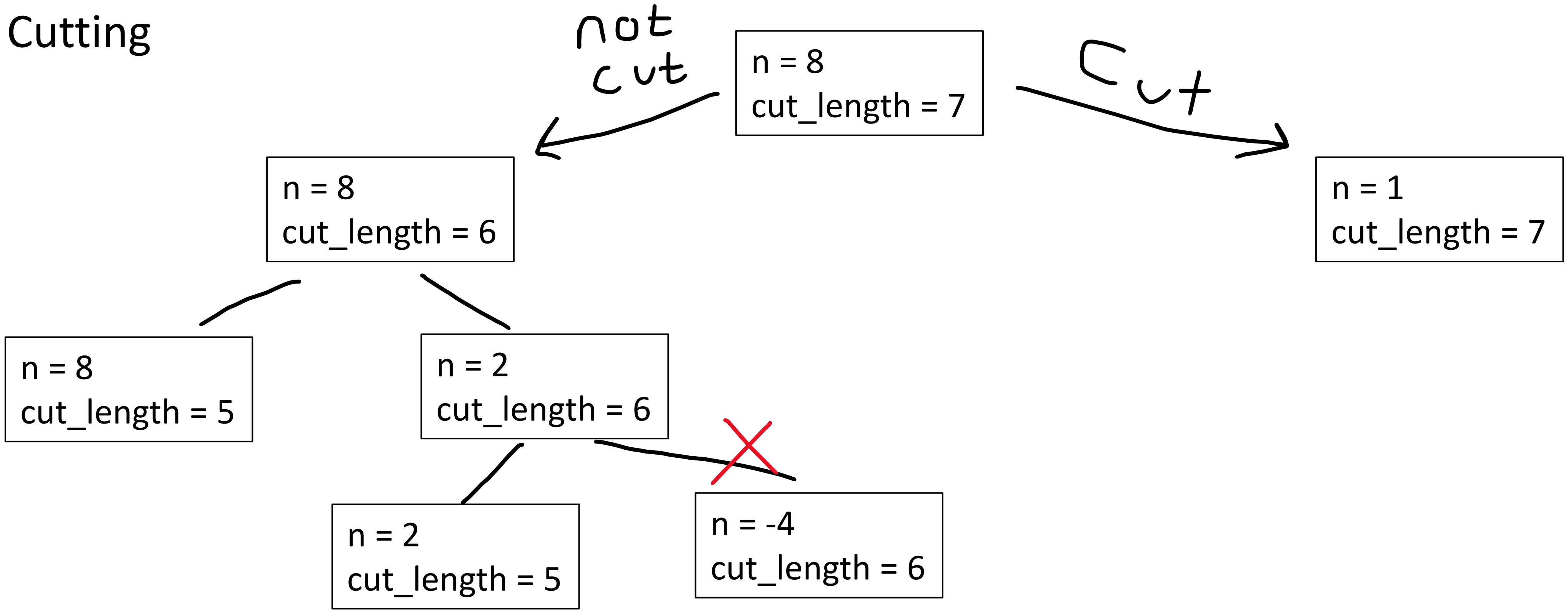
n = 8
cut_length = 5

n = 2
cut_length = 6

n = 2
cut_length = 5

n = -4
cut_length = 6

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

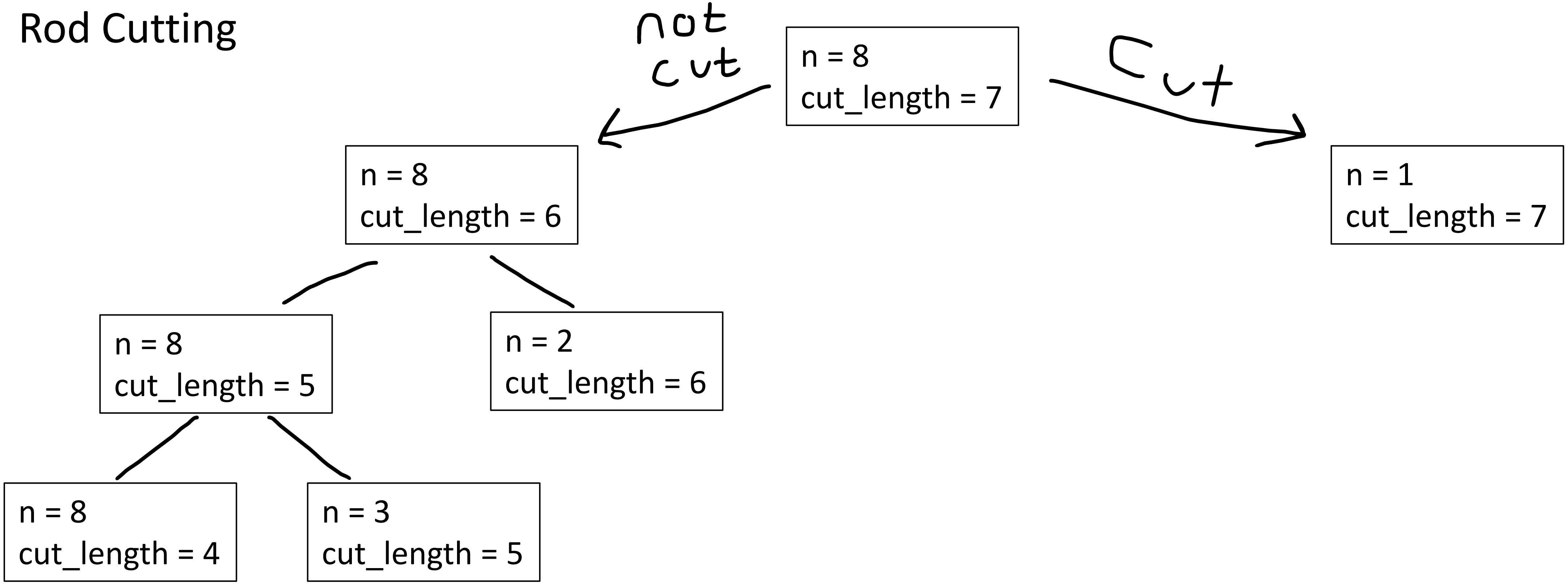
n = 2
cut_length = 6

n = 2
cut_length = 5

n = -4
cut_length = 6

Only make the cut if its possible

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

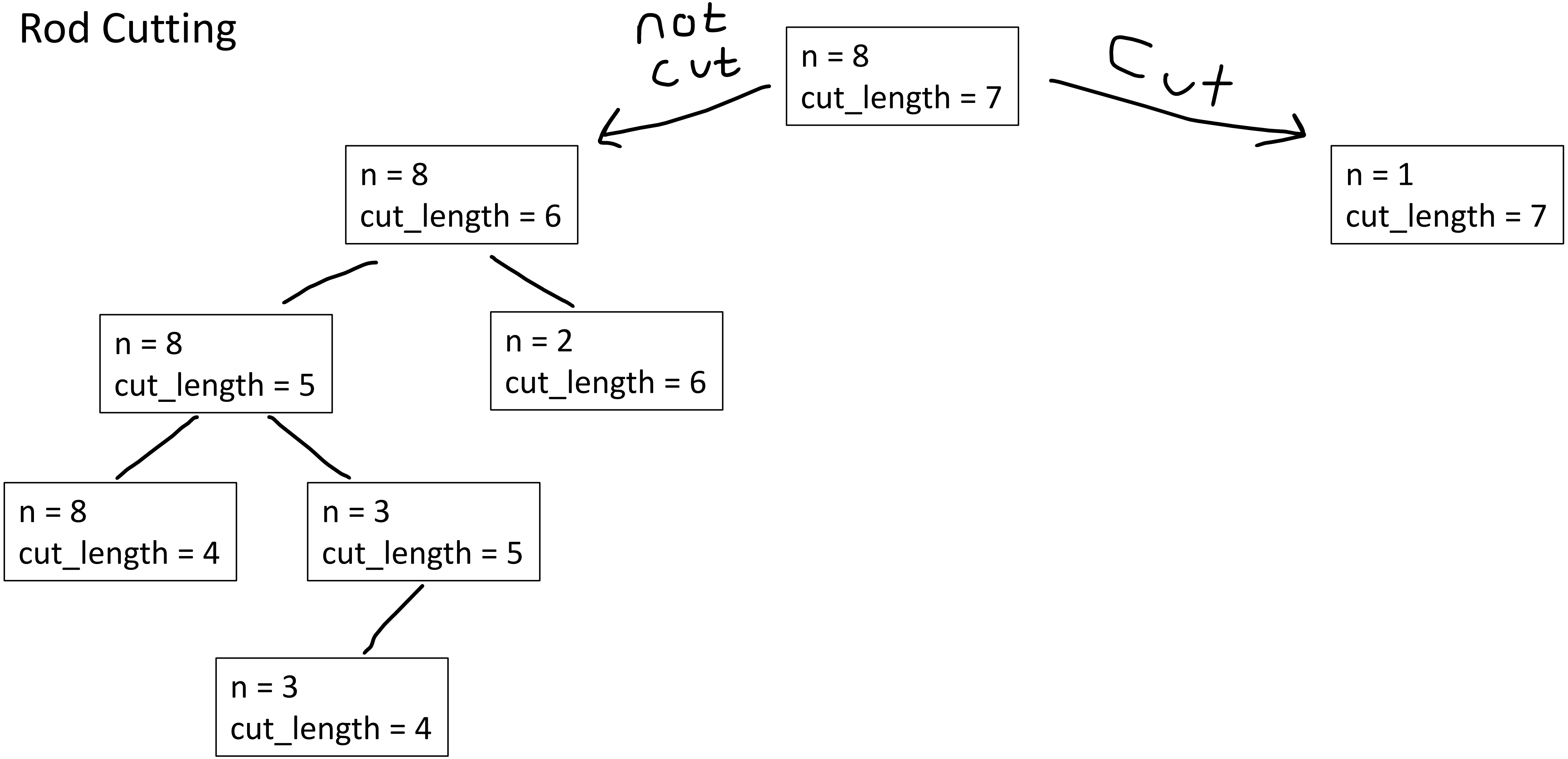
n = 8
cut_length = 5

n = 2
cut_length = 6

n = 8
cut_length = 4

n = 3
cut_length = 5

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

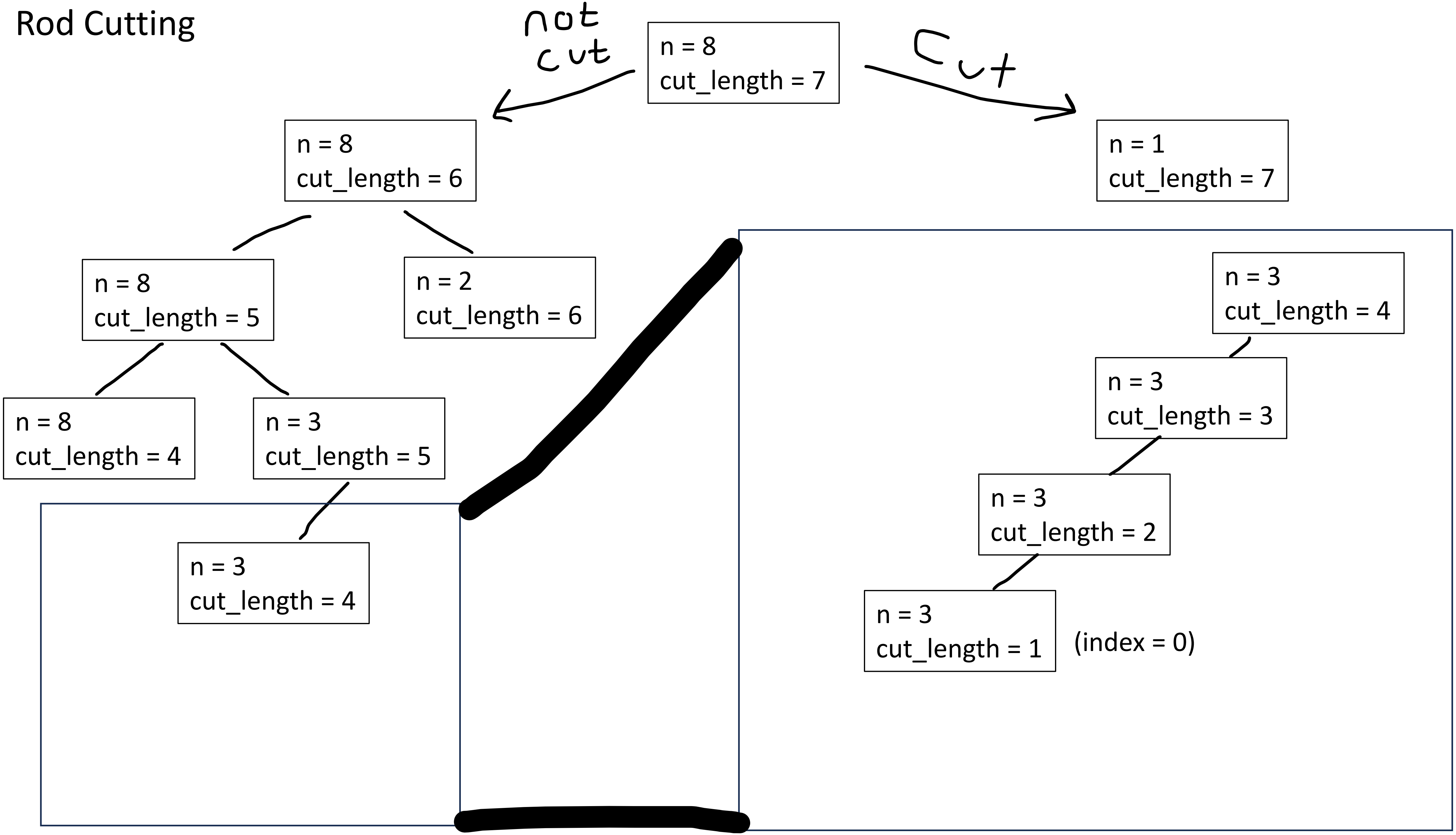
n = 2
cut_length = 6

n = 8
cut_length = 4

n = 3
cut_length = 5

n = 3
cut_length = 4

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

n = 2
cut_length = 6

n = 3
cut_length = 4

n = 8
cut_length = 4

n = 3
cut_length = 5

n = 3
cut_length = 3

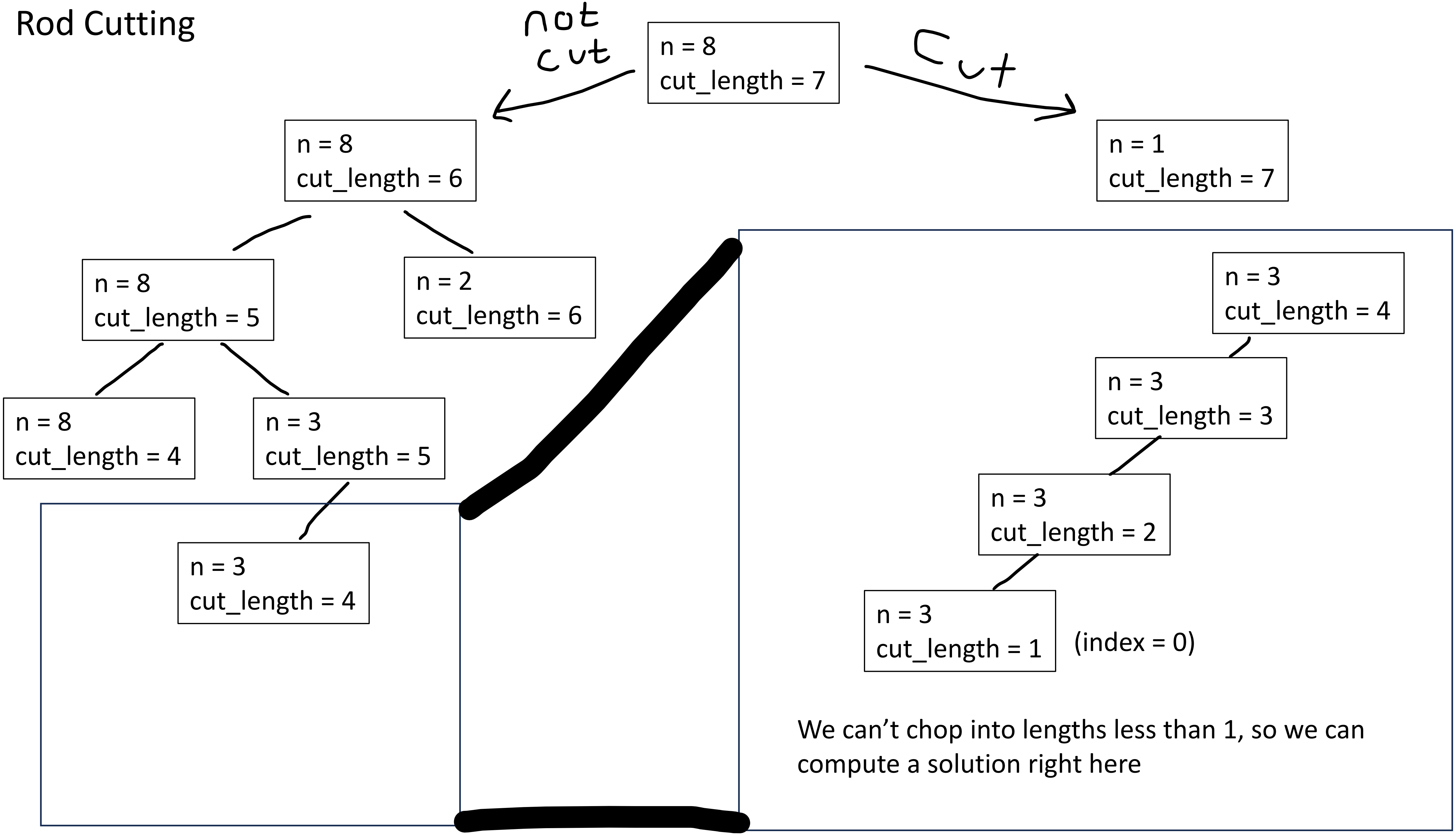
n = 3
cut_length = 2

n = 3
cut_length = 4

n = 3
cut_length = 1

(index = 0)

Rod Cutting



not cut

Cut

n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

n = 2
cut_length = 6

n = 8
cut_length = 4

n = 3
cut_length = 5

n = 3
cut_length = 4

n = 3
cut_length = 4

n = 3
cut_length = 3

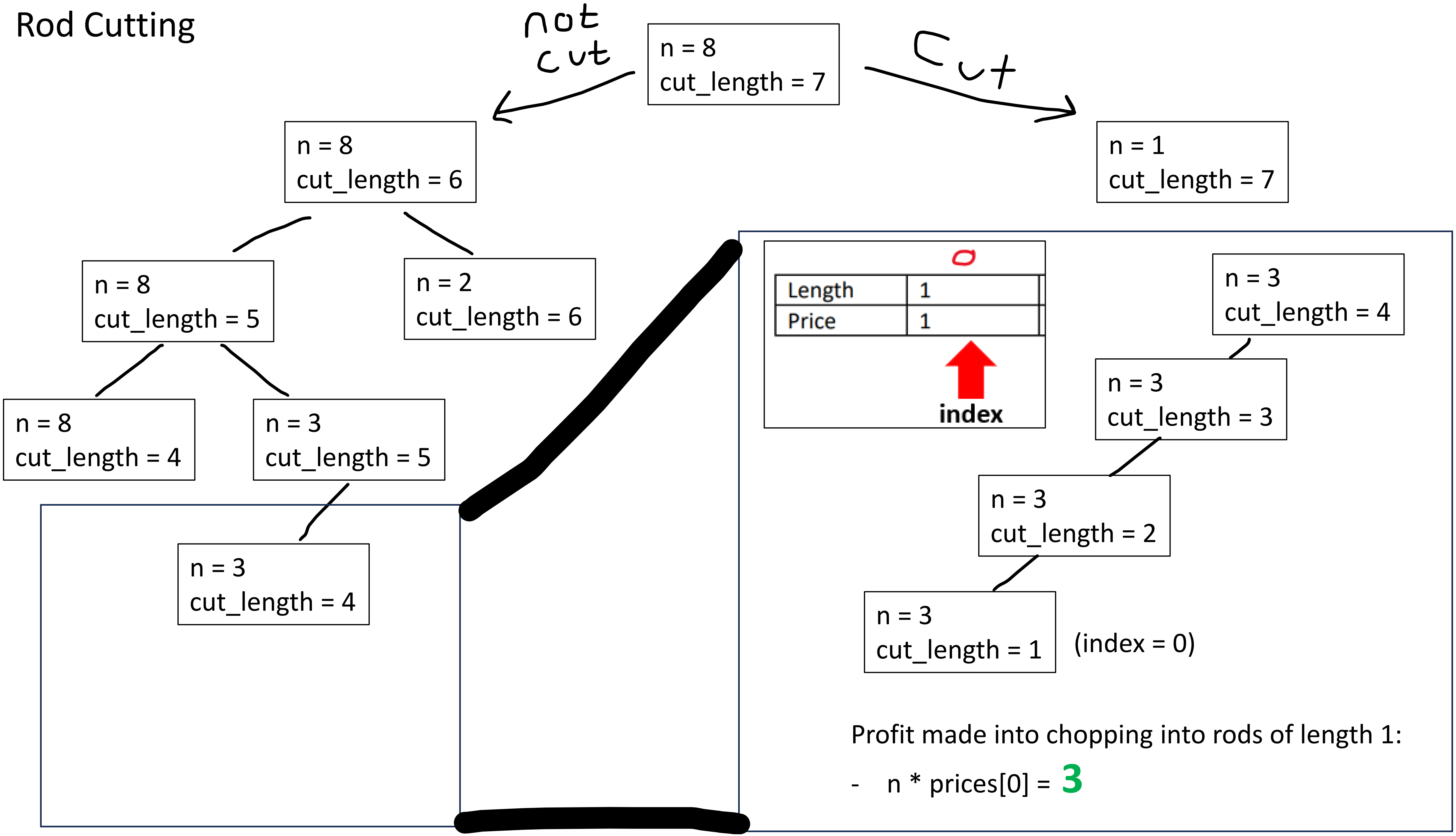
n = 3
cut_length = 2

n = 3
cut_length = 1

(index = 0)

We can’t chop into lengths less than 1, so we can compute a solution right here

Rod Cutting

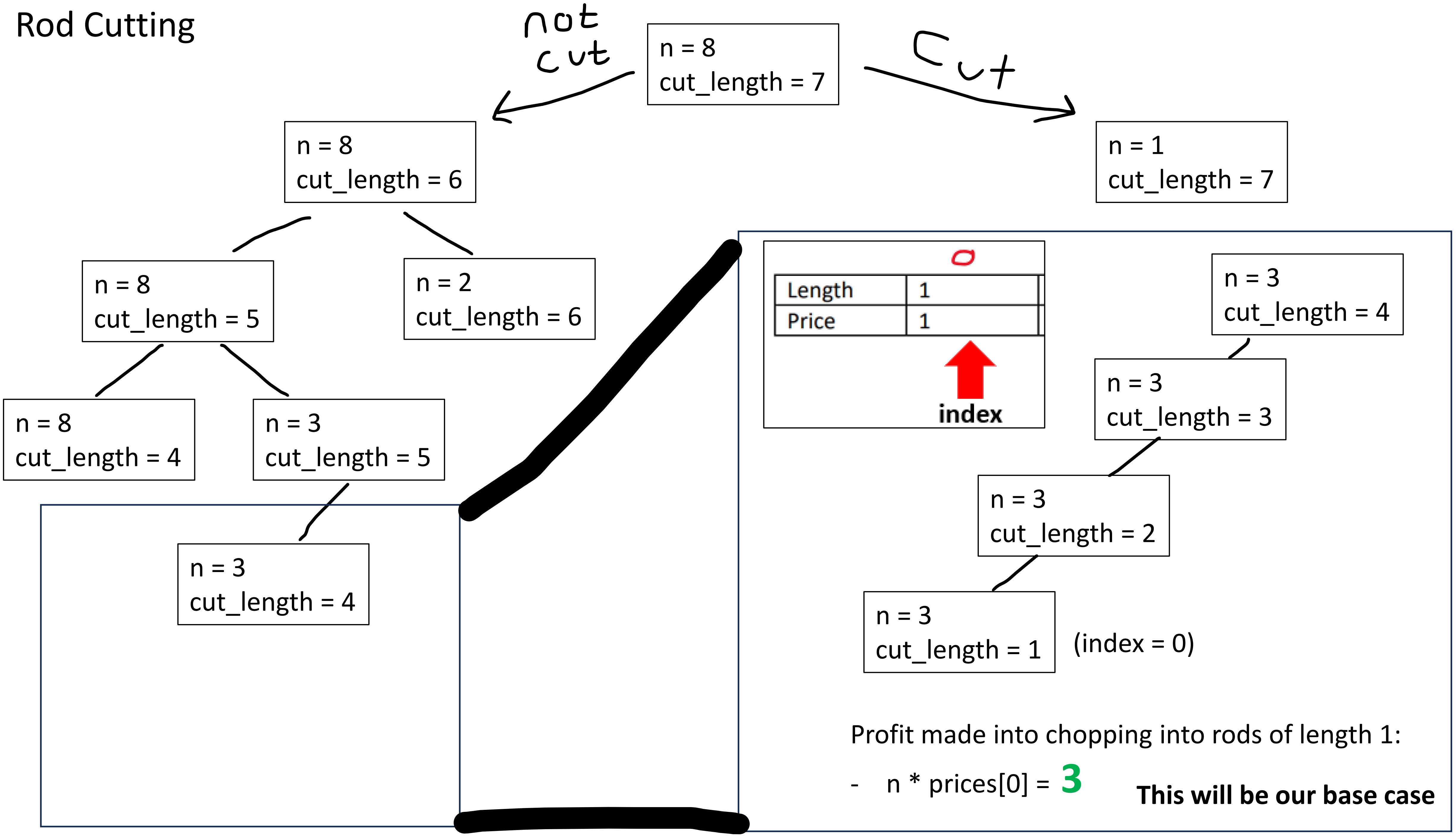


Length	1
Price	1

Profit made into chopping into rods of length 1:

- n * prices[0] = **3**

Rod Cutting



n = 8
cut_length = 7

n = 8
cut_length = 6

n = 1
cut_length = 7

n = 8
cut_length = 5

n = 2
cut_length = 6

n = 8
cut_length = 4

n = 3
cut_length = 5

n = 3
cut_length = 4

Length	1
Price	1

n = 3
cut_length = 4

n = 3
cut_length = 3

n = 3
cut_length = 2

n = 3
cut_length = 1

(index = 0)

Profit made into chopping into rods of length 1:

- n * prices[0] = **3**

This will be our base case

Rod

n = 8
cut_l

LETS TRY TO CODE THIS

If you are confused are the recursion is set up, don’t stress out about it. Its not a big deal.

The goal here is to show how we are using dynamic programming to solve this problem

Thank You!

This class has been very enjoyable to teach. Thank you for deciding to spend part of your Summer with me, and for making this a great experience. I appreciate your kindness, patience, and flexibility.

Things weren't perfect and there were things I wish I did differently, but overall, I am happy with how things turned out.

I hope you enjoyed this class and learned at least *something*. You can now move onto the fun 300 and 400 level CS classes

If I can be of assistance to you for anything in the future (reference, advising, support), please let me know!

Enjoy the rest of your summer!

Connect with me on LinkedIn!

Reese Pearsall (He/Him)
Instructor at Montana State University
Bozeman, Montana, United States · Contact info

I am teaching CSCI 466 (Networks) and CSCI 476 (Computer Security)* in the fall. Now that you have completed 232, you are eligible to take those classes ☺

*you might also need to take programming in C (CSCI 112)