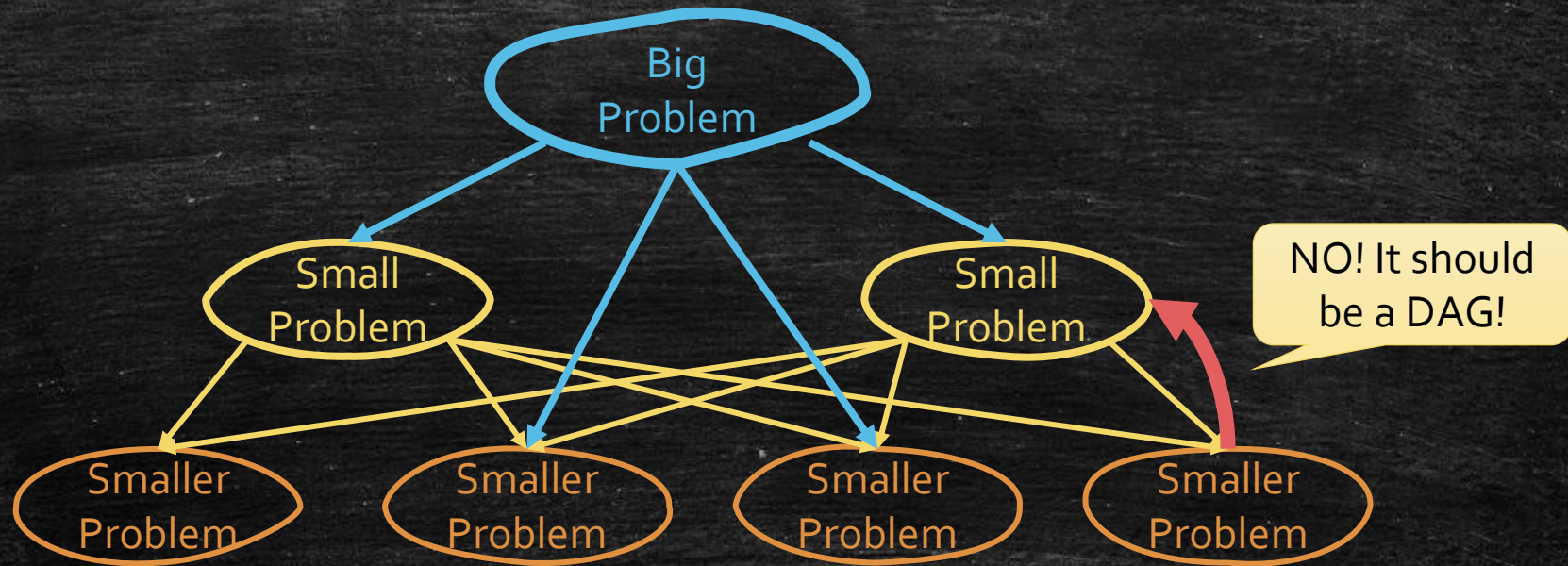


Dynamic Programming

DP improvement

Dynamic Programming



A simpler guideline

- Find subproblems.
- Check whether we are in a **DAG** and find the **topological order** of this DAG. (Usually, by hand.)
- Solve & store the subproblems by the topological order.

Recap the three examples

- Longest Increasing Sequence
 - Subproblem $LIS[i]$: the longest increasing sequence ended by a_i .
- Edit Distance
 - Subproblem $ED[i, j]$: the edit distance for $A[1..i]$ and $B[1..j]$.
- Knapsack
 - Subproblem $f[i, w]$: the maximum value we can get by using first i items and w budget.

How to find these subproblems

- Think from a recursive method
- LIS:
 - We want to find the LIS.
 - It may be ended by every a_i .
 - Solve LIS ended by a_i need to know all LIS ended by $a_{j < i}$.

How to find these subproblems

- Think from a recursive method
- Edit Distance
 - We want to know the Edit Distance.
 - We think how we align the last two character.
 - Different case make us go into different subproblems.
 - We these subproblems can be merged to $ED[i, j]$.

How to find these subproblems

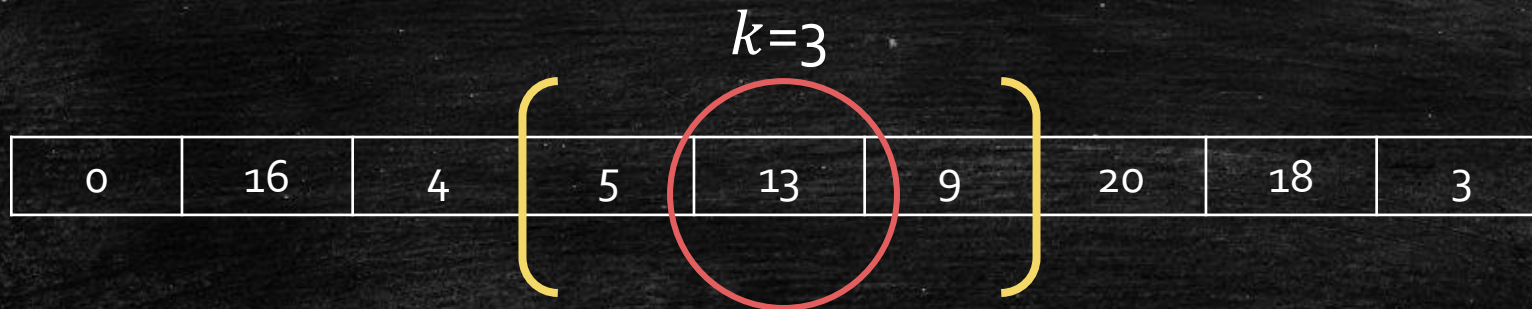
- Think from a recursive method
- Knapsack
 - We want to know the maximum value.
 - We know that for each item, we have two choice: buy it or not.
 - Buy: we have $W - c_i$ budget for other items.
 - Not Buy: we have W budget for other items.
 - Consider we recursive from a_n .
 - Subproblems can be merged to $f[i, w]$.

A Simple but Useful Data Structure.

Priority Queue

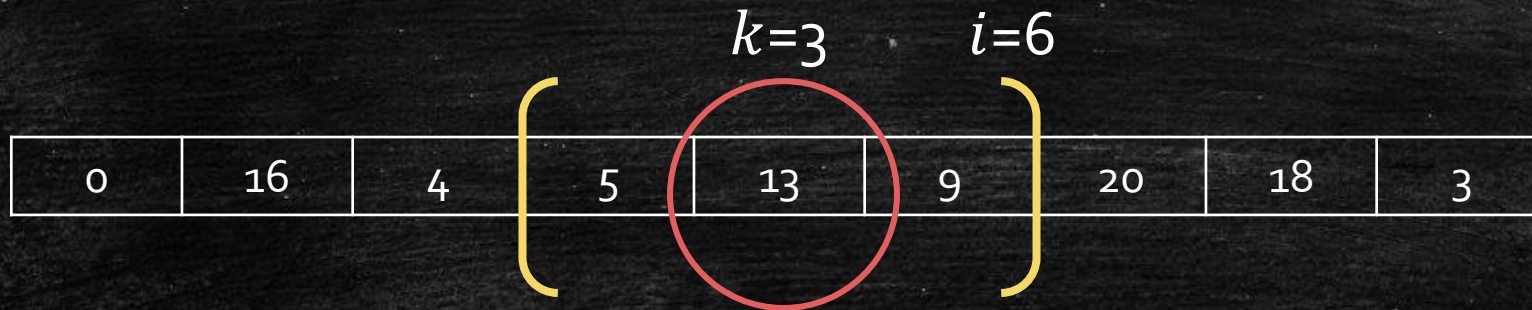
Largest Number in k Consecutive Numbers

- **Input:** A sequence of numbers a_1, a_2, \dots, a_n , and a number k .
- **Output:** The largest number in every k consecutive numbers.



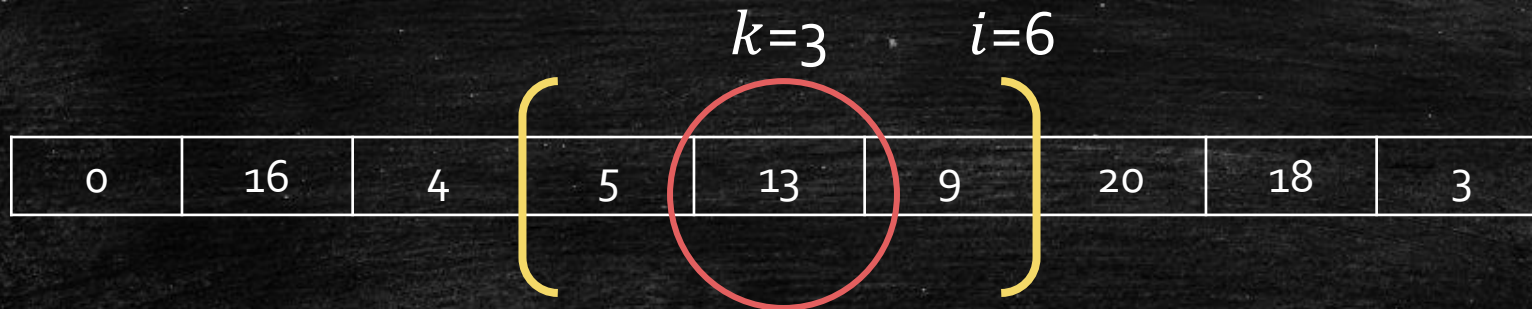
Subproblem Definitions

- $large[i]$: the largest number from a_{i-k+1} to a_i .
- Output: $large[k] \sim large[n]$.



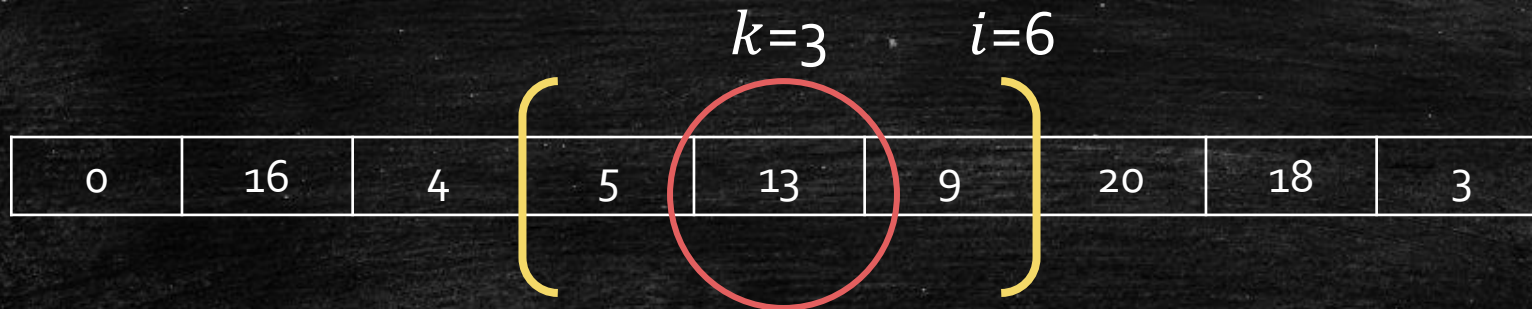
Solving Subproblems

- $large[i]$: the largest number from a_{i-k+1} to a_i .
- Can you find a way to solve $large[i]$ by other subproblems?
 - Brute-force: $large[i] = \max_{j=i-k+1}^i \{a_j\}$.



Solving Subproblems

- $large[i]$: the largest number from a_{i-k+1} to a_i .
- Can you find a way to solve $large[i]$ by other subproblems?
 - Brute-force: $large[i] = \max_{j=i-k+1}^i \{a_j\}$.
 - Tips: from $large[j], j < i$.



Recall Knapsack

- What we always do before:
- $f[i, w]$: the maximum value we can get by using the first i items, and with w budget.
- ~~Use $g[i]$ to store how much budget $f[i]$ uses.~~

$f[i]$	5	10	13	16	21	30	?
--------	---	----	----	----	----	----	---

How to solve $f[i]$ by $f[j < i]$?

We know $f[j]$ but we do not know how much budget it uses!

Key problem: Subproblem definition does not contain enough information!

What kind of information
do we need now?

Observation

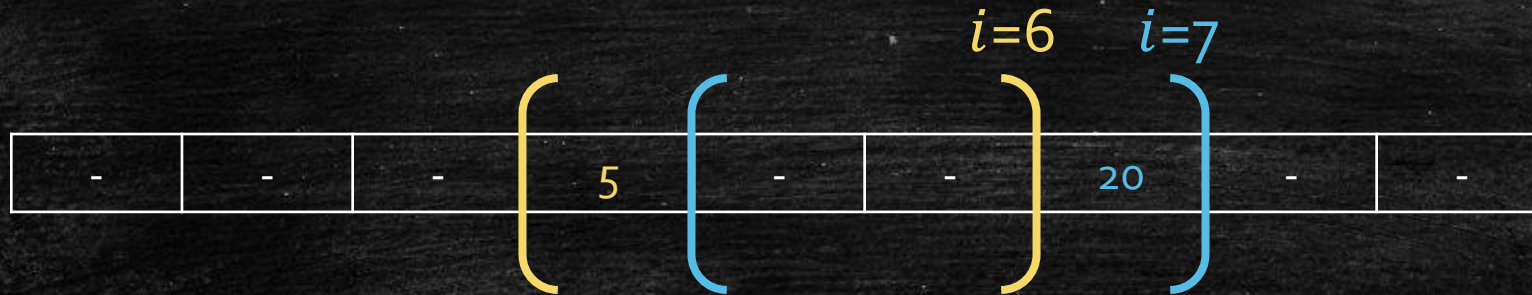
- Compare two $large[i]$ and $large[i - 1]$.
- Difference
 - One entering number: 20
 - One outgoing number: 5
 - Question: how they affect the largest number?



How they affect the largest number

- Difference

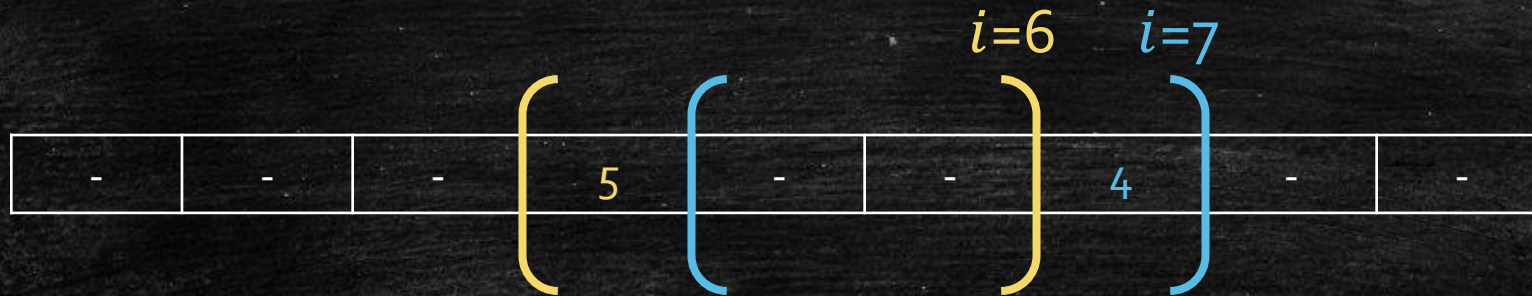
- One entering number: 20
- One leaving number: 5
- Question: how they affect the largest number?
- Case 1: the entering number is the new largest!



How they affect the largest number

- Difference

- One entering number: 20
- One leaving number: 5
- Question: how they affect the largest number?
- Case 2: the leaving number is the previous largest!



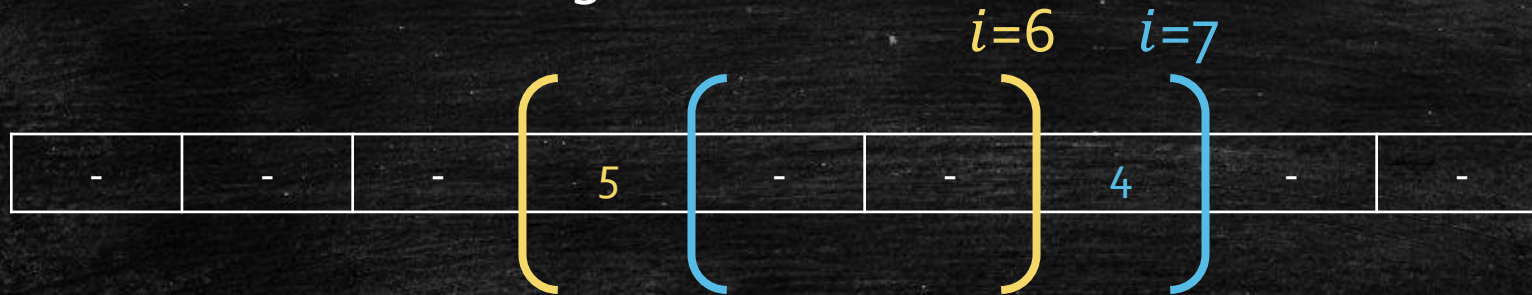
Key problem: We should know what is the previous second largest number.

Ok, let us record it!

How they affect the largest number

- Difference

- One entering number: 20
- One leaving number: 5
- Question: how they affect the largest number?
- Case 3: the leaving number is the previous second largest!
- What is the second largest now?



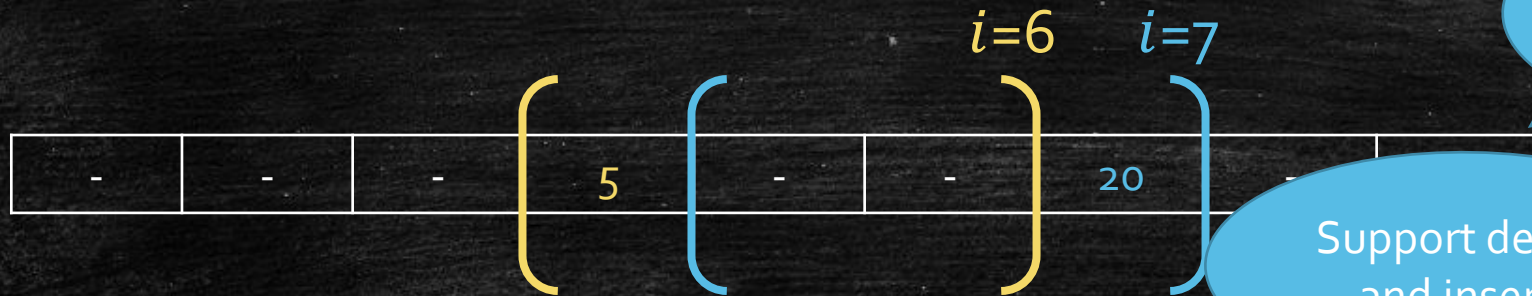
Key problem: We should know what is the previous third largest number.

Ok, let us record it.....

Summarize

- Difference

- One entering number: 20
- One leaving number: 5
- Question: how they affect the largest number?



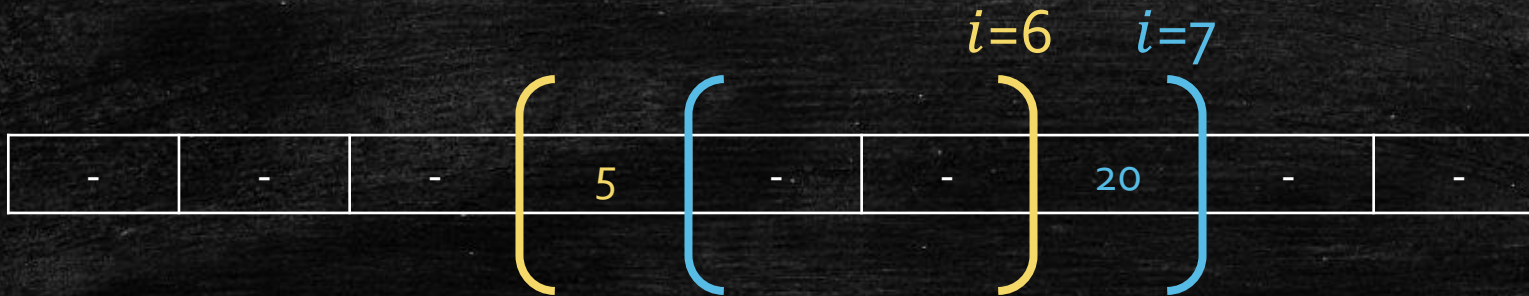
Summarize: We should maintain a data structure!

Support delete and insert!

Heap!
 $O(n \log k)$!

Let us think more!

- New Subproblem: Solving the Heap of $a_{i-k+1} \sim a_i$.
 - Delete (Update & PopMax)
 - Insert
 - FindMax
 - $O(n \log k)$!
- Is Heap too powerful for this problem?
 - We delete and insert only based on the index!



A new Subproblem!

- Think again: why we need the heap?
 - We need to know who is the largest.
 - We need to know who is the **potential largest**.
 - We need to update the **potential largest list**.
- Do we have a better way to maintain this **potential largest list**?
 - Heap views all k numbers as **potential largest**.

Observation

- Who can be the **potential largest** number?

5

13

9

0	16	4	5	13	9	20	18	3
---	----	---	---	----	---	----	----	---

$i=6$

Observation

- Who can be the **potential largest** number?

5

13

9

5 is not a potential largest number because 5 is older than 13 and $5 < 13$.

9 is a potential largest number although $13 > 9$ because 9 is younger.

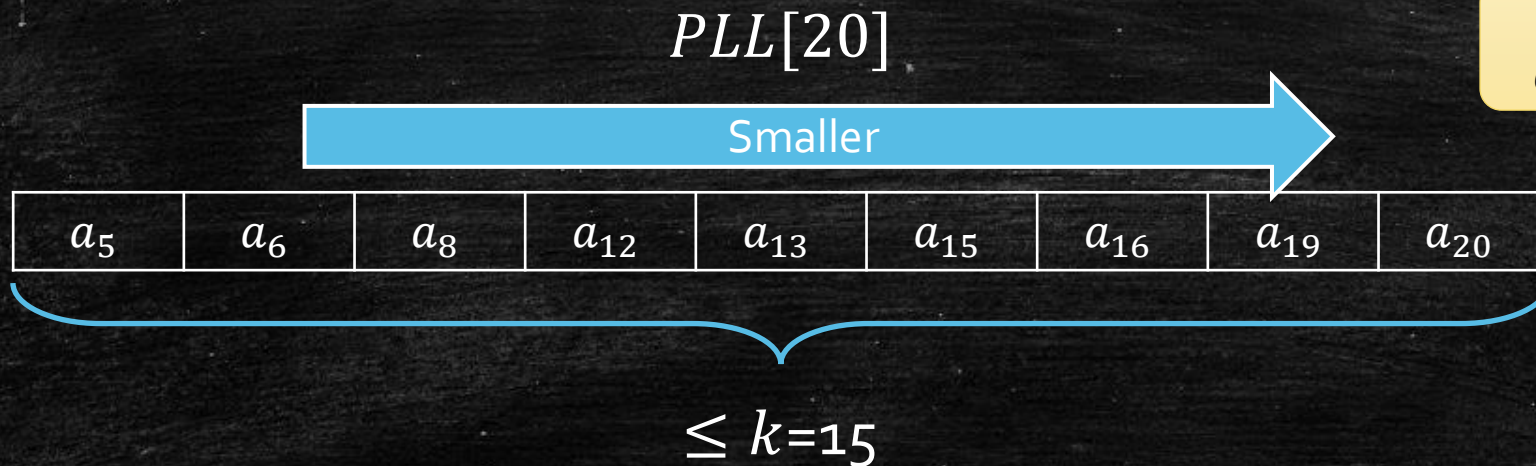
0	16	4	5	13	9	20	18	3
---	----	---	---	----	---	----	----	---

$i=6$

Key Observation: the potential largest list can be smaller than k .

Potential Largest List

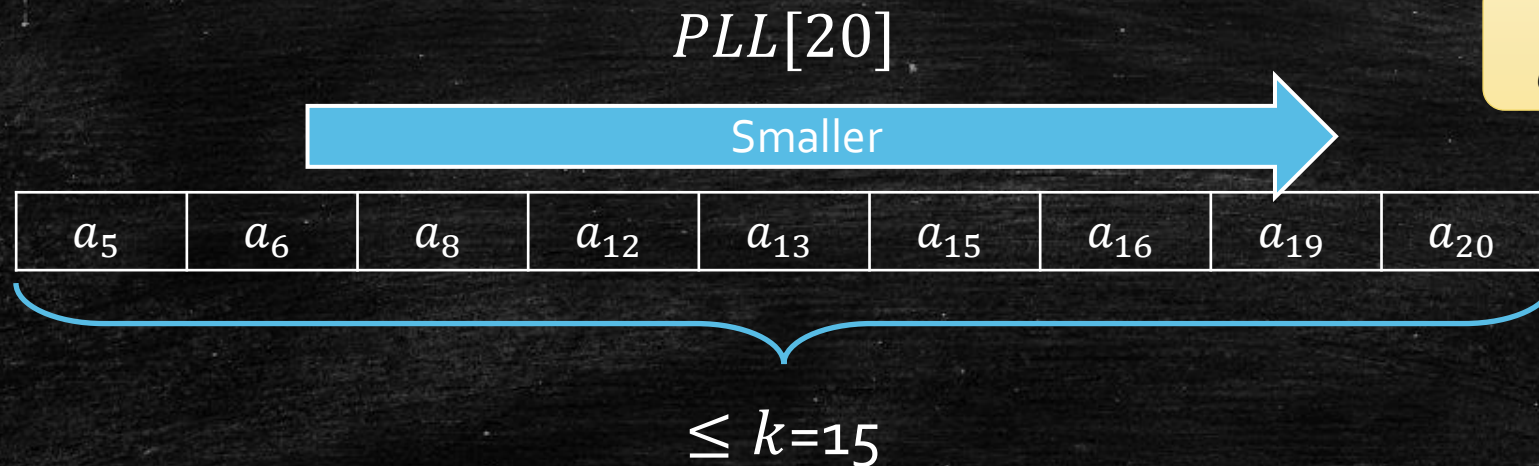
- **Potential Largest List (PLL)**
 - $PLL[i]$: the Potential Largest List for $a_{i-k+1} \sim a_i$.
 - At most k numbers.
 - Sorted by the index.
 - $i - k + 1 \leq \text{Index} \leq i$



Key Property:
 $a_i \geq a_j$ if $i < j$.

How to maintain PLL?

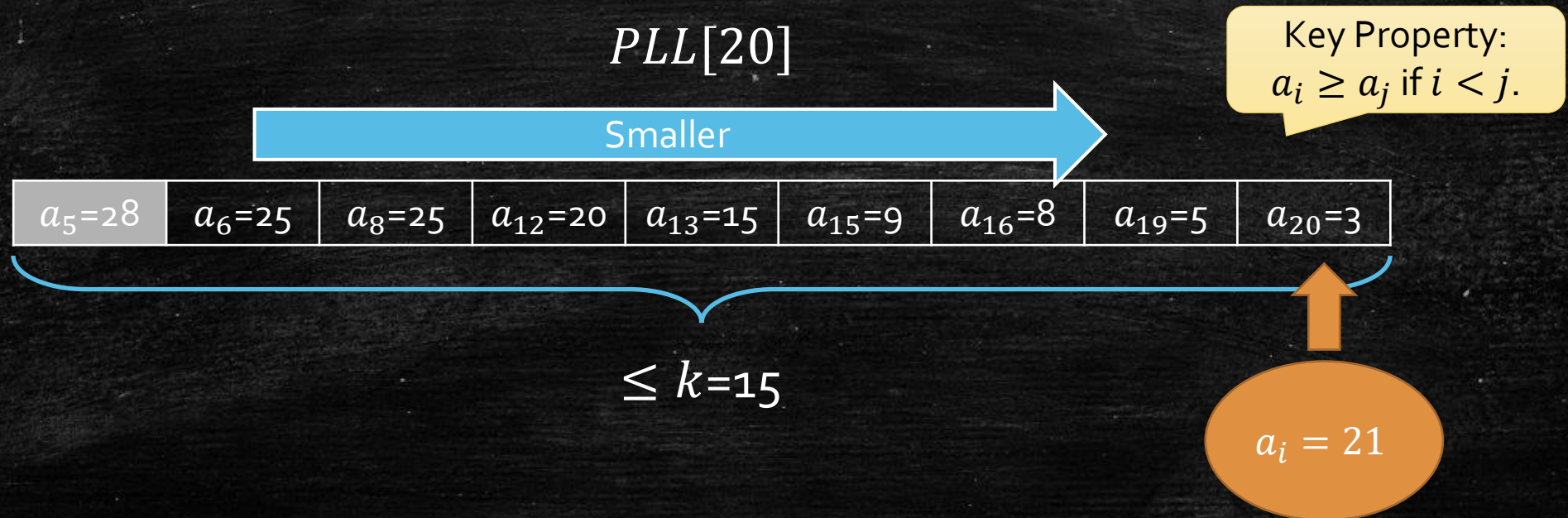
- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.



Key Property:
 $a_i \geq a_j$ if $i < j$.

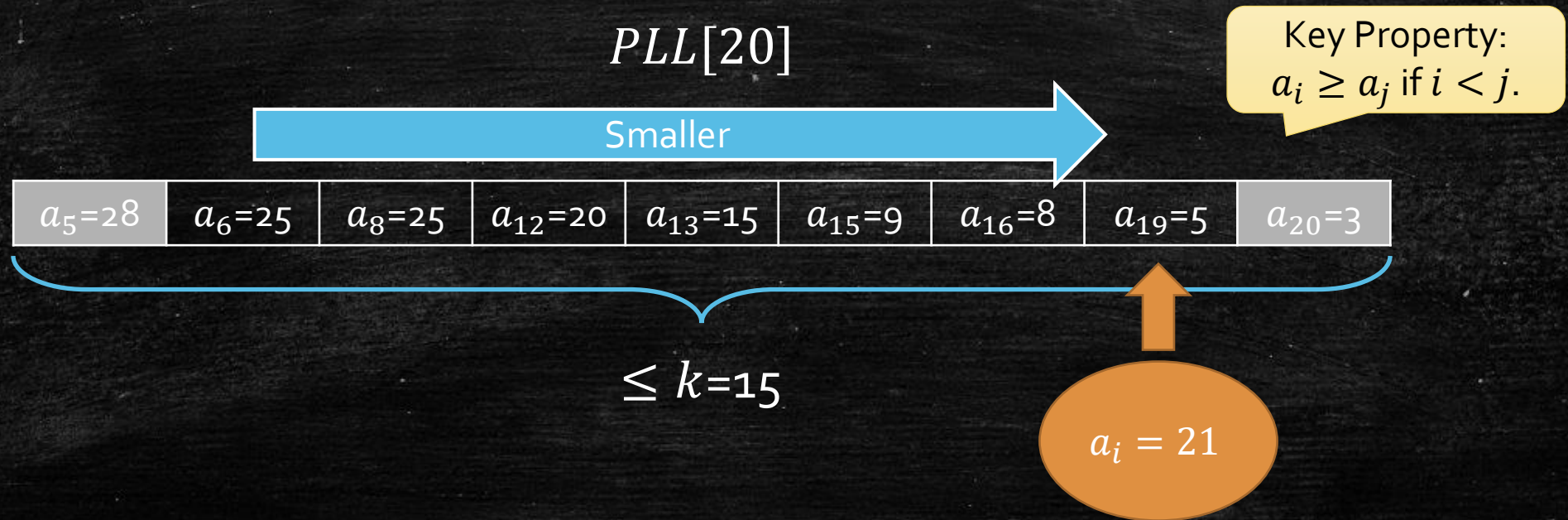
How to maintain PLL?

- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.
- Second, kick numbers by $a_{i=21}$.



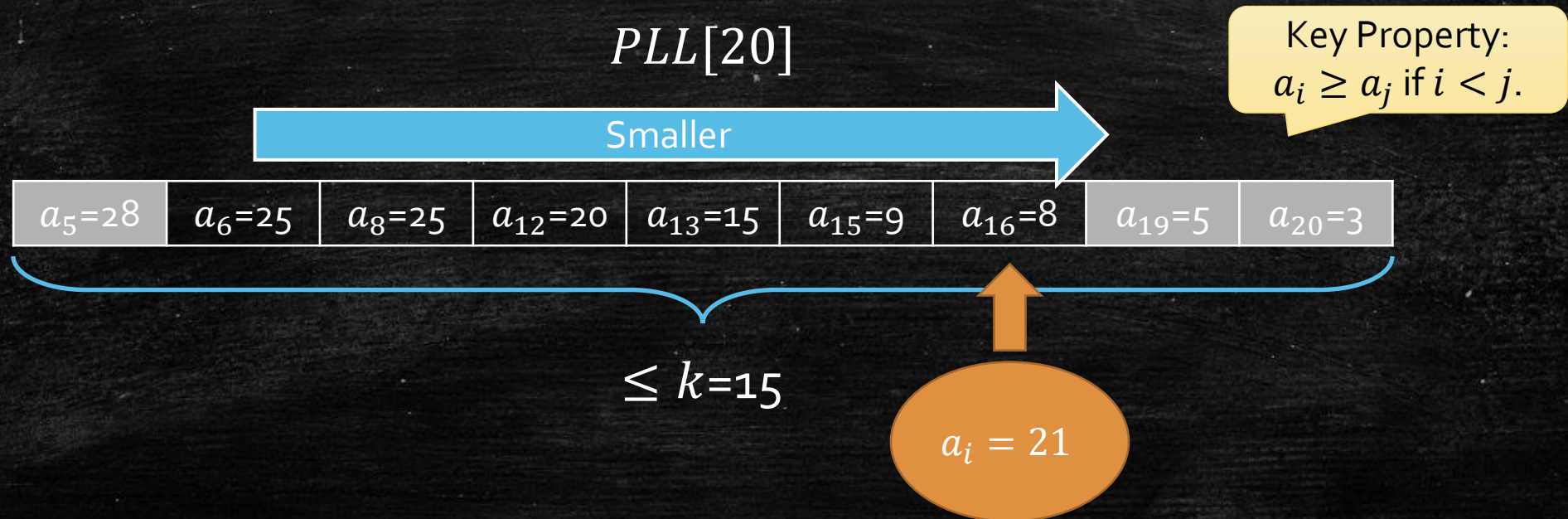
How to maintain PLL?

- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.
- Second, kick numbers by $a_{i=21}$.



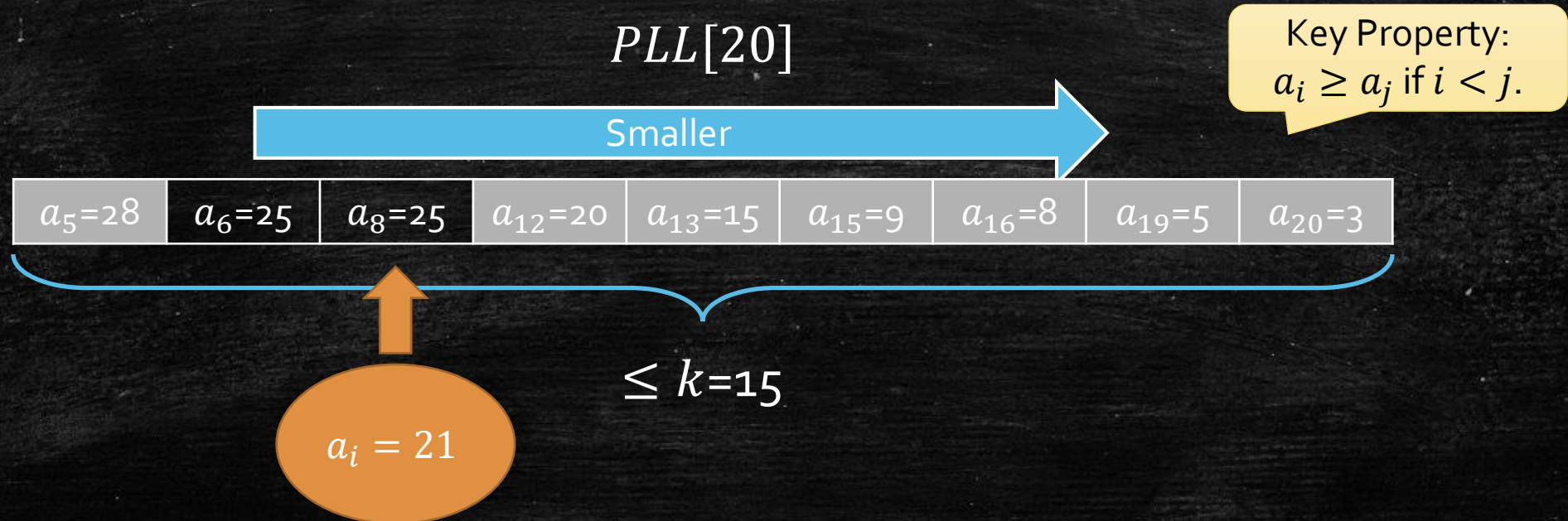
How to maintain PLL?

- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.
- Second, kick numbers by $a_{i=21}$.



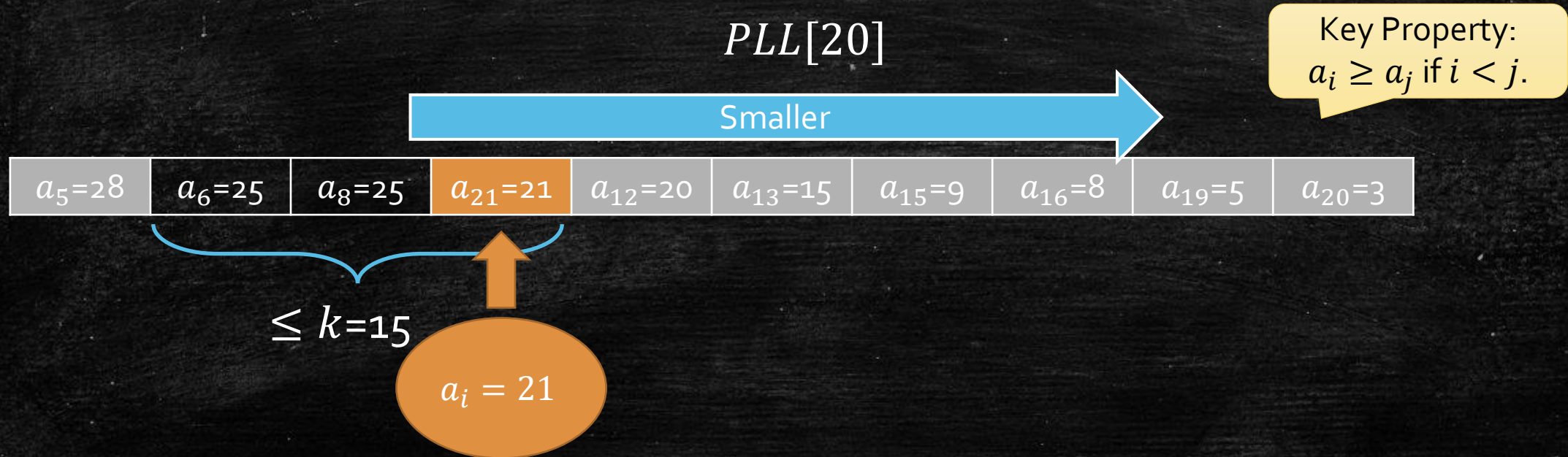
How to maintain PLL?

- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.
- Second, kick numbers by $a_{i=21}$.



How to maintain PLL?

- How to solve $PLL[i = 21]$ by $PLL[i - 1 = 20]$?
- First, kick the number if $index < i - k + 1 = 6$.
- Second, kick numbers by $a_{i=21}$.



Program: updating priority queue

Updating Priority Queue

function *updating*($a[1..n]$, i , k , PLL)

If $PLL.\text{front.index} \leq i - k$

PopFront(PLL)

while $PLL.\text{back.value} \leq a[i]$

PopBack(PLL)

PushBack(PLL , ($\text{index} = i$, $\text{value} = a[i]$))

Largest Number in range k

function *largest*($a[1..n]$, k)

$PLL = \text{NULL}$

for $i = 1$ to n

updating(a , i , k , PLL)

output $PPL.\text{front}.$

Running Time

Updating Priority Queue

```
function updating( $a[1..n]$ ,  $i$ ,  $k$ ,  $PLL$ )  
  if  $PLL.\text{front.index} \leq i - k$   
     $\text{PopFront}(PLL)$   
  while  $PLL.\text{back.value} \leq a[i]$   
     $\text{PopBack}(PLL)$   
   $\text{PushBack}(PLL, (\text{index} = i, \text{value} = a[i]))$ 
```

Largest Number in range k

```
function largest( $a[1..n]$ ,  $k$ )  
   $PLL = \text{NULL}$   
  for  $i = 1$  to  $n$   
    updating( $a$ ,  $i$ ,  $k$ ,  $PLL$ )  
  output  $PLL.\text{front}.$ 
```

$a_i = 21$

Charge to a_5 .

$a_5=28$	$a_6=25$	$a_8=25$	$a_{12}=20$	$a_{13}=15$	$a_{15}=9$	$a_{16}=8$	$a_{19}=5$	$a_{20}=3$
----------	----------	----------	-------------	-------------	------------	------------	------------	------------

$\leq k=15$



Running Time

Updating Priority Queue

```
function updating( $a[1..n]$ ,  $i$ ,  $k$ ,  $PLL$ )  
  if  $PLL.\text{front.index} \leq i - k$   
    PopFront( $PLL$ )  
  while  $PLL.\text{back.value} \leq a[i]$   
    PopBack( $PLL$ )  
  PushBack( $PLL$ , ( $\text{index} = i$ ,  $\text{value} = a[i]$ ))
```

Largest Number in range k

```
function largest( $a[1..n]$ ,  $k$ )  
   $PLL = \text{NULL}$   
  for  $i = 1$  to  $n$   
    updating( $a$ ,  $i$ ,  $k$ ,  $PLL$ )  
  output  $PPL.\text{front}.$ 
```

$a_i = 21$

$a_5=28$	$a_6=25$	$a_8=25$	$a_{12}=20$	$a_{13}=15$	$a_{15}=9$	$a_{16}=8$	$a_{19}=5$	$a_{20}=3$
----------	----------	----------	-------------	-------------	------------	------------	------------	------------

Charge to a_{20} .

$\leq k=15$

Running Time

Updating Priority Queue

```
function updating( $a[1..n]$ ,  $i$ ,  $k$ ,  $PLL$ )  
  if  $PLL.\text{front.index} \leq i - k$   
     $\text{PopFront}(PLL)$   
  while  $PLL.\text{back.value} \leq a[i]$   
     $\text{PopBack}(PLL)$   
   $\text{PushBack}(PLL, (\text{index} = i, \text{value} = a[i]))$ 
```

Largest Number in range k

```
function largest( $a[1..n]$ ,  $k$ )  
   $PLL = \text{NULL}$   
  for  $i = 1$  to  $n$   
    updating( $a$ ,  $i$ ,  $k$ ,  $PLL$ )  
  output  $PPL.\text{front}.$ 
```

$a_i = 21$

$a_5=28$	$a_6=25$	$a_8=25$	$a_{21}=21$	$a_{12}=20$	$a_{13}=15$	$a_{15}=9$	$a_{16}=8$	$a_{19}=5$	$a_{20}=3$
----------	----------	----------	-------------	-------------	-------------	------------	------------	------------	------------

$\leq k=15$

Charge to a_{21} .

Running Time

- The cost of n times updating has been charged to numbers!
- Each number
 - Charged **once** when it is **popped out**.
 - Charged **once** when it is **pushed in**.
- Totally: $O(n)$.

Priority Queue Helps DP

Priority Queue

Longest Increasing Sequence Revisit

- **Input:** A sequence a_1, a_2, \dots, a_n .
- **Output:** the Longest Increasing Subsequence (LIS)
 - $a_{i_1} < a_{i_2} < a_{i_3} \dots < a_{i_k}$
 - $i_1 < i_2 < i_3 \dots < i_k$

1	5	13	2	6	24	15	23	2	16
---	---	----	---	---	----	----	----	---	----

Do you feel that we can
improve?

Is there any monotone thing?

Previous Transfer

- $lis[i] = \max_{a_j < a_i, j < i} \{lis[j] + 1\}$
- Definition: **Potential Prefix**
 - The set of a_j that is possible to be the best prefix of future numbers.

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

Who are the Potential Prefix?

Previous Transfer

- $lis[i] = \max_{a_j < a_i, j < i} \{lis[j] + 1\}$
- Definition: **Potential Prefix**
 - The set of a_j that is possible to be the best prefix of future numbers.

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

It is not because $a[i] > a[j]$ and $lis[i] = lis[j]$

Who are the Potential Prefixes?

New Potential List

- $Sm[len]$: the **smallest ended number** for an increasing subsequence with **length** len .
- Remark: it is enough to record all **Potential Prefixes** (length and number).

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

New Potential List

- $Sm[len]$: the **smallest ended number** for an increasing subsequence with **length** len .
- Remark: it is enough to record all **Potential Prefixes** (length and number).

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

Larger

$sm[len]$	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?
- Difference between $i - 1$ and i ?
 - a_i comes in.
 - It may **become** a potential prefixes and **kick** some potential prefixes.

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]!$

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$



	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[i - 1, len]$

Case 2: $a_i \leq sm[i - 1, len]$

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS.
- it can not update $sm[len]$.

Case 2: $a_i \leq sm[len]$

- It may update $sm[len]$
- it can not create a longer LIS.

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS.
- it can not update $sm[len]$.

Case 2: $a_i \leq sm[len]$

- It may update $sm[len]$
- it can not create a longer LIS.

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS.
- it can not update $sm[len]$.

Case 2: $a_i \leq sm[len]$

- It may update $sm[len]$
- it can not create a longer LIS.

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS.
- it can not update $sm[len]$.

Case 2: $a_i \leq sm[len]$

- It may update $sm[len]$
- it can not create a longer LIS.

↓

$sm[len]$	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS.
- it can not update $sm[len]$.

Case 2: $a_i \leq sm[len]$

- It may update $sm[len]$
- it can not create a longer LIS.

↓

$sm[len]$	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS
- it can not update $sm[len]$

Because we move to here.

Case 2: $a_i \leq sm[len]$

- It **must** update $sm[len]$.
- it can not create a longer LIS.

↓

$sm[len]$	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
	0	1	2	6	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS
- it can not update $sm[len]$

Because we move to here.

Case 2: $a_i \leq sm[len]$

- It **must** update $sm[len]$
- it can not create a longer LIS.

↓

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	$a_i=5$	-	-	-	-	-	-

Updating $sm[len]$!

- How to update $sm[len]$ (Potential Prefixes)?

i

$a[i]$	1	5	13	2	6	5	15	23	2	16
$lis[i]$	1	2	3	2	3	3	-	-	-	-

$a_i = 5$

Case 1: $a_i > sm[len]$

- it can create a longer LIS
- it can not update $sm[len]$

Because we move to here.

Case 2: $a_i \leq sm[len]$

- It **must** update $sm[len]$
- it can not create a longer LIS.

↓

	$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
$sm[len]$	0	1	2	$a_i=5$	-	-	-	-	-	-

The DP Algorithm

- Initialize $sm[0] = 0$
- For $i = 1$ to n
 - Update $sm[len]$ by a_i
 - **It requires $O(\max\{len\} = i)$!**
 - **Remark: now we do not kick everything we pass.**
- Output the largest len such that $sm[len] \neq "-"$.

Recap The Updating

- We need to find the largest len such that $a_i > sm[i - 1, len]$.
- Then we update: $sm[i, len + 1] = a_i$.

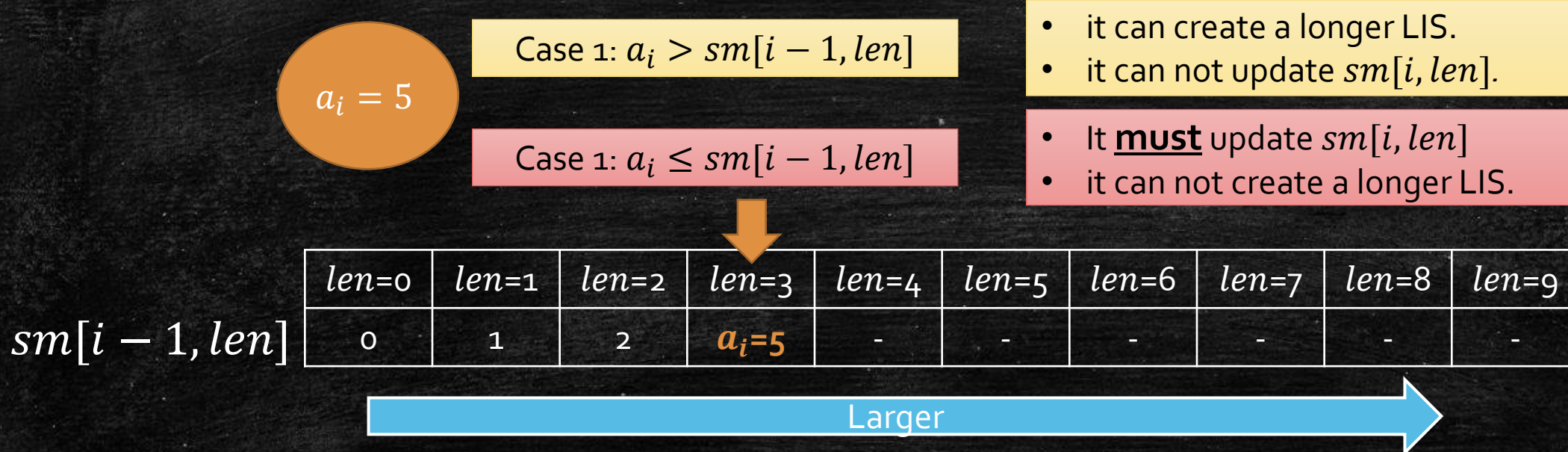
$a_i = 5$

Case 1: $a_i > sm[i - 1, len]$

- it can create a longer LIS.
- it can not update $sm[i, len]$.

Case 1: $a_i \leq sm[i - 1, len]$

- It **must** update $sm[i, len]$
- it can not create a longer LIS.



$len=0$	$len=1$	$len=2$	$len=3$	$len=4$	$len=5$	$len=6$	$len=7$	$len=8$	$len=9$
0	1	2	$a_i=5$	-	-	-	-	-	-

Larger

How to do it efficiently?

Yes! Binary Search!

Now it is better!

- Plan
 - Initialize $sm[0,0] = 0$
- Solve $sm[i, len]$ from $sm[i - 1, len]$ by a_i .
 - It requires $O(\log(len)) = O(\log n)$.
- Output the largest len such that $sm[n, len] \neq "-"$.
- Totally $O(n \log n)$.

The DP Algorithm

- Initialize $sm[0] = 0$
- For $i = 1$ to n
 - Update $sm[len]$ with a_i by binary search.
 - ~~It requires $O(\max\{maxlen\} = i)$!~~
 - **Remark: now we do not kick everything we pass.**
 - It requires $O(\log(maxlen)) = O(\log n)$.
- Output the largest len such that $sm[len] \neq "-"$.

Priority Queue Can Be Stronger

Minimizing Manufacturing Cost

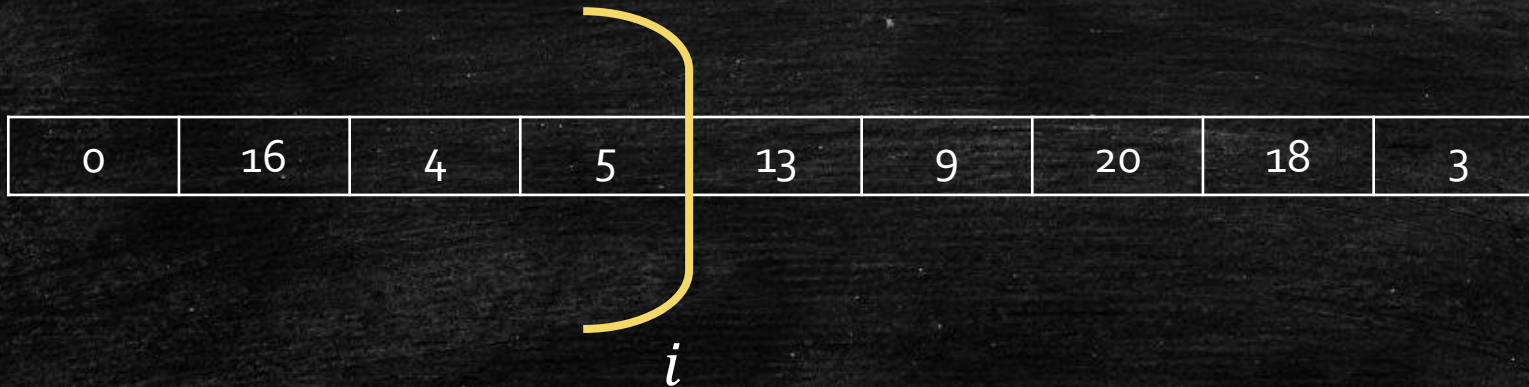
- **Input:** A sequence of items with cost a_1, a_2, \dots, a_n .
- Need to Do:
 - Manufacture these items.
 - Operation $\text{man}(l, r)$: manufacture the items from l to r .
 - $\text{cost}(l, r) = C + (\sum_{i=l}^r a_i)^2$.
- **Output:** The **minimum** cost to manufacture all items.

Discussion

- Cost function: $cost(l, r) = C + (\sum_{i=l}^r a_i)^2$.
- Cost function: $cost(l, r) = C + \sum_{i=l}^r a_i$.
- Cost function: $cost(l, r) = C + (\sum_{i=l}^r a_i)^2$, with $C = 0$.
- Only the first one need to optimize!

Define subproblems

- $f[i]$: the minimum cost for manufacturing item 1 to i .
- How to solve $f[i]$?

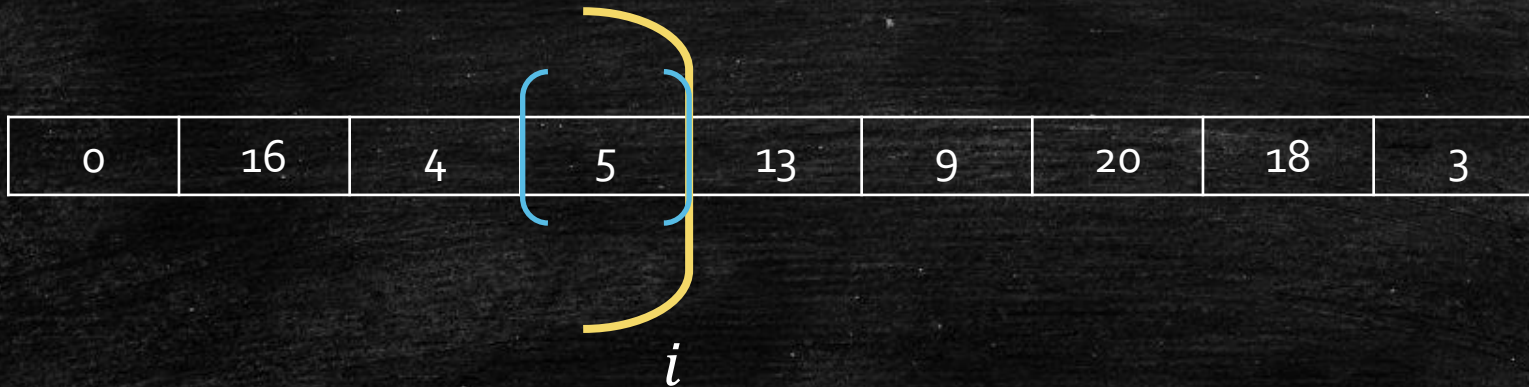


A horizontal array of nine cells containing the values 0, 16, 4, 5, 13, 9, 20, 18, and 3. A yellow bracket is drawn under the first four cells (0, 16, 4, 5), with the label i positioned below the bracket's right end. This indicates that the subproblem $f[i]$ involves the first i items.

0	16	4	5	13	9	20	18	3
---	----	---	---	----	---	----	----	---

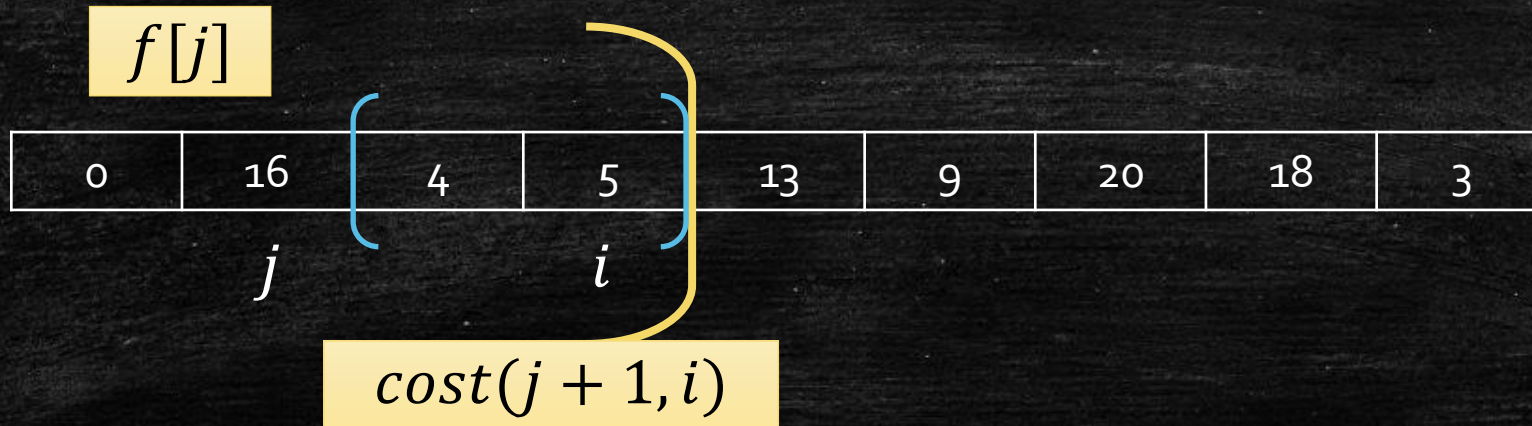
Solving $f[i]$

- $f[i]$: the minimum cost for manufacturing item 1 to i .
- How to solve $f[i]$?
- We can manufacture item i alone.



Solving $f[i]$

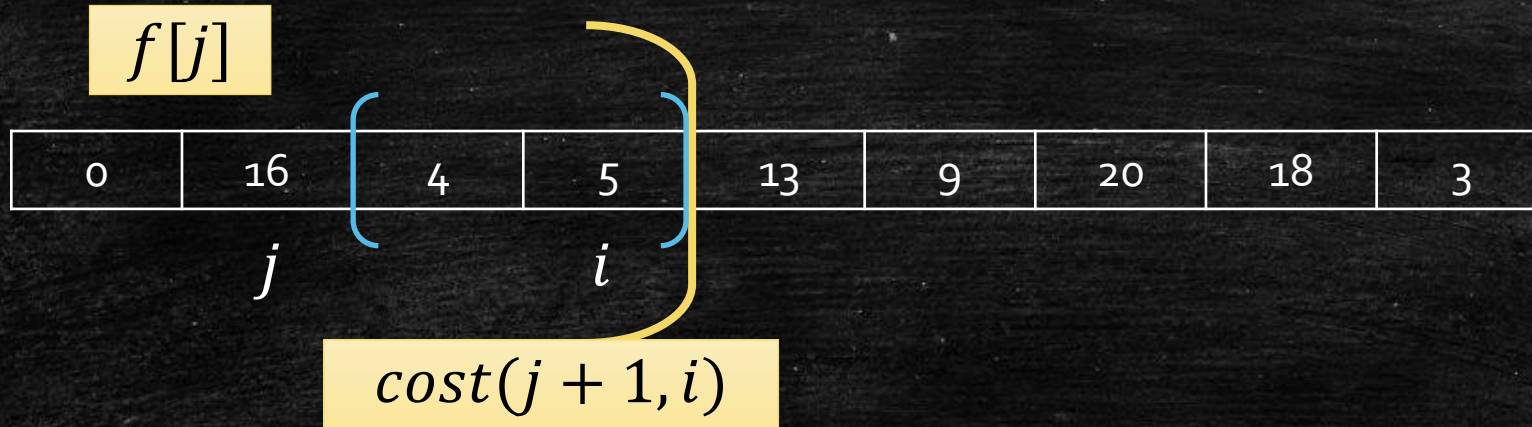
- $f[i]$: the minimum cost for manufacturing item 1 to i .
- How to solve $f[i]$?
- We can also manufacture i along with an interval.
- $f[i] = \min_{j < i} f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2$



DP algorithm

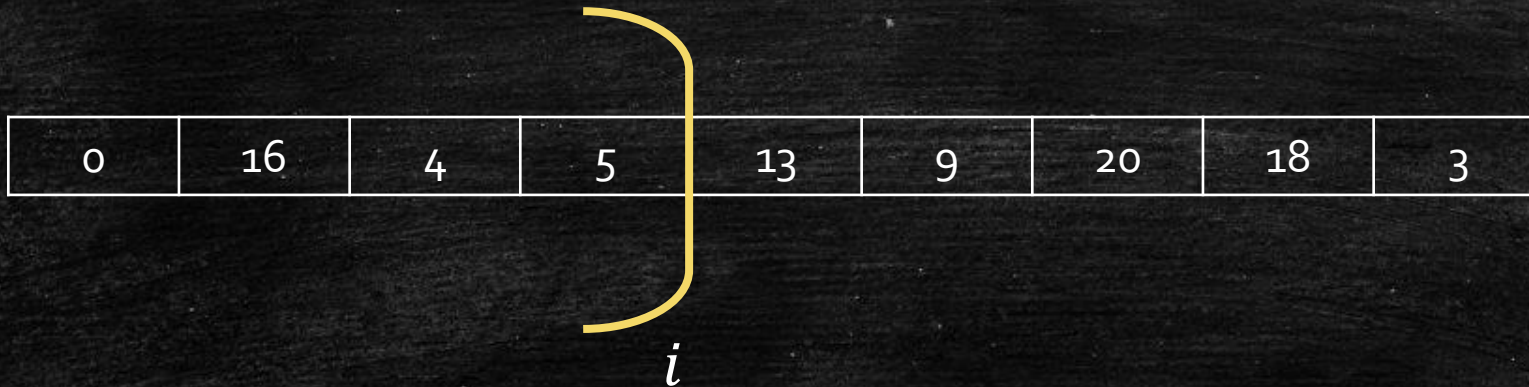
- Define $f[0] = 0$.
- Solve $f[i]$ from 1 to n , and output $f[n]$.
- $f[i] = \min_{j < i} f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2$.

$O(n^2)$



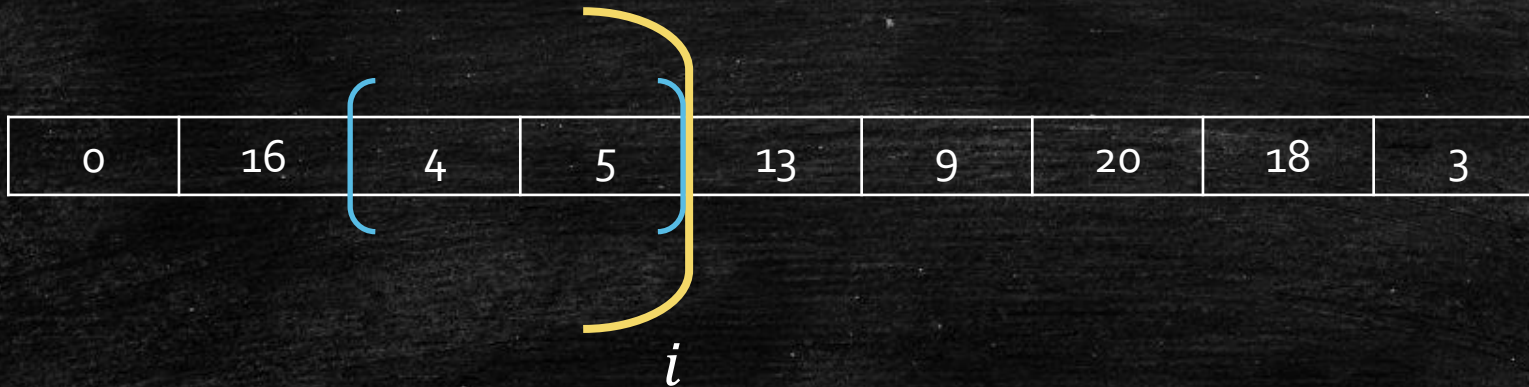
The Potential Idea Again!

- Question: Can every j be a potential prefix for the future?



The Potential Idea Again!

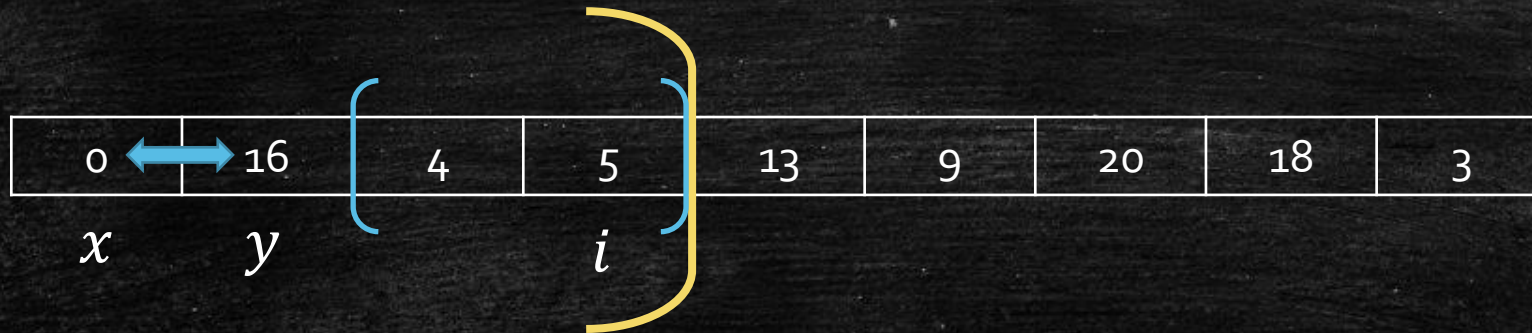
- Question: Can every j be a potential prefix for the future?
- Trade-off
 - Smaller j is better for paid cost.
 - Larger j is better for future cost.



Let us do some math!

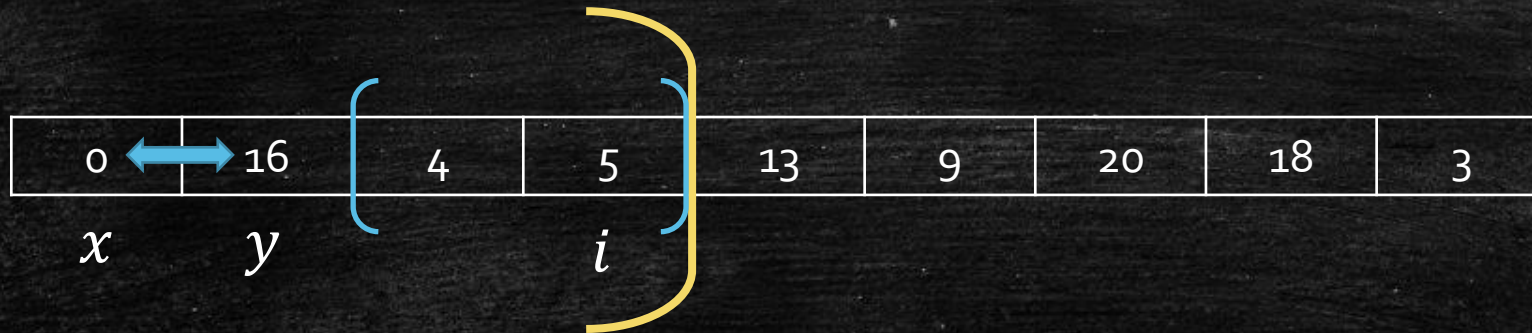
General Question

- $f[i] = \min_{j < i} f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2.$
- When $j = y$ is better than $j = x$ when calculate $f[i]$?



Math Time!

- $f[i] = \min_{j < i} f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2.$
- When $j = y$ is better than $j = x$ when calculate $f[i]$?
- $f[x] + C + \left(\sum_{k=x+1}^i a_k\right)^2 > f[y] + C + \left(\sum_{k=y+1}^i a_k\right)^2$



Math Time!

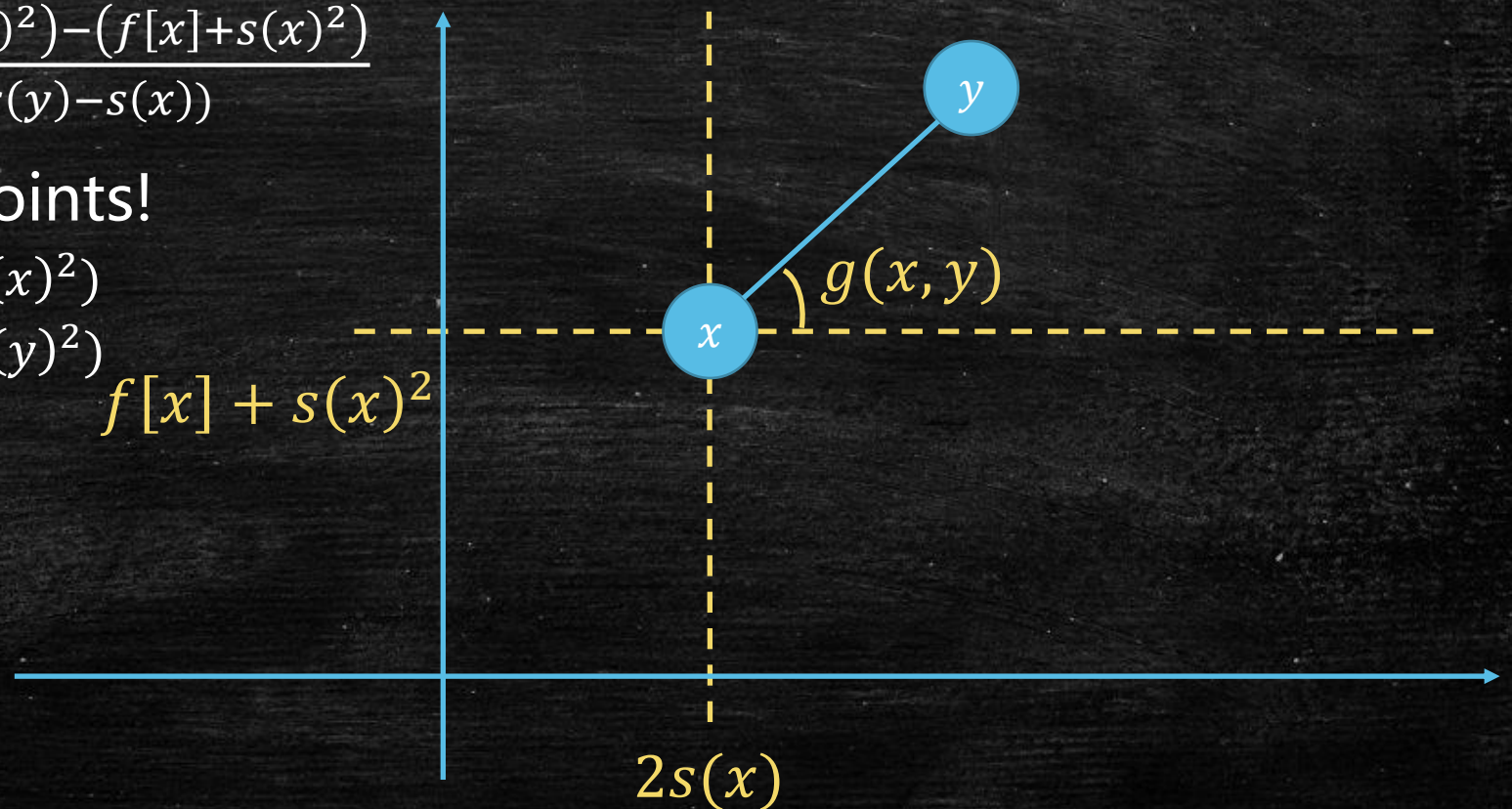
- $f[i] = \min_{j < i} f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2.$
- When $j = y$ is better than $j = x$ when calculate $f[i]$?
- Let $s(i) = \sum_{k=1}^i a_k$

$$\begin{aligned}
 f[x] + C + (s(i) - s(x))^2 &> f[y] + C + (s(i) - s(y))^2 \\
 f[x] - f[y] &> (s(i) - s(y))^2 - (s(i) - s(x))^2 \\
 &= s(y)^2 - s(x)^2 - 2s(i)(s(y) - s(x)) \\
 \frac{(f[y] + s(y)^2) - (f[x] + s(x)^2)}{2(s(y) - s(x))} &< s(i)
 \end{aligned}$$



Math Time!

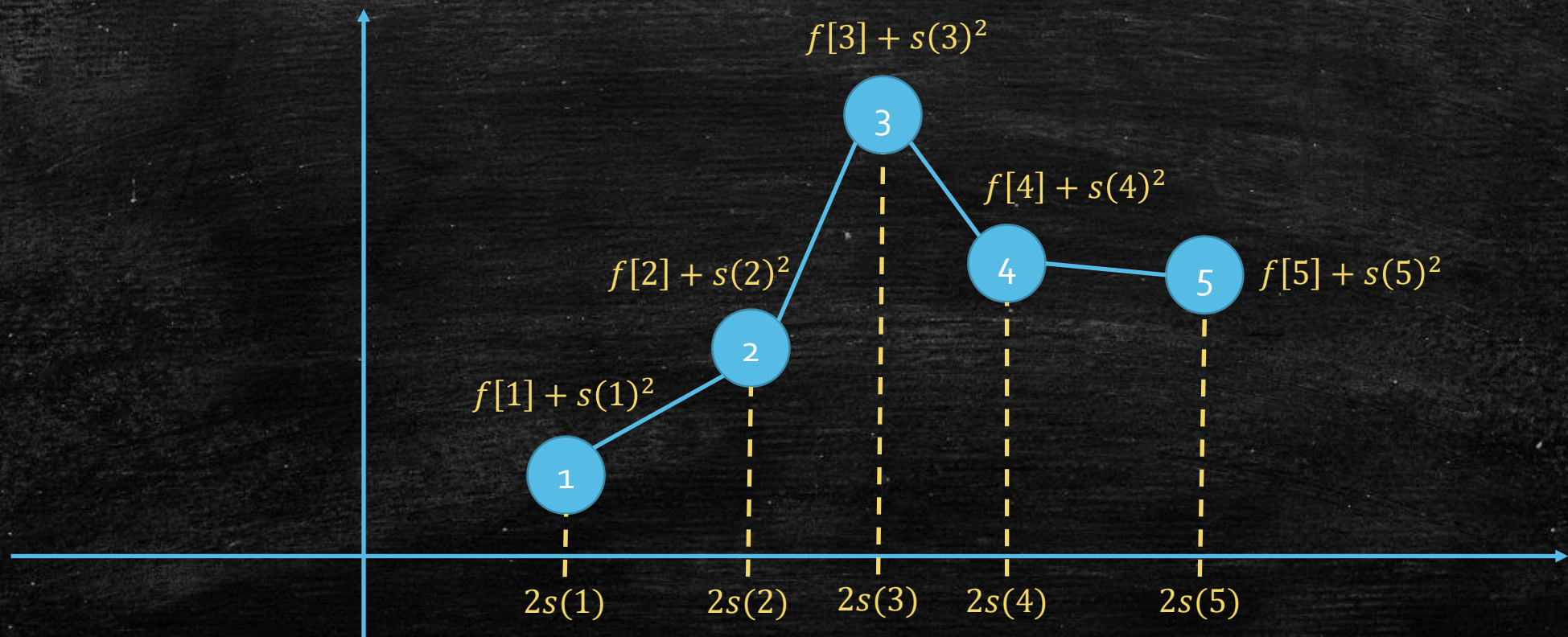
- $\frac{(f[y]+s(y)^2)-(f[x]+s(x)^2)}{2(s(y)-s(x))} < s(i)$
- $g(x, y) = \frac{(f[y]+s(y)^2)-(f[x]+s(x)^2)}{2(s(y)-s(x))}$
- View it as two points!
 - $x: (2s(x), f[x] + s(x)^2)$
 - $y: (2s(y), f[y] + s(y)^2)$



y is better than x for i means the gradient of $x \rightarrow y$: smaller than $s(i)$.

Put everything on the graph

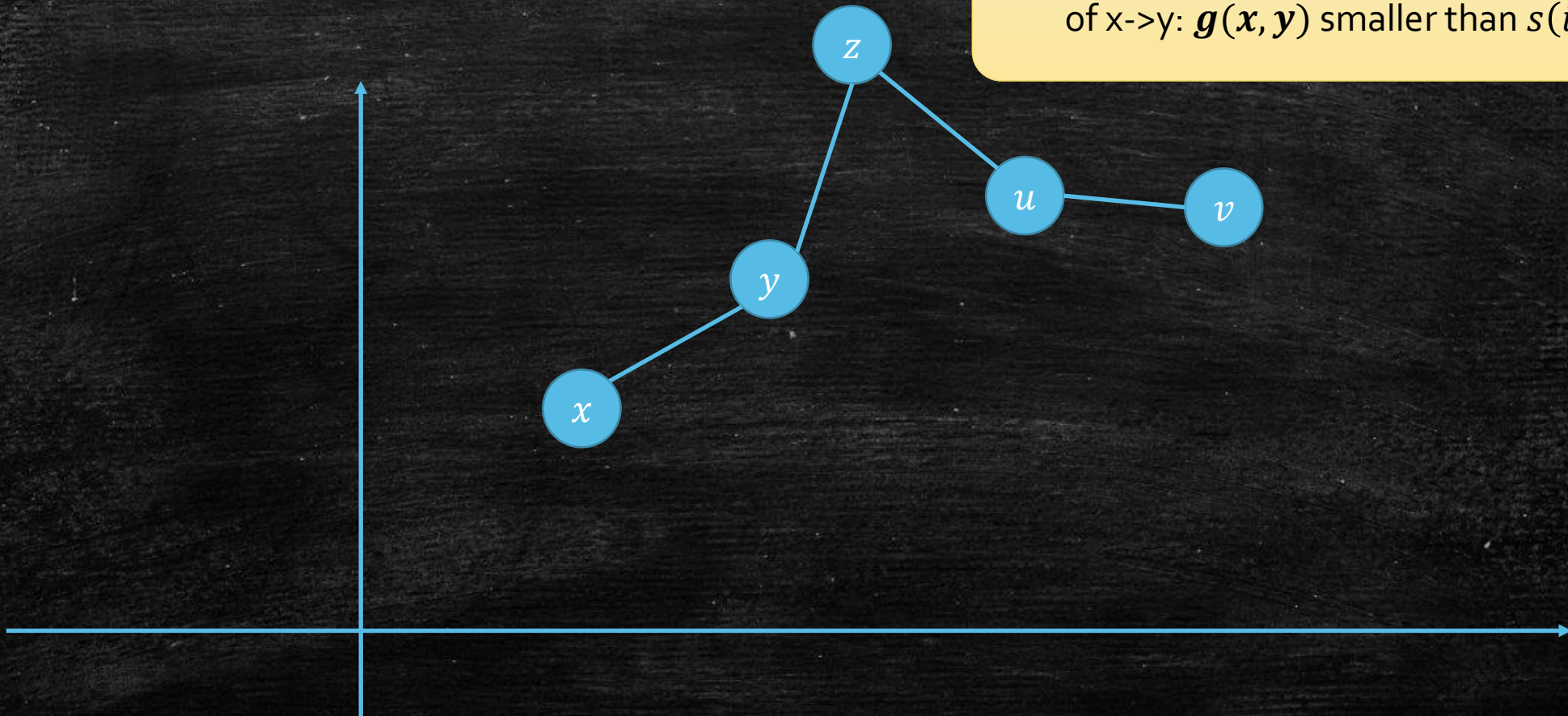
$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$				
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



Who can be kicked out?

Who can be kicked out?

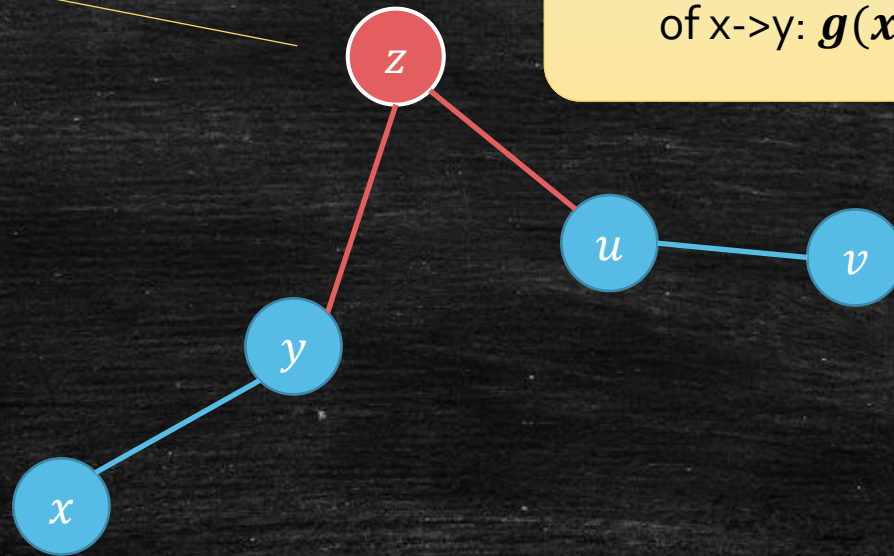
y is better than x for i means the gradient of $x \rightarrow y$: $g(x, y)$ smaller than $s(i)$.



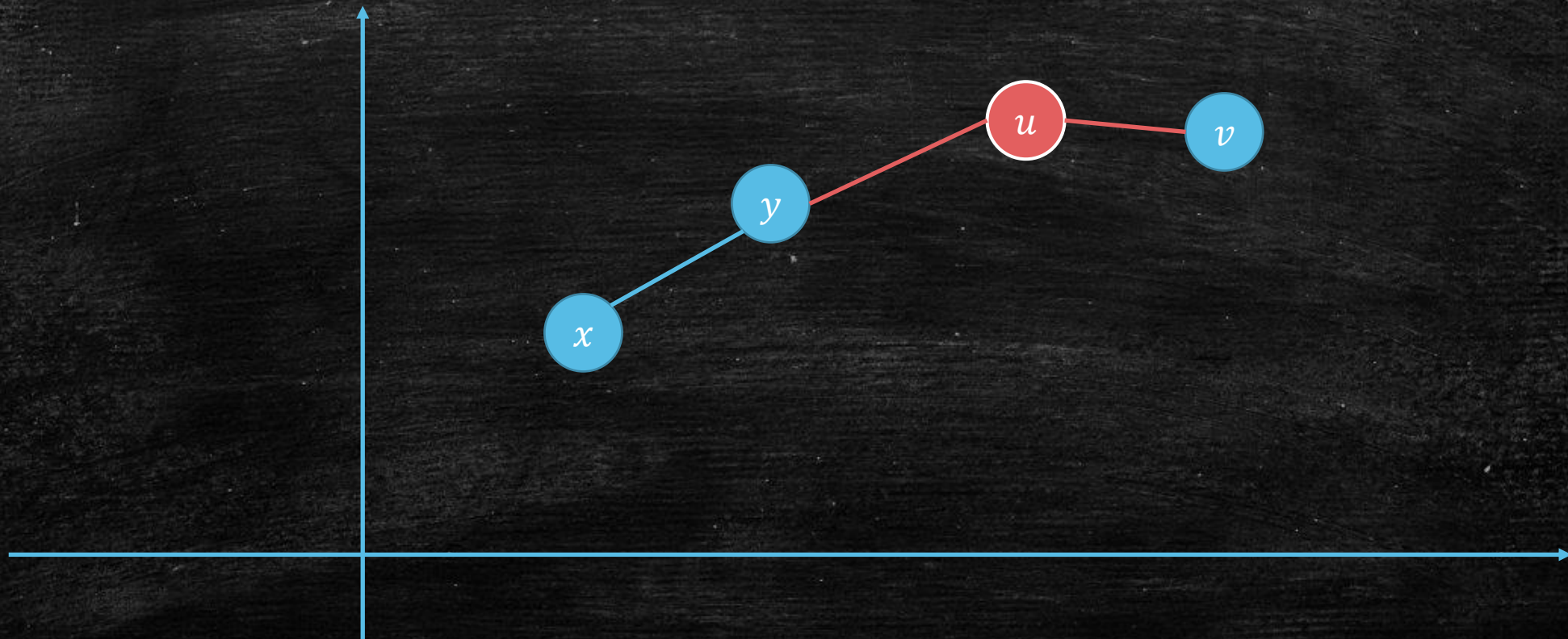
Who can be kicked out?

$g(y, z) > g(z, u)$! If z is better than y , then u is better than u .

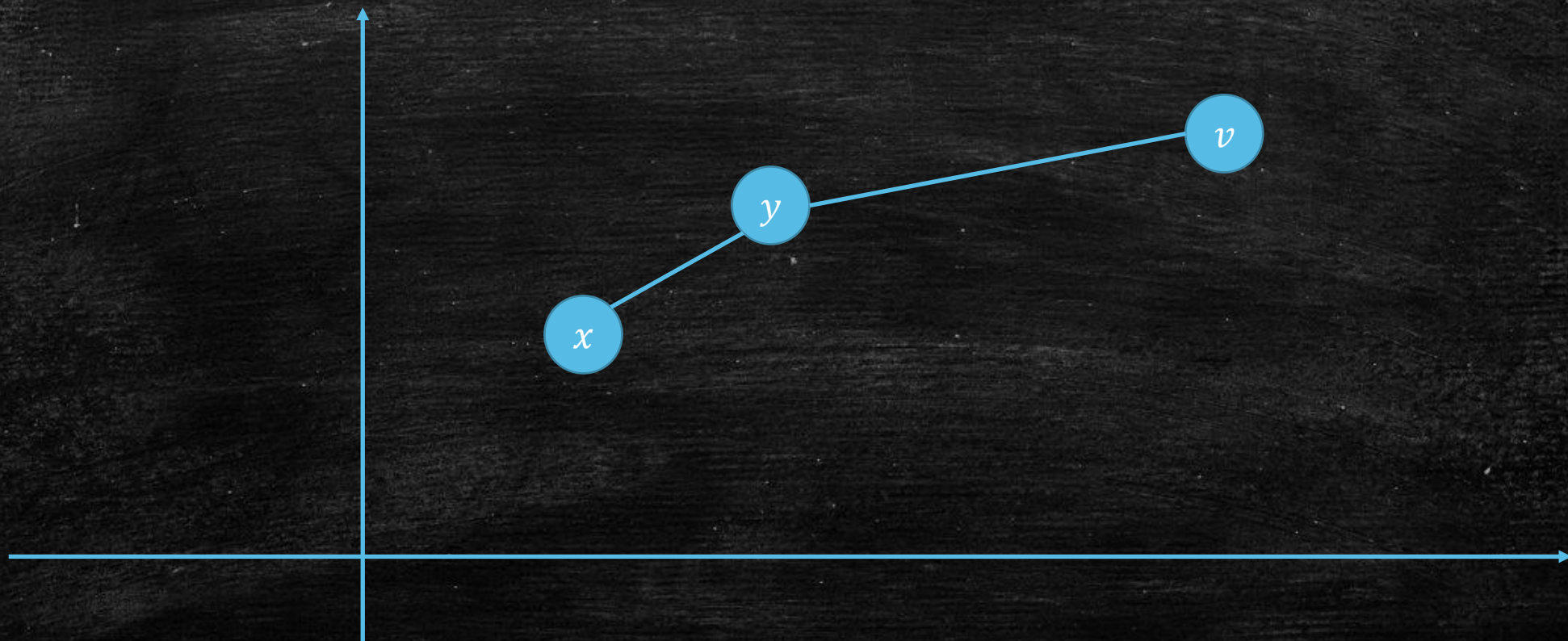
y is better than x for i means the gradient of $x \rightarrow y$: $g(x, y)$ smaller than $s(i)$.



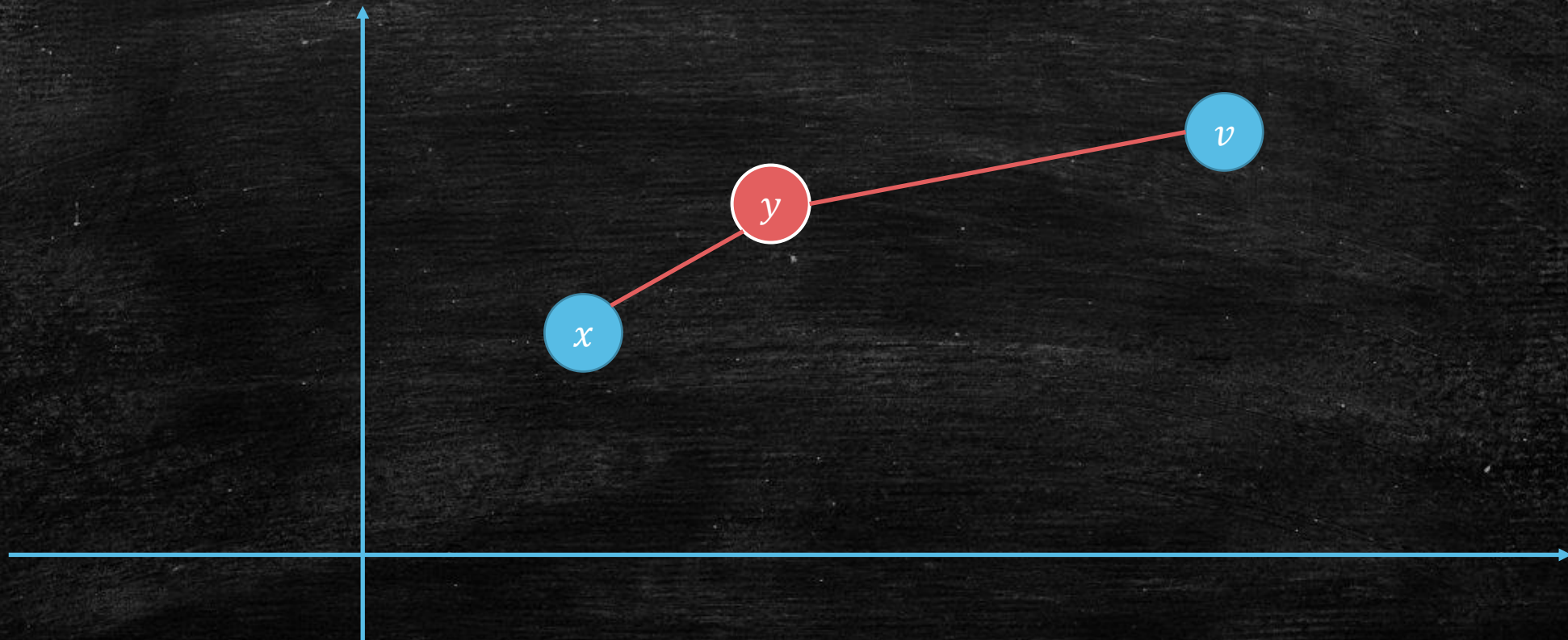
Who can be kicked out?



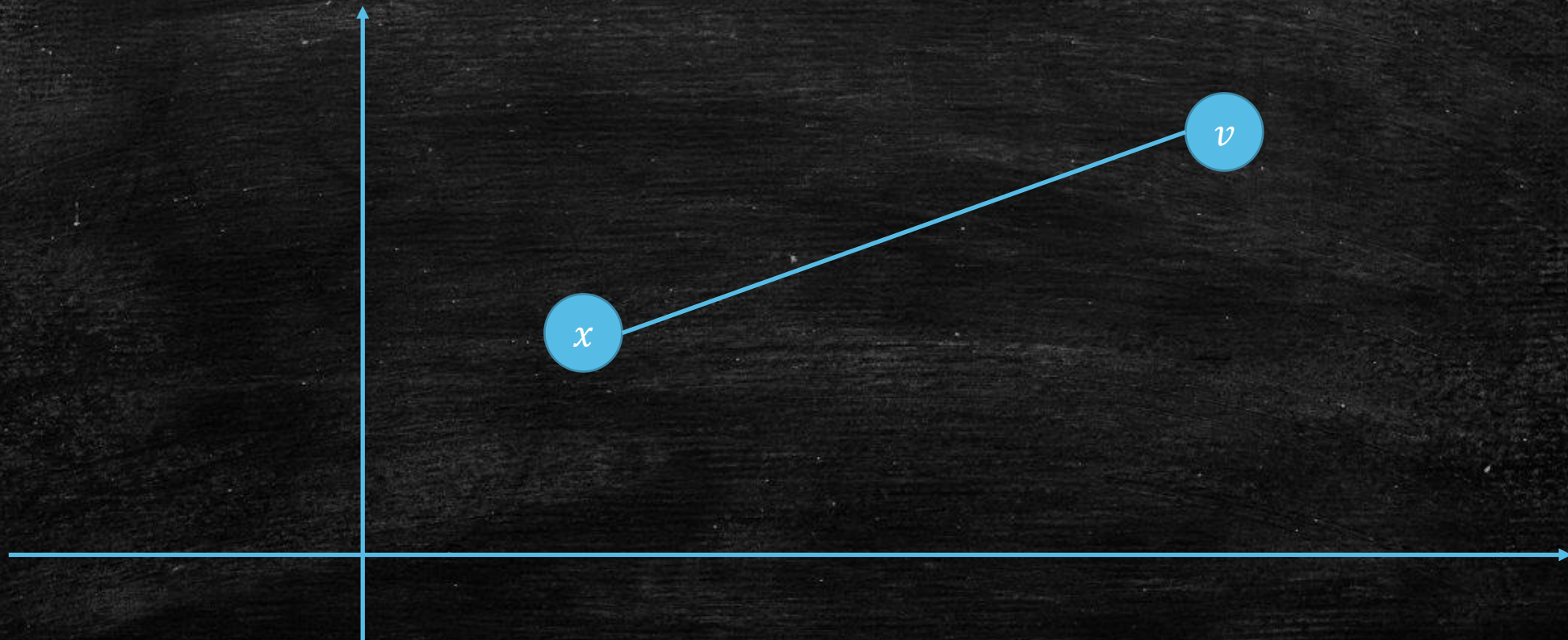
Who can be kicked out?



Who can be kicked out?

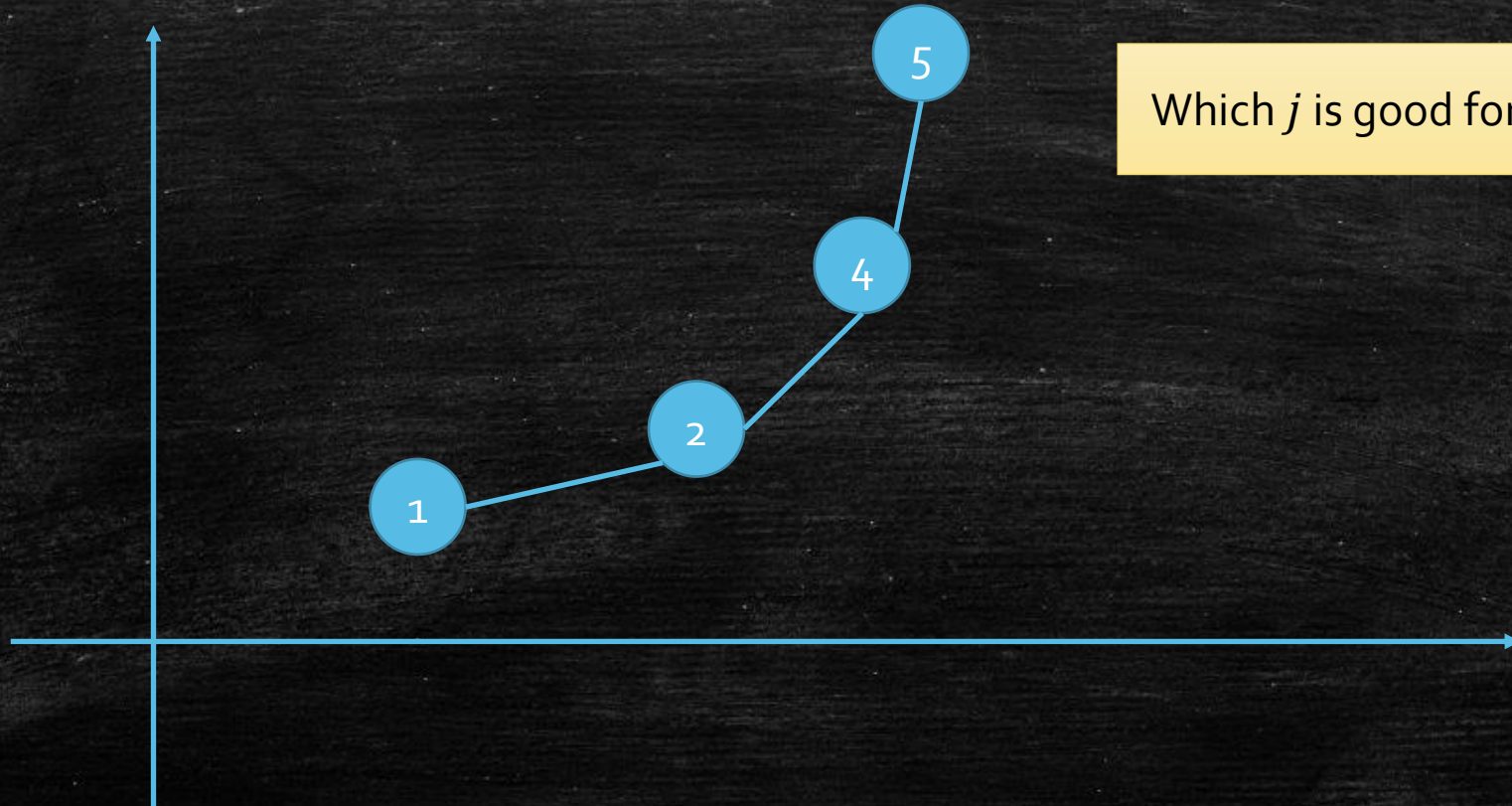


Who can be kicked out?



After Kicking: A Convex Hall.

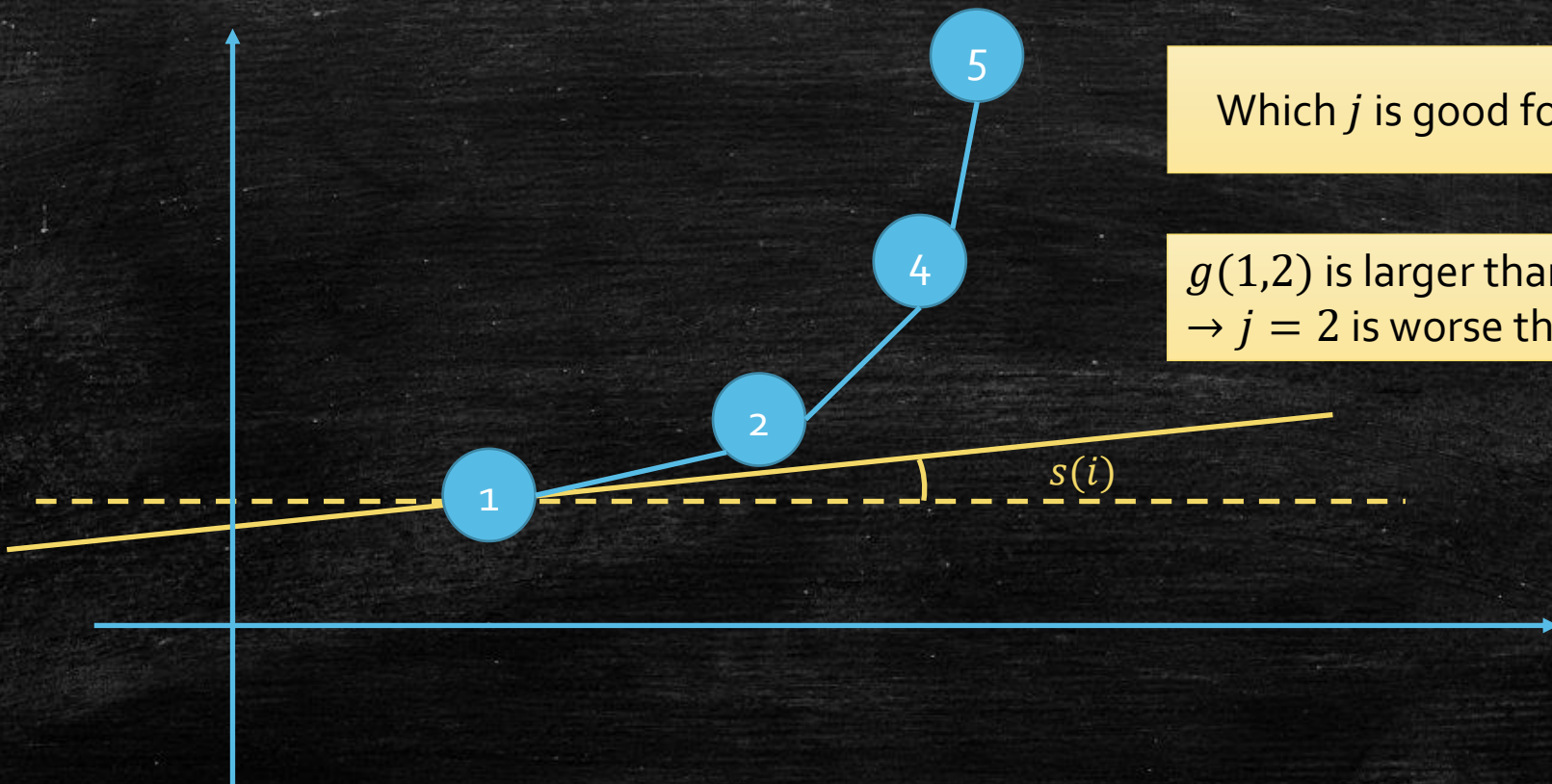
$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



Which j is good for $i = 6$?

After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9

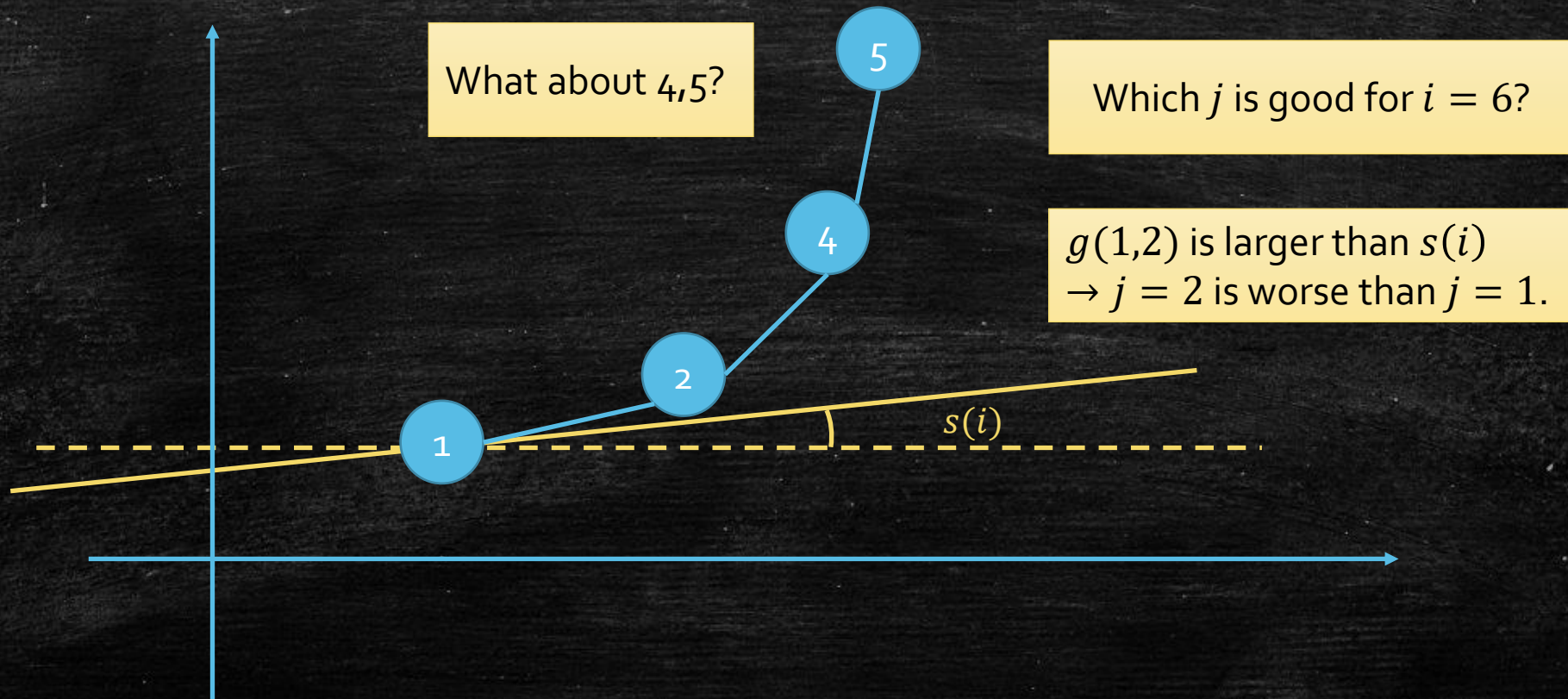


Which j is good for $i = 6$?

$g(1,2)$ is larger than $s(i)$
 $\rightarrow j = 2$ is worse than $j = 1$.

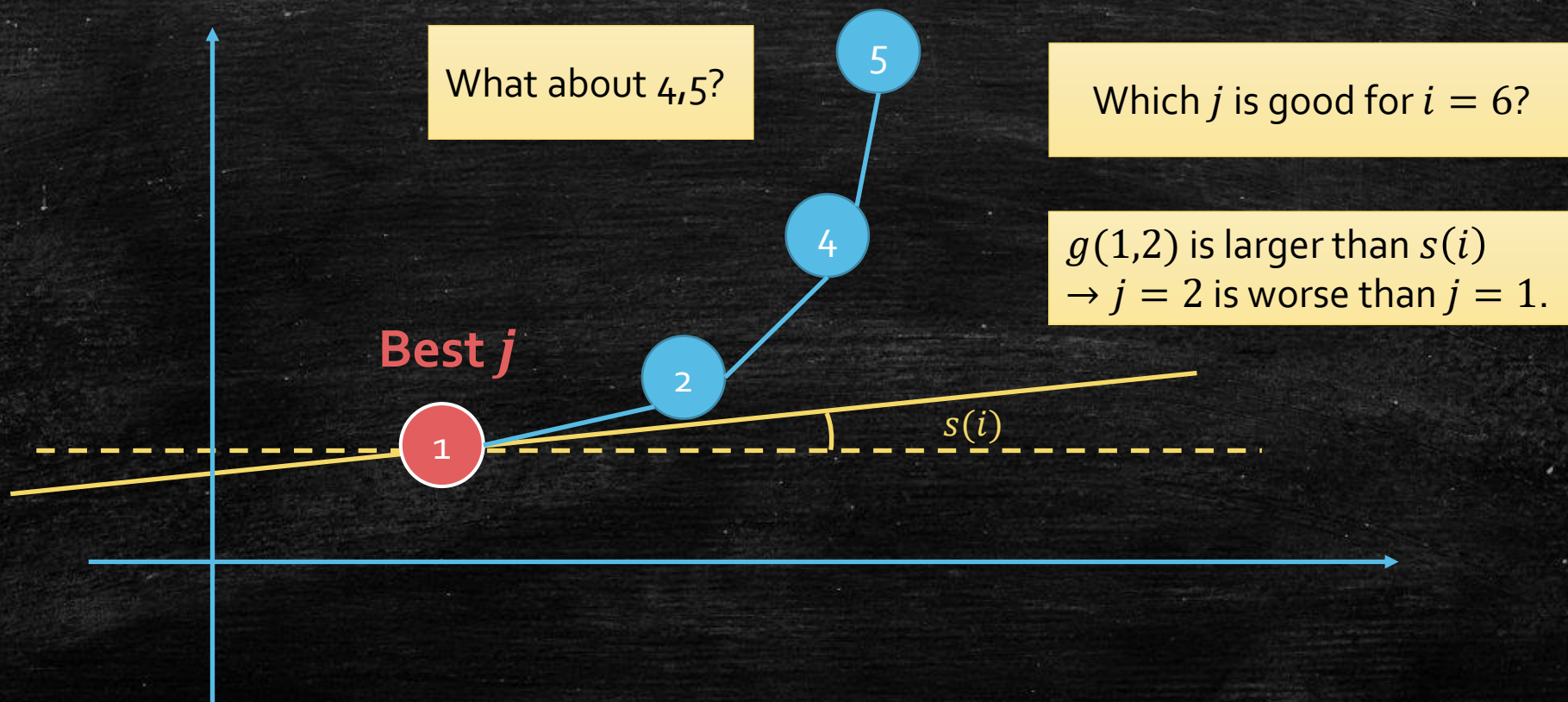
After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



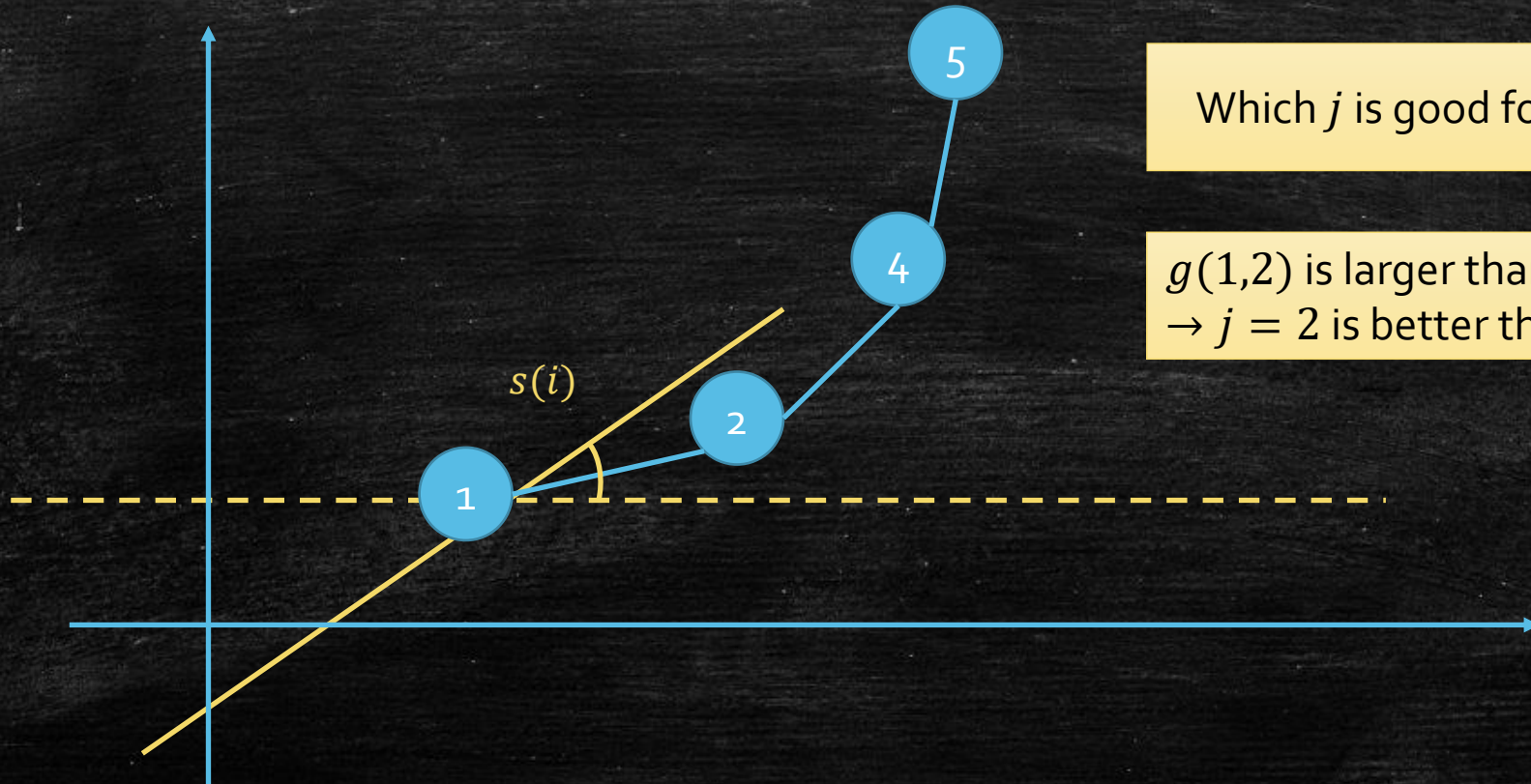
After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9

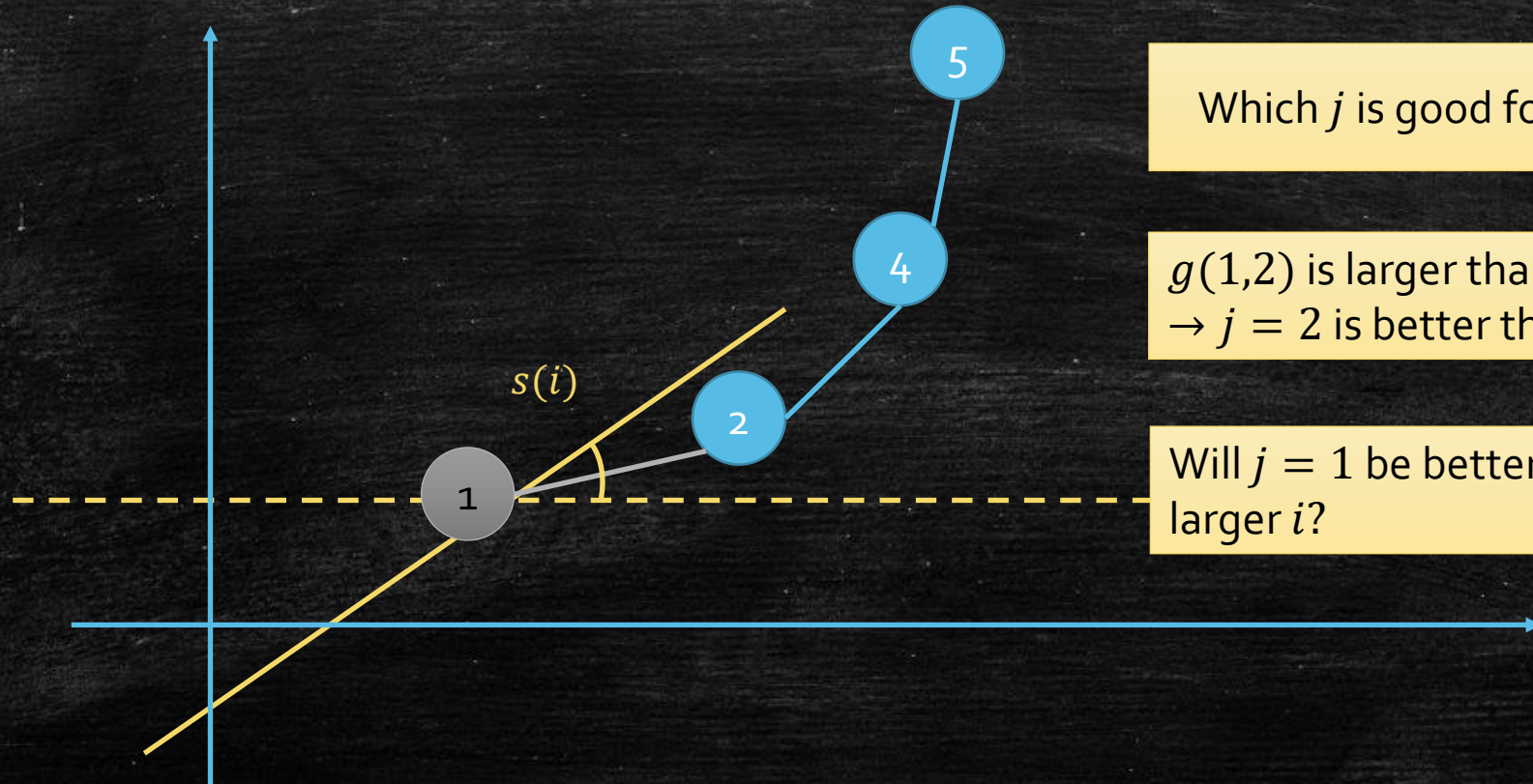


Which j is good for $i = 6$?

$g(1,2)$ is larger than $s(i)$
 $\rightarrow j = 2$ is better than $j = 1$.

After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



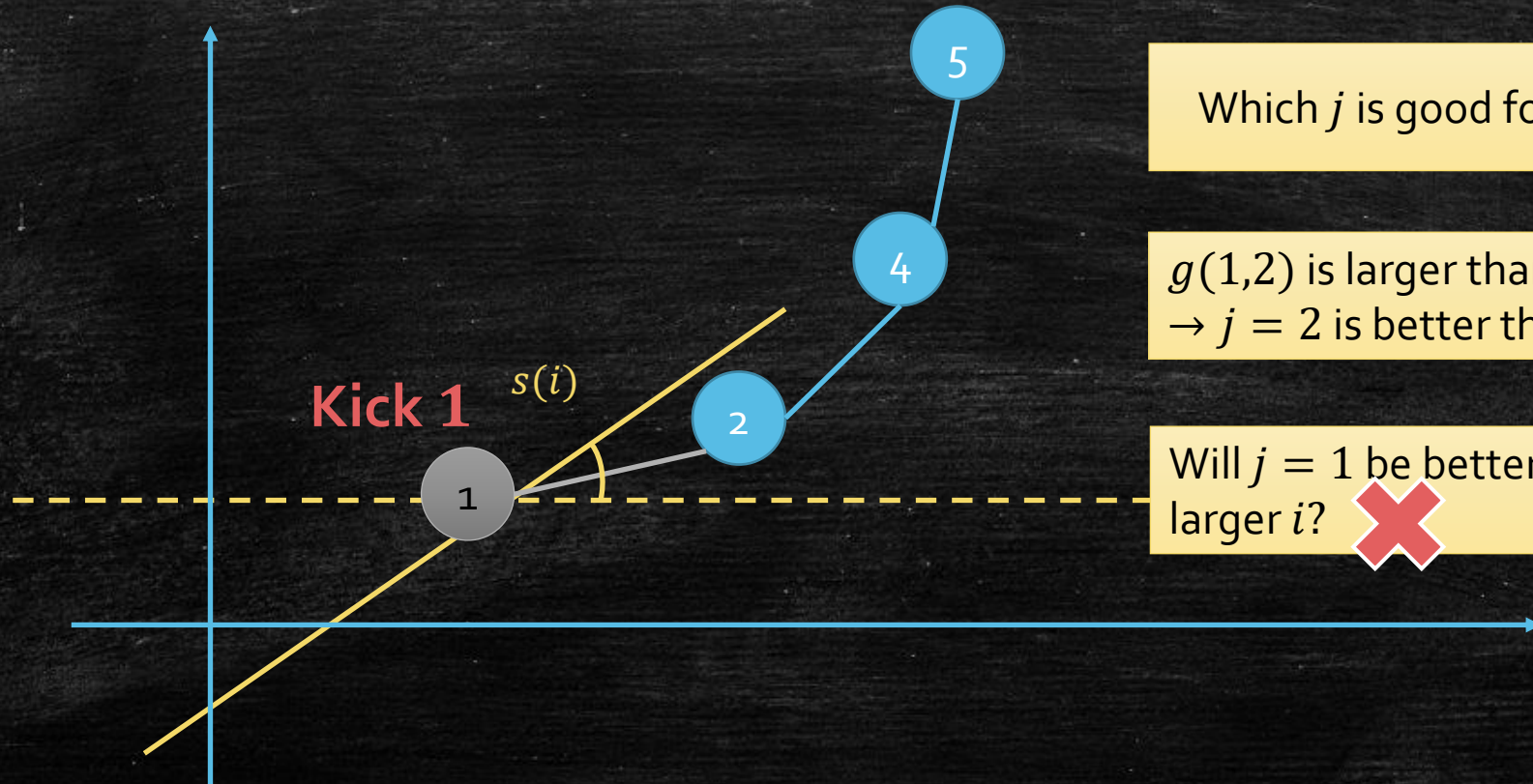
Which j is good for $i = 6$?

$g(1,2)$ is larger than $s(i)$
 $\rightarrow j = 2$ is better than $j = 1$.

Will $j = 1$ be better again for larger i ?

After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



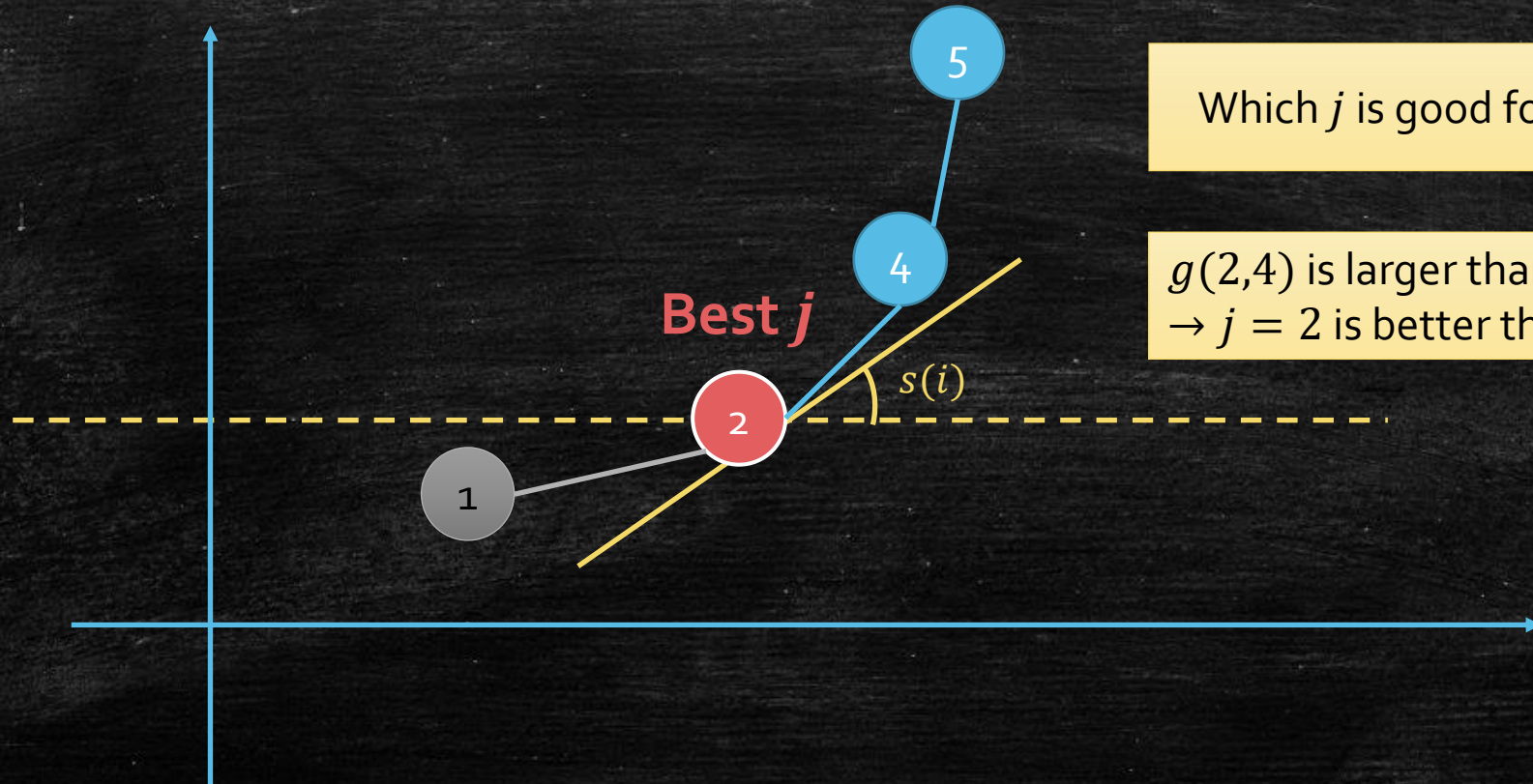
Which j is good for $i = 6$?

$g(1,2)$ is larger than $s(i)$
 $\rightarrow j = 2$ is better than $j = 1$.

Will $j = 1$ be better again for larger i ? 

After Kicking: A Convex Hall.

$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$?			
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9



Which j is good for $i = 6$?

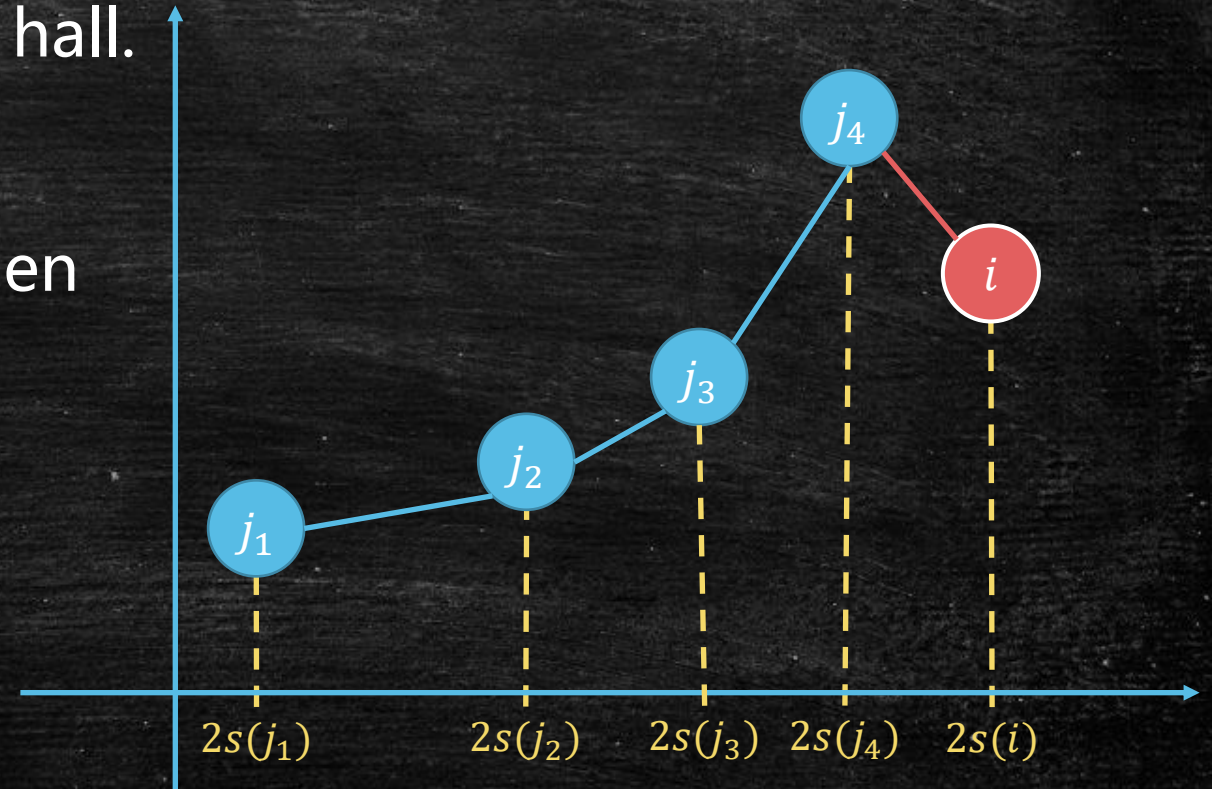
$g(2,4)$ is larger than $s(i)$
 $\rightarrow j = 2$ is better than $j = 4$.

Algorithm for updating $f[i]$.

- Let j_1, j_2, \dots, j_m be the convex hull.
- Loop from $k = 1$
- While $g(j_k, j_{k+1}) \leq s(i)$ then
 - kick j_k
 - $k++$
- Until $g(j_k, j_{k+1}) > s(i)$
- j_k is the **best!**
- $f[i] = f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2 = f[j] + C + (s(i) - s(j))^2$

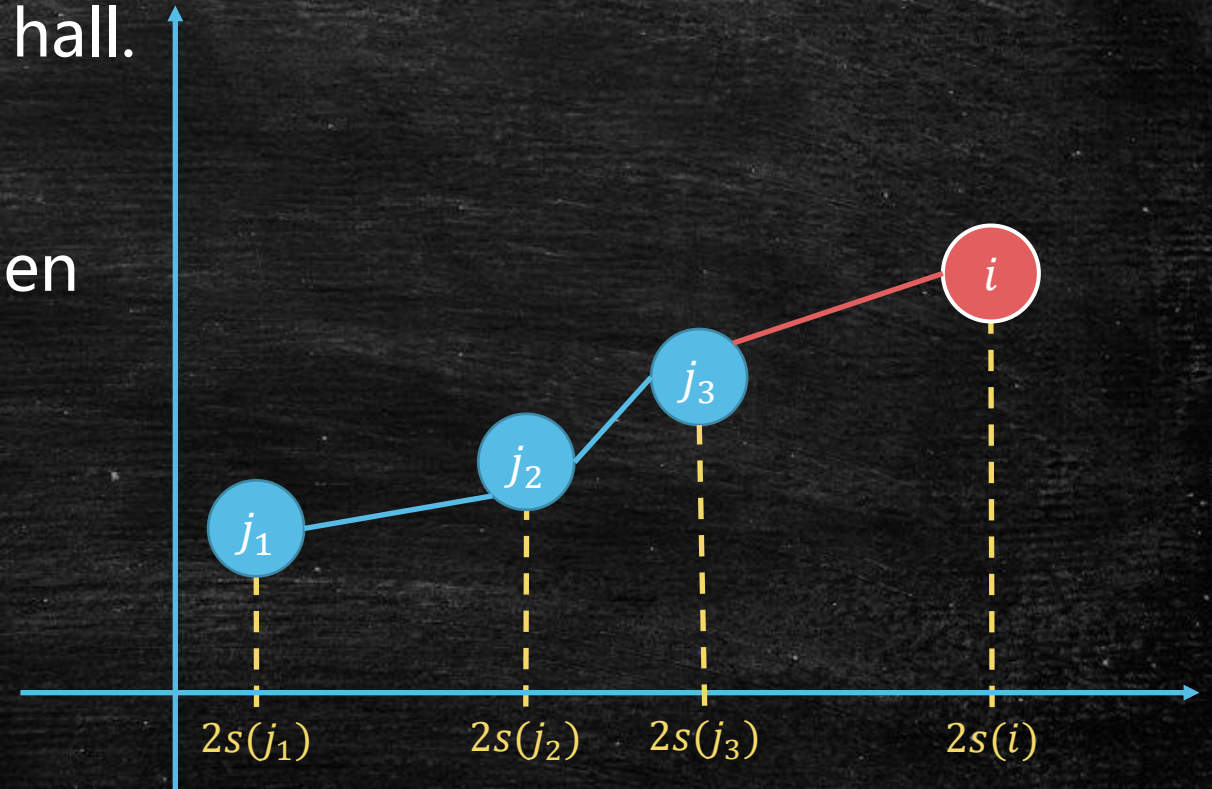
Algorithm for updating the convex hall

- Let j_1, j_2, \dots, j_m be the convex hall.
- Loop from $k = m$
- While $g(j_{k-1}, j_k) \geq g(j_k, i)$ then
 - kick j_k
 - $k--$
- Until $g(j_k, j_{k+1}) < g(j_k, i)$
- $j_{k+1} \leftarrow i$



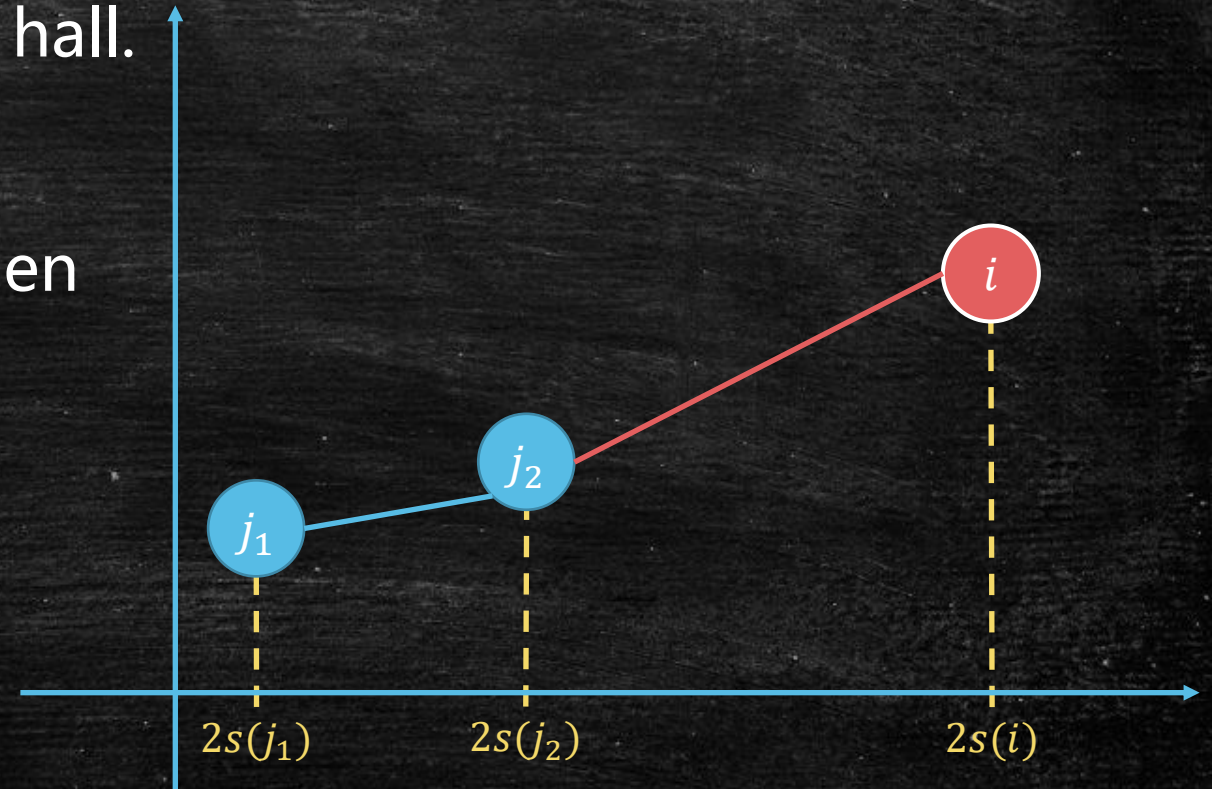
Algorithm for updating the convex hull

- Let j_1, j_2, \dots, j_m be the convex hull.
- Loop from $k = m$
- While $g(j_{k-1}, j_k) \geq g(j_k, i)$ then
 - kick j_k
 - $k--$
- Until $g(j_k, j_{k+1}) < g(j_k, i)$
- $j_{k+1} \leftarrow i$



Algorithm for updating the convex hull

- Let j_1, j_2, \dots, j_m be the convex hull.
- Loop from $k = m$
- While $g(j_{k-1}, j_k) \geq g(j_k, i)$ then
 - kick j_k
 - $k --$
- Until $g(j_k, j_{k+1}) < g(j_k, i)$
- $j_{k+1} \leftarrow i$



The DP Algorithm

- Complete the DP
 - $f[0] = 0$
 - For $i = 1$ to n
 - Calculate $f[i]$ from the potential convex hull.
 - Insert i into the convex hull.

Running time?

- Complete the DP

- $f[0] = 0$

- For $i = 1$ to n

- Calculate $f[i]$ from the potential convex hull.
 - Insert i into the convex hull.

Algorithm for updating $f[i]$.

- Let j_1, j_2, \dots, j_m be the con

Charge to i

1. Loop from $k = 1$

2. While $g(j_k, j_{k+1}) \leq s(i)$ then

- kick j_k
- $k++$

Charge to j_k

3. Until $g(j_k, j_{k+1}) > s(i)$

4. j_k is the **best**!

Charge to i

$$5. f[i] = f[j] + C + \left(\sum_{k=j+1}^i a_k\right)^2 = f[j] + C + (s(i) - s(j))^2$$

Algorithm for updating the convex hull

- Let j_1, j_2, \dots, j_m be the co

Charge to j_k

Loop from $k = m$

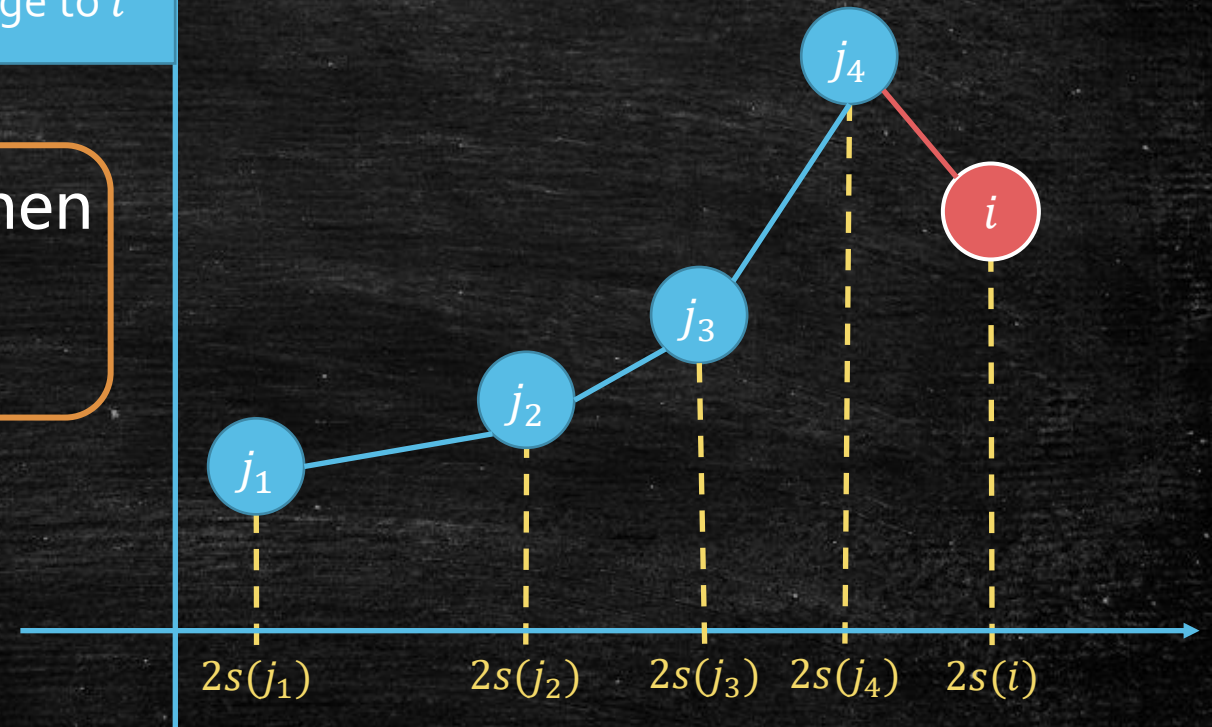
Charge to i

2. While $g(j_{k-1}, j_k) \geq g(j_k, i)$ then
- kick j_k
- $k--$

3. Until $g(j_k, j_{k+1}) < g(j_k, i)$

4. $j_{k+1} \leftarrow i$

Charge to i



Running time?

- Complete the DP

- $f[0] = 0$

- For $i = 1$ to n

- Calculate $f[i]$ from the potential convex hall.

- Insert i into the convex hall.

- Total Charged Cost for Each i

- When it is kicked \rightarrow once

- When it is calculated \rightarrow once

- When it is inserted \rightarrow once

Product of Sets

- **Input:** n sets L_1, L_2, \dots, L_n .
- **Output:** The minimum number of operations to calculate $L_1 \times L_2 \dots \times L_n$.
- What is $L_1 \times L_2$?
- $L_1 = \{a, b, c\}, L_2 = \{x, y\}$
- $L_1 \times L_2 = \{(a, x), (a, y), (b, x), (b, y), (c, x), (c, y)\}$
- **General Form:** $L_1 \times L_2 = \{(a, b) \mid a \in L_1, b \in L_2\}$

What is the cost of different calculation?

- We want $L_1 \times L_2 \times L_3$
- Two different calculations
 - $(L_1 \times L_2) \times L_3$
 - $L_1 \times (L_2 \times L_3)$
- Two different costs
 - $m_1m_2 + m_1m_2m_3$
 - $m_1m_2m_3 + m_2m_3$
- $m_1, m_2, m_3, \dots, m_n$ are crucial!

Can you design a DP for it?

A simple DP

- **Subproblem:** $c(i, j)$ is the cost for calculating $L_i \times L_2 \dots \times L_j$.
- How to update $c(i, j)$?
 - Case 1: $c(i, j) = c(i + 1, j) + m_i m_{i+1} \dots m_j$
 - Case 2: $c(i, j) = c(i, i + 1) + c(i + 2, j) + m_i m_{i+1} \dots m_j$
 - ...
 - Case ?: $c(i, j) = c(i, j - 1) + m_i m_{i+1} \dots m_j$
 - Case k : $c(i, j) = c(i, k) + c(k + 1, j) + m_i m_{i+1} \dots m_j$
- **Transfer:** $c(i, j) = m_i m_{i+1} \dots m_j + \min_{i \leq k < j} \{c(i, k) + c(k + 1, j)\}$
- Remark: we use $w(i, j)$ to denote $m_i m_{i+1} \dots m_j$.

Running Time

- What about the running time?
- n^2 subproblems, we use n iterations to calculate each.
- $O(n^3)$
- The topological order.

$c(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$					
$i = 2$					
$i = 3$					
$i = 4$					
$i = 5$					

Improvement!

F. Frances Yao STOC 1980

Key Property We Observe

- **DP formula**

- $c(i, j) = w(i, j) + \min_{i \leq k < j} \{c(i, k) + c(k + 1, j)\}$

- **Weight function**

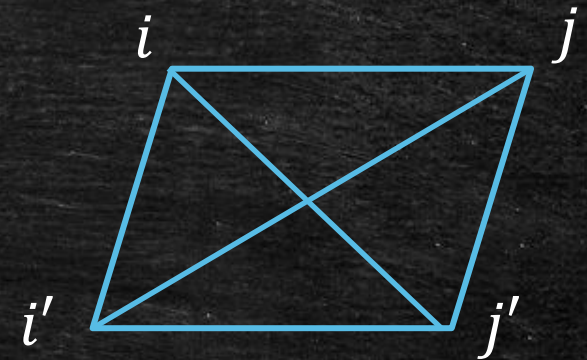
- $w(i, j) = m_i m_{i+1} \dots m_j$

- **Quadrangle Inequality (QI)**

- $\forall i \leq i' \leq j \leq j', w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$

- **Monotonicity**

- $\forall i \leq i' \leq j \leq j', w(i', j) \leq w(i, j')$



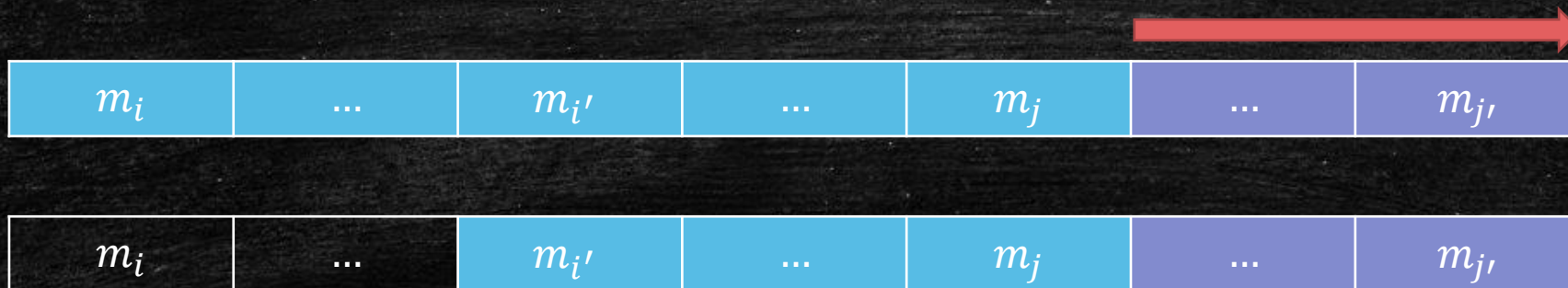
Understand QI and Monotonicity

- **Monotonicity**

- $\forall i \leq i' \leq j \leq j', w(i', j) \leq w(i, j')$
- The weight function is increasing.

- **Quadrangle Inequality (QI)**

- $\forall i \leq i' \leq j \leq j', w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$
- $w(i, j') - w(i, j) \geq w(i', j') - w(i', j)$
- Larger size w increase larger.



Check $w(i, j)$

- Monotonicity

- $\forall i \leq i' \leq j \leq j', w(i', j) \leq w(i, j')$

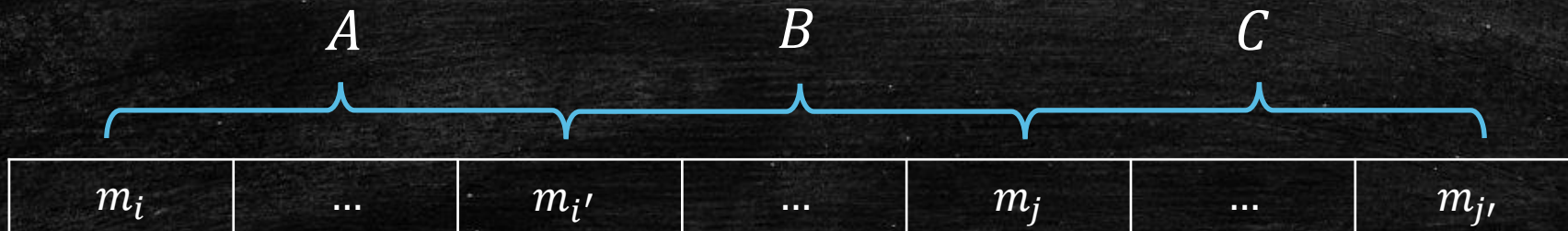
- Quadrangle Inequality (QI)

- $\forall i \leq i' \leq j \leq j', w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$

- Prove QI

- $AB + BC \leq B + ABC$

- $AB(C - 1) \geq B(C - 1)$

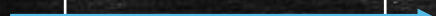






QI for w implies QI for c

- It implies.....

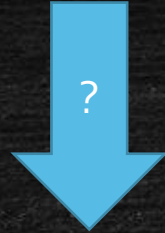
- Quadrangle Inequality (QI) for $c(i, j)$

$$\forall i \leq i' \leq j \leq j', \quad c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$$

$c(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$			$c(i, j)$		$c(i, j')$
$i = 2$					
$i = 3$			$c(i', j)$		$c(i', j')$
$i = 4$					
$i = 5$					

Hold on

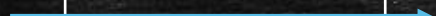




Quadrangle Inequality (QI): $w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$.
Monotonicity: $w(i', j) \leq w(i, j')$



Quadrangle Inequality (QI) for $c(i, j)$
 $\forall i \leq i' \leq j \leq j', \quad c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$

Assume we have it!

- Quadrangle Inequality (QI) for $c(i, j)$
 - $\forall i \leq i' \leq j \leq j', c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$
- How to design a better DP?

$c(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$			$c(i, j)$		$c(i, j')$
$i = 2$					
$i = 3$			$c(i', j)$		$c(i', j')$
$i = 4$					
$i = 5$					

Go back to the potential idea.

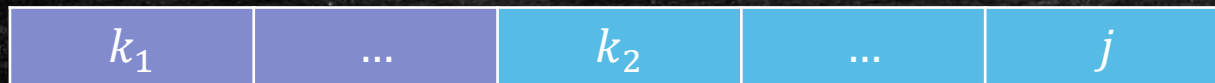
What is the best k for $c(i, j)$?

- Consider again when k_2 is better than k_1 .
- $c(i, k_1) + c(k_1 + 1, j) > c(i, k_2) + c(k_2 + 1, j)$
- $c(k_1 + 1, j) - c(k_2 + 1, j) > c(i, k_2) - c(i, k_1)$
- Fix i , which one is better depends on
 - $c(k_1 + 1, j) - c(k_2 + 1, j) > c(i, k_2) - c(i, k_1)$
- Fix j , which one is better depends on
 - $c(i, k_2) - c(i, k_1) < c(k_1 + 1, j) - c(k_2 + 1, j)$

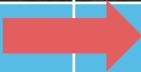


The Monotonicity when Comparing k_1, k_2

- The condition of k_2 is better than k_1
 - $c(k_1 + 1, j) - c(k_2 + 1, j) > c(i, k_2) - c(i, k_1)$
- Fix i , consider what happens for different j .
 - $c(k_1 + 1, j) - c(k_2 + 1, j)$ is non-decreasing on j !



The Monotonicity when Comparing k_1, k_2

$c(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$			k_1, k_2 	k_1, k_2	
$i = 2$					
$i = 3$					
$i = 4$					
$i = 5$					

- $c(k_1 + 1, j) - c(k_2 + 1, j)$ is non-decreasing on j !
- If k_2 is better than k_1 at $j = 3$, it is also better at $j = 4$.
- $c(i, k_2) - c(i, k_1) < c(k_1 + 1, j) - c(k_2 + 1, j) \leq c(k_1 + 1, j') - c(k_2 + 1, j')$

What does it mean

- The best k have some monotonicity w.r.t. j .
- Let $k(i, j)$ be the best k $c(i, j)$ selected.

Monotonicity for $k(i, j)$.

$k(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	Non-decreasing 				
$i = 2$					
$i = 3$					
$i = 4$					
$i = 5$					

- $c(i, j) = w(i, j) + \min_{k(i, j-1) \leq k < j} \{c(i, k) + c(k + 1, j)\}$
- Is that enough?

Is that enough?

- $c(i, j) = w(i, j) + \min_{k(i, j-1) \leq k < j} \{c(i, k) + c(k + 1, j)\}$
- No!
- The cost maybe $1 + 2 + 3 + 4 + \dots + n - 1 + n$ for each row.
- Still $O(n^3)$!

Monotonicity for $k(i, j)$.

$k(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$					
$i = 2$					
$i = 3$			k'	k	
$i = 4$				k''	
$i = 5$					





Non-decreasing

- $c(i, k_2) - c(i, k_1)$ is non-increasing on i .
- k_2 is better? $c(i, k_2) - c(i, k_1) < c(k_1 + 1, j) - c(k_2 + 1, j)$

Monotonicity for $k(i, j)$.

$k(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$					
$i = 2$					
$i = 3$			k'	k	
$i = 4$				k''	
$i = 5$					

Non-decreasing

- $c(i, k_2) - c(i, k_1)$ is non-increasing on i .
- k_2 is better? $c(i, k_2) - c(i, k_1) < c(k_1 + 1, j) - c(k_2 + 1, j)$
- If k_2 is better at i , then it is still better at $i' > i$.

A new DP approach

$k(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$					
$i = 2$					
$i = 3$			k_1	$k_1 \leq k \leq k_2$	
$i = 4$				k_2	
$i = 5$					

- $c(i, j) = w(i, j) + \min_{k(i, j-1) \leq k < k(i+1, j)} \{c(i, k) + c(k+1, j)\}$
- What is a good order for this approach?

A new DP approach

$k(i, j)$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$					
$i = 2$					
$i = 3$			k_1	$k_1 \leq k \leq k_2$	
$i = 4$				k_2	
$i = 5$					

- $c(i, j) = w(i, j) + \min_{k(i, j-1) \leq k < k(i+1, j)} \{c(i, k) + c(k+1, j)\}$
- We know k_1 and k_2 in the topological order!

DP algorithm

- Initialize $c[i, i] = 0$ for all i
- For $\Delta = 1$ to $n - 1$
 - For $i = 1$ to $n - \Delta$
 - $j = i + \Delta$
 - $c[i, j] = w(i, j) + \min\{c(i, k) + c(k + 1, j)\}$.
 - Searching range k from $k(i, j - 1)$ to $k(i + 1, j)$
 - $k(i, j) \leftarrow$ the best k

Running Time

- Initialize $c[i, i] = 0$ for all i
- For $\Delta = 1$ to $n - 1$
 - For $i = 1$ to $n - \Delta$
 - $j = i + \Delta$
 - $c[i, j] = w(i, j) + \min\{c(i, k) + c(k + 1, j)\}$.
 - Searching range k from $k(i, j - 1)$ to $k(i + 1, j)$
 - $k(i, j) \leftarrow$ the best k
- $Time = \sum_{i=1}^{n-\Delta} k(i + 1, i + \Delta) - k(i, i + \Delta - 1)$
 $= k(n - \Delta + 1, n) - k(1, \Delta) \leq n$

Don't forget to check why QI is correct for c .

Prove QI for c

- **Given**

- $w(i, j) = m_i m_{i+1} \dots m_j$
- **Quadrangle Inequality (QI)**
 - $\forall i \leq i' \leq j \leq j', w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$
- **Monotonicity**
 - $\forall i \leq i' \leq j \leq j', w(i', j) \leq w(i, j')$

- **To prove**

- $c(i, j) = w(i, j) + \min_{i \leq k < j} \{c(i, k) + c(k + 1, j)\}$
- **Quadrangle Inequality (QI)**
 - $\forall i \leq i' \leq j \leq j', c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$

Prove by Induction

- Prove by Induction with $(j' - i)$
- **Base case:** $(i = j')$: easy to check.
- **Inductive:** $j' - i = \Delta$
 - Hypothesis
 - $\forall j' - i < \Delta, i \leq i' \leq j \leq j', c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$
 - To prove
 - $\forall j' - i = \Delta, i \leq i' \leq j \leq j', c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$

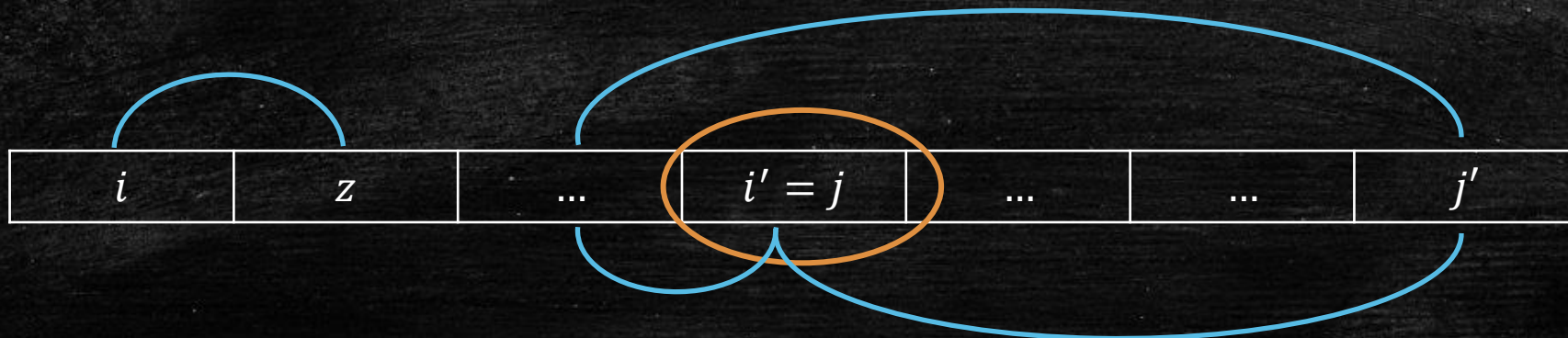
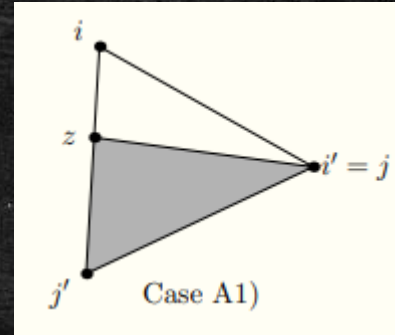
A special case: $i' = j$

- Quadrangle Inequality (QI) for $c(i, j)$

$$\forall i \leq i' \leq j \leq j', \quad c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$$

- To prove $c(i, j) + c(j, j') \leq c(i, j')$
- $c(i, j')$ is minimized at a point in $[i, j']$.
- $c(i, j') = c(i, z) + c(z + 1, j') + w(i, j')$

$$\geq c(i, z) + \mathbf{c(z + 1, j)} + \mathbf{c(j, j')} + w(i, j')$$

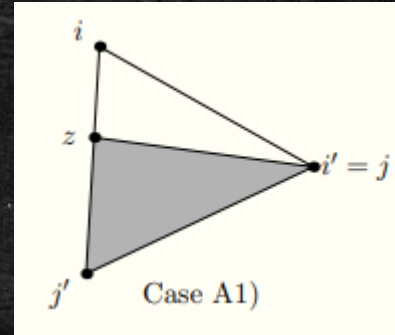


A special case: $i' = j$

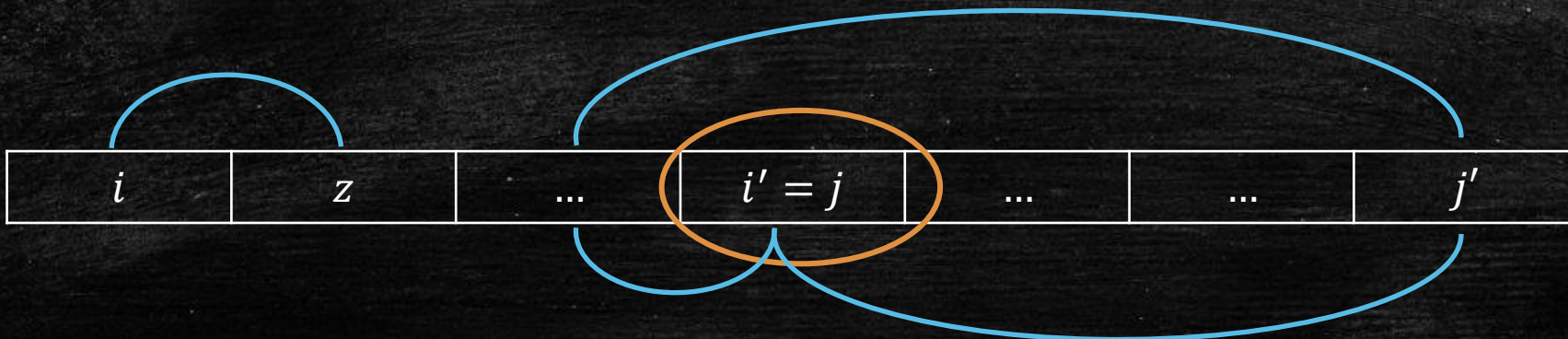
- Quadrangle Inequality (QI) for $c(i, j)$

$$\forall i \leq i' \leq j \leq j', \quad c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$$

- To prove $c(i, j) + c(j, j') \leq c(i, j')$
- $c(i, j')$ is minimized at a point in $[i, j']$.
- $c(i, j') = c(i, z) + c(z + 1, j') + w(i, j')$



$$\geq \mathbf{c(i, z)} + \mathbf{c(z + 1, j)} + c(j, j') + \mathbf{w(i, j')} \geq c(i, j) + c(j, j')$$



General case: $i' \neq j$

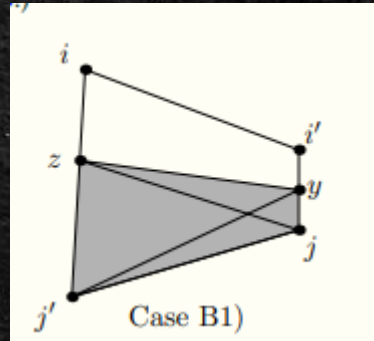
- Quadrangle Inequality (QI) for $c(i, j)$

$$\forall i \leq i' \leq j \leq j', \quad c(i, j) + c(i', j') \leq c(i', j) + c(i, j')$$

- $c(i, j') = c(i, z) + c(z + 1, j') + w(i, j')$

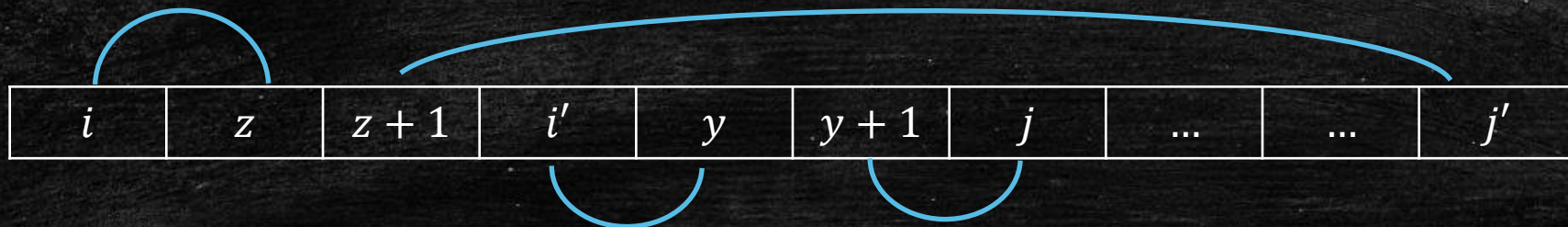
- $c(i', j) = c(i', y) + c(y + 1, j) + w(i', j)$

- $c(i, j') + c(i', j) = c(i, z) + c(z + 1, j') + w(i, j') \\ + c(i', y) + c(y + 1, j) + w(i', j)$



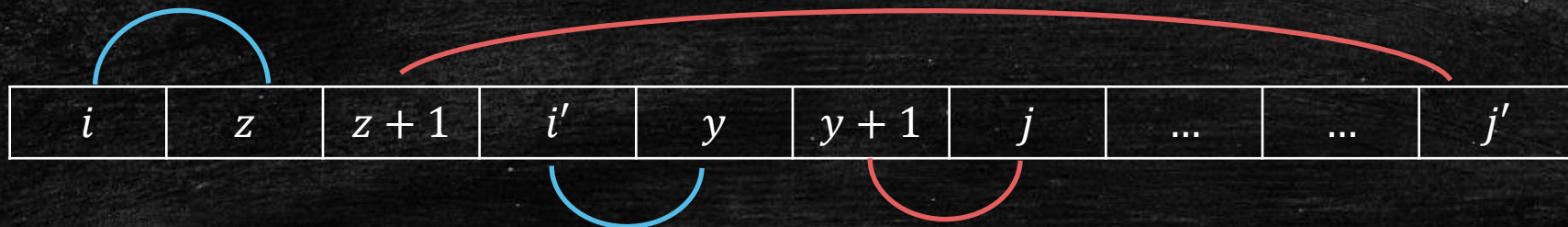
General case: $i' \neq j$

- $$c(i, j') + c(i', j) = c(i, z) + c(z + 1, j') + w(i, j') \\ + c(i', y) + c(y + 1, j) + w(i', j)$$



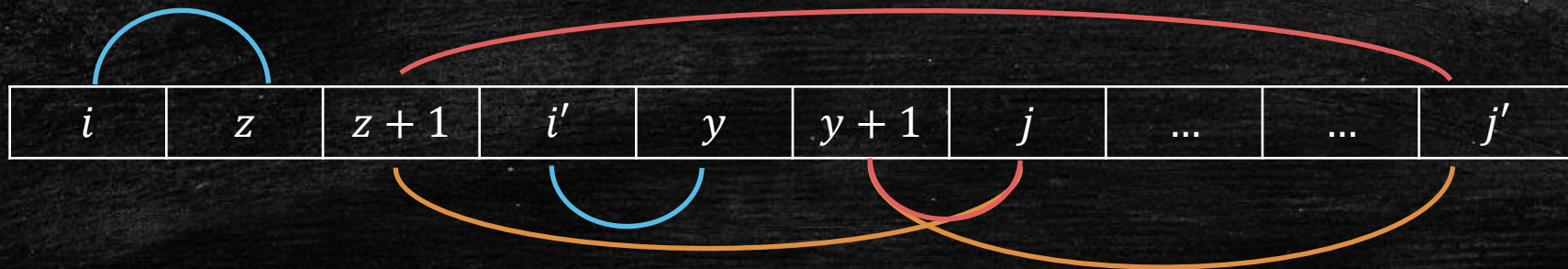
General case: $i' \neq j$

- $c(i, j') + c(i', j) = c(i, z) + c(z + 1, j') + w(i, j')$
 $+ c(i', y) + c(y + 1, j) + w(i', j)$



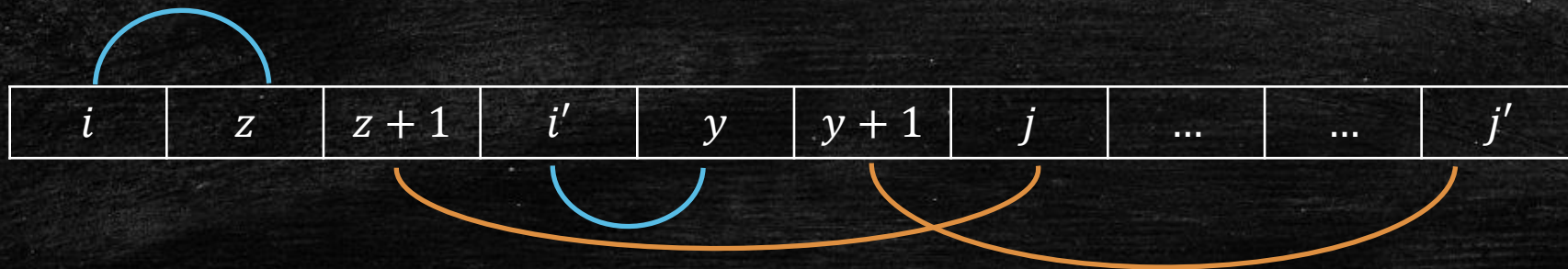
General case: $i' \neq j$

- $c(i, j') + c(i', j) = c(i, z) + c(z + 1, j') + w(i, j')$
 $+ c(i', y) + c(y + 1, j) + w(i', j)$
- Inductive Hypothesis
 - $c(z + 1, j') + c(y + 1, j) \geq c(z + 1, j) + c(y + 1, j')$



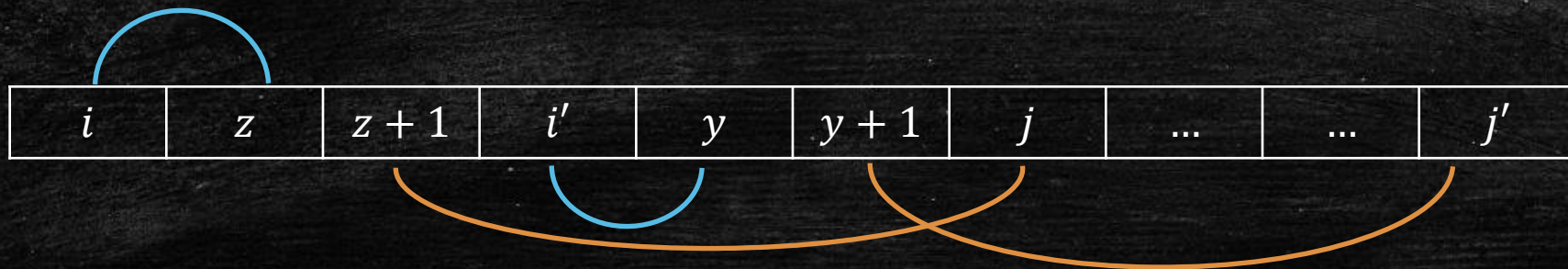
General case: $i' \neq j$

- $c(i, j') + c(i', j) \geq c(i, z) + c(z + 1, j) + w(i, j')$
 $+ c(i', y) + c(y + 1, j') + w(i', j)$
- Inductive Hypothesis
 - $c(z + 1, j') + c(y + 1, j) \geq c(z + 1, j) + c(y + 1, j')$



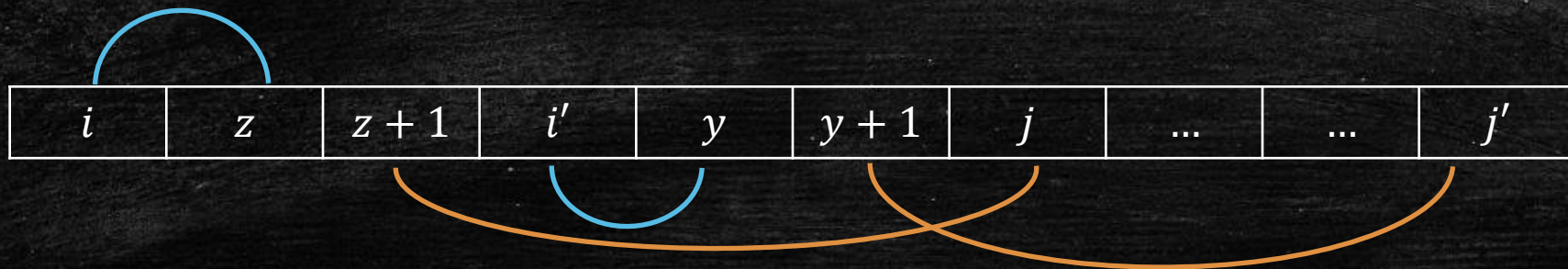
General case: $i' \neq j$

- $c(i, j') + c(i', j) \geq c(i, z) + c(z + 1, j) + w(i, j')$
 $+ c(i', y) + c(y + 1, j') + w(i', j)$
- Inductive Hypothesis
 - $c(z + 1, j') + c(y + 1, j) \geq c(z + 1, j) + c(y + 1, j')$
- QI of w
 - $w(i, j') + w(i', j) \geq w(i, j) + w(i', j')$



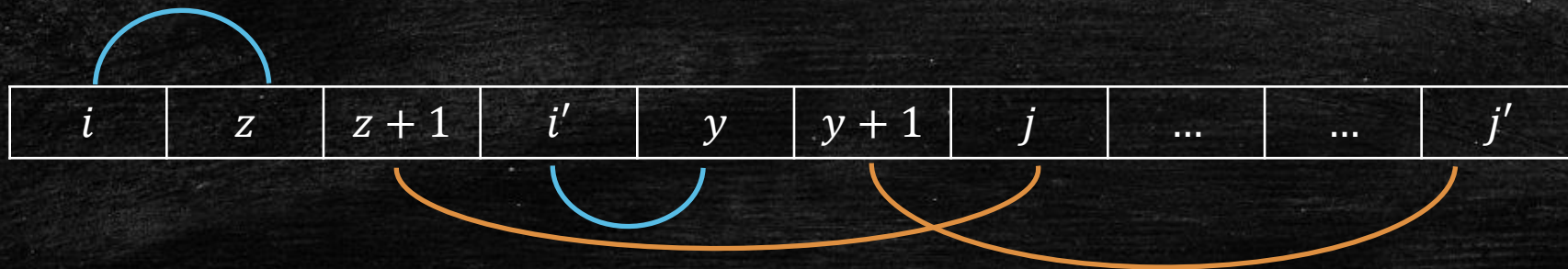
General case: $i' \neq j$

- $c(i, j') + c(i', j) \geq c(i, z) + c(z + 1, j) + w(i, j)$
 $+ c(i', y) + c(y + 1, j') + w(i', j')$
- Inductive Hypothesis
 - $c(z + 1, j') + c(y + 1, j) \geq c(z + 1, j) + c(y + 1, j')$
- QI of w
 - $w(i, j') + w(i', j) \geq w(i, j) + w(i', j')$



General case: $i' \neq j$

- $c(i, j') + c(i', j) \geq c(i, z) + c(z + 1, j) + w(i, j) \geq c(i, j)$
 $+ c(i', y) + c(y + 1, j') + w(i', j') \geq c(i', j')$
- Inductive Hypothesis
 - $c(z + 1, j') + c(y + 1, j) \geq c(z + 1, j) + c(y + 1, j')$
- QI of w
 - $w(i, j') + w(i', j) \geq w(i, j) + w(i', j')$



Today's goal

- Recap the **Guideline** of DP! (Most Important)
- Learn how to improve DP by **Priority Queue**!
- Learn the tool: **Priority Queue**.
- Example
 - Largest Number in k Consecutive Numbers
 - Longest Increasing Sequence
 - Minimizing Printing Cost