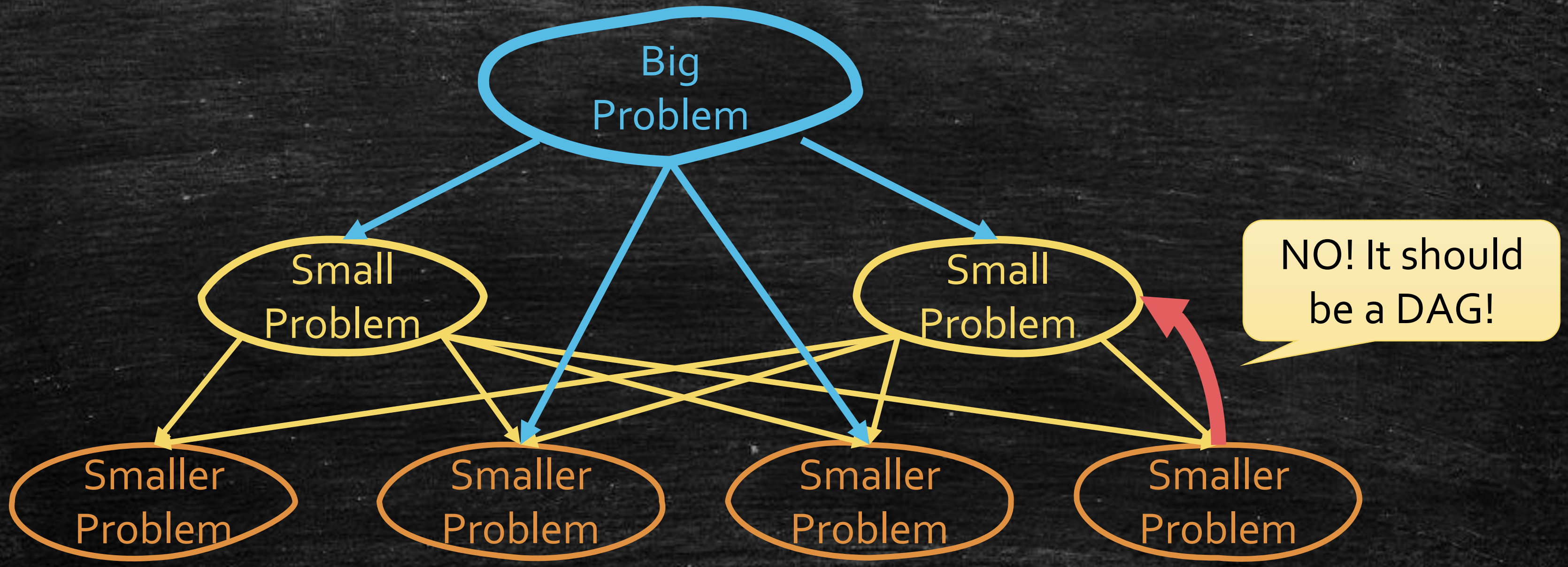
Dynamic Programming

Smarter Subproblem Definitions

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 $f[i, w]$.

Understand Bellman-Ford as A DP

Bellman-Ford

Function bellman_ford(G, s)
    $dist[s] = 0, dist[x] = \infty$ for other $x \in V$
    while $\exists dist[x]$ is updated
        for each $(u, v) \in E$
            $dist[v] = \min\{dist[v], dist[u] + d(u, v)\}$

Lemma 1

After k rounds, $dist(v)$ is the shortest distance of all **k-edge-path (path with at most k edges)**.

Define subproblems

- $dist[k, v]$: the shortest distance from s to v among all ***k*-edge-path (path with at most k edges)**.

Observation 2

The shortest distance of all **$|V|$-edge-path** can not be shorter than the shortest distance of all **$(|V|-1)$ –edge-path** unless there is a **Negative Cycle.**

Bellman-Ford

function bellman_ford(G, s)
    $dist[0, s] = 0, dist[0, x] = \infty$ for other $x \in V$
    for $k = 1$ to $|V|$
        for each $(u, v) \in E$
            $dist[k, v] = \min\{dist[k-1, v], dist[k-1, u] + d(u, v)\}$

Solving Subproblems

- $dist[k, v] = \min\{dist[k-1, v], dist[k-1, u] + d(u, v)\}$

$f[k, v]$	s	v_2	v_3	v_4	v_5	v_6	v_7	…	$v_{\|V\|}$
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1									
2									
3						$f[k, v]$			
…									
$\|V\|$									

All Pair Shortest Path

- **Input:** A directed graph $G(V,E)$, and a weighted function $d(u,v)$ for all $(u,v) \in E$.
- **Output:** Distance $d(u,v)$, for **all vertex pair** u, v.

What can we do?

- Naïve Plan:
  - Run $|V|$ times Bellman-Ford
  - $O(|V|^2|E|)$
- Improve it by an integrated DP!
  - Floyd-Warshall Algorithm!
  - $O(|V|^3)$
  - History from Wikipedia:

History and naming [edit]

The Floyd–Warshall algorithm is an example of dynamic programming, and was published in its currently recognized form by Robert Floyd in 1962.[3] However, it is essentially the same as algorithms previously published by Bernard Roy in 1959[4] and also by Stephen Warshall in 1962[5] for finding the transitive closure of a graph,[6] and is closely related to Kleene's algorithm (published in 1956) for converting a deterministic finite automaton into a regular expression.[7] The modern formulation of the algorithm as three nested for-loops was first described by Peter Ingerman, also in 1962.[8]

Define subproblems

- Bellman-Ford: $dist[k, v]$: the shortest distance from s to v among all **k-edge-path (path with at most k edges)**.
- A very natural generalization!
- Natural Generalization: $dist[k, u, v]$: the shortest distance from u to v among all **k-edge-path (path with at most k edges)**.

Natural Generalization

- Natural Generalization: $dist[k,u,v]$: the shortest distance from u to v among all **k-edge-path (path with at most k edges)**.
- Transfer:
  - $dist[k,u,v] = \min_{(s,v)\in E}\{dist[k-1,u,s] + d(s,v)\}$
- Time:
  - $|V|$ rounds
  - In one round, an edge can be used to update $|V|$ distance.
  - Totally $O(|V|^2|E|)$!

No improvement!

Solving Subproblems

- $dist[k, u, v] = \min_{(s,v) \in E} \{dist[k-1, u, s] + d(s, v)\}$

$\boldsymbol{k-1}$	v_1	v_2	v_3	…	$v_{\|V\|}$
v_1					
v_2					
v_3					
v_4					
…					
$v_{\|V\|}$					

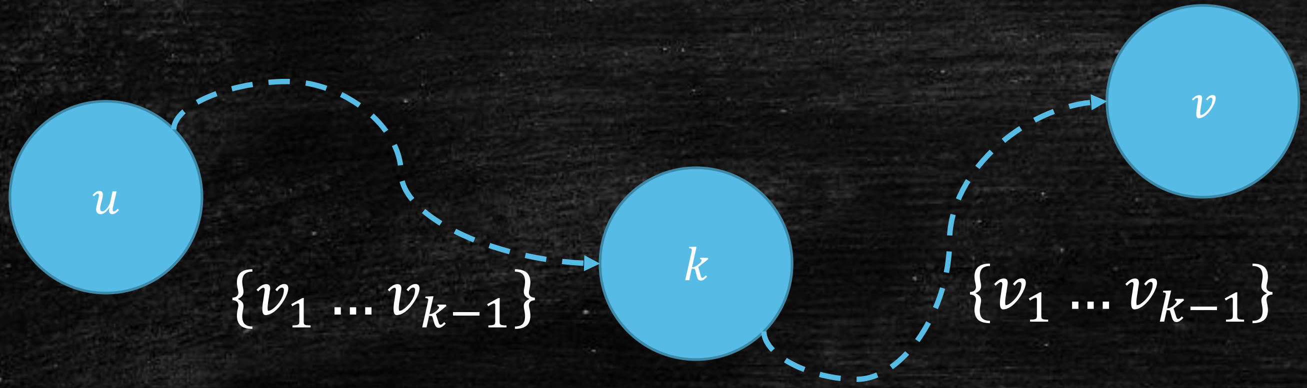
$\boldsymbol{k}$	v_1	v_2	v_3	…	$v_{\|V\|}$
v_1					
v_2					
v_3					
v_4			$f[k, u, v]$		
…					
$v_{\|V\|}$					

Floyd-Warshall: Subproblems

- Natural Generalization: $dist[k, u, v]$: the shortest distance from u to v among all **k-edge-path (path with at most k edges)**.
- Floyd-Warshall: $dist[k, u, v]$: the shortest distance from u to v that **only across inter-vertices in $\{v_1 \dots v_k\}$**.
- Remark:
  - We can label vertices from 1 to $|V|$.
  - $dist[0, u, v]$ is exactly $d(u, v)$ or ∞. (allow 0 inter-vertex)
  - $dist[|V|, u, v]$ is exactly what we want!

Floyd-Warshall: Solving Subproblems

- $dist[k, u, v]$: the shortest distance from u to v that only **across inter-vertices in $\{v_1 \dots v_k\}$.**
- Solve $dist[k, u, v]$ (give addition power k to all pairs)
  - Case 1: the shortest path do not go across k.
  - Case 2: the shortest path go across k.
  - $dist[k, u, v] = \min\{dist[k-1, u, v], dist[k-1, u, k] + dist[k-1, k, v]\}$



Solving Subproblems

- $dist[k, u, v] = \min\{dist[k-1, u, v], dist[k-1, u, k] + dist[k-1, k, v]\}$

$\boldsymbol{k-1}$	v_1	v_2	v_3	…	$v_{\lvert V \rvert}$
v_1					
v_2					
v_3					
v_4					
…					
$v_{\lvert V \rvert}$					

$\boldsymbol{k}$	v_1	v_2	v_3	…	$v_{\lvert V \rvert}$
v_1					
v_2					
v_3					
v_4			$f[k, u, v]$		
…					
$v_{\lvert V \rvert}$					

DAG and Topological

- $dist[k, u, v]$ only depends
  - $dist[k-1, u, v]$
  - $dist[k-1, u, k]$
  - $dist[k-1, k, v]$
- We initialize $dist[0, u, v] = d(u, v)$ for all (u, v).
- Solve them from $k = 1$ to n is a topological order.
- Running Time: $3 \cdot O(|V| \cdot |V| \cdot |V|)$

Floyd-Warshall

Floyd-Warshall $O(|V|^3)$

function floyd_warshall(G)
    $dist[0, u, v] = d(u, v)$ for all $(u, v) \in E$, $dist[0, u, v] = \infty$ otherwise.
    for $k = 1$ to $|V|$
        for $u = 1$ to $|V|$
            for $v = 1$ to $|V|$
                $dist[k, u, v] = \min\{dist[k-1, u, v], dist[k-1, u, k] + dist[k-1, k, v]\}$

Floyd-Warshall: a simpler implement

Floyd-Warshall

function floyd_warshall(G)
    $dist[u,v] = d(u,v)$ for all $(u,v) \in E$, $dist[u,v] = \infty$ otherwise.
    for $k = 1$ to $|V|$
        for $u = 1$ to $|V|$
            for $v = 1$ to $|V|$
                $dist[u,v] = \min\{dist[u,v], dist[u,k] + dist[k,v]\}$

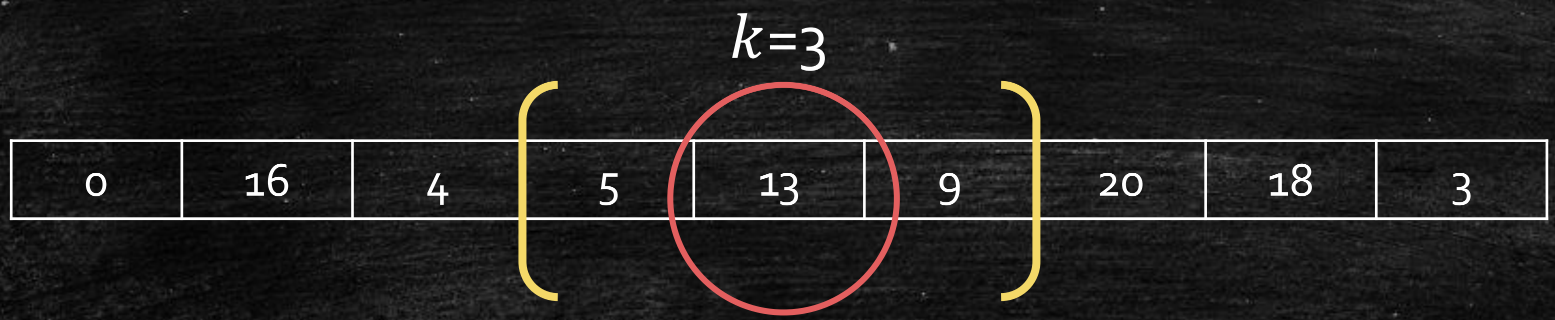
$O(|V|^3)$ running time but $O(|V|^2)$ space! Why it is correct?

More Smarter Subproblem Definitions

Priority Queue

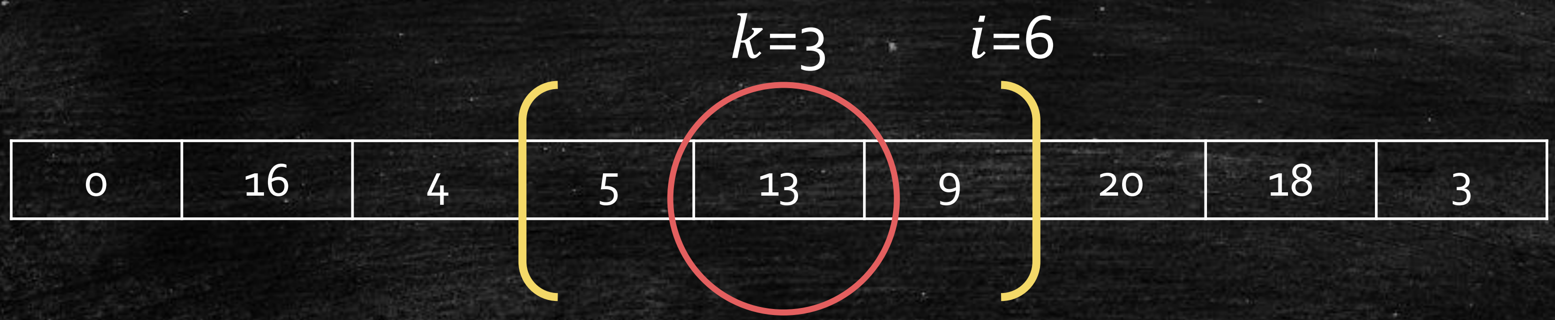
Largest Number in k Consecutive Numbers

- **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: $large[k]$~$large[n]$.



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?
  - Tips: from $large[j], j < i$.

k=3 i=6

0	16	4	5	13	9	20	18	3

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?
  - Tips: from $large[j], j < i$.
  - Brute-force: $large[i] = \max_{j=i-k+1}^{i}\{a_i\}$

k=3 i=6

0	16	4	5	13	9	20	18	3

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 budge $f[i]$ uses.~~

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

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

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

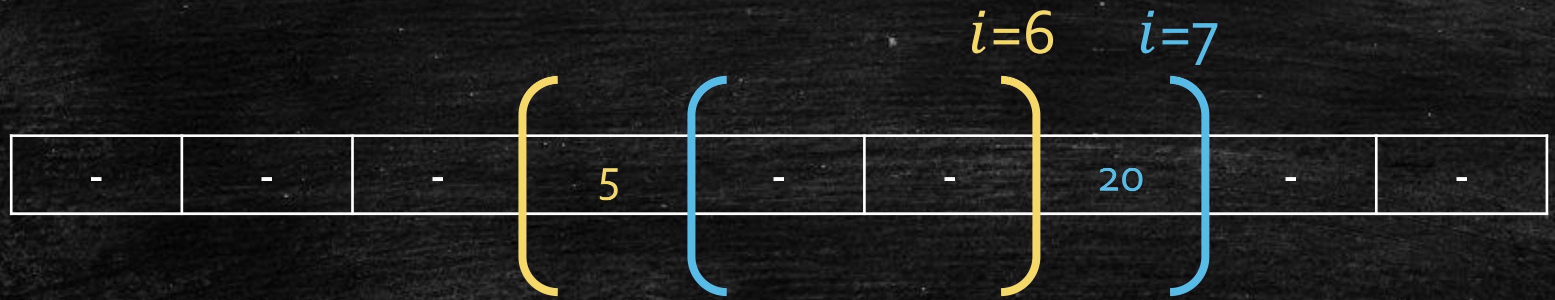
- Compare two $large[i]$ and $large[i-1]$.
- Difference
  - One entering number: 20
  - One outgoing number: 5
  - Question: how they affect the largest number?

i=6 i=7

0	16	4	5	13	9	20	18	3

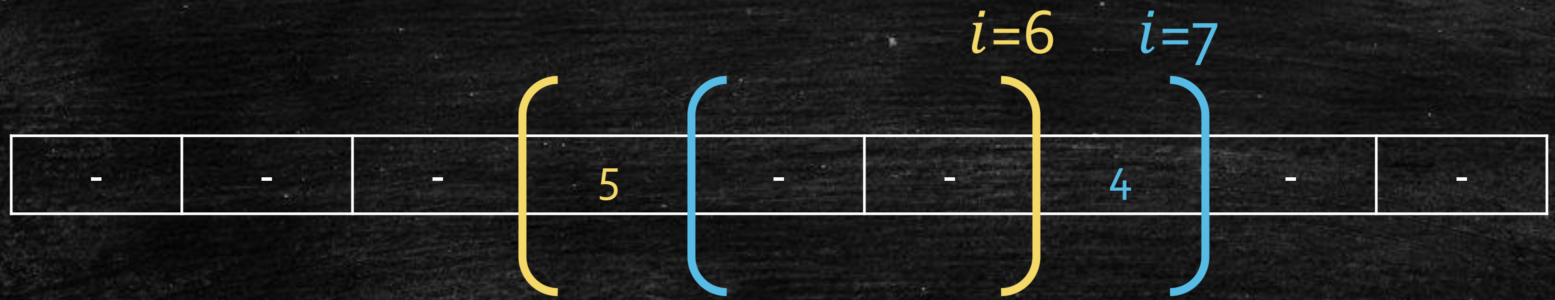
How they affect the largest number

- 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!

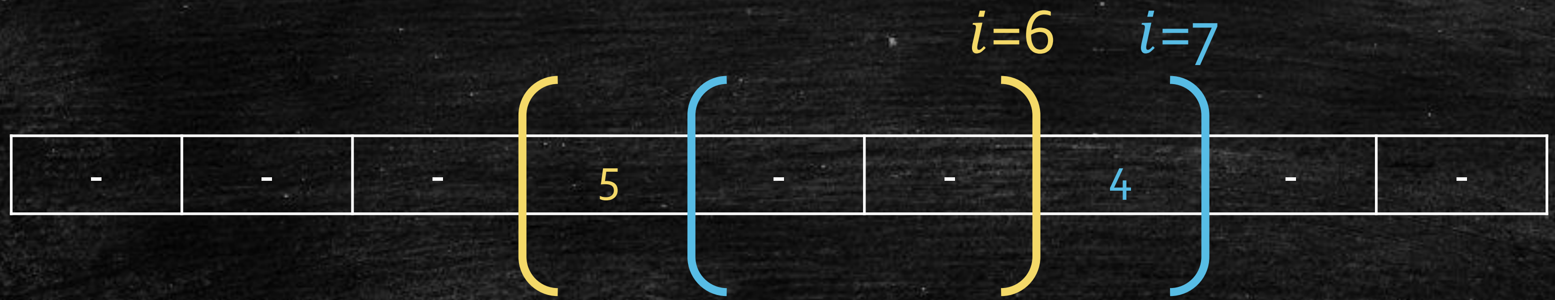


Key problem: We should know what is the previous second largest number.

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!

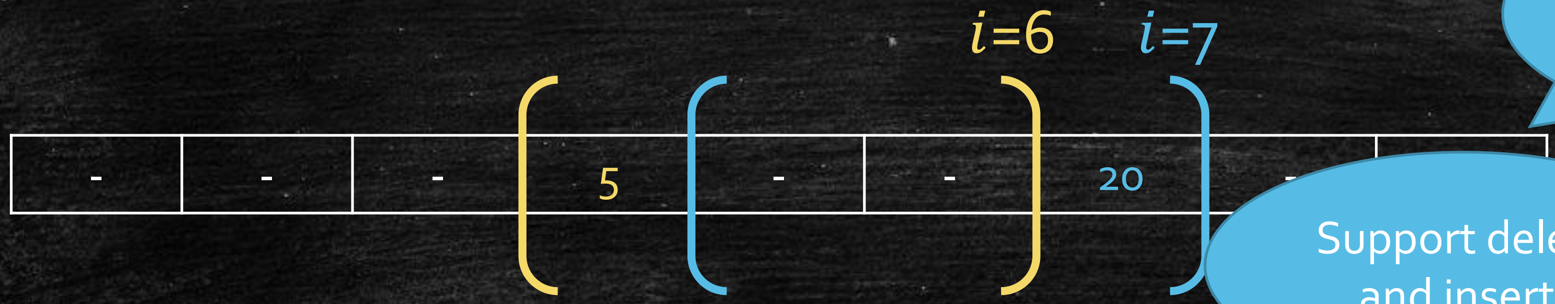


Key problem: We should know what is the previous third largest number.

Ok, let us record it.....

Summarize

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



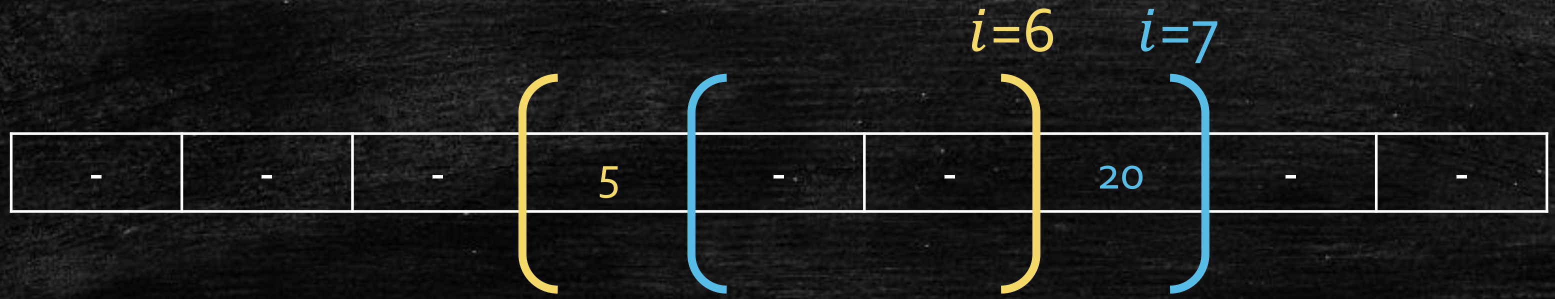
Data Structure
$O(n \log k)$!

Support delete
and insert!

Summarize: We should maintain a data structure!

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 it too powerful?
  - We delete and insert only based on the index!



A new Subproblem!

- Think again: why we need the heap?
  - We need two 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

i=6

0	16	4	5	13	9	20	18	3

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.

i=6

0	16	4	5	13	9	20	18	3

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.
  - $i-k+1 \le \text{Index} \le i$

$PLL[20]$

Smaller

a_5	a_6	a_8	a_{12}	a_{13}	a_{15}	a_{16}	a_{19}	a_{20}

$\le k$=15

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

How to maintain PLL?

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

$PLL[20]$

Smaller

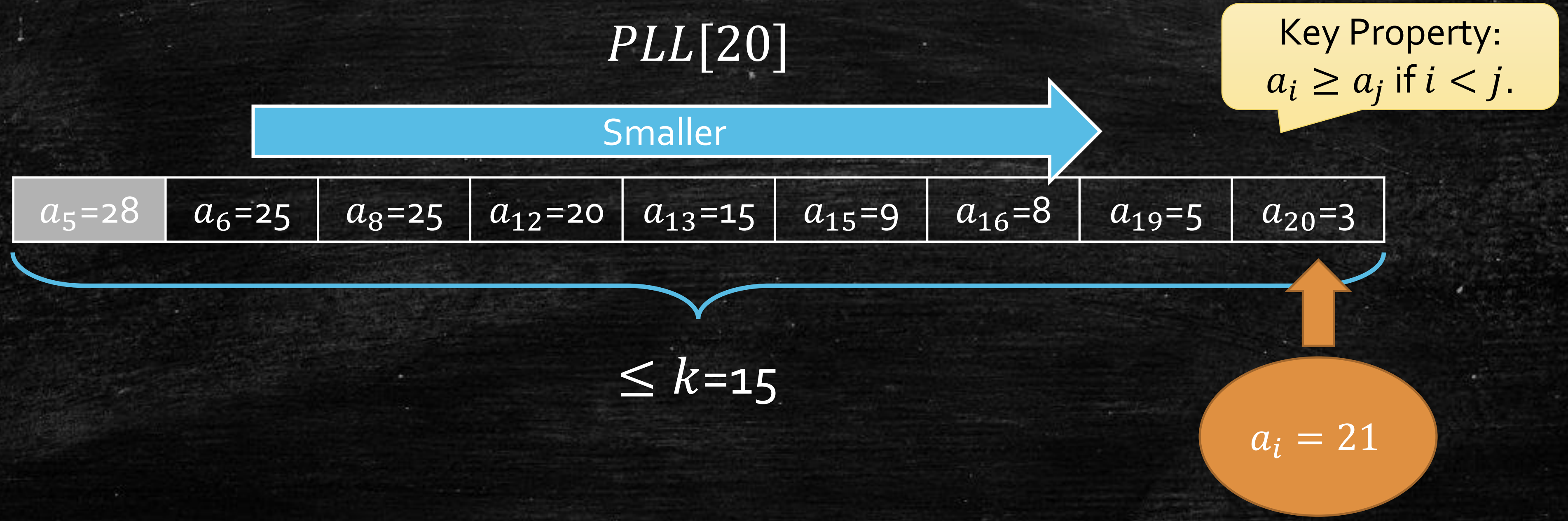
a_5	a_6	a_8	a_{12}	a_{13}	a_{15}	a_{16}	a_{19}	a_{20}

$\leq k$=15

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

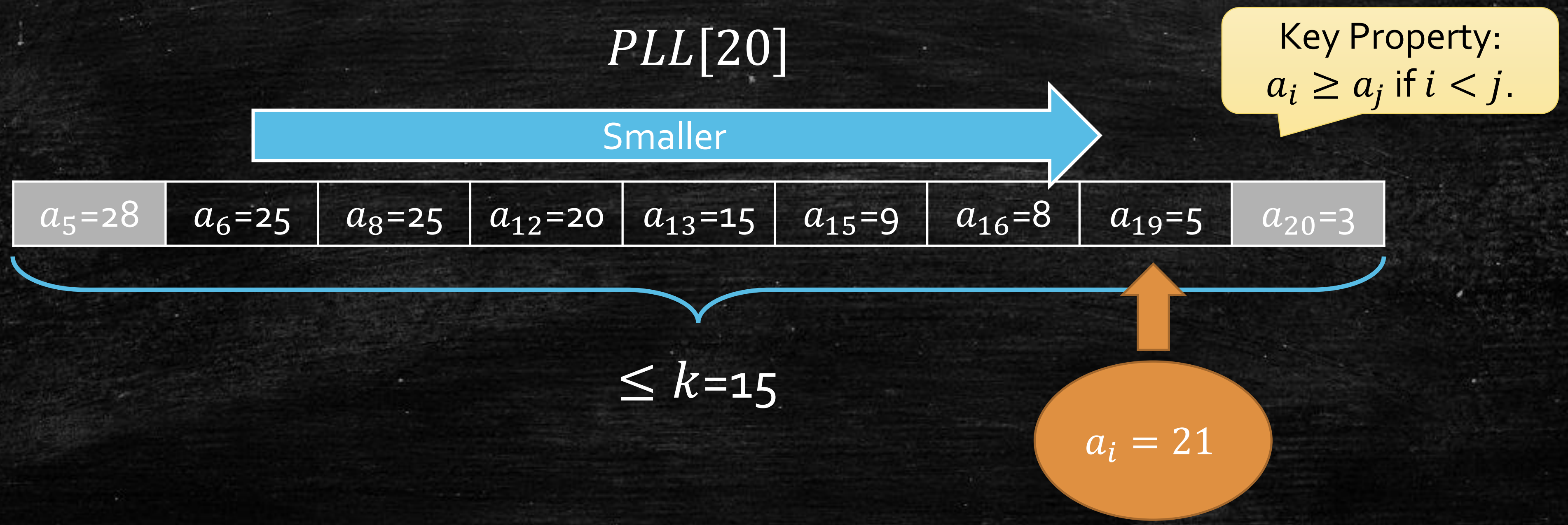
How to maintain PLL?

- 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 $a_{i=21}$.



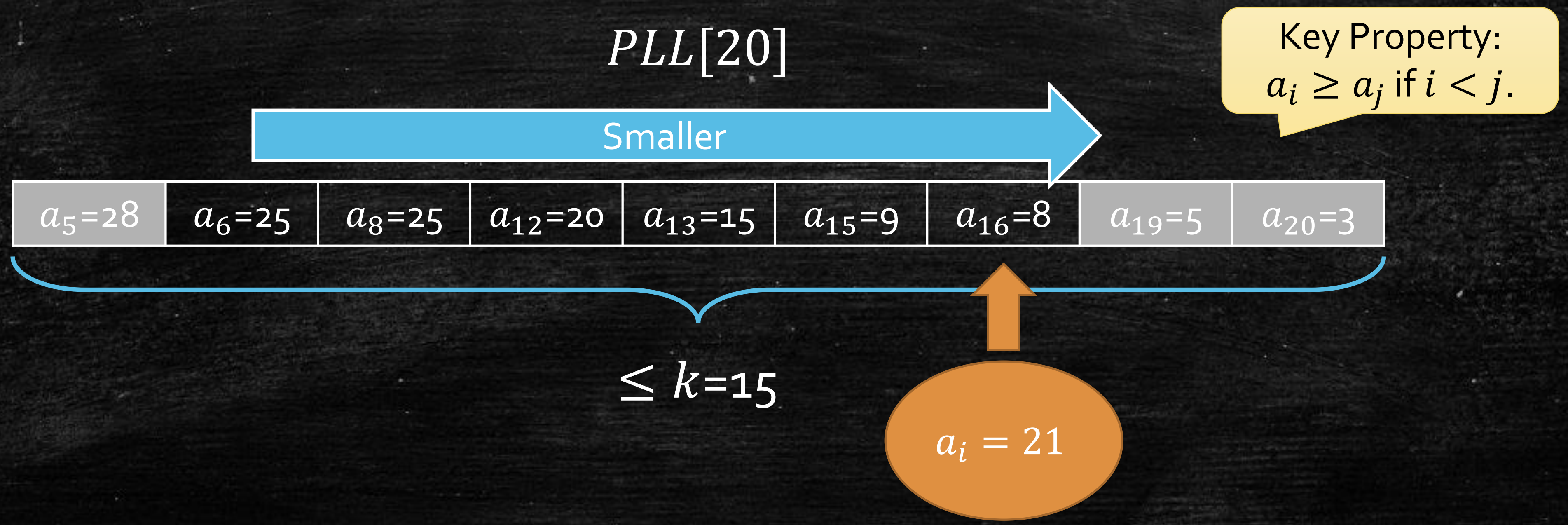
How to maintain PLL?

- 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 $a_{i=21}$.



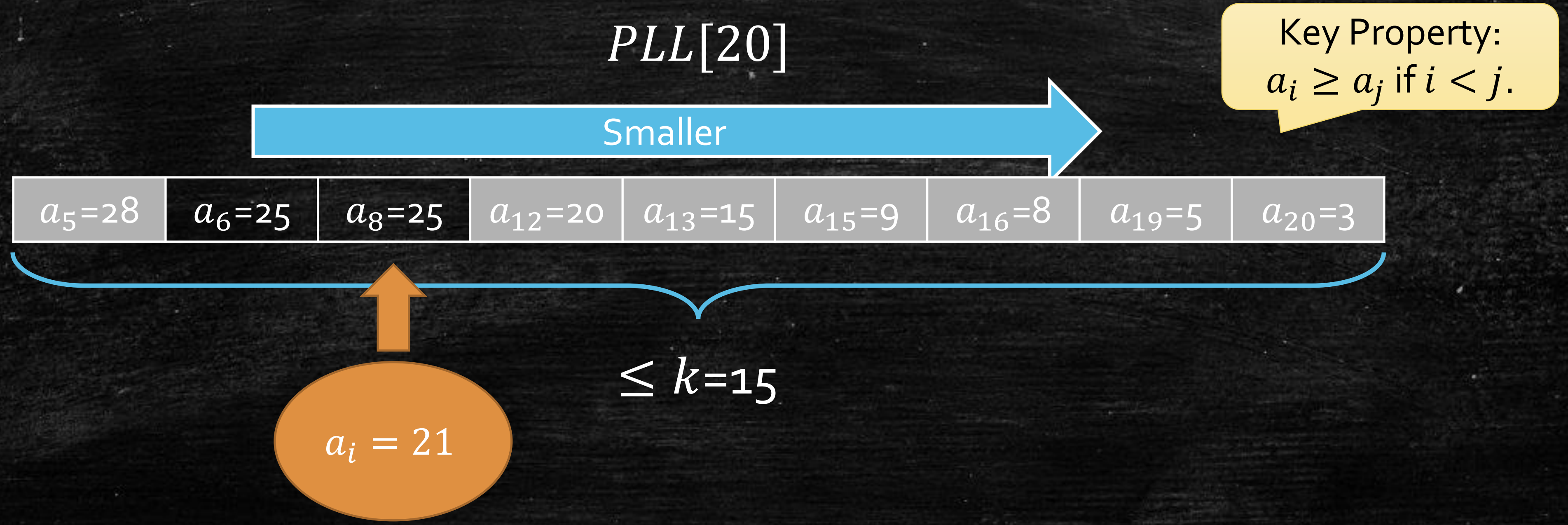
How to maintain PLL?

- 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 $a_{i=21}$.



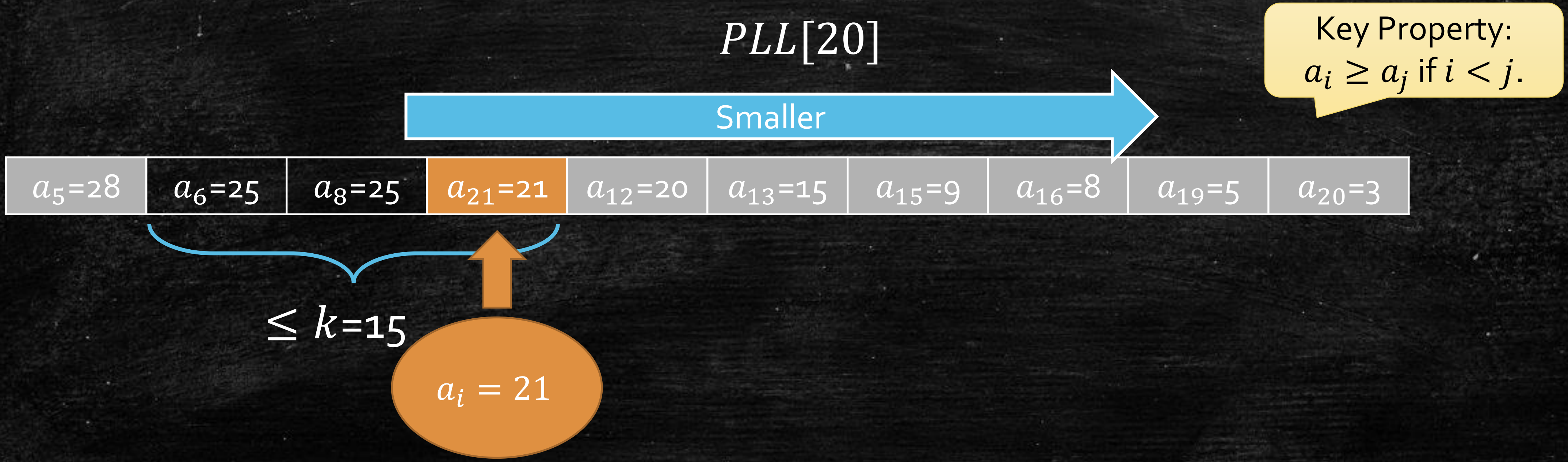
How to maintain PLL?

- 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 $a_{i=21}$.



Largest Number in k Consecutive Numbers

- Keep Inserting $a_1 \sim a_k$ & kicking to make $PLL[k]$.
- Solve every $PLL[k < i \leq n]$ by inserting & kicking.
- We can easily get $large[i]$ by $PLL[i]$.
- It is efficient: $O(n)$! Each number at most:
  - Inserted once.
  - Kicked once.
  - **Pass once** (because once we pass, we kick it).

It is an important idea for DP improvement!

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?

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 prefix of future numbers.

$a[i]$	1	5	13	2	6	24	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

Who are the Potential Prefix?

Previous Transfer

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

$a[i]$	1	5	13	2	6	24	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 Subproblem!

- $Sm[i, len]$: 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	24	15	23	2	16
$lis[i]$	1	2	3	2	3	-	-	-	-	-

New Subproblem!

- $Sm[i, len]$: the **smallest ended number** for an increasing subsequence with length len by using $a_1 \dots a_i$.
- Remark: it is enough to record all Potential Prefixes (length and number).

i

$a[.]$	1	5	13	2	6	24	15	23	2	16
$lis[.]$	1	2	3	2	3	-	-	-	-	-

Larger

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

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- 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
$sm[i, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \ldots]$?
- Difference between $i - 1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- Difference between $i - 1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

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

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

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \leq i, \dots]$?
- Difference between $i - 1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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 may 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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- Difference between $i - 1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

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

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

- It may 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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- Difference between $i-1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

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

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

- It may 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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- Difference between $i-1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

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

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

- It may 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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \le i, \dots]$?
- Difference between $i-1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

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

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

- It may 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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \leq i, \dots]$?
- Difference between $i - 1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

- it can create a longer LI
- it can not update $sm[i, le$

Because we move to here.

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
$sm[i-1, len]$	0	1	2	6	-	-	-	-	-	-

Solving $sm[i, len]$!

- How to solve $sm[i, len]$ (Potential Prefixes)?
  - By $sm[j \leq i, \ldots]$?
- Difference between $i-1$ and i?
  - a_i comes in.
  - It may **become** a potential prefixes and **kick** some potential prefixes.

$a_i = 5$

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

- it can create a longer L
- it can not update $sm[i, le$

Because we move to here.

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
$sm[i-1, len]$	0	1	2	a_i=5	-	-	-	-	-	-

Longest Increasing Subsequence with $sm[\cdot]$.

- Plan
  - Initialize $sm[0,0] = 0$
- Solve $sm[i, len]$ from $sm[i-1, len]$ by a_i.
- Output the largest len such that $sm[n, len] \neq$"-".

Still Not Finished!

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

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
$sm[i-1, len]$	0	1	2	a_i=5	-	-	-	-	-	-

Larger

How to do it efficiently?

Yes! Binary Search!

Recap the updating

- We need to find the largest len such that $a_i > sm[i-1, len]$.
  - Find it by binary search, we only need $O(\log(\max len = i))$!
- 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
$sm[i-1, len]$	0	1	2	a_i=5	-	-	-	-	-	-

Larger

Now it is better!

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

One more Interesting problem.

Minimizing Manufacturing Cost

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

0	16	4	5	13	9	20	18	3

i

Solving $f[i]$

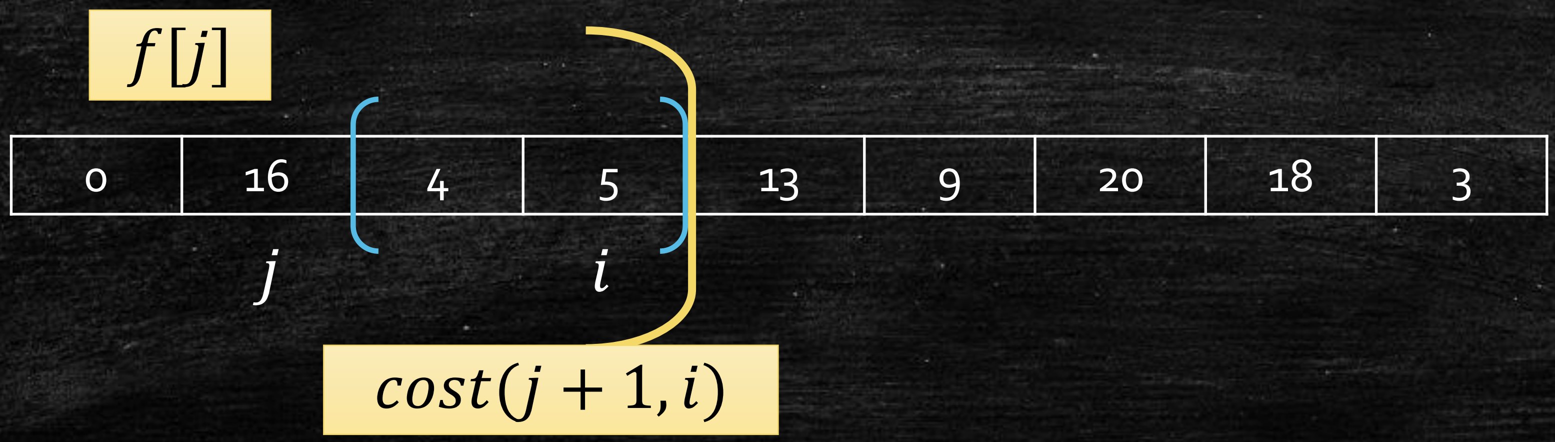
- $f[i]$: the minimum cost for manufacturing item 1 to i.
- How to solve $f[i]$?
- We can manufacture item i alone.

0	16	4	5	13	9	20	18	3

i

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



DP algorithm

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

$O(n^2)$

$f[j]$

0	16	4	5	13	9	20	18	3
	j		i					

$cost(j+1, i)$

The Potential Idea Again!

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

0	16	4	5	13	9	20	18	3

i

The Potential Idea Again!

- Question: Can every j be a potential prefix for the future?
- Maybe........ I can find nothing.

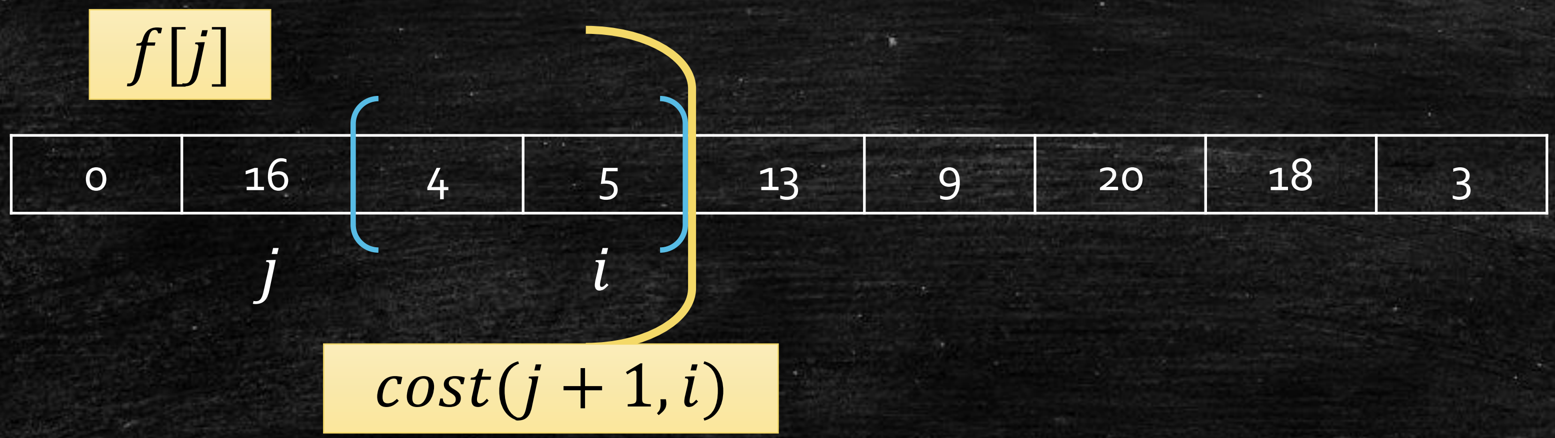
0	16	4	5	13	9	20	18	3

i

Let us do some math!

Math Time!

- $f[i] = \min\limits_{j<i} f[j] + C + \left(\sum_{k=j+1}^{i} a_k\right)^2.$
- Consider j $= x$ and j $= y$, when x is better than y for 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!

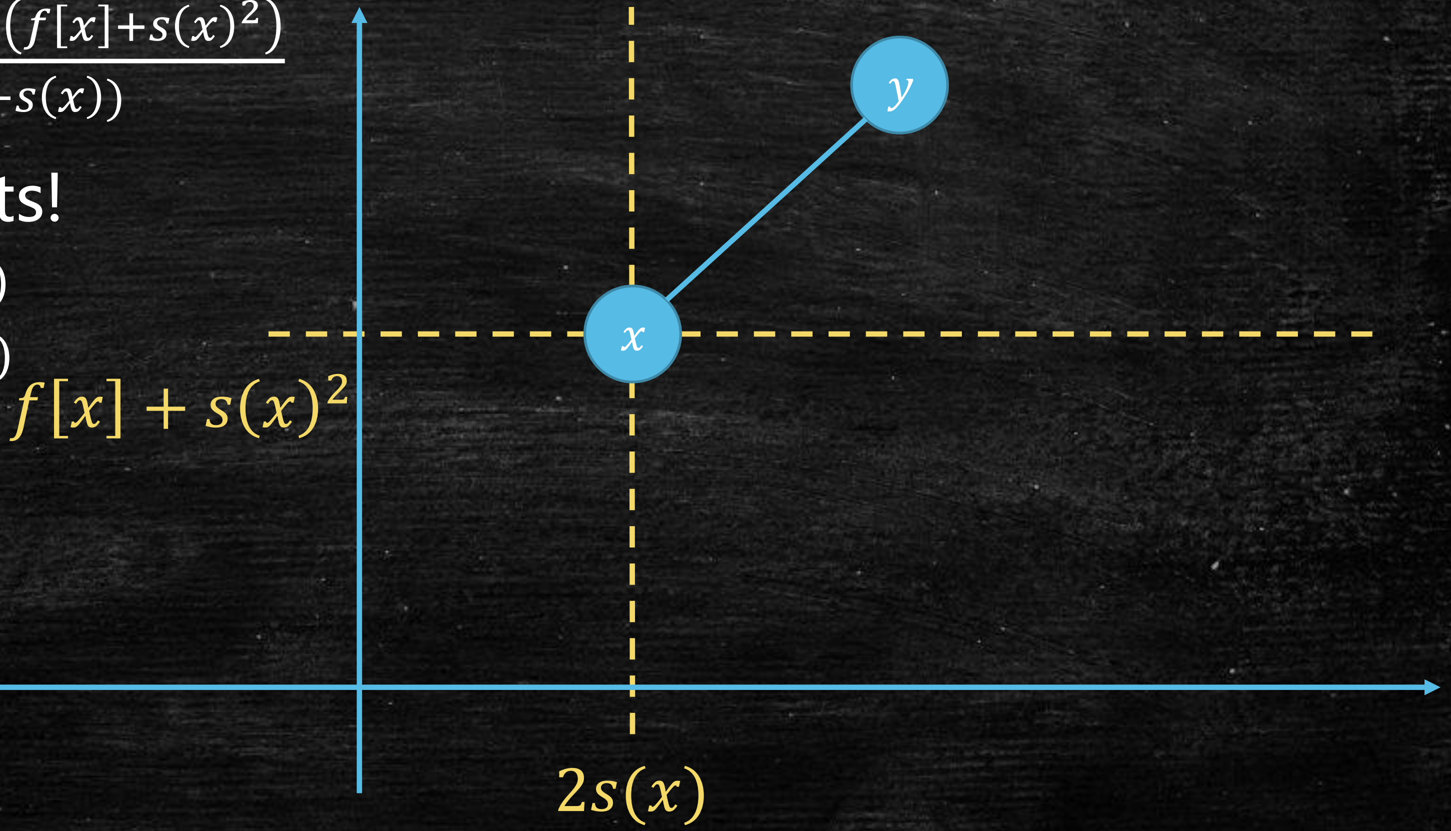
- $f[i] = \min_{j<i} f[j] + C + \left(\sum_{k=j+1}^{i} a_k\right)^2.$
- Consider j = x and j = y, when y is better than x for i?
- $f[x] + C + \left(\sum_{k=x+1}^{i} a_k\right)^2 > f[y] + C + \left(\sum_{k=y+1}^{i} a_k\right)^2$
- Let $s(i) = \sum_{j=1}^{i} a_k.$
- $f[x] - f[y] > \left(s(i) - s(y)\right)^2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

  $= s(y)^2 - s(x)^2 - 2s(i)(s(y) - s(x))$
- $\frac{(f[y]+s(y)^2)-(f[x]+s(x)^2)}{2(s(y)-s(x))} < 2s(i)$

Math Time!

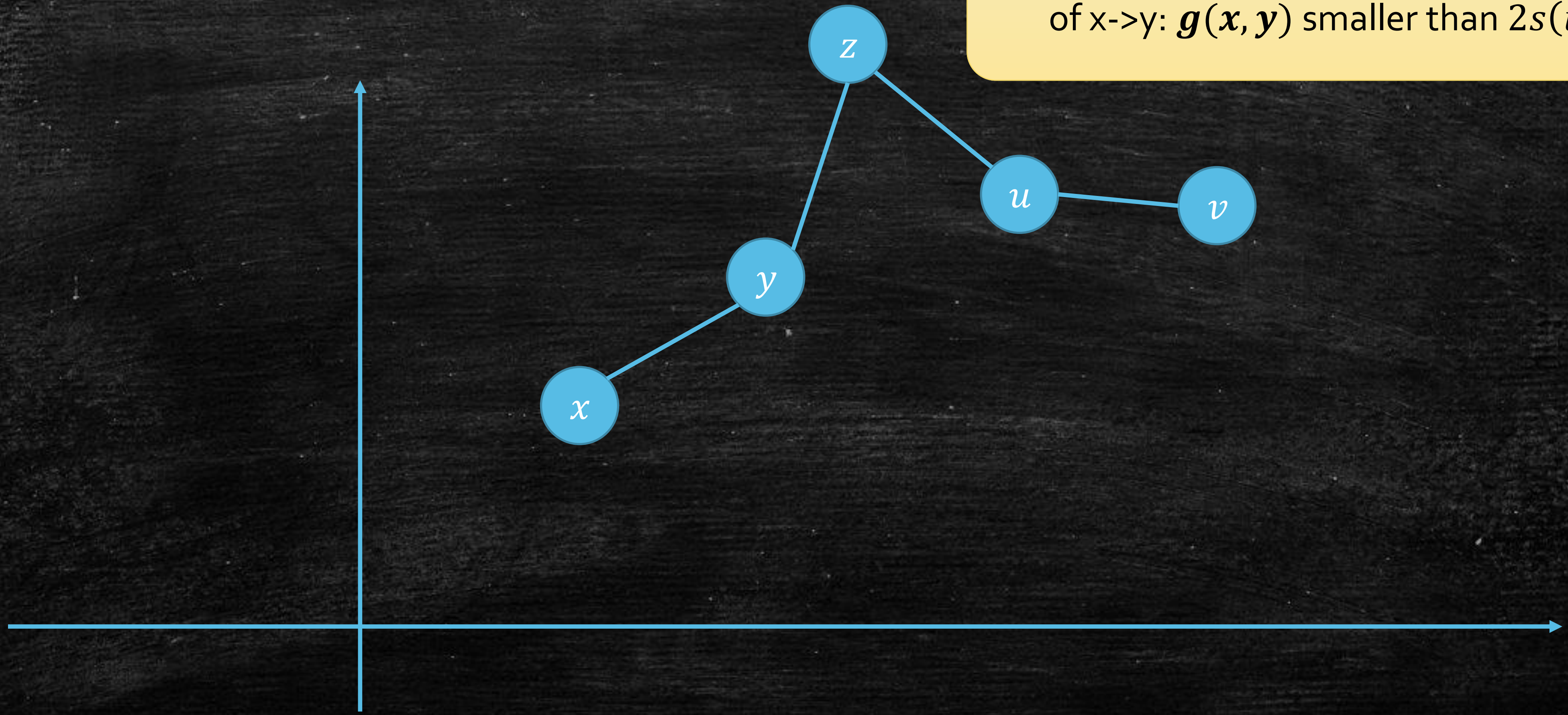
- $\frac{(f[y]+s(y)^2)-(f[x]+s(x)^2)}{2(s(y)-s(x))} < 2s(i)$
- $g(x,y) = \frac{(f[y]+s(y)^2)-(f[x]+s(x)^2)}{2(s(y)-s(x))}$
- View it as two points!
  - x: $(2s(x), f[x] + s(x)^2)$
  - y: $(2s(y), f[y] + s(y)^2)$

y is better than x for i means the gradient of x->y: $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y})$ smaller than $2s(i)$.



Who can be kicked out?

Who can be kicked out?

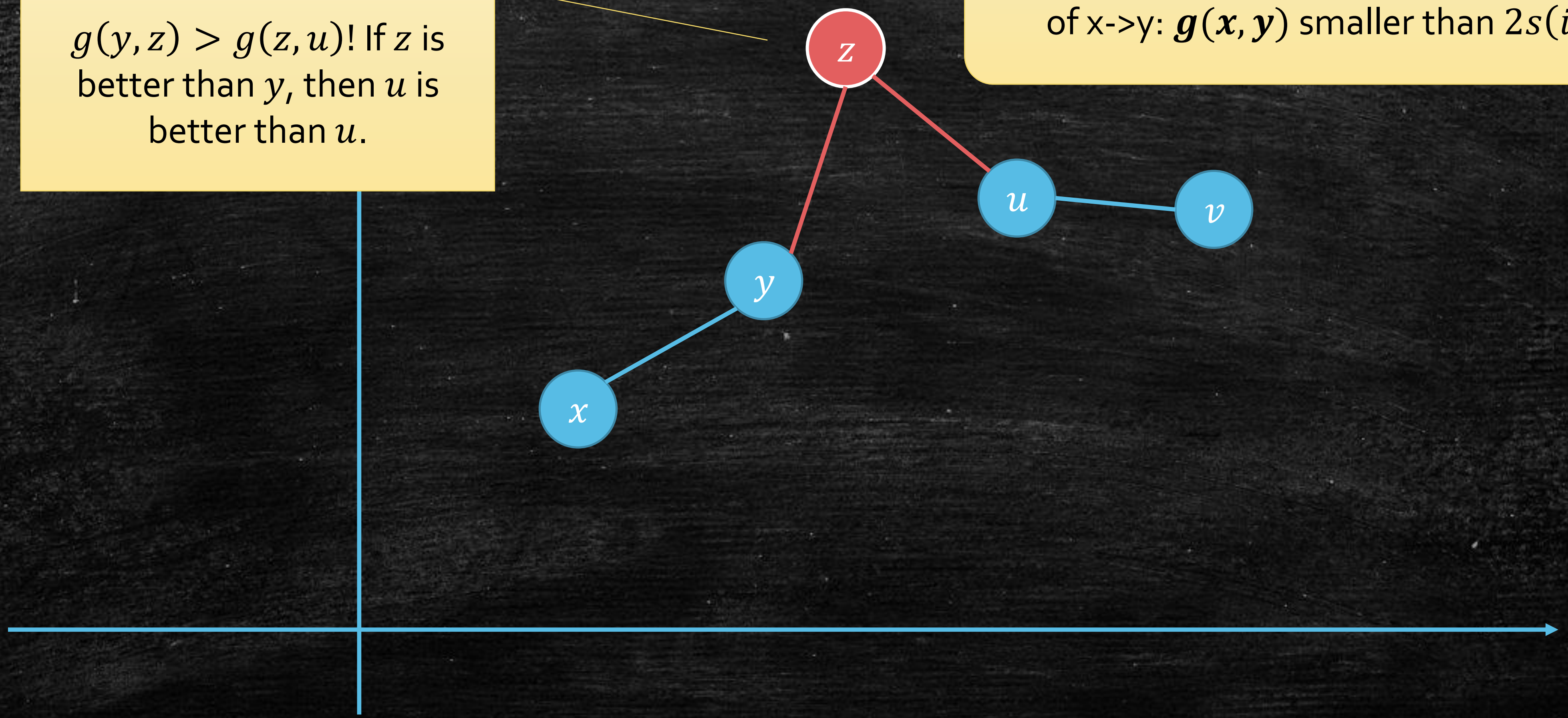


y is better than x for i means the gradient of x->y: $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y})$ smaller than $2s(i)$.

Who can be kicked out?

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

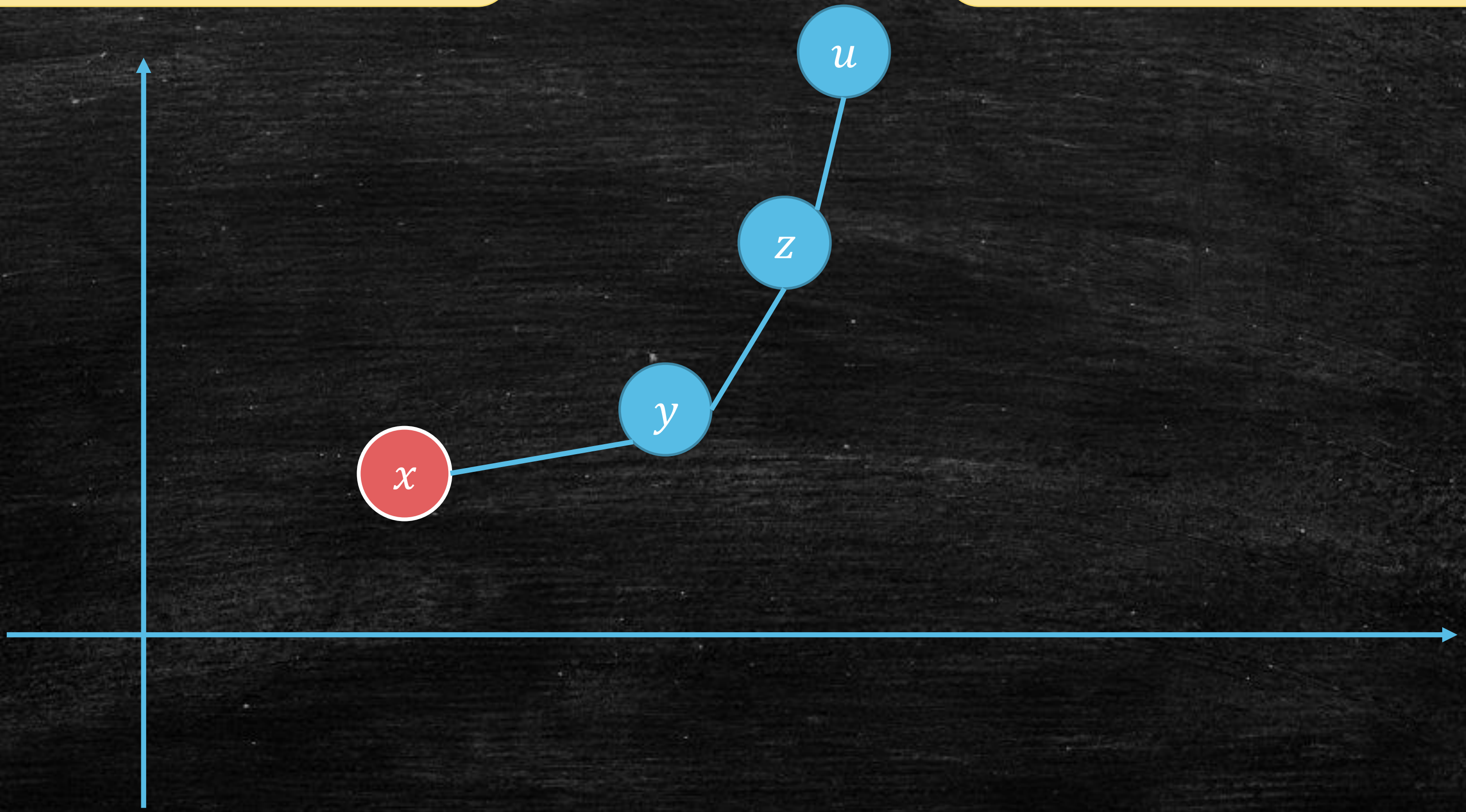
y is better than x for i means the gradient of x->y: $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$ smaller than $2s(i)$.



After Kicking: A Convex Hall.

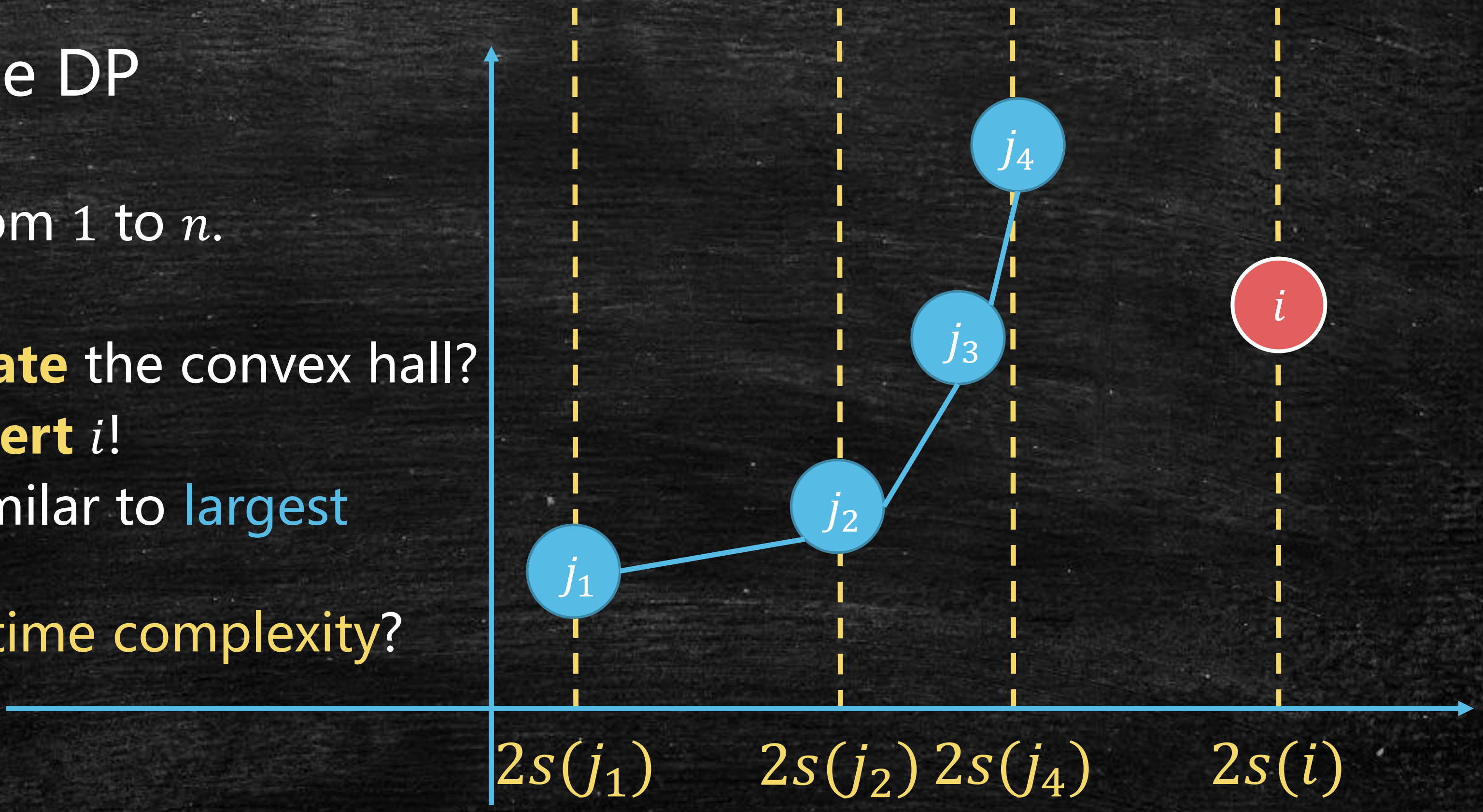
What if $g(x, y) < 2s(i)$?
Kick x!

y is better than x for i means the gradient of x->y: $\boldsymbol{g(x, y)}$ smaller than $2s(i)$.



Discussion

- Complete the DP
  - $f[0] = 0$
  - Solve $f[i]$ from 1 to n.
  - Output $f[n]$.
  - How to **update** the convex hall?
  - We need **insert** i!
  - Tips: very similar to largest number!
  - What is the time complexity?



Today's goal

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