

CS 331, Fall 2026

Lecture 5 (9/9)

Today: - Scheduling

- Longest increasing subsequence

- Subset sum

Scheduling (Part II, Section 3.1)

Input: L is n tuples in $\mathbb{R}^2$

$[l_i, r_i]$ with $l_i < r_i \forall i \in [n]$

$\uparrow \nearrow$

endpoints of i th interval

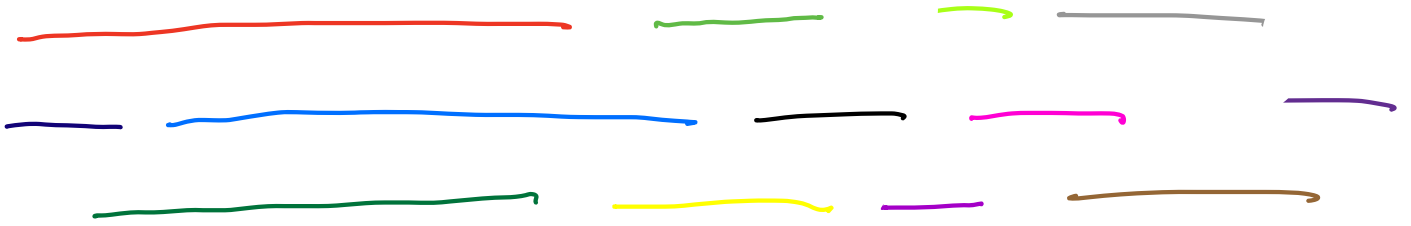
Output: Maximum $|S|$ for nonoverlapping $S \subseteq [n]$

i.e. $\forall i \neq j \in [n], [l_i, r_i] \cap [l_j, r_j] = \emptyset$

Not OK

OK

Example



Q: How many can we schedule?

It is not so easy.

Naive algo: "try everything"

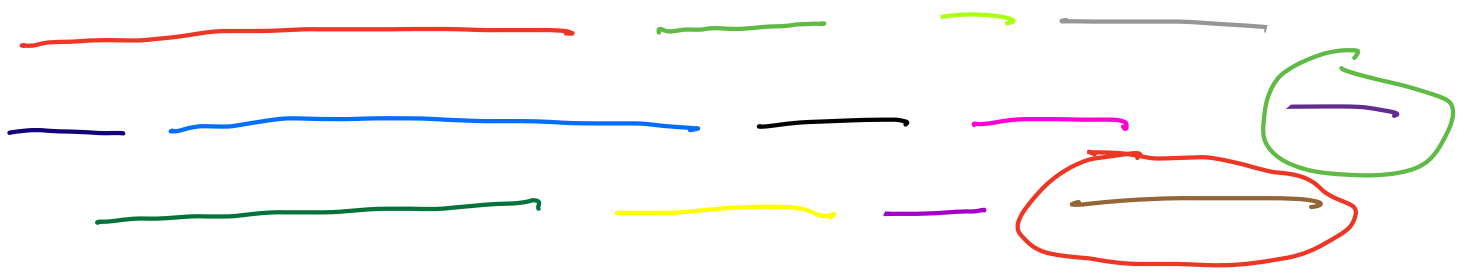
Problem: there's 2^n possible $S \subseteq \{n\}$...

Idea: DP?

$\text{Best}(i)$ = largest subset taking interval i ?

Issue 1: What order? (not sorted)

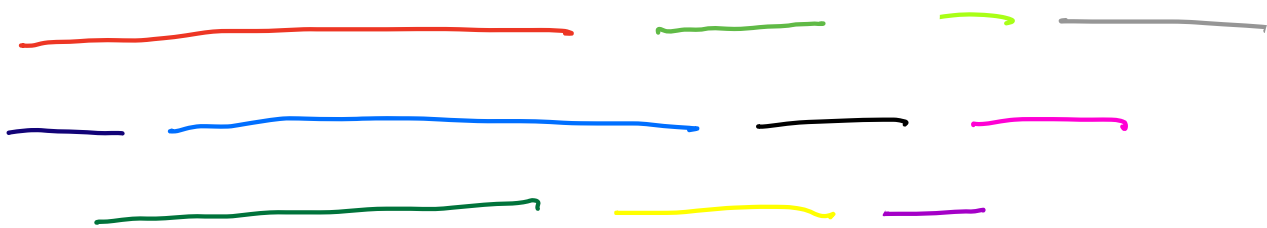
2: how to recurse?



Intuition: Let's get rid of "last" interval —

It means we can't take — (overlaps)

Skip ahead to:



Idea: 1) Sort L

2) Define special problems $Best(i)$

3) ...

4) Profit?

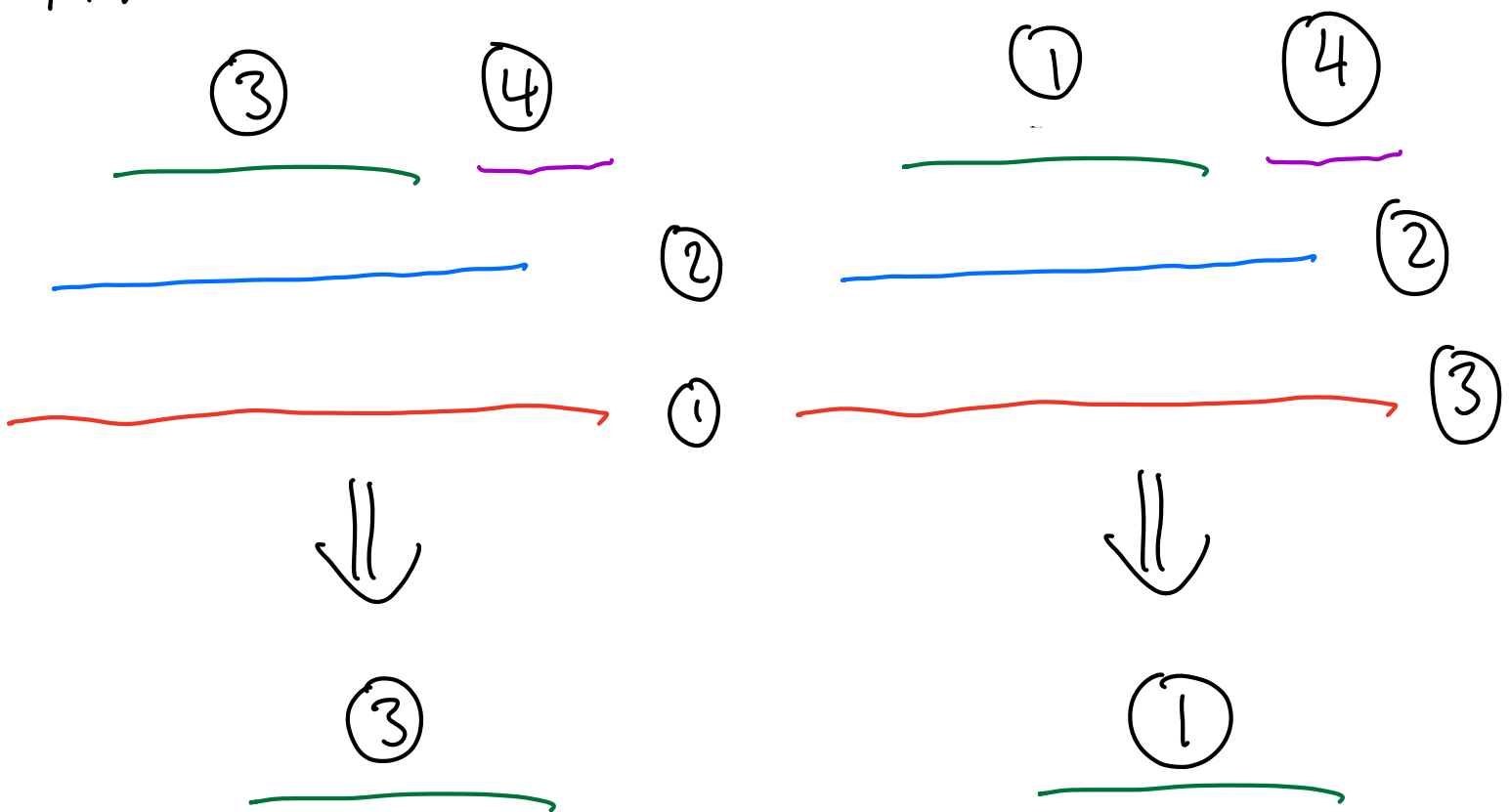
Step 2: let

$\text{Best}(i)$ = largest nonoverlapping subset
of $L[i:j]$ (prefix of L)

Step 1: How to sort?

- left endpoint?
- right endpoint?

Ans: we should do whatever lets us recurse.



Left sort: not a prefix

Right sort: a prefix
(memoized)

Claim: Suppose L sorted by right endpoint.

Removing overlaps w/ last interval

gives a prefix of L .

Proof:



let $P(n)$ be last interval with $r_{P(n)} < l_n$

Overlapping: $P(n)+1, P(n)+2, \dots, n$ $r_i \geq r_{P(n)+1} \geq l_n$

Non-overlapping: $1, 2, \dots, P(n)$ $r_i \leq r_{P(n)} < l_n$


prefix

1) Sort L by right endpoint

2) Define special subproblems:

$Best[j]$ = largest nonoverlapping subset
of $L[1:j]$ (prefix of L)

3) Recursion

$$Best[j] = \max \left(\underset{\substack{\uparrow \\ \text{don't include} \\ \text{interval } j}}{Best[j-1]}, \quad 1 + \underset{\substack{\uparrow \\ \text{include} \\ \text{interval } j}}{Best[P(j)]} \right)$$

Here, $P(j)$ is last acceptable interval:

$$\underset{\substack{\text{(can take)}}}{r_{P(j)}} < l_j \leq \underset{\substack{\text{(can't take)}}}{r_{P(j)+1}}$$

Runtime analysis: 1) $O(n \log(n))$

2) N/A

3) $O(n) \times O(\log(n))$

$\underbrace{\hspace{2cm}}$ # subproblems $\underbrace{\hspace{2cm}}$ compute $P[i]$
via binary search

4) $O(n \log(n))$ time
(before, $\exp(n)$)

😊

Extension

Recover the subset?



We memoized everything,
can infer the path.

Work backwards!

$S \leftarrow S \cup \{n\}$

Extension

Weighted scheduling

Same idea, but interval i has weight $W(i)$

Goal: maximize $\sum_{i \in S} W(i)$ for non-overlapping S

Motivation: not all intervals created equal

e.g. $W(i) = r_i - l_i$ (by length)

$W(i) = 1$ (vanilla scheduling)

Strategy: $Best(i) = \text{max weight non-overlapping subset of first } i \text{ intervals.}$

$$Best(i) = \max \left(Best(i-1), Best(p(i)) + W(i) \right)$$

Longest increasing subsequence (Part III, Section 3.2)

Input: L is a list of n elements in $\mathbb{R}$

Output: Longest increasing subsequence of L

$$S \subseteq [n], L[i] \leq L[j] \quad \forall i < j \in S$$

Example

5, 10, 7, 1, 8, 3, 2, 6, 12, 4, 9, 11

(random permutation of $[12]$)

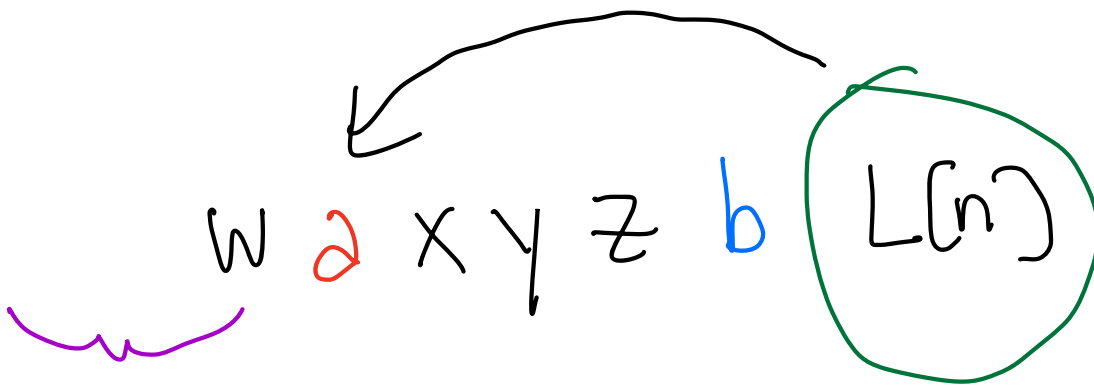
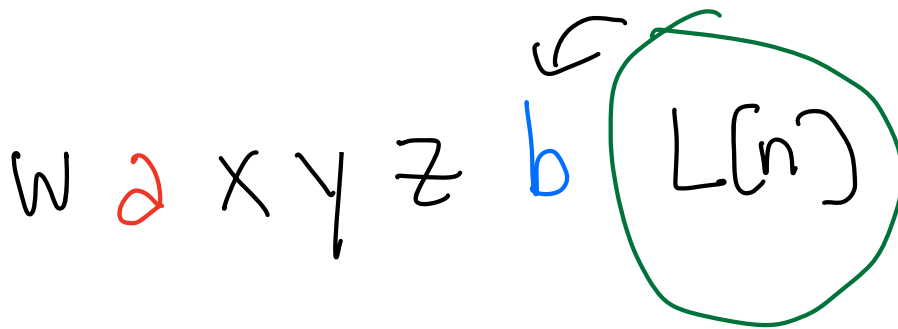
Again, not so easy! "try everything" $\Rightarrow 2^n$ tries

$\text{Best}(i) =$ longest increasing subsequence
ending on $L[i]$

(prefix also OK: just not as clean)

How to recurse?

if b is small,
rather not take



less options
to continue growing
the sequence

takeaway: there are tradeoffs...

Solution: try everything.

$$\text{Best}[j] = \max_{\substack{i \in [j-1] \\ L[i] \leq L[j]}} \text{Best}[i] + 1$$

$L[i] \leq L[j]$

Best continuation.

We let it = 0

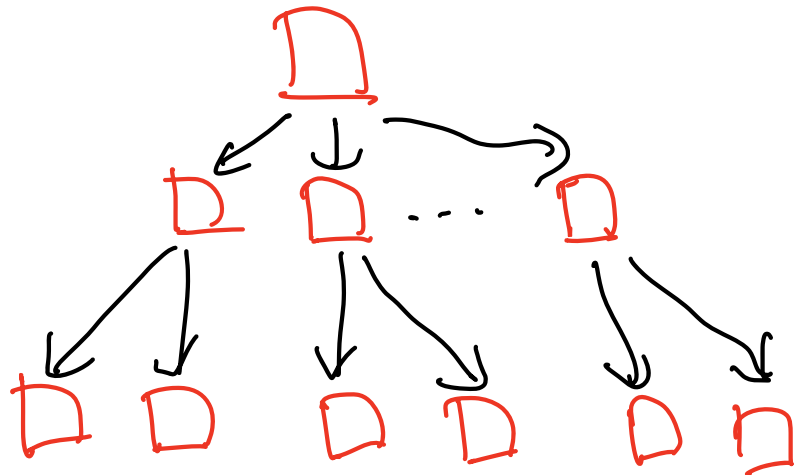
if there are no $L[i] \leq L[j]$.

include $L[j]$

Takes $O(n)$ time per $j \in [n]$,
 $O(n^2)$ time total.

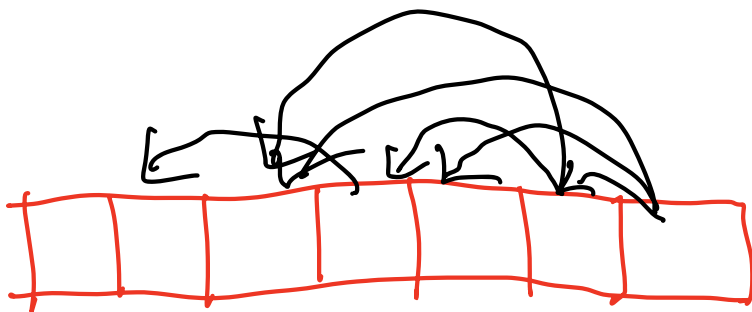
Intuition:

Normal recursion



Smart recursion

Still can have
large branching
factor - just reusing
the same n problems



Subset Sum (Part III, Section 3.3)

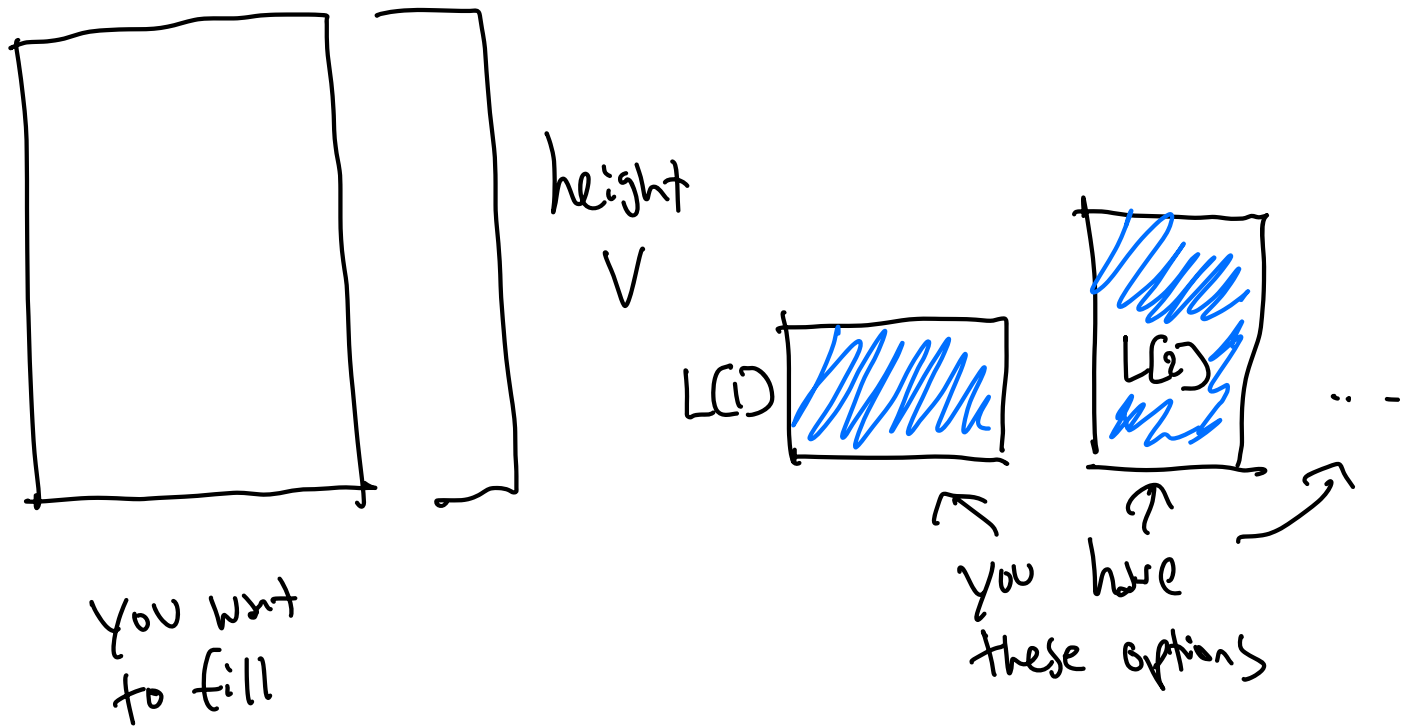
Input: L is a list of n natural #s
1, 2, 3, 4, ...

$V \in \mathbb{N}$ is a target value

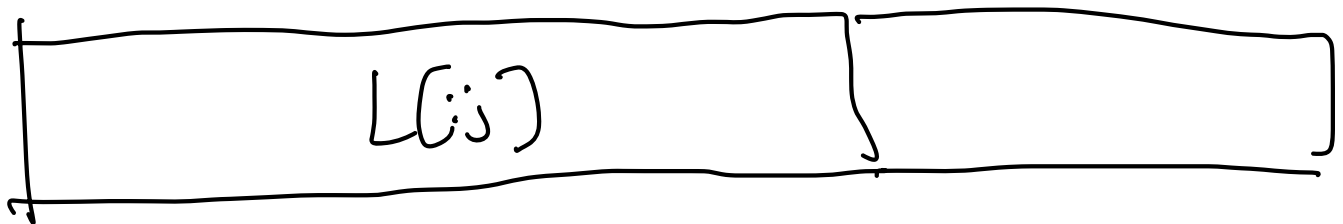
Output: True if $\exists S \subseteq [n], \sum_{i \in S} L[i] = V$

False else

Intuition:



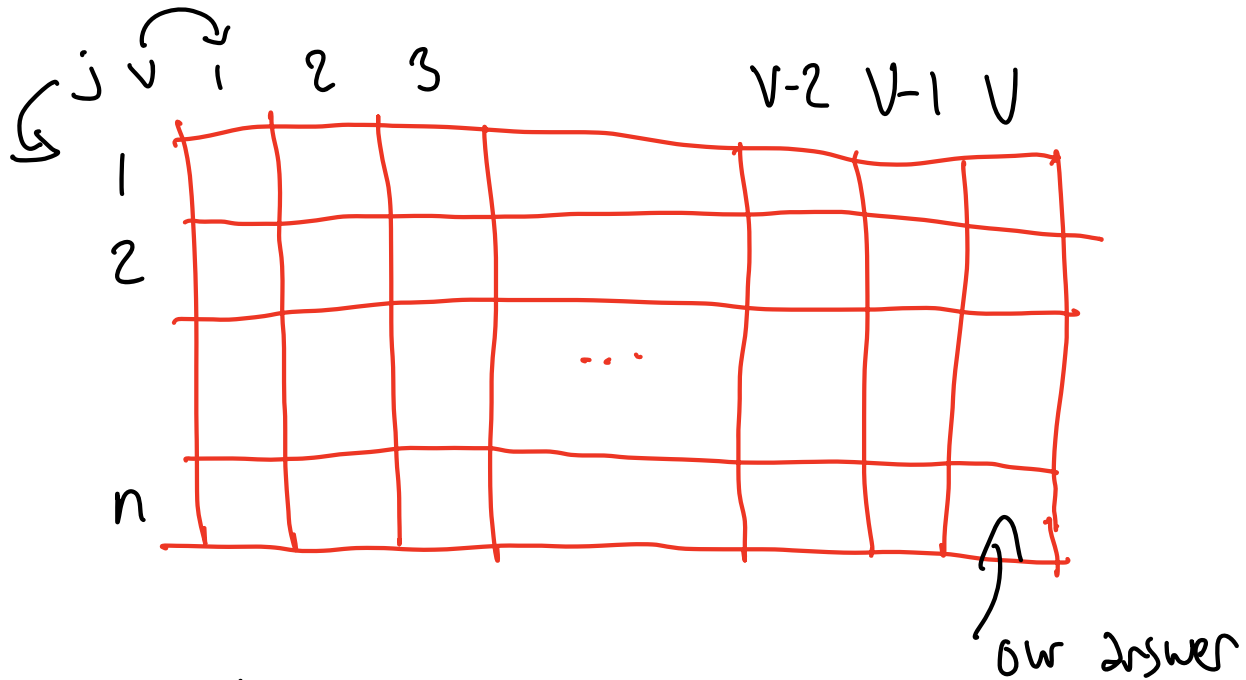
How to define subproblems?



if false, doesn't
yield much into later...

need to consider values

$S[j][v]$ = Can you hit target v using first j items?



Formula:

$$S[j][v] = S[j-1][v]$$

$$\text{OR } S[j-1][v - L[j]]$$

What order?

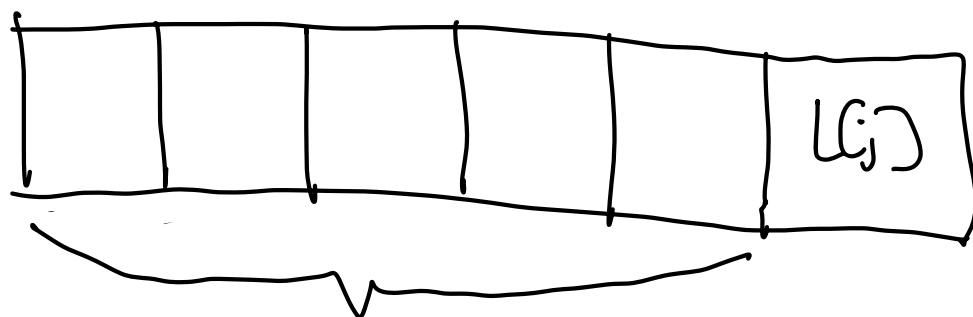
Row by row. Formula depends on prev row.

Time: $O(nV)$. (good if V small)

Bonus: Faster LIS

Solvable in $O(n \log n)$ time.

Intuition: Can we make length- k using $L[j]$?



is there length $k-1$?

may as well store Smallest end

Goal: Iterate thru $j \in [n]$

After time j :



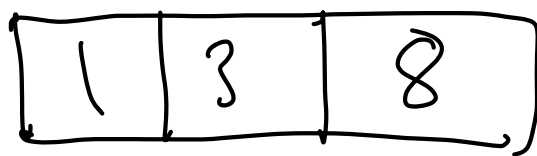
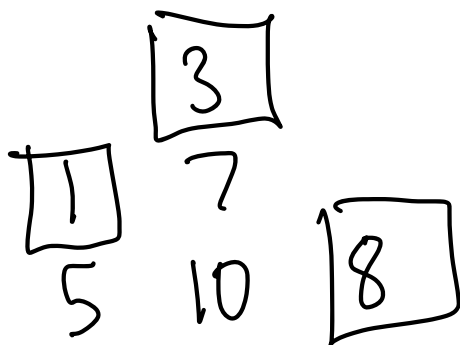
Smallest end of length- k increasing subseq. of $L[j]$

Example

5, 10, 7, 1, 8, 3, ($j=6$)

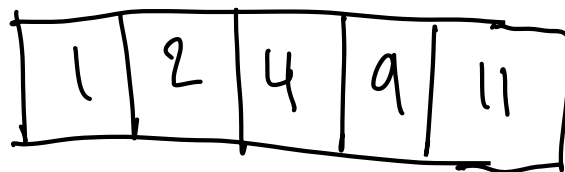
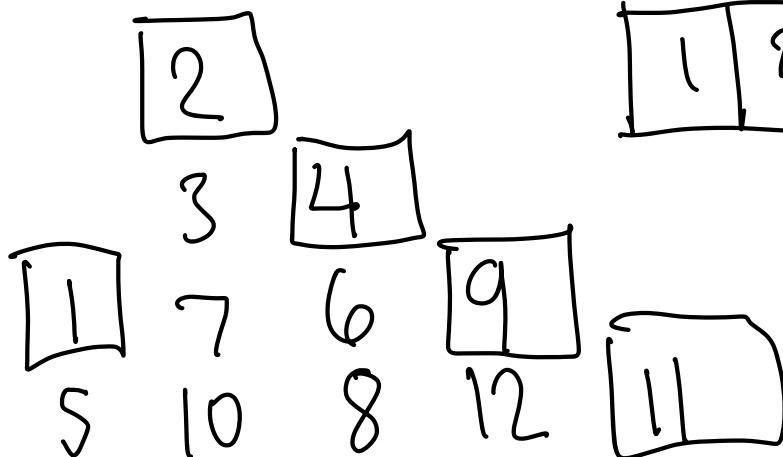
2, 6, 12, 4, 9, 11 ($j=12$)

$j=6$



S

$j=12$



Observations: • S always sorted

• Incoming element unique stack

- earlier? not smaller than current head

- later? remove $L[j]$, smaller than prev. head

• $\log(n)$ time per iter

→