DYNAMIC PROGRAMMING

Learning Goals

- Describe + benchmark Max Weight Independent Set problem
- Create a recurrence for MWIS on a line
- Design a Dynamic Programming alg for MWIS ✓

Announcements

Blank notes

Exit Tickets

Exchange argument redo

n levels, $n/2$ levels

$\boxed{\text{Use Dynamic Programming for Closest Points?}}$

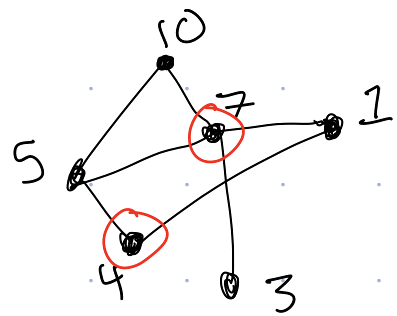
Brute force

Max Weight Independent Set

Optimization Problem

Input: Graph $G = (V, E)$

weights $w : V \to \mathbb{Z}^+$



Output: $S \subseteq V$ s.t.

- If $\{u, v\} \in E$ $\quad \neg(u \in S \wedge v \in S)$ } Constraint
- Maximizes $W(S) = \sum_{v \in S} w(v)$

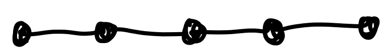
objective function

Applications:

- Cell tower transmission
- Choose franchise locations
- Party invite
- Scheduling
- ~~House robbing~~

General Graph: HARD

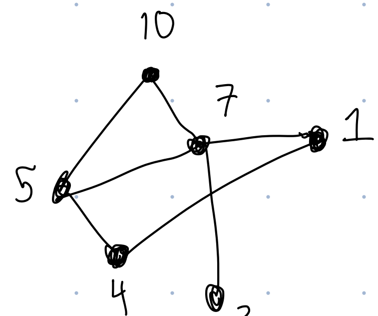
Line Graph: Easy, with



Dynamic Programming

Max Weight Independent Set (MWIS)

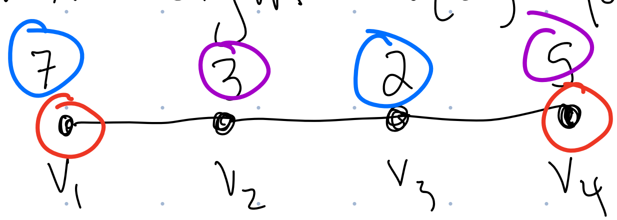
<u>Input</u>: Graph $G = (V, E)$
weights $w : V \to \mathbb{Z}^+$



<u>Output</u>: $S \subseteq V$ s.t.

- If $\{u,v\} \in E$, $\neg(u \in S \wedge v \in S)$ ⇐ "Independent Set Condition" "constraint"
  (not, and)
- Maximizes $W(S) = \sum_{v \in S} w(v)$ ⟸ "Weight of S"

What is max weight $W(S)$ for MWIS of this line graph:



A) 0 B) 8 C) 9 $\boxed{\text{D) 12}}$

Dynamic Prog. Approach

Recall: # of n bit strings with 2 consecutive ones

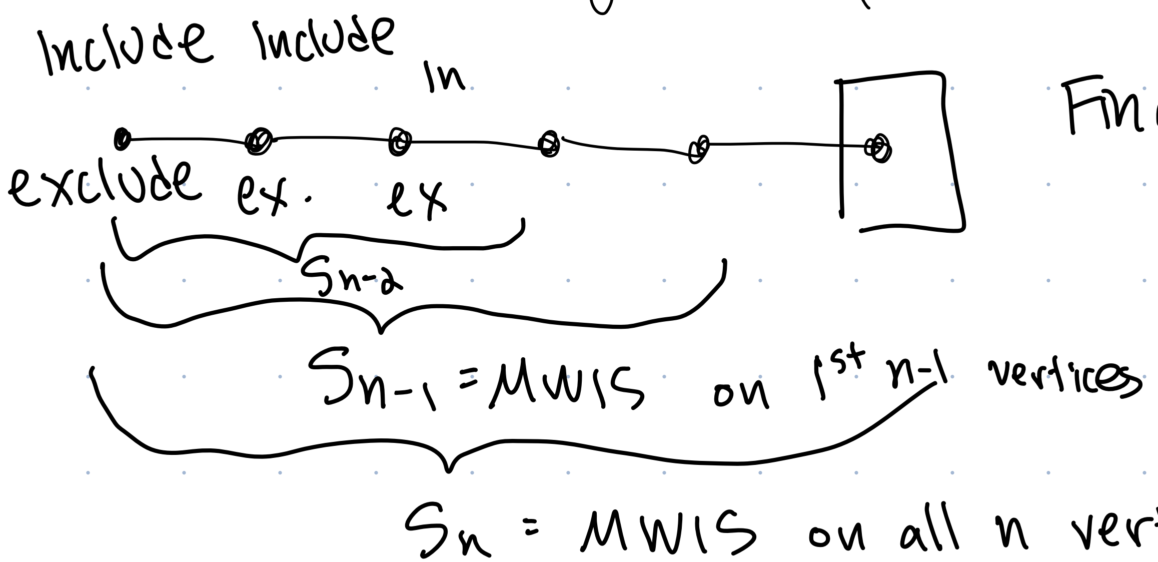
0,1 0,1 0,1

_ _ _ _ _ _ $\boxed{}$ 2 options: 0 or 1

_ _ _ _ _ 0 How many strings?

_ _ _ _ _ 1 How many strings?

To create a DP alg, (often) need to conceptualize the optimal solution as a sequence of choices

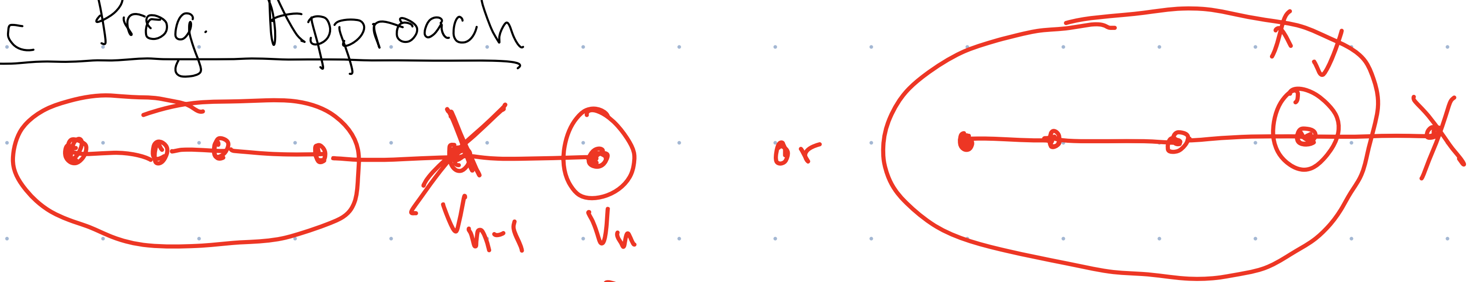


Final choice:

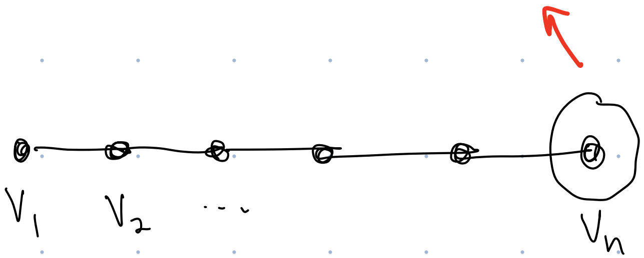
$V_n \in S$

$V_n \notin S$

Dynamic Prog. Approach



MWIS



2 cases: $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

- $v_n \in S$, then $S_n = S_{n-2} \cup \{v_n\}$

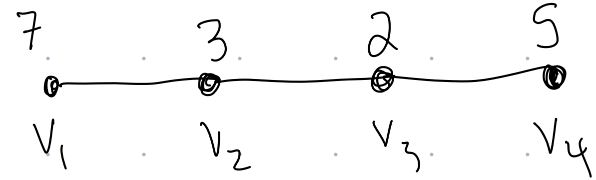
  Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$, $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$
- $v_n \notin S$, then $S_n = S_{n-1}$

Name, pronouns, best thing watched/listened to recently

Write pseudocode for brute force approach, analyze runtime

Brainstorm greedy, divide + conquer approaches

Brute Force $O(n2^n)$




```
MWIS(G=(V,E), w)
  maxS ← ∅
  MaxW ← 0
  For each set S ⊆ V:          O(2^n)
    If S is I.S.:
      w ← W(S)
      if w > maxW then
        maxS ← S
        maxW ← w
  Return maxS
```

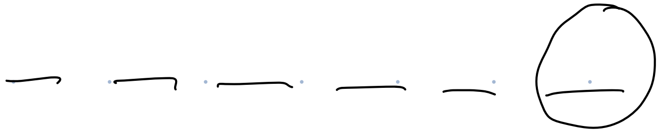
0110
0001
0010

```
S is I.S.:                     O(n)
  For i ← 1 to n-1:
    If v_i ∈ S and v_{i+1} ∈ S:
      Return False
  Return true
```

```
W(S):                          O(n)
  W ← 0
  For i ← 1 to n
    If v_i ∈ S
      W ← W + w(v_i)
  return W
```

Dynamic Prog. Approach

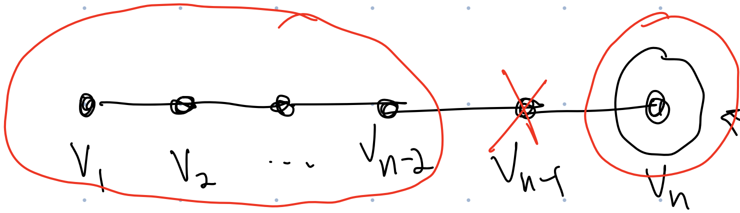
Recall: # of n bit strings with 2 consecutive ones



$\leftarrow$ 2 cases, 0 or 1

$\downarrow$ $T(n-1 \ldots) \ldots$ $\quad$ $\downarrow$ $T(n-2) \ldots$

MWIS

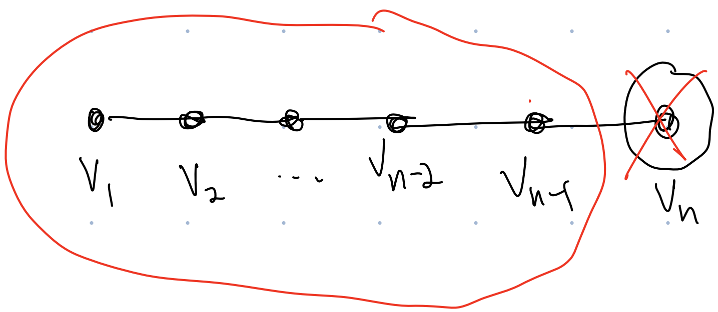


$\leftarrow$ 2 cases : $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

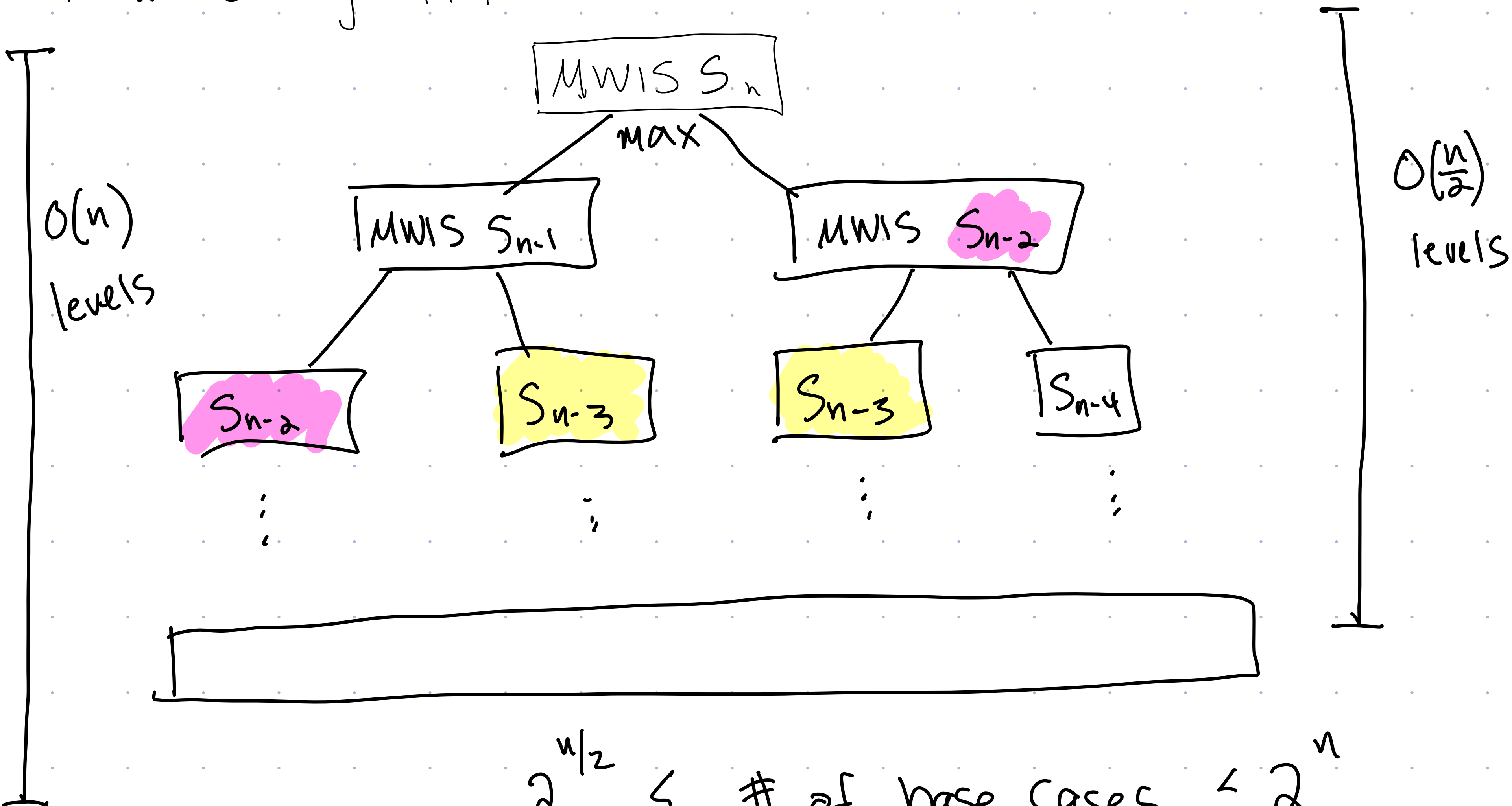
Options:
S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$
$S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

- $v_n \in S$, then $S_n = \underline{S_{n-2}} \cup \{v_n\}$
- $v_n \notin S$, then $S_n = \underline{S_{n-1}}$



$$S_n = \begin{cases} \text{Max Weigh} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \text{Base case} \end{cases}$$

Recursive Algorithm:

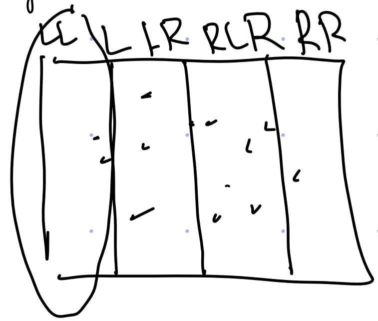


$2^{n/2} < \#$ of base cases $< 2^n$

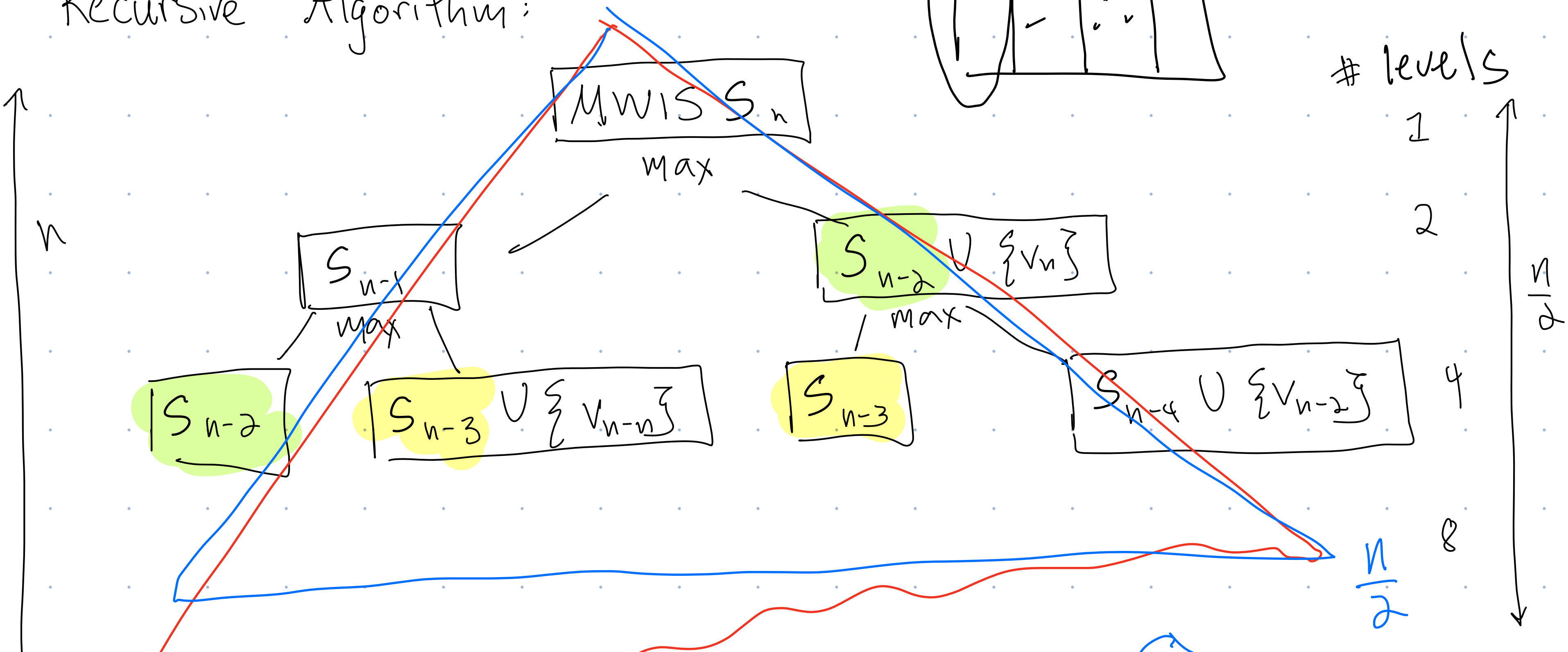
$$S_n = \begin{cases} \max \text{ weight} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \text{base case} \end{cases}$$

Only 2 possible options, take larger!

★ And base case! Later...



Recursive Algorithm:



$2^n >$ # of nodes in tree $> 2^{n/2}$

Each node requires $\Omega(1)$ time $\rightarrow \Omega(2^{n/2})$ algorithm

asymptotic lower bound

Bad!

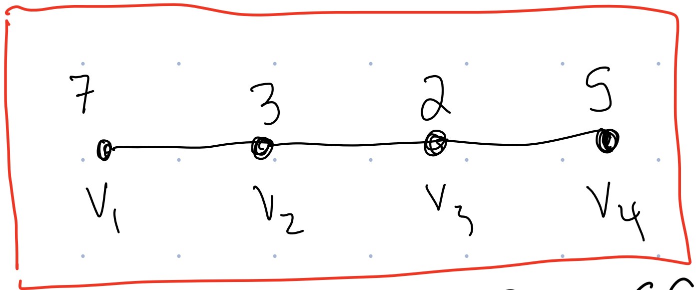
How many unique subproblems are there?

A) $\sqrt{n}$ B) $\frac{n}{2}$ $\boxed{\text{C) } n}$ D) n^2

$S_n, S_{n-1}, S_{n-2} \cdots S_1$

Dynamic Programming Idea: Store subproblems in an array + look up

$$S_n = \begin{cases} \max\text{ wt} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \phi \quad \text{if } n=0 \\ \{v_1\} \quad \text{if } n=1 \end{cases}$$



$$S_4 = \begin{cases} \max\text{ wt} \begin{cases} S_3 \\ S_2 \cup \{v_4\} \end{cases} \end{cases}$$

S_0	S_1	S_2	S_3	S_4
ϕ	$\{v_1\}$	$\{v_1\}$	$\{v_1, v_3\}$	$\{v_1, v_4\}$

Trick: Start at smallest subproblem and go up

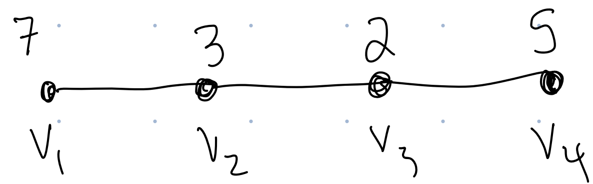
Trick: Create a size 0 problem

Trick: Store objective function value, not object itself

$$S_n = \begin{cases} n \geq 1: \text{max wt} \begin{cases} \underbrace{S_{n-1}}_{v_n \notin S} \\ \underbrace{S_{n-2} \cup \{v_n\}}_{v_n \in S} \end{cases} \\ \phi \quad \text{if } n=0 \\ \{v_1\} \quad \text{if } n=1 \end{cases} \Rightarrow W(S_n) = \begin{cases} \max\{\underbrace{W(S_{n-1})}_{v_n \notin S}, \underbrace{W(S_{n-2}) + w(v_n)}_{v_n \in S}\} \\ 0 \quad \text{if } n=0 \\ w(v_1) \quad \text{if } n=1 \end{cases}$$

array A

$W(S_0)$	$W(S_1)$	$W(S_2)$	$W(S_3)$	$W(S_4)$
0	7	7 ~~0+3~~	~~7~~ 7+2=9	~~9~~ 7+5=12



$\longrightarrow$

0	7	7	9	12

ϕ	$\{v_1\}$	$\{v_1\}$	$\{v_1, v_3\}$	$\{v_1, v_4\}$

MWIS on a Line (G, w):

// Create array of objective function values

1. Initialize array A of length $n+1$.
2. $A[0] \leftarrow 0$
3. $A[1] \leftarrow w(v_1)$
4. for $i \leftarrow 2$ to n:

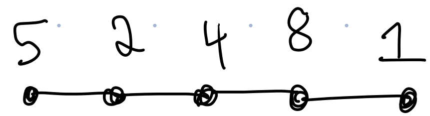
   $A[i] \leftarrow \max \{ \overbrace{A[i-1]}^{v_n \notin S}, \overbrace{A[i-2] + w(v_i)}^{v_n \in S_n} \}$ $O(n)$

// Determine optimal Set

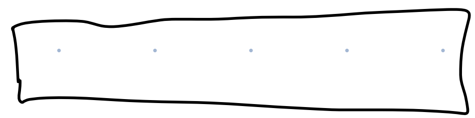
6. $S \leftarrow \emptyset$ // variable will hold optimal sets

7. $i \leftarrow n$ // index to walk backwards thru arrays

ex:



A



$S = \{ \quad \}$

Runtime: $O(n)$

7. // Work backwards through FOR loop code

O(n) While $i \geq 2$ // include vertex v_i?

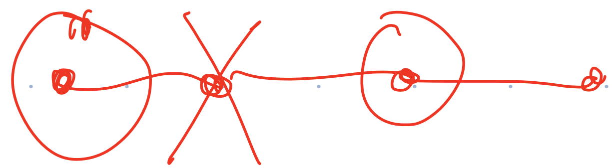
    if $A[i] = A[i-1]$:
        $i \leftarrow i-1$
    else
        $S \leftarrow S \cup \{v_i\}$
        $i \leftarrow i-2$

$A[i] \leftarrow \max\{A[i-1], A[i-2]+w(v_i)\}$

8 // Base case(s)

    If $i == \boxed{1}$
        $\boxed{S \leftarrow \{v_1\} \cup S}$

9 Return S



Proof: Explanation of recurrence relation

Why "dynamic programming"