Learning Goals

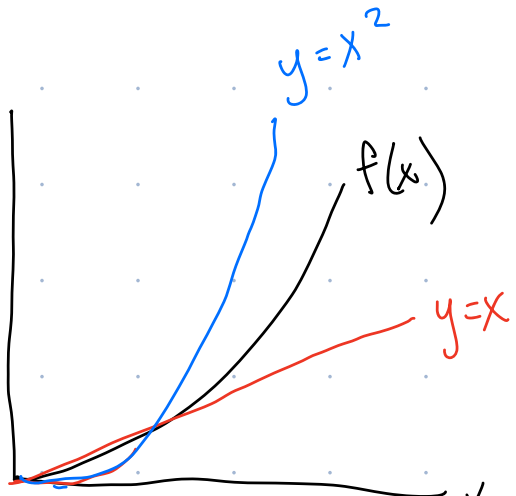
- Describe + benchmark Max Weight Independent Set problem
- Create a recurrence for MWIS on a line
- Design a Dynamic Programming alg for MWIS

Announcements

- Accessing TA feedback

Exit Tickets

- Greedy algorithm
- Only take 1 arm of recursion
- Omega Ω and recursive tree analysis
- 3 vertices at a time = Divide + Conquer
- House Robber Problem?

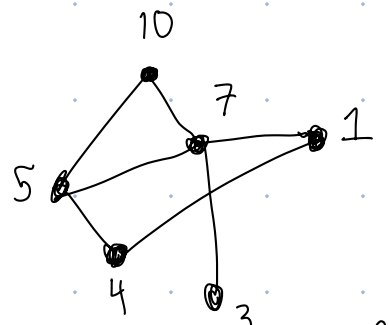


$f(x) = O(x^2)$

$f(x) = \Omega(x)$

Max Weight Independent Set

Input: Graph $G = (V, E)$
Weights $w: V \to \mathbb{Z}^+$



Output: $S \subseteq V$ 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"

Objective function

Applications:

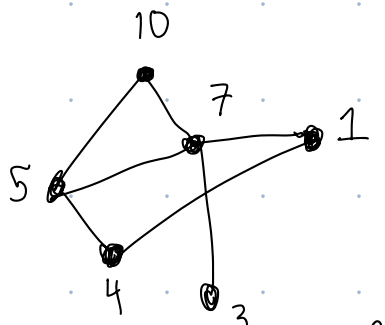
- Cell tower transmission
- Choosing franchise locations
- Party Invites
- Scheduling
- House robbing (I have some ethical concerns)

General Graph: HARD

Line Graph: Easy, using dynamic programming

Max Weight Independent Set (MWIS)

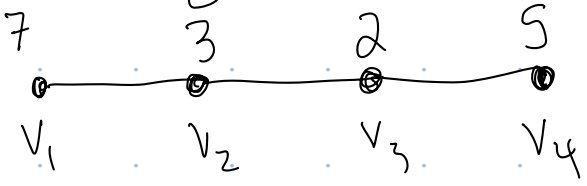
Input: Graph $G = (V, E)$
weights $w: V \to \mathbb{Z}^+$



Output: $S \subseteq V$ 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:

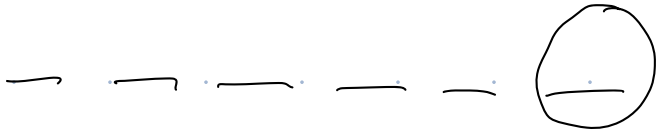


A) 0 B) 8 C) 9 D) 12

↑ $\{v_1, v_4\}$

Dynamic Prog. Approach

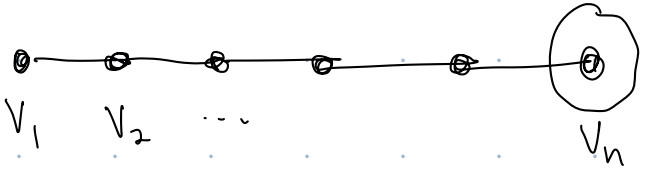
Recall: # of n bit strings with 2 consecutive ones



← 2 cases, 0 or 1

↙ $T(n-1\ldots)$ ···· ↓ $T(n-2)$ ···

MWIS



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

Let's call S_i the MWIS of first i vertices

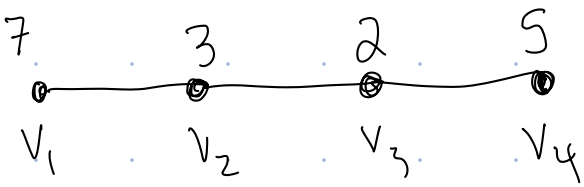
- $v_n \in S$, then $S_n =$ ___ Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$
- $v_n \notin S$, then $S_n =$ ___ $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

Name, pronouns, best thing watched/listened to

Write pseudocode for brute force approach, analyze runtime

Brainstorm greedy, divide + conquer approaches

Brute Force




```
MWIS(G = (V, E), w)
max S ← ∅
max W ← 0
O(2^n) For each set S ⊆ V:
    If S is I.S.:                    O(n)
        w ← W(S)
        if w > max W then            O(1)
            max S ← S
            max W ← w
Return max S
```

$O(n2^n)$ Bad!

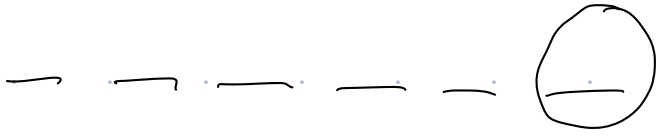
```
S is I.S.:                           O(n)
For i ← 1 to n − 1:
    If v_i ∈ S and v_{i+1} ∈ S:
        Return False
Return true
```

$O(2^n \cdot n)$ Bad!

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

Dynamic Prog. Approach

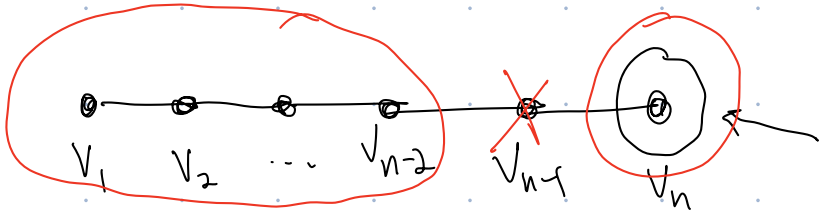
Recall: # of n bit strings with 2 consecutive ones



← 2 cases, 0 or 1

↙ $T(n-1\ldots)$ .... ↓ $T(n-2)$ ....

MWIS

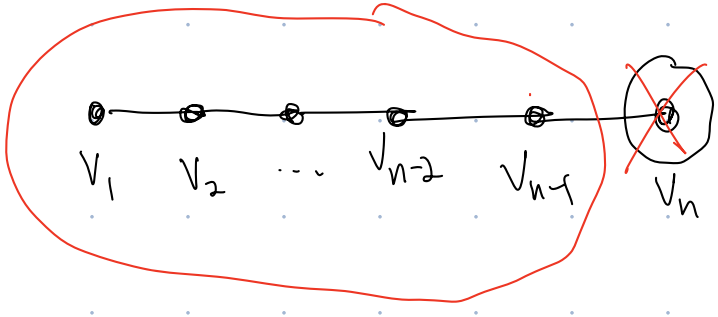


← 2 cases: $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

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

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

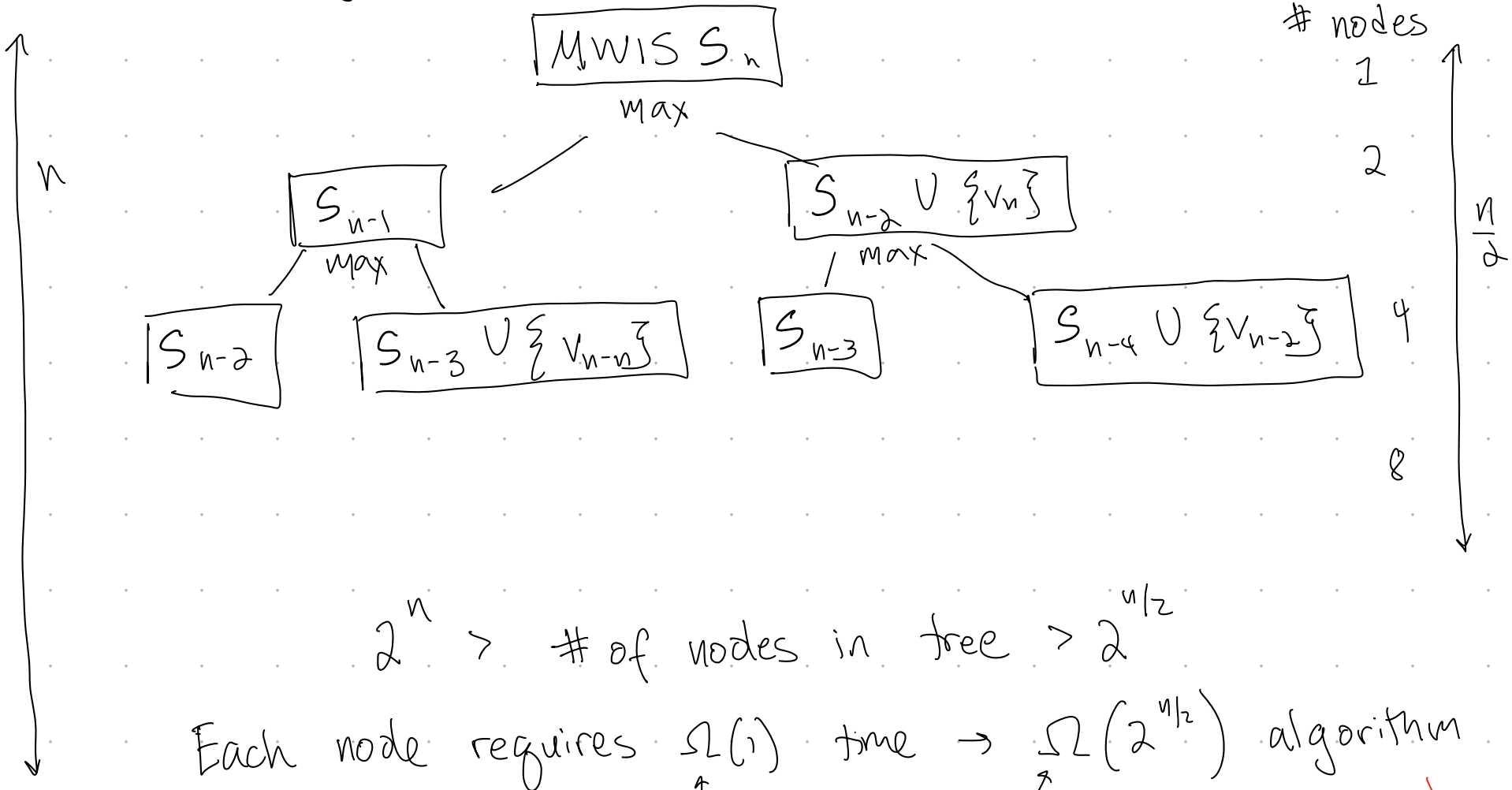


$$S_n = \begin{cases} \max \text{ weight} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \text{base case} \end{cases}$$

← 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 → $\Omega(2^{n/2})$ algorithm

↑ asymptotic lower bound

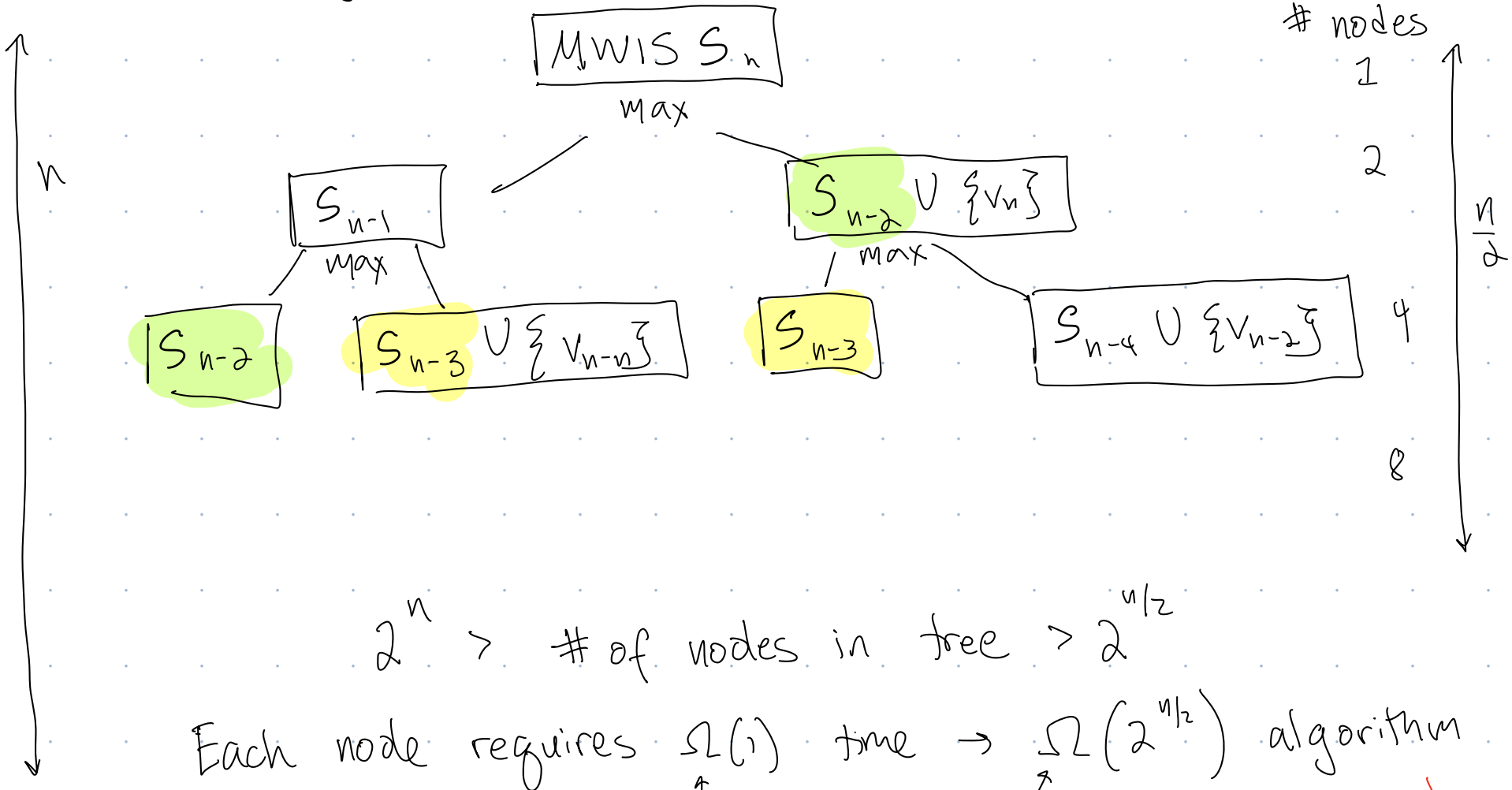
Bad!

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

← 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 → $\Omega(2^{n/2})$ algorithm

↑ asymptotic lower bound

Bad!

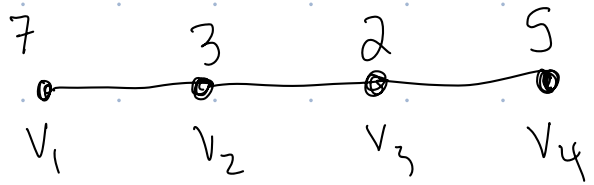
How many unique subproblems are there?

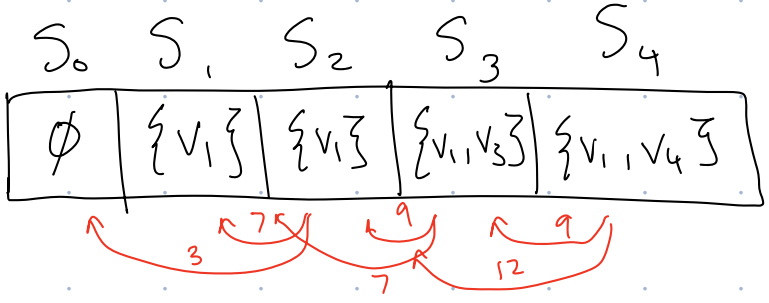
A) $\sqrt{n}$ B) $\frac{n}{2}$ C) n D) n^2

$G_1, G_2, G_3, G_4 \ldots G_n$

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

$$S_n = \begin{cases} \max\text{wt} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \text{B.C.} \end{cases}$$





Trick 1: Fill up this way

Trick 0: Start at empty problem

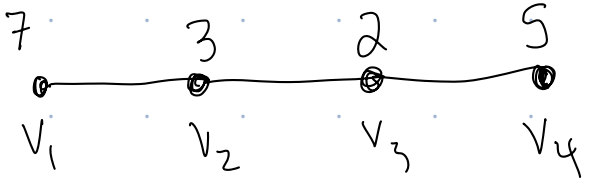
Trick 2: Store Objective Function Value

$$S_n = \begin{cases} n>1: \max_{\text{wt}} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ n \le 1: \text{B.C. } n=0,1 \end{cases} \rightarrow W(S_n) = \begin{cases} n>1: \max \begin{cases} W(S_{n-1}) \\ W(S_{n-2}) + w(v_n) \end{cases} \\ n=1 \quad w(v_1) \\ n=0 \quad \phi \end{cases}$$

array A

$W(S_0)$	$W(S_1)$	$W(S_2)$	$W(S_3)$	$W(S_4)$
0	7	7	9	15

0+3, 7, 7+2, 7+5, 9



Go slowly

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

// Determine optimal set

5. $S \leftarrow \emptyset$
6. $i \leftarrow n$
7. While $i \geq 1$:
   - if $A[i] = A[i-1]$:
     - $i \leftarrow i-1$
   - else
     - $i \leftarrow i-2$
     - $S \leftarrow S \cup \{v_n\}$
8. Return S

ex:

A

$S = \{ \quad \}$

Runtime: $O(n)$

Proof: Explanation of recurrence relation

Ethics: Why "dynamic programming"