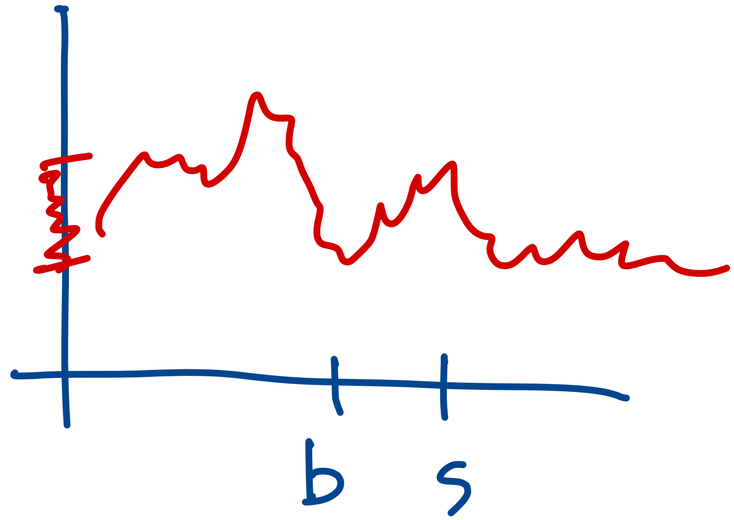
Lecture 23: Profits and Weighted Intervals

COSC 311 *Algorithms*, Fall 2022

Overview

1. Profit Maximization via Dynamic Programming
2. Weighted Interval Scheduling



Profit Maximization

Input. Array a of size n

- $a[i]$ = price of Alphabet stock on day i

Output. Indices b (buy) and s (sell) with $1 \leq b \leq s \leq n$ that maximize profit

- $p = a[s] - a[b]$

Another (Recursive) Procedure?

- consider last day, n
- two cases for optimal solution:
  1. max profit achieved by selling on day n
  2. max profit achieved by selling before day n

Questions.

1. In case 1, how should we determine buy date?
   - find minimum price in $a[1..n]$
2. In case 2, how should we compute max profit?
   - recursively find max profit for $a[1..n-1]$

Recursive Procedure

```
MaxProfit(a, n):
  if n = 1 then return 0
  min <- FindMin(a, n)
  max <- MaxProfit(a, n-1)
  return max(a[n] - min, max)
```

Base case

min stock value to day n

Running time?

- $\Theta(n^2)$
- n method calls of sizes $n, n-1, n-2, \ldots, 1$
- running time is $\Theta(n + (n-1) + \cdots + 1) = \Theta(n^2)$

$\frac{n(n+1)}{2}$

Memoizing `MaxProfit`

Create two arrays:

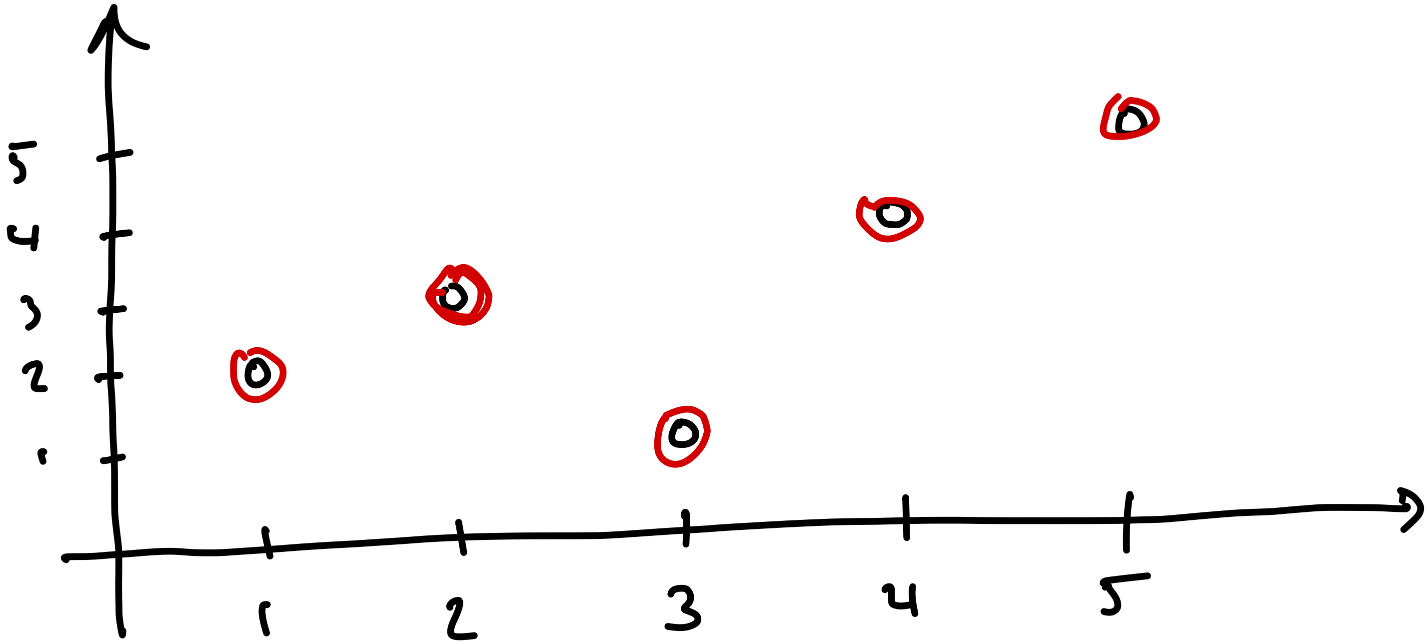
1. `min[i]` stores minimum value in `a[1..i]`
2. `max[i]` stores maximum profit achievable by selling up to time `i`

Question. How to update these arrays?

$$\text{min}[i+1] \leftarrow \text{Min}(\text{min}[i], a[i+1])$$

$$\text{max}[i+1] \leftarrow \text{Max}(\text{max}[i], a[i+1] - \text{min}[i+1])$$

Example



min :	2	2	1	1	1
max :	0	1	1	3	4

Memoized Maximum Profit

```
MMaxProfit(a):
  initialize arrays min, max
  min[1] <- a[1]
  max[1] <- 0
  for i from 2 to n do
    min[i] <- Min(min[i-1], a[i])
    max[i] <- Max(max[i-1], a[i] - min[i])
  endfor
  return max[n]
```

Correctness

Claim. For every i, `max[i]` stores the maximum profit achievable by selling on a day $s \leq i$.

Proof. Induction on i...

Running Time?

```
MMaxProfit(a):
    initialize arrays min, max
    min[1] <- a[1]
    max[1] <- 0
    for i from 2 to n do
      min[i] <- Min(min[i-1], a[i])
      max[i] <- Max(max[i-1], a[i] - min[i])
    endfor
    return max[n]
```

O(n)

O(1)

n-1

O(1)

O(1)

O(n)

Optimization

Can do without arrays for `min` and `max`

```
MMaxProfit(a):
  min <- a[1]
  max <- 0
  for i from 2 to n do
    min <- Min(min, a[i])
    max <- Max(max, a[i] - min)
  endfor
  return max[n]
```

Exercise

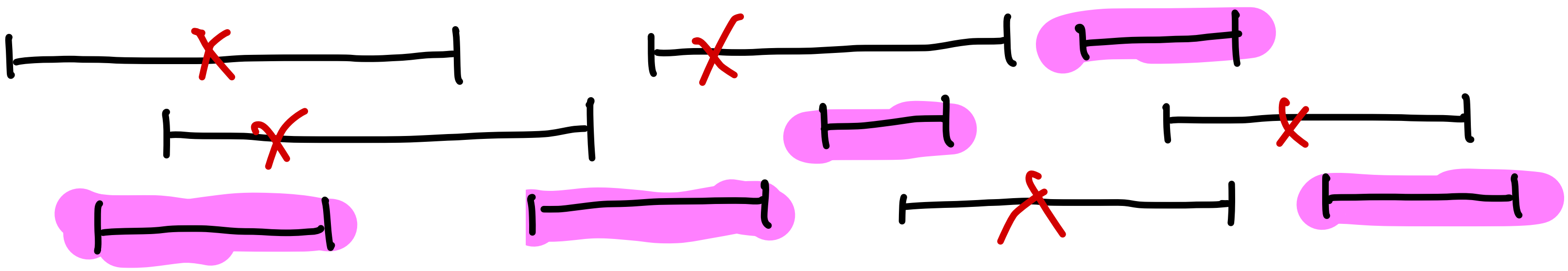
Update `MMaxProfit` to return the buy/sell days in addition to the maximum achievable profit.

Weighted Interval Scheduling

Previously

Interval Scheduling:

Input. A set R of n intervals
$r_1 = [s_1, t_1], r_2 = [s_2, t_2], \ldots, r_n = [s_n, t_n]$



Output. A collection of intervals from R that is:

1. *feasible* no two intervals overlap
2. *maximum* the largest possible feasible collection

Maximum feasible collection can be found in $O(n \log n)$ time using a greedy algorithm

Today

Weighted Interval Scheduling:

Input.

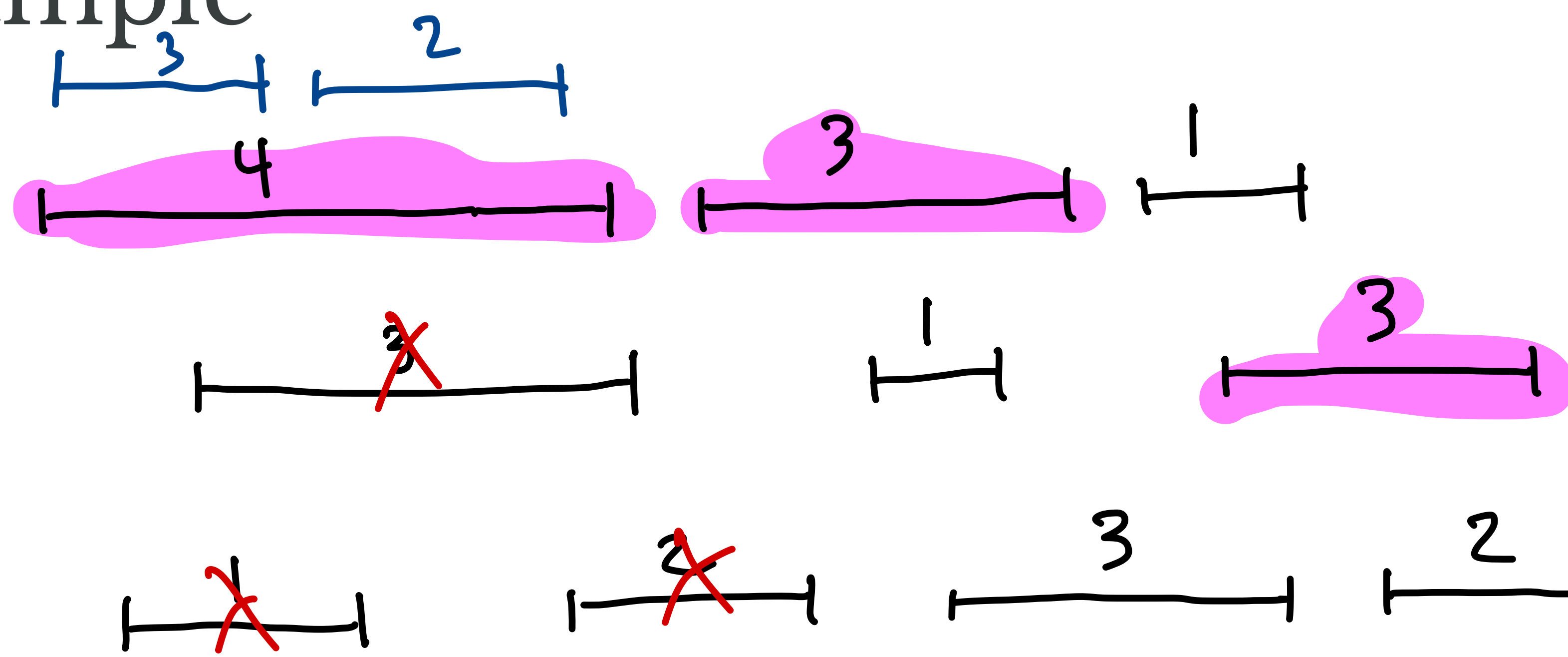
1. A set R of n intervals
   $r_1 = [s_1, t_1], r_2 = [s_2, t_2], \ldots, r_n = [s_n, t_n]$
2. For each interval $r \in R$, a **weight** $w(r) > 0$
   - e.g., weight = profit from serving request r

Output. A collection of intervals from R that is

1. *feasible* no two intervals overlap
2. *maximum weight* choice maximizes sum of $w(r)$ for chosen r

Note: equivalent to (unweighted) interval scheduling when all weights are the same

Example



Total weight 4+3+3 = 10

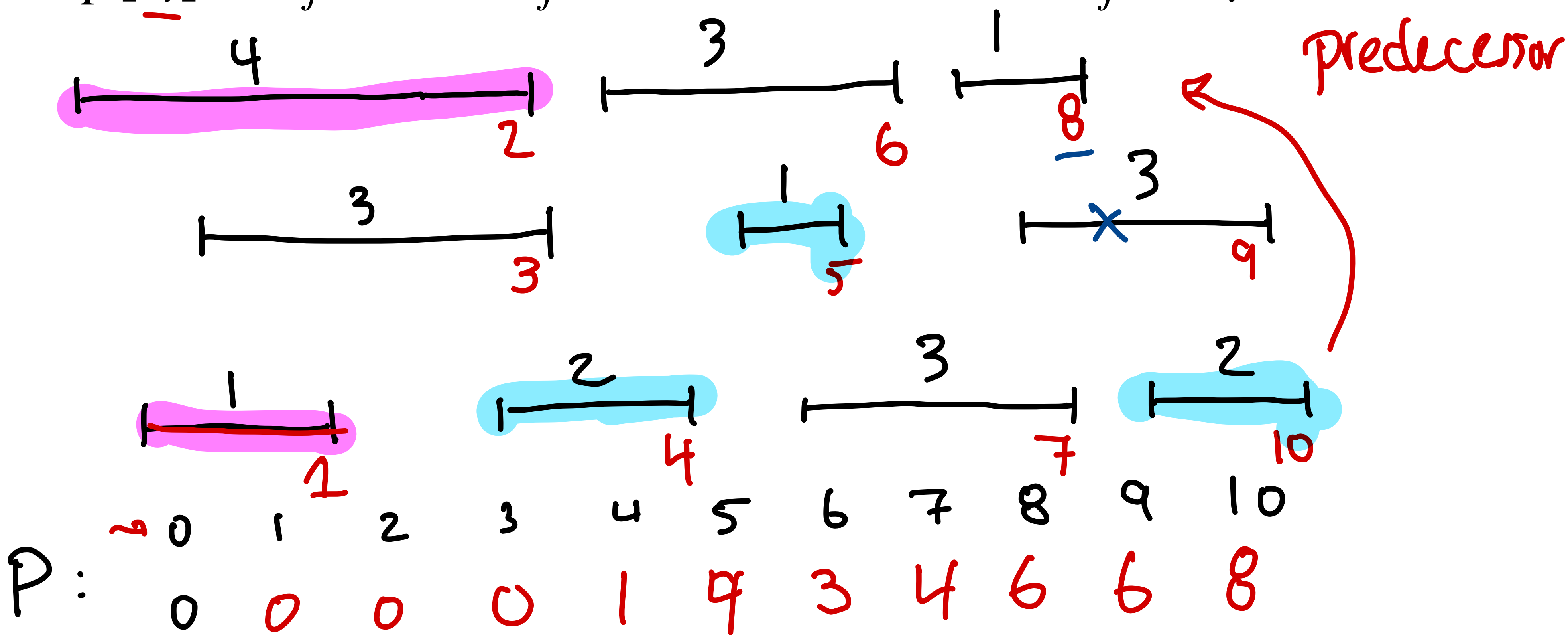
Question: Pick an interval to include or not to include?

Exercise

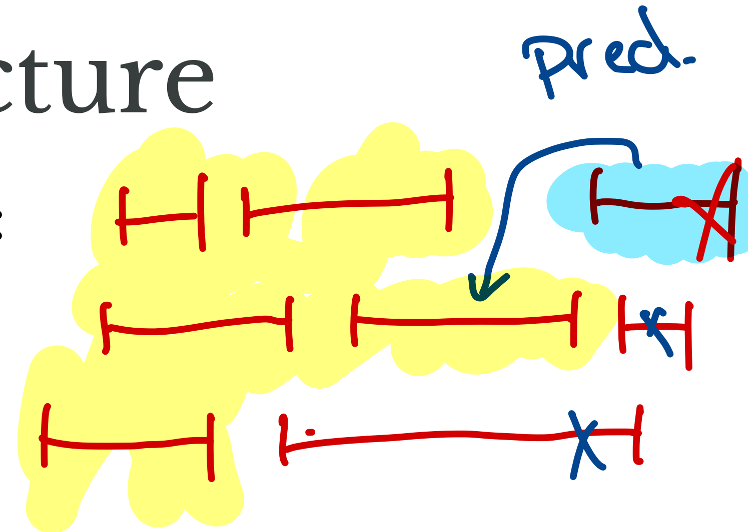
Construct an example for which the greedy algorithm for *unweighted* interval scheduling does not find a maximum weight solution.

Pre-processing

1. Sort intervals by end date
   - $[s_1, t_1], [s_2, t_2], \ldots, [s_n, t_n]$
   - $t_1 \leq t_2 \leq \cdots \leq t_n$
2. For each interval $r_i = [s_i, t_i]$ define
   - $p[r_i] = r_j$ where r_j is last interval with $t_j < s_i$



Optimal Solution Structure



Two cases for optimal solution opt:

1. last interval r_n is in opt
2. last interval r_n is not in opt

Questions.

1. What is the structure of optimal solution in case 1?

   Opt = r_n + Opt(from 1 to P[n])

2. What is the structure of optimal solution in case 2?

   Opt = opt from 1 to n-1

A Recursive Solution

```
MaxWeightSchedule(w, p, n):
  if n = 0 then return 0
  opt-n <- w[n] + MaxWeightSchedule(w, p, p[n])
  opt-no-n <- MaxWeightSchedule(w, p, n-1)
  return Max(opt-n, opt-no-n)
```

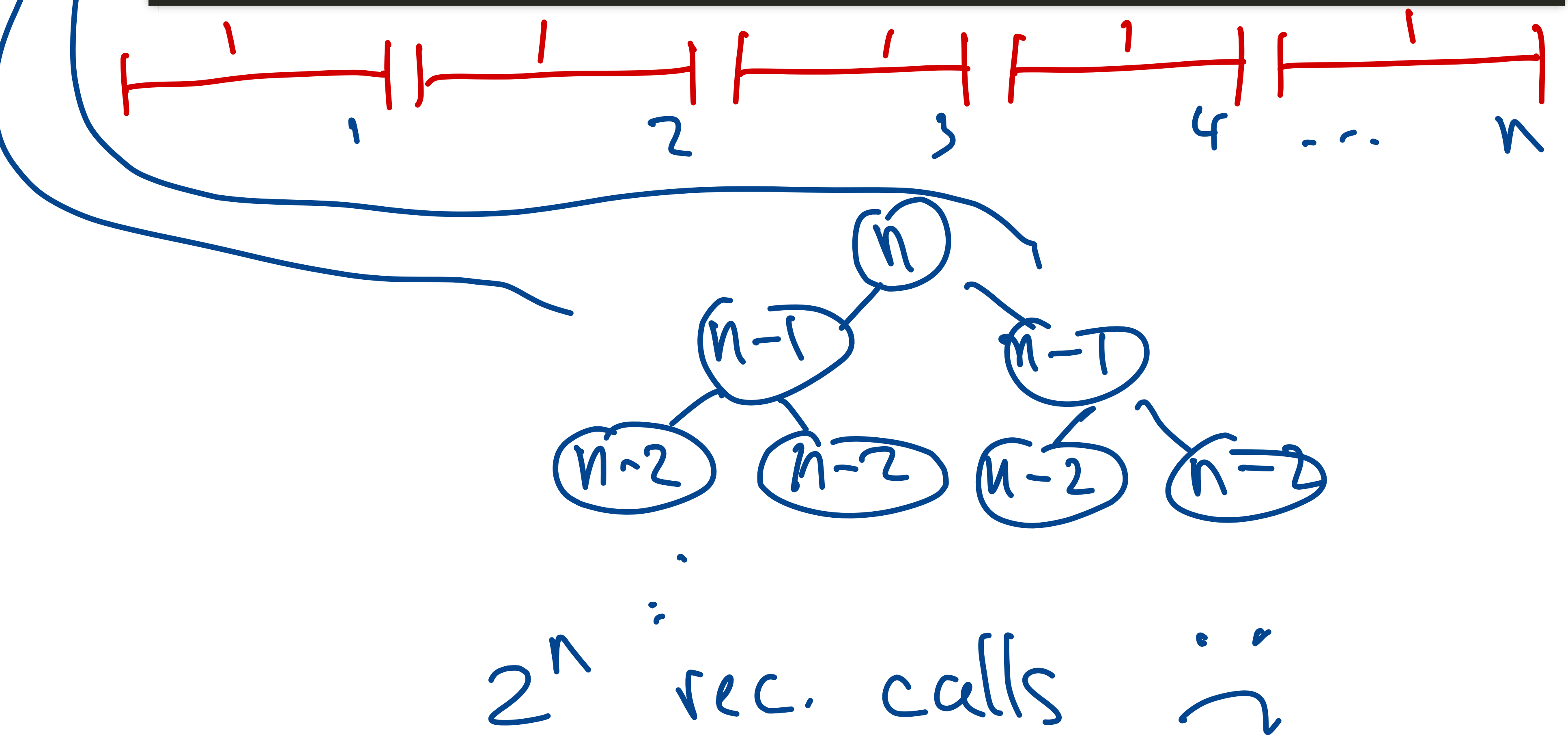
opt sol inc. n

opt not inc. n

Correctness?

Running Time?

```
MaxWeightSchedule(w, p, n):
  if n = 0 then return 0
  opt-n <- w[n] + MaxWeightSchedule(w, p, p[n])
  opt-no-n <- MaxWeightSchedule(w, p, n-1)
  return Max(opt-n, opt-no-n)
```



Recursion to Iteration

Idea. Store array `max`:

- `max[i]` is maximum weight of schedule consisting of intervals $r_1, r_2, \ldots, r_i$

Question. How to initialize/update `max` values?