Dynamic Programming

General approach to solving opt. problems using ~~p~~ dynamic programming

1- Characterize the structure of an opt. solution

2- Recursively define the value of an opt. solution.

3- Compute the value of an opt. sol. in a bottom up fashion

4- Construct an opt. sol. from computed info.

General approach to solving opt. problems using dynamic Programming

1- Characterize the structure of an opt. sol.

2- Recursively define the value of an opt. sol.

3- Compute the value of an opt. sol. in a bottom up ~~to~~ fashion

4- construct an opt. sol. from computed info.

Problem Def.

We have 1 resource

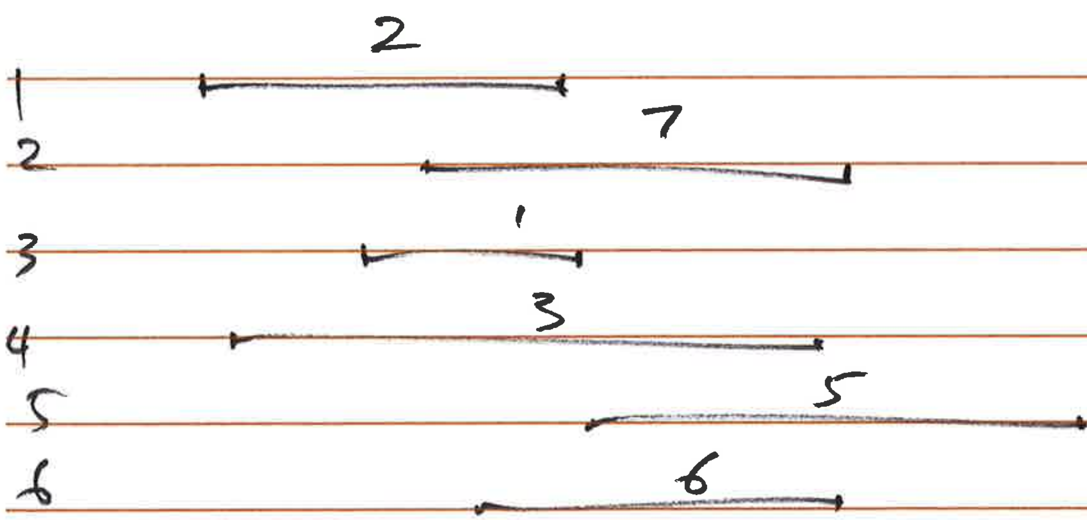
" " $\underline{n}$ requests labeled $\underline{1}$ to $\underline{n}$

each request has start time s_i

finish time f_i

weight w_i

Goal: Select a subset $S \subseteq \{1..n\}$ of mutually compatible intervals to maximize $\sum_{i \in S} w_i$



observation:

- Either job i is part of the opt. sol. or it isn't

Case 1. if it is, value of the opt. sol. = w_i + value of the opt. sol. for the subproblem that consists only of ~~Compati~~ compatible requests w/ i.

Case 2: if it isn't, value of the opt. sol. = ~~Value~~ value of opt. sol. w/o job i.

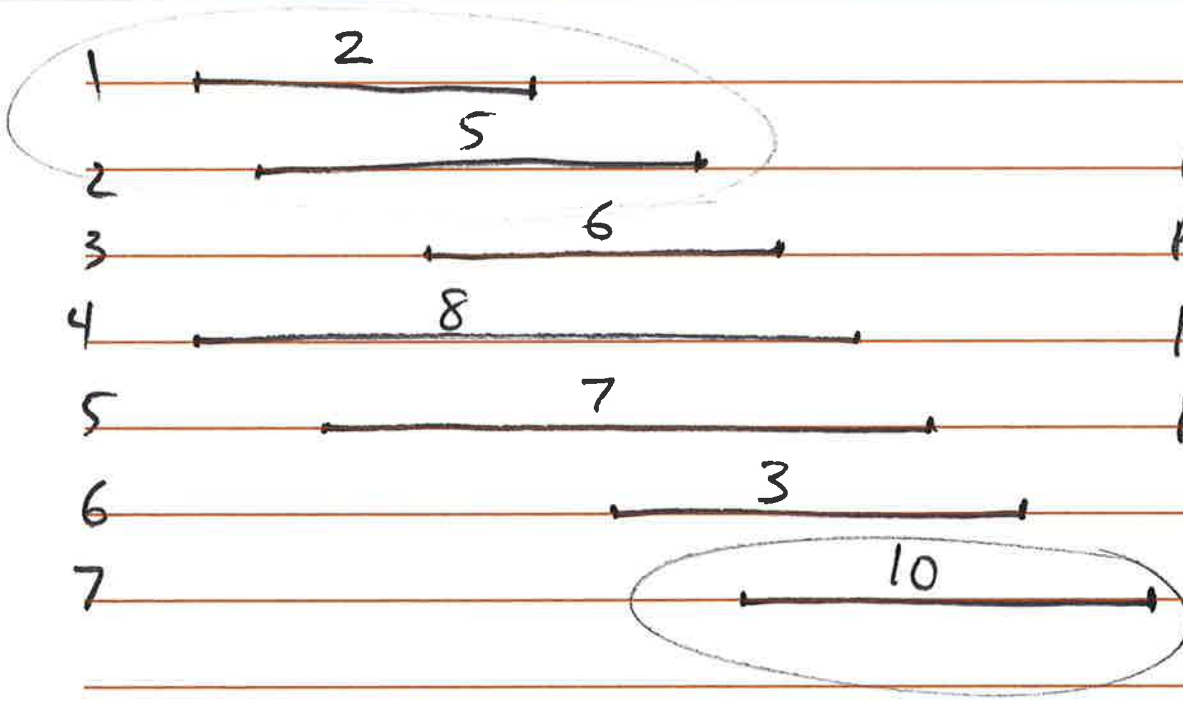
explore both paths recursively and find out which is opt!

value of opt. sol. = Max (value for case 1, value for case 2)

Sort requests in order of non-decreasing finish time

$$f_1 \leq f_2 \leq \cdots \leq f_n$$

Define $p(j)$ for ~~an~~ an interval j to be the largest index $i < j$ such that interval i & j are disjoint



$p(1) = 0$

$p(2) = 0$

$p(3) = 0$

$p(4) = 0$

$p(5) = 0$

$p(6) = 1$

$p(7) = 2$

Def. Let O_j denote the opt. solution to the problem consisting of requests $\{1..j\}$

Let $OPT(j)$ denote the value of O_j

$O_7 = \{2, 7\}$

$OPT(7) = 15$

Case 1: $j \in O_j \Rightarrow OPT(j) = w_j + OPT(p(j))$

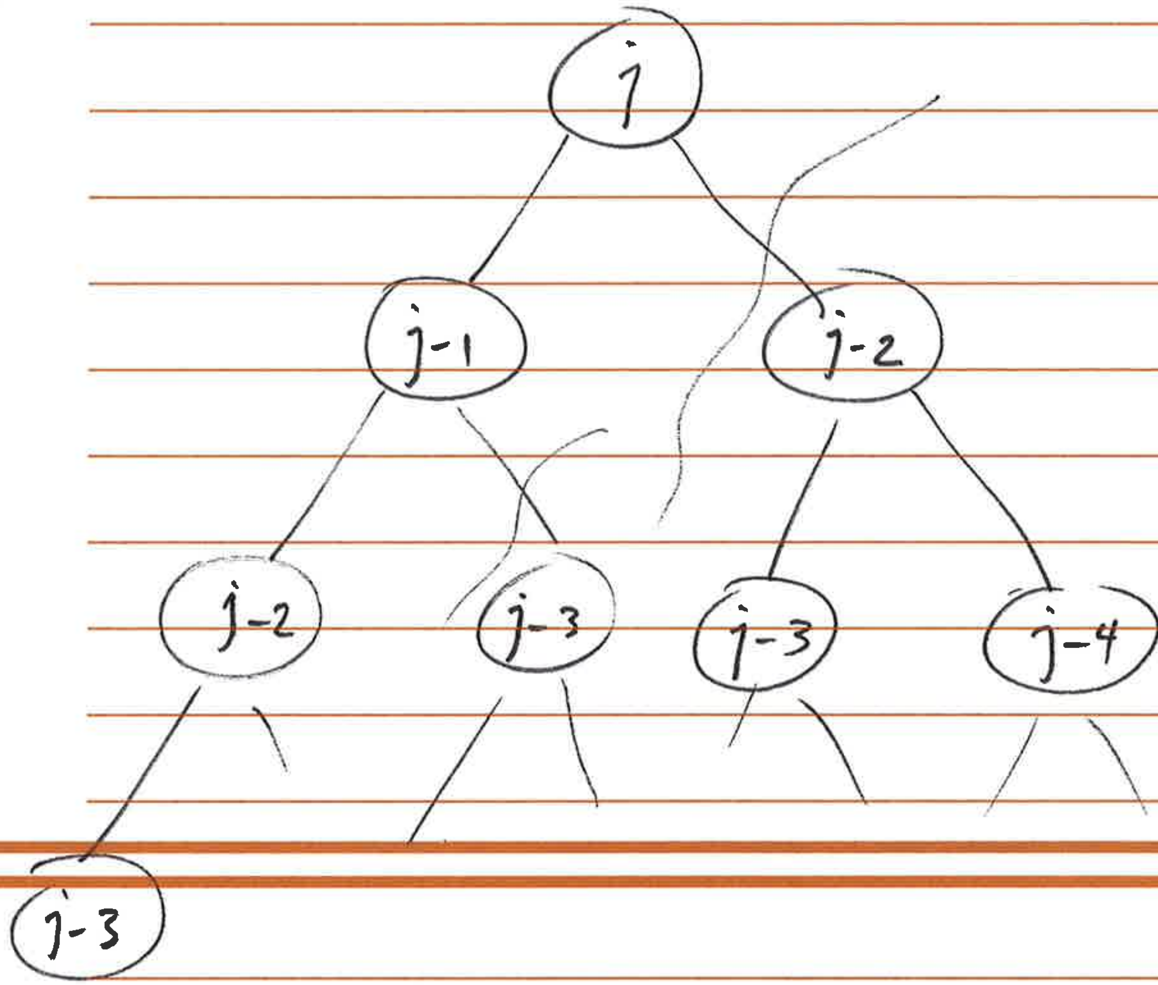
Case 2: $j \notin O_j \Rightarrow OPT(j) = OPT(j-1)$

Solution:

```
Compute-opt(j)
  if j=0 then
    return 0
  else
    return Max(w_j + Compute-opt(p(j)),
               Compute-opt(j-1))
  endif
```

runs in exponential time

$$T(n) = T(n-1) + T(n-2) + \cancel{c}$$



memoization

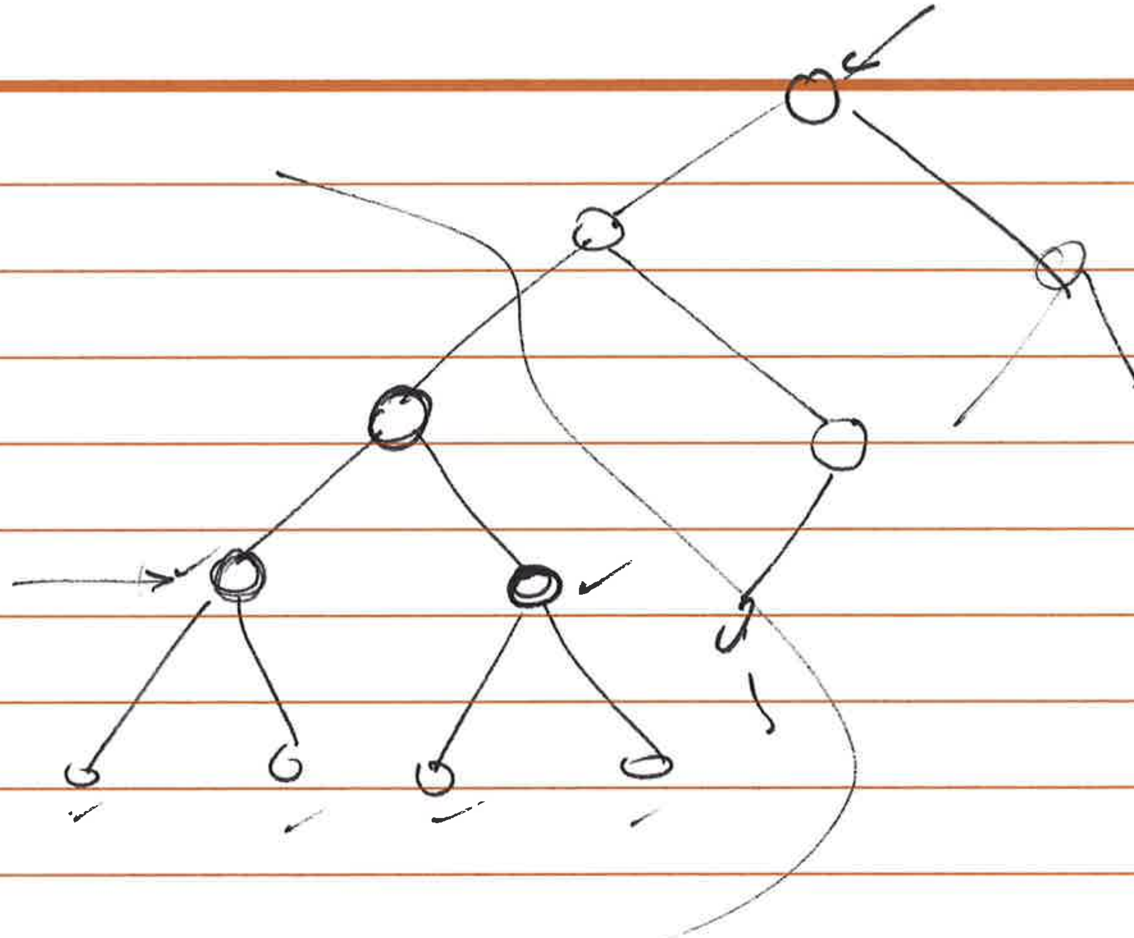
store the value of Compute-opt in a globally accessible place the first time we compute it. Then simply use this value in place of all future recursive calls.

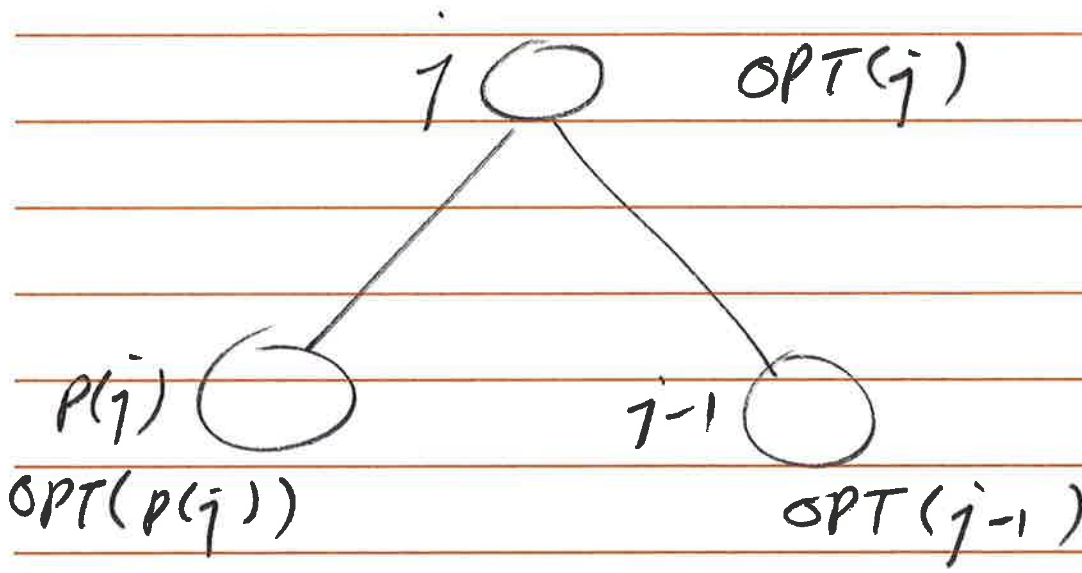
```
M_Compute_opt ( j )
  if j=0 then
     return 0
  else if M[j] is not empty then
     return M[j]
  else
     define M[j] = Max( w_j +
          M_compute_opt( p(j) ),
          M_Compute_opt( j-1 ) )
  return M[j]
```

$O(n)$

overall time =

Sort	$O(n \lg n)$
Construct $p(j)$	$O(n \lg n)$
time spent in M_Compute_opt	$\leq O(n)$
total	$O(n \lg n)$





j
OPT(j)
p(j)
j-1
OPT(p(j))
OPT(j-1)

j belongs to O_j iff

$$\underbrace{w_j + OPT(p(j))}_{} \geq OPT(j-1)$$

→ Find-Solution

if $j > 0$ then

if $w_j + M[p(j)] \geq M[j-1]$ then

output j together w/ the

results of Find-Solution$(p(j))$

else

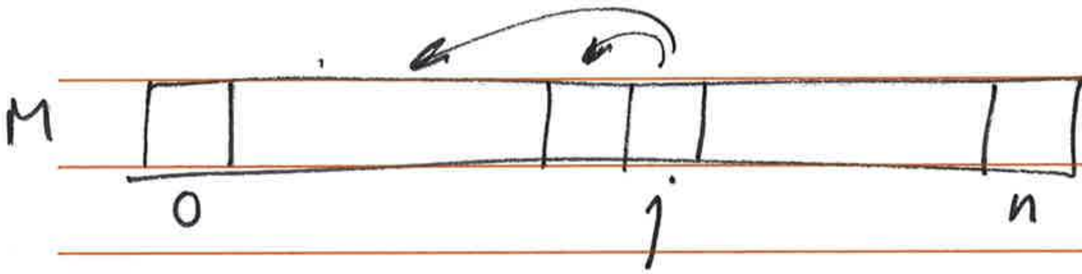
output the results of

Find-solution$(j-1)$

endif

endif

→ takes $O(n)$



$M[0] = 0$

for $i = 1$ to n

$\quad M[j] = \mathrm{Max}(w_j + M[p(j)], M[j-1])$

endfor

takes $O(n)$