CSC373

ALGORITHMS

Max-WEIGHTED INTERVAL SCHEDULING (RECAP)

Given : n intervals $[s_i, f_i]$ $\quad s_i < f_i$
weights $wt(i)$

Output : A subset I of non-overlapping intervals with maximum total wt $\sum_{i \in I} wt(i)$

PREPROCESS : SORT BY FINISH TIMES

$f_1 \leq f_2 \leq \ldots \leq f_n$

COMPUTE $p(j)$ = max-index $i < j$ st interval i does not overlap with j

RECURSION PROVED :

$$OPT(\{1, \ldots, j\}) = \max \{ OPT(\{1, \ldots, j-1\}), wt(j) + OPT(\{1, \ldots, p(j)\}) \}$$

BASE CASE :

$$OPT(\{\}) = 0$$

MAX-WT-IS (RECURSIVE)

1. Sort intervals by finishing time
2. Compute $p(j)$ for all j by Binary Search
3. RETURN R-OPT(n)

$p(j)$ is such that $f_{p(j)} < s_j \leq f_{p(j)+1}$

j does not overlap w/ $\{1, \ldots, p(j)\}$

$OPT(\{1, \ldots, j\}) \equiv OPT(j)$

```
R-OPT(j)
  IF j == 0
    RETURN 0
  RETURN MAX( R-OPT(j-1),
              wt(j) + R-OPT(p(j)) )
```

What happens if $p(j) = j-1$ $\forall j$

$T(n) = 2T(n-1) + O(1)$

$\Omega(2^n)$!

EXPONENTIAL

ALL SUBPROBLEMS ARE OF TYPE $OPT(\{1, \ldots, j\})$

MAX-WT-IS

1. Sort intervals by finishing time $O(n \log n)$
2. Compute $p(j)$ for all j by binary search $O(n \log n)$
3. $M = [-1] * (n+1)$ $O(n)$
4. $M[0] = 0$

RETURN M-OPT(n) $\equiv$ OPT($\{1, \ldots, n\}$)

M-OPT(j):

IF M[j] == -1 $O(n)$ (IF NOT BEEN COMPUTED)

{ M[j] = MAX(M-OPT(j-1), wt(j) + M-OPT(p(j)))

Executed exactly once for each j

RETURN M[j]

MEMOIZE

USING M[j]

j = 0, ... n

M[j] = OPT($\{1, \ldots, j\}$)

SUBPROBLEMS

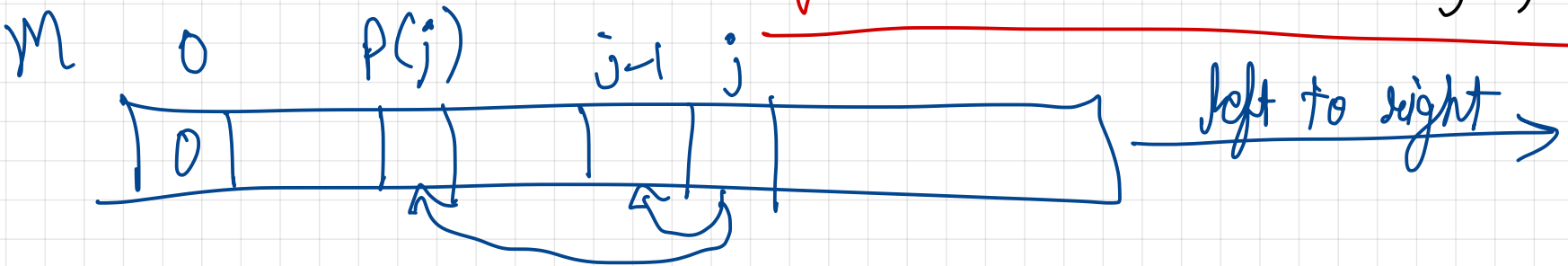
n OPT($\{1, \ldots, n\}$)

⋮

1 OPT($\{1\}$)

0 OPT($\{\}$)

OPT($\{1, \ldots, j\}$) = max wt of a non-overlapping subset of $\{1, \ldots, j\}$



MAX-WT-IS (NON-RECURSIVE)

1. Sort intervals by finishing time — $O(n \log n)$
2. Compute $p(j)$ for all j by Binary Search — $O(n \log n)$

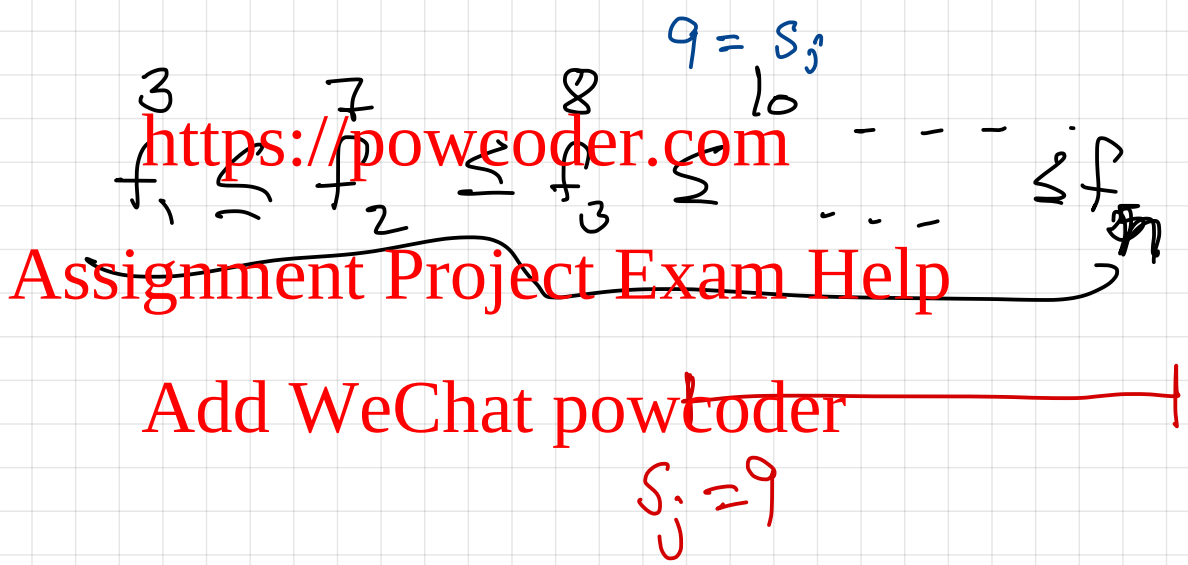
3. $M = [-1] * (n+1)$
4. $M[0] = 0$ $n+1$ excluded
5. FOR j IN RANGE $(1 : n+1)$: — $O(n)$

$$M[j] = \max\{ M[j-1], wt(j) + M[p(j)] \}$$

RETURN $M[n]$ — $O(n \log n)$

Adv of non-recursive:

- Avoid recursive overhead
- Recursion depth irrelevant



RECOVERING THE OPTIMAL SOLUTION

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max(OPT(j-1), \\ \quad wt(j) + OPT(p(j))) \end{cases}$$

$OPT(j)$ = max wt of a subset of non-overlapping intervals from $\{1, \ldots, j\}$

$S(j)$ = max wt, subset from $\{1, \ldots, j\}$ non-overlapping

$$S(j) = \begin{cases} \emptyset & j = 0 \\ S(j-1) & \text{if } OPT(j) = OPT(j-1) \\ \{j\} \cup S(p(j)) & \text{o/w} \end{cases}$$

KNAPSACK PROBLEM

GIVEN : n items with weight w_i, cost c_i; budget B

GOAL : Find a subset I of items with total cost $\sum_{i \in I} c_i \leq B$ and maximum possible weight $\sum_{i \in I} w_i$

ASSUME : All c_i and B are integers

$OPT(b, \{1, \ldots, j\})$ = Weight of a subset of items from $\{1, \ldots, j\}$ satisfying the budget constraint $\sum_{i \in I} c_i \leq b$ & with maximum total wt $\sum_{i \in I} w_i$

budget

$OPT(B, \{1, \ldots, n\})$ ← Original Problem

$OPT(b, \{1, \ldots, j\})$

don't pick j → best of → $OPT(b, \{1, \ldots, j-1\})$

pick j (only if $b \geq c_j$) → $w_j + OPT(b - c_j, \{1, \ldots, j-1\})$

$b - c_j$: cost; $OPT(b, \{\}) = 0$ ($j = 0$); made to you

$$= \begin{cases} 0 & \\ OPT(b, \{1, \ldots, j-1\}) & , j > 0 \ \& \ b < c_j \\ MAX\left(OPT(b, \{1, \ldots, j-1\}), \right. & \text{o/w} \\ \quad \left. w_j + OPT(b - c_j, \{1, \ldots, j-1\}) \right) & \end{cases}$$

Identify Subproblems

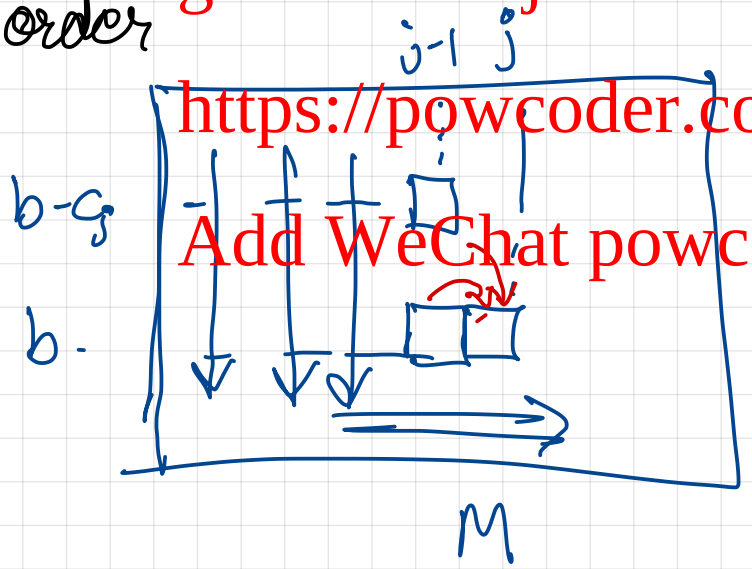
can't be on arbitrary subset

$OPT(b, \{1, \ldots, j\})$ $\quad 0 \le b \le B$, $0 \le j \le n$

Memoization data-structure

$M[b, j] = OPT(b, \{1, \ldots, j\})$

Evaluation order



column-wize
left to right

KNAPSACK

M ← 2-d array [0...B, 0...n]

base { FOR b in RANGE(0, B+1):
M[b, 0] = 0

FOR j in RANGE(1 : n+1):

FOR b in RANGE(0, B+1):

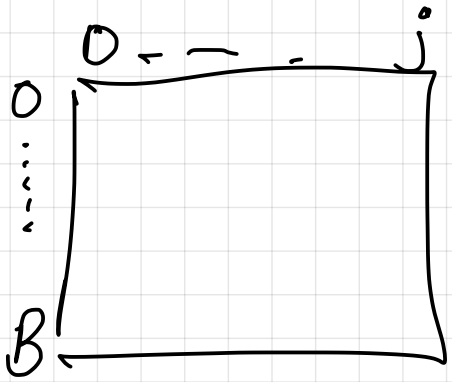
IF $b \geq c_j$

$M[b,j] = MAX(M[b,j-1],$
$w_j + M[b-c_j, j-1])$

ELSE $M[b,j] = M[b,j-1]$

RETURN M[B, n]

excluded



$M[b,j]$
$= OPT(b, \{1, \ldots, j\})$

$\Theta(nB)$ time
space

$\Theta(nB)$

does this algo run in polynomial time in size of the input?

exponential

in input size

Since B is represented using $O(\log B)$ bits

pseudo-polynomial

bits to represent

$2^k \longrightarrow k$ bits

$B \longrightarrow \log B$ bits

$\underbrace{10000}_{k \text{ zeros}}$

$$OPT(b, \{1, \ldots, j\}) = \begin{cases} 0 & j = 0 \\ OPT(b, \{1, \ldots, j-1\}) & j > 0 \ \& \ b < c_j \\ \max\left(OPT(b, \{1, \ldots, j-1\}),\ w_j + OPT(b - c_j, \{1, \ldots, j-1\}) \right) & \text{o/w} \end{cases}$$

$$S(b, j) = \begin{cases} \{\} & j = 0 \\ S(b, j-1) & j > 0,\ b < c_j \\ & \text{if } OPT(b, j) = OPT(b, j-1) \\ S(b, j-1) & \\ \{j\} \cup S(b - c_j, j-1) & \text{o/w} \end{cases}$$

```
j = n
b = B
best = []
while j > 0 :
    if OPT(b, j) == OPT(b, j-1)
        j ← j-1
    else
        best.append(j)
        j ← j-1
        b ← b-c_j
return best
```