# COP3503 2/22/2022

## FROm now on ⇒ Algorithms!
## 3 Categories of types of Algs

1. Greedy
2. Divide + Conquer
3. Dynamic Programming

→ Are ones where you take the "greedy" step (what seems to be the easiest) w/o considering all the possibilities ⇒ OPPOSITE of BRUTE FORCE

My categorization of problems in relation to greedy strategies

(1) The natural greedy idea solves the problem correctly! ⇒ fractional knapsack

Most Dangerous

(2) Where a greedy solution exists, but the most natural greedy idea is INCORRECT! ⇒ single room scheduling

(3) No known greedy solution exists but lots of people think there is one. ⇒ 0/1 Knapsack

# Fractional Knapsack

Thief has knapsack capacity W lbs.

| | | |
|---|---|---|
| Item$_1$ | \$3/lb | 10 lbs available |
| Item$_2$ | \$5/lb | 6 lbs |
| Item$_3$ | \$2/lb | 15 lbs |
| Item$_4$ | \$8/lb | 5 lbs |

$W = 17$

Sort by value/lb and take in order!

$\$8 \times 5 + \$5 \times 6 + \$3 \times 6 = \$88$

ran out of room (leave 4 lbs of Item$_1$)

Prove optimal via exchange argument.

My knapsack K ⇒ produced via algorithm

Yours K' you claim is more valuable

K' must be diff K.

⇒ if you didn't take all of Item$_2$ or Item$_4$ then you subbed it w/ something less valuable of equal weight ⇒ mine IS better!

# Single Room Scheduling

n requests to use room

| | Item | Start | End | |
|---|---|---|---|---|
| | 1 | 10 | 40 | x |
| 1 | 2 | 5 | 25 | |
| | 3 | 20 | 80 | x |
| | 4 | 30 | 90 | x |
| 2 | 5 | 35 | 40 | |
| | 6 | 45 | 80 | x |
| 3 | 7 | 50 | 60 | |
| 4 | 8 | 70 | 80 | |

Can schedule $I_i$ and $I_j$ if

$I_i.end \leq I_j.start$

or

$I_j.end \leq I_i.start$

Ideas: Shortest Event

| | | |
|---|---|---|
| ✓ | 5 | 100 |
| | 99 | 101 |
| ✓ | 100 | 500 |

} If I take this I get wrong ans

Start Time

| | |
|---|---|
| 5 | 1000 (oops!) |
| 6 | 7 |
| 7 | 8 |
| 8 | 9 |
| 9 | 10 |

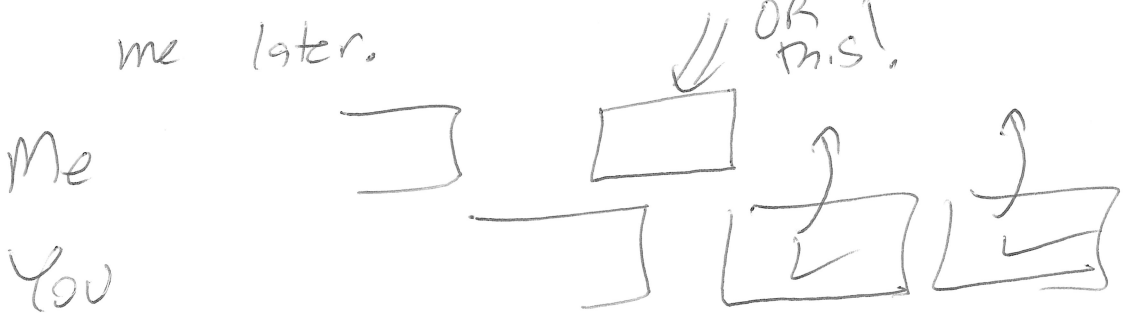
etc.

# SoA End Time

This works. Sort. go through the items and add to schedule if it's not causing a conflict

My 1st event ends earliest.
Let my schedule be S.
Assume a better schedule S' exists.

→ When my 1st event ends I am NOT behind.
So if they beat me, they have to "pass" me later.




any thing you do in the future, I can do also! Either I did you did + you didn't get ahead OR I scheduled other stuff that ended before you!

# My visual

This also works: Sort by start time and process the list backwards!

# 0-1 Knapsack

Thief knapsack capacity W lbs.

Several items

| W | V |
|---|---|
| 3 lbs | $6 |
| 5 lbs | $9 |
| 1 lb | $3 |
| 4 lbs | $5 |
| 10 lbs | $16 |

You either take or don't take

W = 17

Exercise - come up w/ 3 greedy ideas to solve
- create break cases for each!

NO KNOWN GREEDY SOLUTION EXISTS!

## Coin Changing

Consider a coin system with denominations $d_1 < d_2 < d_3 \ldots < d_k$ where $d_i \mid d_{i+1}$ for all $i$.

1, 5, 10, 20, 100, 500, 1000, $d_1 = 1$.

Given a positive int $n$, use the fewest # of coins to make change for $n$ cents

$$787¢ = 1 \times 500¢ + 2 \times 100¢ + 4 \times 20¢ + 1 \times 5¢ + 2 \times 1¢$$

10 coins.

Algorithm greedily give as many of the most expensive coins as possible + repeat!

A competing different set of change would substitute one of my bigger coins for multiple smaller coins, due to divisibility!

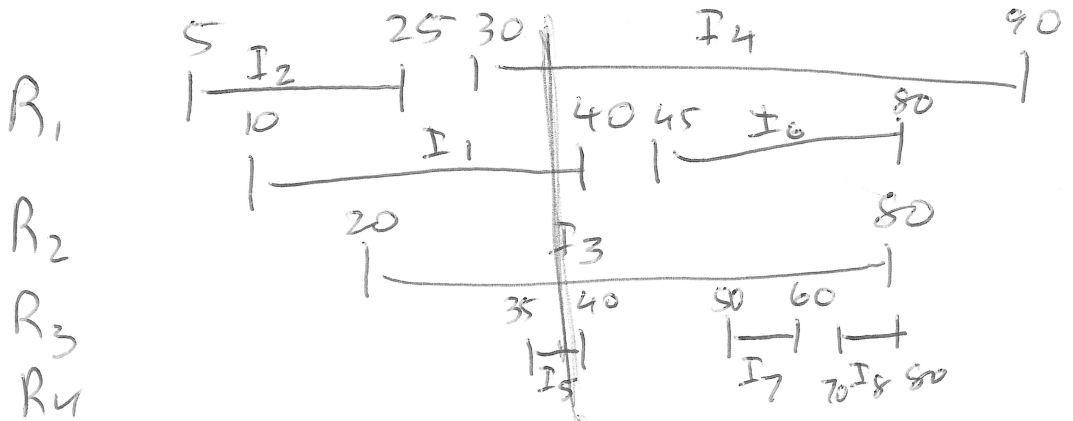
for US coins (1, 5, 10, 25) this works
proof => Brute force upto 50¢.
Does NOT work for arbitrary denominations!

---

## Multiple Room Scheduling

Same input as single room scheduling.
Must schedule ALL events
Goal: Minimize # of rooms used

Sort by start time.
When an event starts, put it in the lowest unused room #.




Min # Rooms = 4

4 rooms simultaneously in use!

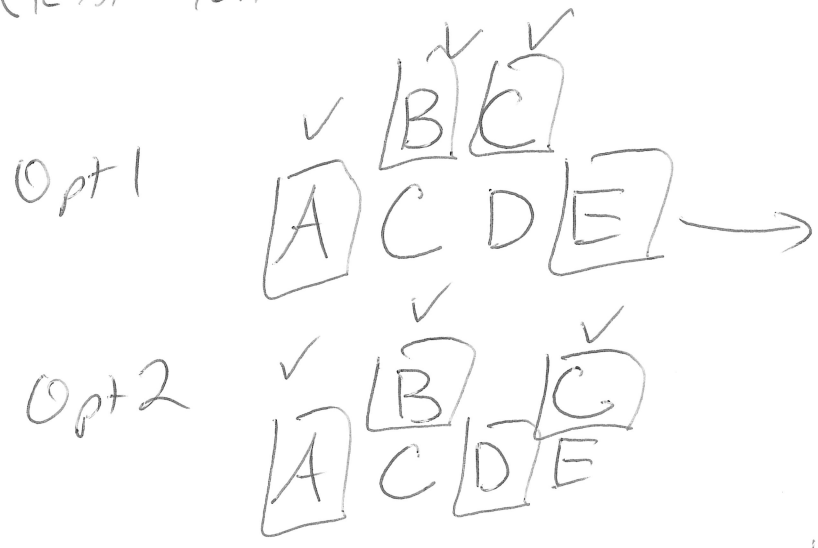
# Containers

Shipments: A, C, B, D, E, C, B, F, C, ...
↑

B C
B C
A C D E F

When each new crate comes stack it so its letter ≤ top OR create new stack
Goal: Minimize # stacks

for each crate put it on the leftmost one (least letter ≥ current letter).




C
this is better
E
vs
D

Every placement Opt 2 could do Opt 1 can do also!

# Last Alphabetic Subsequence

DCQRMABQMNOSRACBDCA

QMNABA is subsequence

Original sequence is $s_1, s_2, s_3 \ldots s_n$

A subsequence is $s_{a_1}, s_{a_2}, s_{a_3} \ldots s_{a_k}$

where $a_1 < a_2 < s_3 \ldots < a_k$.

Observation #1: Starting letter must be "last" one the appears alphabetically.

Simple Solution to run in $O(n)$ time,
$n$ = # letters

Maintain a Stack, every time we get a letter pop off each letter on stack that comes before the new letter.