COL1000: Introduction to Programming

Algorithmic Thinking (Merge & Merge Sort)

Subodh Sharma | Lec 33 | Nov 4

Announcement

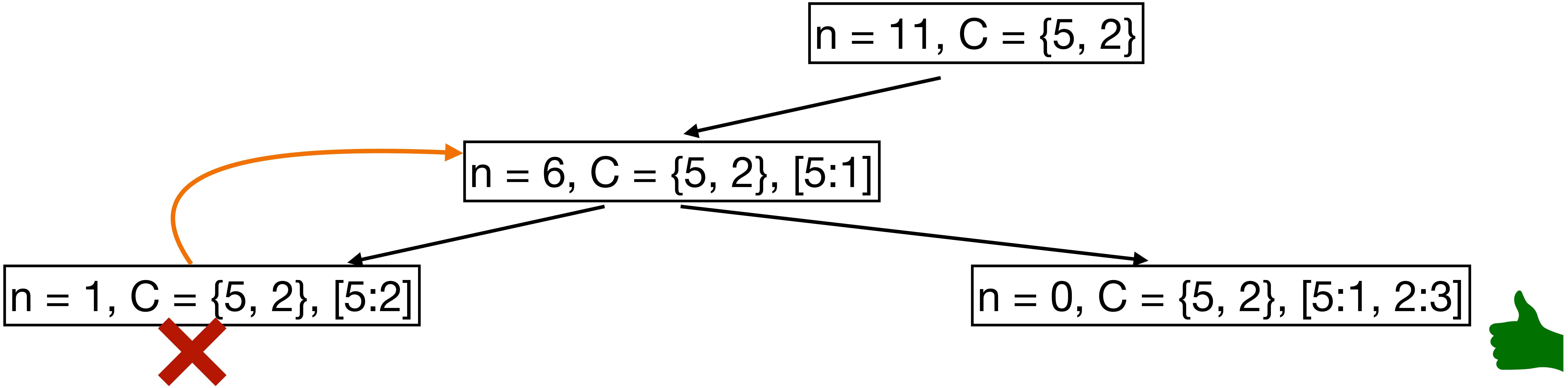
- (Usually) Monday's — 5 pm to 7 pm; Doubt learning sessions in Bharti 419
  - This time — 3 students visited me!

Greedy is Suboptimal

- **Suboptimal** — C= {1,3, 4} T = 6 —> 4, 1,1, —> 3 coins; 3 +3 —> 2 coins (optimal)
- Greedy approach is **optimal only for canonical coins**
  - Let $G(x)$ be the # of coins in greedy approach and $Opt(x)$ be the real optimal solutions to meet target x
    - Then a system is canonical $iff\ G(x) = Opt(x)$
    - **(Optional)** Look at **Kozen-Zaks theorem** to establish when Greedy can be optimal

Greedy is Suboptimal

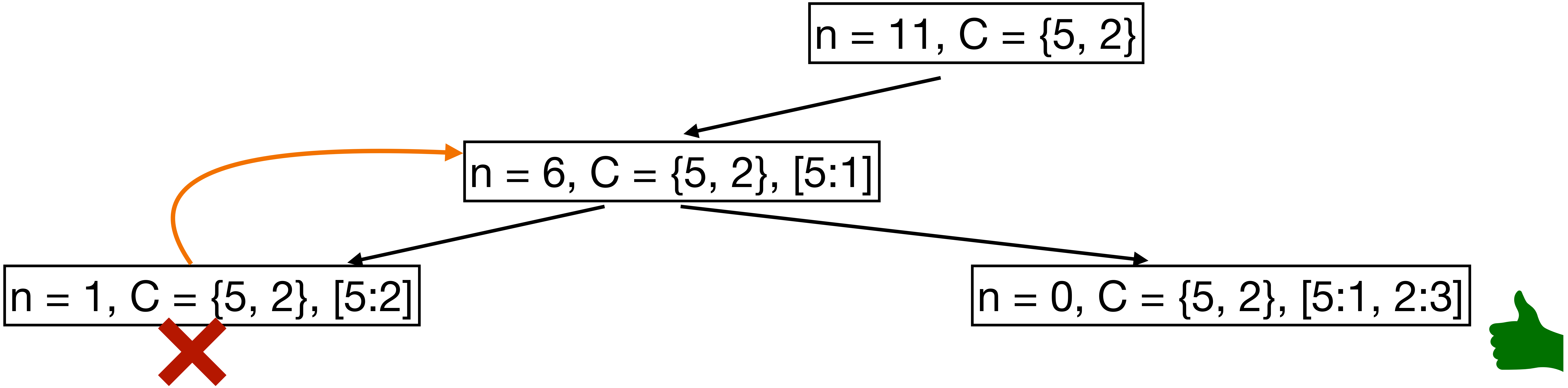
- Simple Greedy strategy is suboptimal — Why?
  - False assumption: that the largest coin always participate in an optimal solution
- Alternate strategy — backtrack + greedy



Greedy is Suboptimal

- Alternate strategy — backtrack + greedy (two variants - any solution, optimal)

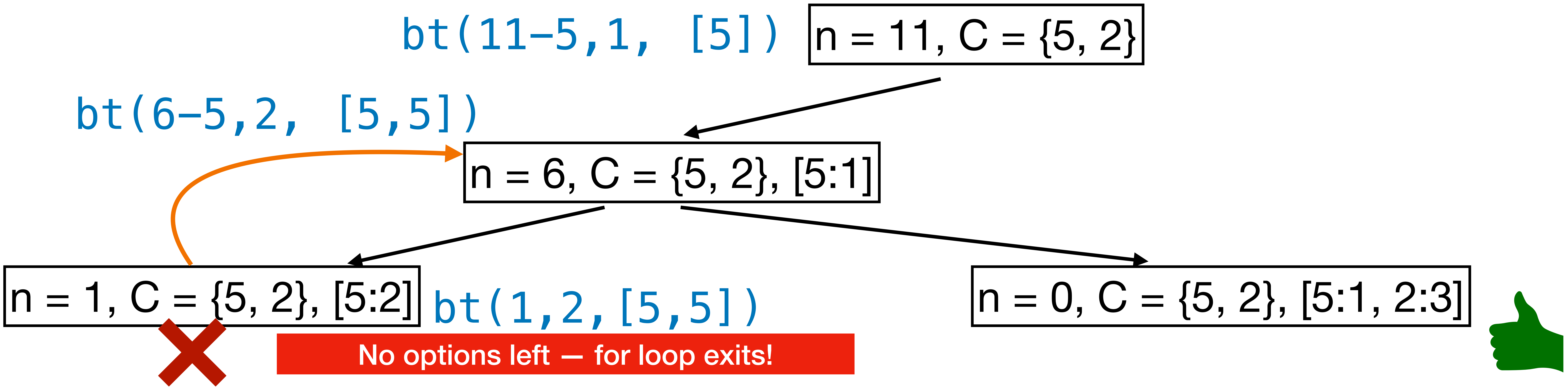
```python
for coin in denom:
    (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])
```



Greedy + Backtrack - Any Solution

- Alternate strategy — backtrack + greedy (two variants - any solution)

```
for coin in denom:
     (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])
```



Greedy + Backtrack - Any Solution

- **Alternate strategy — backtrack + greedy (two variants - any solution)**

```
for coin in denom:
    (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])

    #  But what if a solution is found — i.e. bt returns with a valid resCnt?

return (None, None)
```

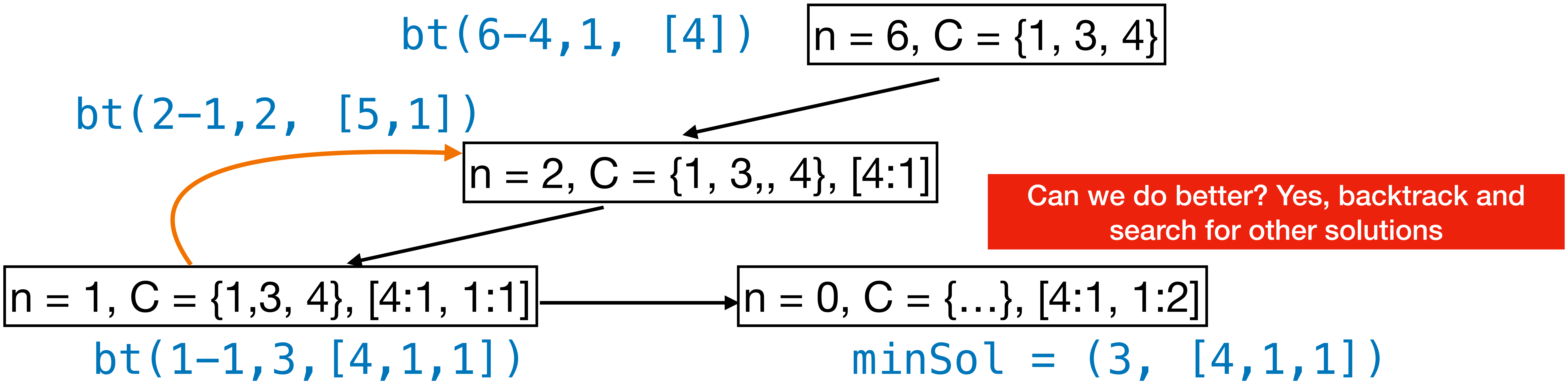
Greedy + Backtrack - Any Solution

```
def coin_change_bt_any(denom, target):

    denom = sorted(denom, reverse=True)

    def bt(remaining , count, coins_used):

         # base case 1
        if remaining == 0:
            return (count, coins_used)

        for coin in denom:

            if remaining < coin:
                continue

            (resCnt, resCoins) = bt (remaining - coin, count+1, coins_used + [coin])
            # If solution is found
            if resCnt is not None:
                return (resCnt, resCoins)
        # Can't progress from this state -- no solution in this path
        return (None, None)

    (resCnt, resCoins) = bt(target, 0, [])
    return (resCnt, resCoins) if resCnt is not None else (-1, [])
```

Greedy + Backtrack - Optimal Solution

- **Note that for optimality (with backtrack), you must maintain the minimum (or best) solution while looking for other solutions**

```
for coin in denom:
    (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])
```



Greedy + Backtrack - Optimal Solution

- Note that for optimality (with backtrack), you must maintain the minimum (or best) solution while looking for other solutions

```
for coin in denom:
     (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])
```

bt(6-4,1, [4])

n = 6, C = {1, 3, 4}

CurrMinSol = (3, [4,1,1])

bt(2-1,2, [5,1])

n = 2, C = {1, 3,, 4}, [4:1]

n = 1, C = {1,3, 4}, [4:1, 1:1]

bt(1-1,3,[4,1,1])

Greedy + Backtrack - Optimal Solution

- **Note that for optimality (with backtrack), you must maintain the minimum (or best) solution while looking for other solutions**

```
for coin in denom:
        (resCnt, resCoins) = bt(remaining-coin, count+1, coins_used+[coin])
```

bt(6-3,1, [3])

CurrMinSol = (3, [4,1,1])

n = 6, C = {1, 3, 4}

bt(3-3,2, [3,3])

n = 3, C = {1, 3, 4}, [3:1]

MinSol < CurrMinSol =>
CurrMinSol = MinSol

n = 0, C = {1, 3, 4}, [3:2]

MinSol = (2, [3,3])