188: Best time to Buy and Sell Stock

prices = [2, 4, 1] ⇒ integer array with stock prices day wise

k - transactions, ⇒ buy atmost k times and sell k times.

In above case for k=2, answer is 2.

(buy on 1st day and sell on 2nd day)

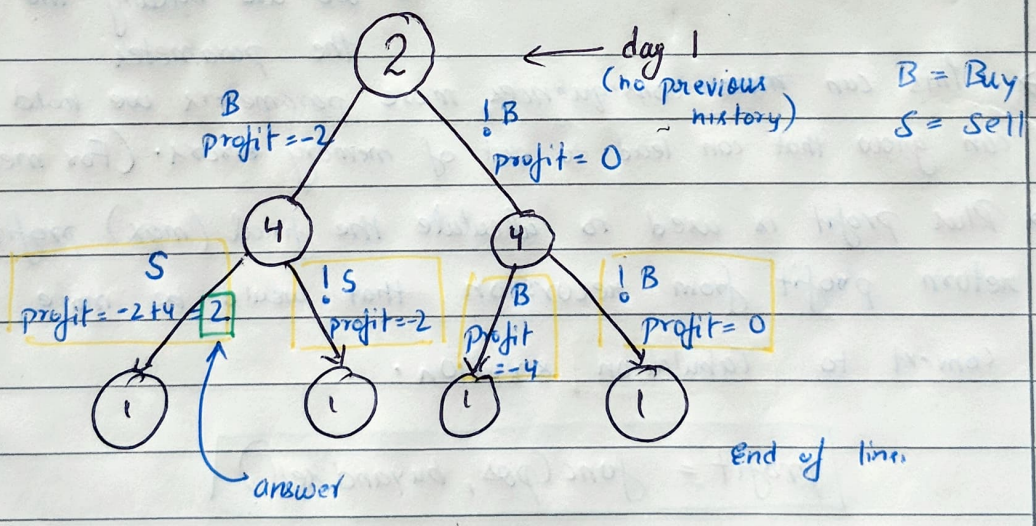
Note:- Cannot purchase multiple stocks on same day.

Logic

At any day:-

Buy / Not buy (If last transaction was sell)

Sell / Not sell (If last transaction was buy)



We can create a recursive solution from this approach.

Steps

1) Figure out how many parameters are required in the recursive solution.

Increase buy count

You buy or sell depends on last transaction

func (pos, buy, sell, last_trans, profit)

current position in prices array

How many times sold

profit upto current recursion.

2) See if the number of parameters can be reduced.

buy
sell
last_trans
} → All can be represented by only one variable.

buyandsell — When we buy we increment it by 1
When we sell we increment it by 1

Buy is only possible when last transaction was sell and Sell is only possible when last transaction was buy.

∴ when buyandsell %2 == 0 You Buy
else: You sell

3) func (pos, buyandsell, profit)

↑ we are holding the profit in the parameter.

→ This can have consequences, more parameters we hold, stack size can grow that can lead to out of memory errors. (For memorization)

→ Plus profit is used to calculate the final (max) profit, if we return profit from recursion, that would be more easy to convert to Tabulation solution.

profit = func(pos, buyandsell)

4) Base cases:-

if pos >= len(prices) or buyandsell >= 2*k : return 0

5) Recurrence relation:

Options are do nothing (move to next day) OR
Buy or sell and move to nex day.

F(pos, byandsell) = F(pos+1, buyandsell)
if buyandsell %2 == 0
F(pos, buyan

do_nothing = F(pos+1, buyandsell)

do_something = 0

if buyandsell % 2 == 0:

    do_something = F(pos+1, buyandsell+1) - prices[pos]

else:

    do_something = F(pos+1, buyandsell+1) + prices[pos]

~~return~~

F(pos, buyandsell) = max(do_nothing, do_something)