Today's agenda

- Dungeon Princess
- Buy and Sell Stock 1
- Buy and Sell Stock 2
- Buy and Sell Stock 3

- youtube.com → ~~1D~~ ~~DP~~ → CP
  - Problem exists

→ interview ~~DP~~ → ~~2D~~
  - ~~2D~~ DP

Q) Dungeon Princess

↳ Given mat[N][M] where each cell indicates health gained.

→ Find out min health required at (0,0) so that we can reach [N-1, M-1].

Note: ① movements : right or bottom.

② if health reaches 0, at any place you are dead.

③ we are starting at (0,0).

Ex:

	0	1
0	-3	-5
1	-2	1

6 → 4 5 6 7

// find min cost Path??

Ex:

	0	1	2	3
0	-3	2	4	-7
1	-6	5	-4	6
2	-15	-8	3	-4
3	7	4	-2	-7

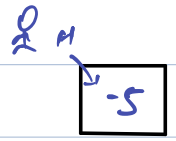
minH(0,0 → 3,3) → minH req to start at (0,0)

right → minH(0,1 - 3,3)

down → minH(1,0 - 3,3)

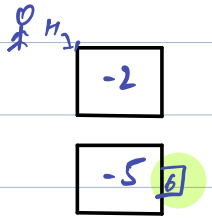
dp[i][j] ------> dp(i+1)(j) and dp(i)(j+1)

Case 1:



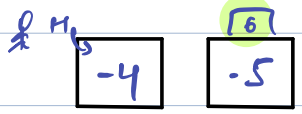
$H + (-5) = 1 \Rightarrow H = 1 + 5 = 6$

Case 2:



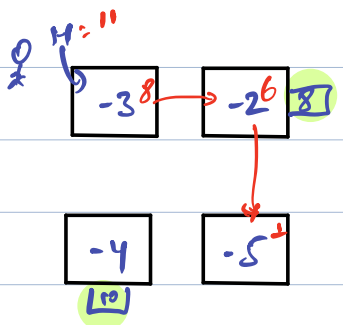
$H + (-2) = 6 \Rightarrow H = 6 + 2 = 8$

Case 3:



$H + (-4) = 6 \Rightarrow H = 6 + 4 = 10$

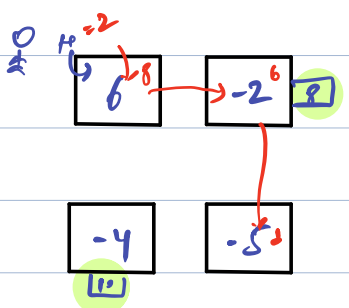
Case 4:



$H + (-3) = \min(8, 10)$

$H = 8 + 3 = 11$

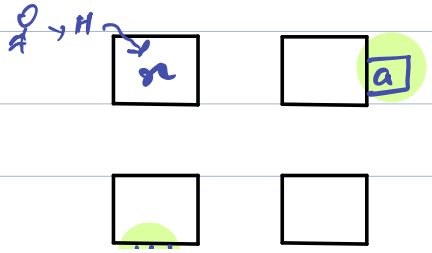
Case 5:



$H + (+6) = \min(8, 10)$

$H = 8 - 6 = 2$

Generic case:



$H + (x) = \min(a, b)$

$H = \min(a, b) - x$

(b)

$$minH(i,j) = \min\left(minH(i,j+1), minH(i+1,j)\right) - arr[i][j]$$

//Pseudo code

```
int dp[N][M] = {-1};

int minH (int mat[N][M], int i, int j) {
    if (i>=N || j>=M) {return INT_MAX;}
    if (i==N-1 && j==M-1) { return max(1, 1-mat[i][j]);}
    if (dp[i][j] != -1) {return dp[i][j];}

    int a = minH (mat, i+1, j);
    int b = minH (mat, i, j+1);

    dp[i][j] = max(1, min(a,b) - mat[i][j]);
    return max(1, min(a,b) - mat[i][j]);

}
```

mat

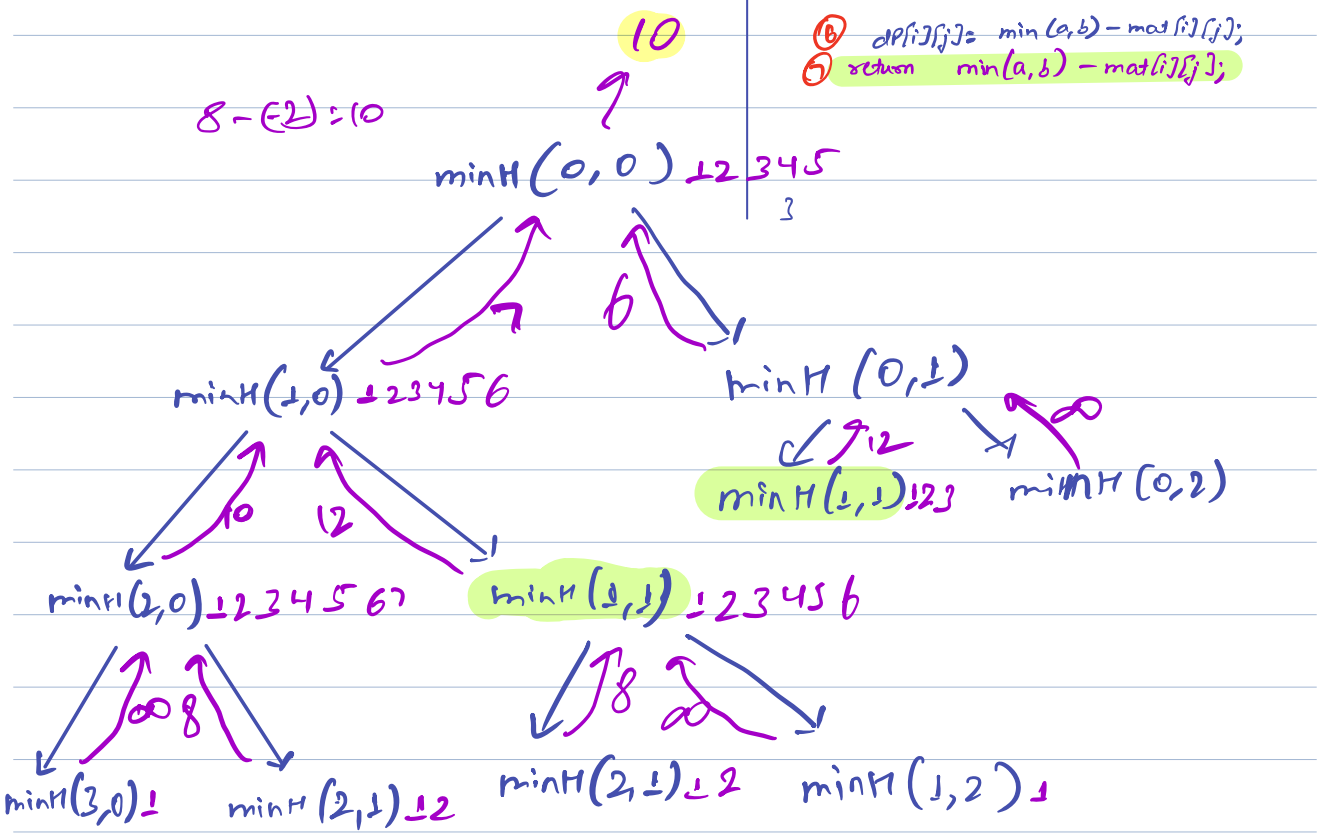
	0	1
0	-4	6
1	3	-4
2	-2	-7

dP

	0	1
0	~~-1~~ 10	~~-1~~ 6
1	~~-1~~ 7	~~-1~~ 12
2	~~-1~~ 10	-1

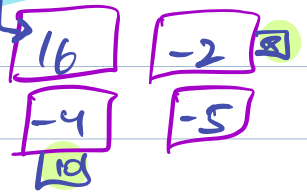
```
int minH (int mat[][], int i, int j) {
① if (i>=N || j>=M) {return INT_MAX;}
② if (i==N-1 && j==M-1) {return 1-mat[i][j];}
③ if (dp[i][j] != -1) {return dp[i][j];}

④ int a = minH (mat, i+1, j);
⑤ int b = minH (mat, i, j+1);

⑥ dp[i][j] = min(a,b) - mat[i][j];
⑦ return min(a,b) - mat[i][j];
}
```

8 - (-2) = 10



edge case

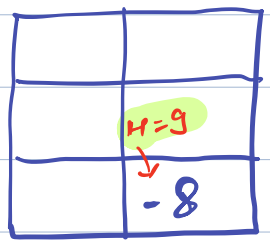
H = -8 = 1

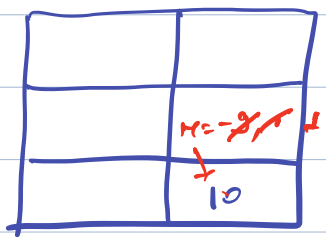


→ H + 16 = min(8, 10)

H = 8 − 16 = max(−8, 1) = 1

eg.





1 − (−8) = 9

1 − 10 = −9

```
if (i == n-1 && j == m-1) { return max(1, 1 - mat[i][j]); }
```

Break till 10:35 pm

* Buy and Sell Stocks I

↳ given an array `Prices` where `Prices[i]` is the price of a given stock on the i^{th} day. Return the maximum Profit by doing a single transaction. → Buy - sell

Ex: Prices[] = { 7 1 5 3 6 4 } — ans = 5

(0 1 2 3 4 5; B = 1, S = 6)

//Brute force

↳ consider all Pairs, Pick the maximum

Prices[] = { 7 1 5 3 6 4 } (0 1 2 3 4 5)

T.C: $O(N^2)$ S.C: $O(1)$

//Optimal

Prices[] = { 7 1 5 3 6 4 } (0 1 2 3 4 5; i = 3)

− (6 4 2 5 3)

↓ max = 5

↓ max = -ve ⇒ 0

$0 - (i-1)^{th}$ → Buy at min Price

i^{th} day → selling

//Pseudo code

```
int BuyandSell (int Prices[N]) {
    int ans=0;
    int minval=Prices[0];

    for (int i=1; i<N; i++) {
        int Profit = Prices[i] - minval;
        ans = max (ans, Profit);
        minval = min (minval, Prices[i]);
    }

    return ans;
}
```

T.C: O(N)

S.C: O(1)

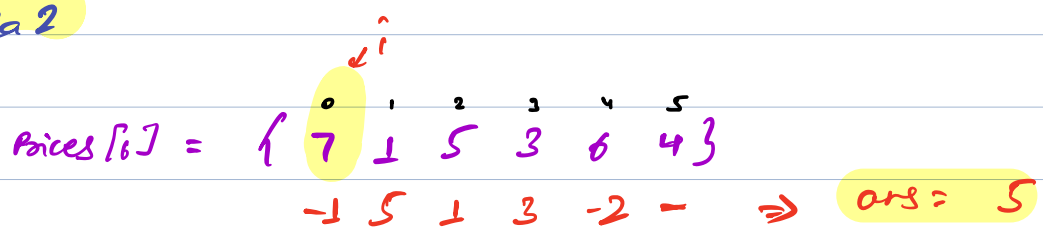
minval: ~~1~~ 1 ans: ~~0~~ 4

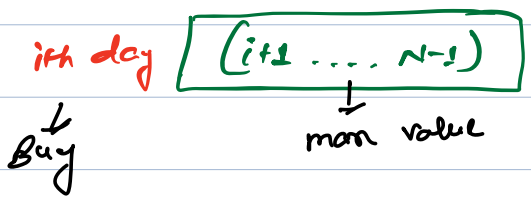
i

Prices[6] = { 7 1 5 3 6 4 }
(indices 0 1 2 3 4 5)

Profit: 5 - 1 = 4

// idea 2


i
Prices[i] = { 7 1 5 3 6 4 }
0 1 2 3 4 5
-1 5 1 3 -2 -
⇒ ans = 5


ith day
(i+1 .... N-1)
Buy
max value

Q) Buy and Sell Stocks 2

↳ Same as Previous Problem but we can do as many transactions as you want.

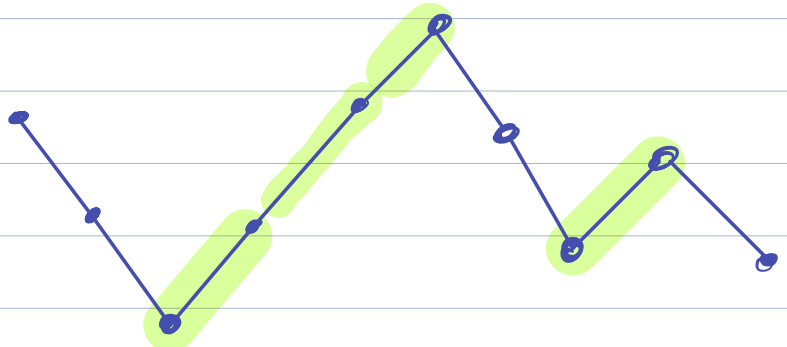
Note: you can't buy again until you sell the previous stock. → B-S-B-S-B-S

Buy the dip

Ex: Prices[i] = { 7 1 5 3 6 4 } → ans: 7

(0 1 2 3 4 5)

(1 → 5 : 4, 3 → 6 : 3)



// Pseudo code

```
int ans = 0;
for (int i = 1; i < N; i++) {
    if (arr[i] > arr[i-1]) {
        ans = ans + (arr[i] - arr[i-1]);
    }
}
return ans;
```

T.C: O(N)

S.C: O(1)

Q) Buy and Sell Stocks 3

↳ Same Problem statement, atmost 2 transactions allowed. → B-S-B-S

Ex: Prices[] = { 7 1 5 3 6 4 } → ans = 7

(0 1 2 3 4 5)

(1 → 5 : 4) (3 → 6 : 3)

//Optimal idea

Prices[] = { 7 1 5 3 6 4 }

(0 1 2 3 4 5)

0 → 5 : 5

4 → 7 : 3

0 → 3 : 3

Prices[] = { 7 1 5 | 3 6 4 }

(0 1 2 3 4 5)

leftProfit = 0 0 4 4 5 5

rightProfit = 5 5 3 3 0 0

ith split

int Profit = (leftProfit[i] + rightProfit[i+1])

⇓

ans = max(ans, Profit);

If you didn't come this far only to come this far.