" DP on stocks "

DP-35 Best time to buy and sell stock.

Problem Best time to buy and sell stock ,,

arr() → [7, 1, 5, 3, 6, 4] Maximize the profit

Brute force → try every possible sell and buy way and find the maximum profit but time → $O(N^2)$

idea 7 1 5 3 6 4.

" if you are selling it on the ith day . you must buy it on the minimum price from (1st - (i-1)th) day "

```
int maxprofit ( vector<int> nums)
{
    int mini = nums[0];
    int ans = 0;
    int n = nums.size();
    for( int i=1; i<n; i++){
        int cost = num[i] - mini;
        ans = max( ans, cost);
        mini = min( mini, nums[i]);
    }
    return ans;
}
```

Time complexity → O(N)

DP→36 Best time to buy and sell stocks II

Same problem (just add condition : can purchase and sell any number of time

my solution

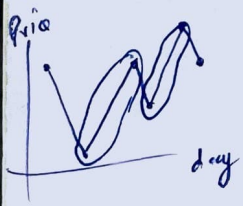
```
int maxprofit ( vector<int> & nums) {
    int n = nums.size();
    int maxi = 0;
    for (int i = 0; i < n-1; i++){
        if ( num[i+1] > num[i]){
            maxi += (nums[i+1] - num[i]);
        }
        else { continue; }
    }
    return maxi;
}
```

O(n) O(1)

logic

we must consider only that part which give us benefit

example (7 1 5 3 6 4)



official example 7 1 5 3 6 4 can buy and sell at any date

so we have (a lot of ways) → (Try all ways) → (Best answer)

Recursion

How to write recursion?

(i) Express in term of (index, buy) — variable that tell us either i can buy or not.

like 7 ① 5 3/f 6 4 at 3'y

[already purchased a stock not sold yet then; can't buy another stock]

(ii) Explore possibilities on that day

(iii) Take the max profit made

(iv) Base case.

(i) f(ind, buy)

f(0,1) → [starting on 0th day with buy 1 can give me maximum profit]

[buy → 1 → can buy
→ 0 → can't buy]

(ii) if(buy){ → take / not take
profit1 =
}
else { profit2 = ..}

```
if (buy) {
    profit = max { -prices(ind) + f(ind+1, 0)   → take (purchased)
                   0            + f(ind+1, 1) } → not take (not purchase)
else {
    profit = max { prices(ind) + f(ind+1, 1);   → sold
                   0           + f(ind+1, 0); } → don't sold
}
```

(iii) Base case → when array is finished no more profit

if(ind == n) return 0;

Memorization and Recursion code

```
int fun(int ind, int buy, vector<int>& nums, int n, vector<vector<int>>& dp)
{
    if(ind == n) return 0;
    if(dp[ind][buy] != -1) return dp[ind][buy];
    int profit = 0;
    if(buy) {
        profit = max( -nums[ind] + fun(ind+1, 0, nums, n, dp)
                      0 + fun(ind+1, 1, nums, n, dp) );
    else {
        Profit = max( nums[ind] + f(ind+1, 1, nums, n, dp)
                      0 + fun(ind+1, 0, nums, n, dp)
    }
    return dp[ind][buy] = profit;
}
```

Tabulation

steps to convert memorization to tabulation

(i) convert base case if(ind == n) return 0, dp[n][0] = dp[n][1] = 0

(ii) write loop condition

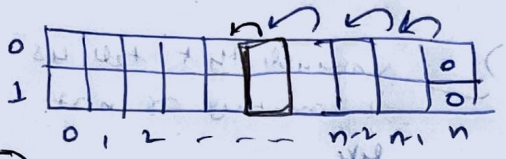


Fig - 1

(i) we are traversing from n to 0 and dp[0][1] would be our final ans

```
for(int i = n-1, i >= 0; i--)
    for(int j = 0; j < 2; j++)
```

(iii) copy memo, recursion condition

(iv) convert fun(---) into dp to complete code

Code

```
int maxprofit(vector<int>& nums){
    int n = nums.size();
    vector<vector<int>> dp(n+1, vector<int>(2,0));
    dp[n][0] = dp[n][1] = 0;
    for(int i = n-1; i>=0; i--){
        for(int j=0; j<2; j++){
            int profit = 0;
            if(j){
                profit = max(-nums[i] + dp[i+1][0], dp[i+1][1]);
            } else {
                profit = max(nums[i] + dp[i+1][1], dp[i+1][0]);
            }
            dp[i][j] = profit;
        }
    }
    return dp[0][1];
}
```

O(n×2) → Time
O(n×2) → space

optimization

- we just need only previous (ind+1) data to calculate next step so we can space optimize it

- also we can do this optimization using first approach to make it solve in O(n)

DP-37 Best time to Buy and sell stocks III

Problem statement Prices = [3, 3, 5, 0, 0, 3, 1, 4] → same as last problem

We have to find maximum profit but with a extra condition

* We can do maximum 2 transitions

Solution → Try all possible ways + Pick the best one + Recursion

How to write recursion

① Express function in term of variables, like ind, buy

But Here we also get 1 extra condition to keep track of no of transition. So

* let add another variable (cap) → start with 2 and with each sold we decrement value of cap.
* when value of cap == 0, no more transition can done

fun (ind, buy, cap) {

② Explore all possibilities on that day

③ Take the maximum profit made

④ write base case

1st ⟹

fun (ind, buy, cap) {

ind → traverse the days in Buy and Sell stock

buy → can tell that we can purchase or not

cap → if (2 → we can do Sell 2 times), (1 → can do sell 1 time), 0 → no more buy or sell

```
if ( buy) {
    profit = max( -num[ind] + fun( ind+1, 0, cap),
                   0 + fun( ind+1, 1, cap) );
}
else
    profit = max( +num[ind] + fun( ind+1, 1, cap),
```

Memorization and Recursive code

```
int fun ( int mid, int buy, int cap, int n, vector<int> nums,
          vector<vector<vector<int>>> &dp) {
    if ( mid == n) return 0;
    if ( cap == 0) return 0;
    if( dp[mid][buy][cap] != -1) return dp[mid][buy][cap];
    if (buy) {
        profit = max (-nums[mid] + fun( mid+1, 0, cap, n, nums, dp),
                                   fun(mid+1, 1, cap, n, nums, dp)
    }
    else
    {
        profit = max( nums[mid] + fun(mid+1, 1, cap-1, n, nums, dp),
                                   fun ( mid+1, 0, cap, n, nums, dp)
    }
    return dp[mid][buy][cap] = profit;
}
int maxprofit ( vector<int> & nums, int n) {
    vector< vector < vector<int>>> dp( n+1, vector<vector<int>>( 2,
                    vector<int>( 3,-1)));
    return fun( 0, 1, 2, n, nums, dp);
}
```

Time = O(N×2×3)

Space = O(N×2×3) + O(N)

Tabulation steps to convert memorization into tabulation

(i) change Base case , here we can neglect because they already have

(ii) write loop condition same value '0'.

In memorization		In tabulation
(i) we are traversing from (0 → n)	+	we will traverse (n-1 → 0)
(ii) Buy we took 1	+	Buy 0 → 1
(iii) Cap we took 2	+	cap 0 → 2

But (cap == 0) already covered so we take it from 1.

(iii) ans returned as

fun(0, 1, 2, n, nums, dp) → return dp[0][1][2];

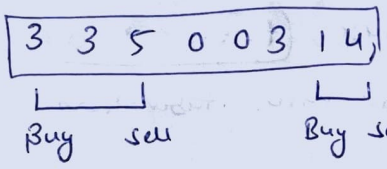
Tabulation code

```
int maxProfit ( vector<int>& nums, int n){

  Vector< vector < vector<int>>> dp( n+1, vector< vector <int >>(2,
          Vector<int> ( 3, 0)));

    for ( int ind = n-1; ind >=0; ind --) {
      for ( int buy = 0 : buy<2 : buy ++) {
        for ( int cap = 0 : cap<3 ; cap ++) {

      if ( buy) {
      dp(ind)(buy)(cap) =   max ( -nums[ind] + dp(ind+1)[0][cap],
      }                                dp(ind+1)[1](cap) ) ;
      else {
      dp(ind)(buy)(cap) = max (  nums[ind] + dp(ind+1)[1][cap-1],
      ]                                dp(ind+1)[0][cap]);
    )}
  )
      return dp[0][1][2];
)
```

Another approach Instead of using dp[N][2][3] (ind, buy, cap)

we can use dp[N][4] (ind, transactions)

idea :-



1. we know the maximum limit is 2.
2. so we can say that we can have maximum 2 Buy 2 Sell
3. with each buy and sell we will increment value of 'trans' and once it reach 4 no more transaction will possible

So recursion function

```
fun( ind, trans) {

  if( ind == n || trans == 4)
    return 0;

  if ( trans%2 == 0) → buy
  max { - prices(ind) + fun( ind+1, trans+1) ,
        0 + fun ( ind+1 , trans) }

  else
  max { + prices ( ind) + fun (ind+1, trans+1)
        0 + fun( ind+1, trans ) }
```

Memorization + Recursion approach :

```
int fun ( int ind, int t, int n, vector<int>& nums,
                    vector<vector<int>>& dp ){
    if( ind == n || t == 4) return 0;
    if( dp[ind][t] != -1) return dp[ind][t];
    int profit = 0;
    if( t%2 == 0){
        profit = max ( -nums[ind] + fun ( ind+1, t+1, n, nums, dp),
                        0 + fun( ind+1, t, n, nums, dp) );
    else{
        profit = max( nums[ind] + fun ( ind+1, t+1, n, nums, dp),
                        0 + fun( ind+1, t, n, nums, dp) );
    }
    return dp[ind][t] = profit;
}

int maxprofit ( vector<int>& nums){
    int n = nums.size();
    vector<vector<int>> dp( n+1, vector<int> (4,-1));
    return fun( 0, 0, n, nums, dp);
}
```

Tabulation

How to convert memorization into tabulation

① step 1 → declare required size dp. → Here we will need n×5 dp

[Because we have to store base t==4 → 0]

② convert Base cases of nums into tabulation

```
for(int i=0; i<5; i++){
    dp[n][i] = 0;
}
```

```
for( int i=0; i<n; i++)
    dp[i][4] = 0;
}
```

③ convert recursion into (loops) → Just opposite traversing of memorization on Memo

ind → (0 → n) ⟶ (n-1 → 0)] on tabulation

t → (0 → 4) ⟶ (3 → 0)]

④ copy paste Main part of memo

```
for( int i = n-1; i>=0; i--){
    for (int j = 3; j>=0; j--){
    }
}
```

Code

```
int maxprofit ( vector<int> & nums, int n){
vector <vector <int>>dp( n+1, vector<int> (5,0));
for(int i = 0 ; i<5 ; i++){
      dp[n][i] = 0;
}
for( int j=0; j<n ; j++){
      dp[j][4] = 0 ;
}
for ( int i = n-1; i>=0; i--){
    for ( int j=3 ; j>=0 ; j--){
    if ( j%2 ==0){
      dp[i][j] = max ( -nums[i] + dp[i+1][j+1] , dp[i+1][j]);
    else {
        dp[i][j] = max ( nums[i] + dp[i+1][j+1], dp[i+1][j]);
      }
    }
  }
  return dp[0][0];
}
```

time = O(N×5)
space = O(N×5)

DP → 38 Best time to Buy and sell stock 4

Problem statement

- We have given an nums = [3, 2, 6, 5, 0, 3], and k = 2
- And we have to calculate maximum profit and we can do maximum k transactions.

idea / logic →

- In the previous ques we can do maximum tranachoins upto 2 but now it will be k so only 1 change is required to get correct answer

when k = 2
we use dp[ind][buy][3]
n ↙ 2 ↙ 2 for k, 1 for basecase

dp[n+1][2][k+1]
for ind ↙ buy ↙ no of transachoins

(Repeat same algo like last time)

Memorization + Recursion

```
int fun ( int mid, int buy, int k, int n, vector<int>& nums,
          vector<vector<vector<int>>>& dp) {
    if ( mid == n || k==0) return 0;
    if ( dp[mid][buy][k] != -1) return dp[mid][buy][k];
    int profit =0;
    if( buy) {
        profit = max( -nums[mid] + fun(mid+1, 0, k, n, nums, dp)
                      0 + fun(mid+1, 1, k, n, nums, dp) );
    else {
        profit = max( nums[mid] + fun( mid+1, 1, k-1, n, nums, dp)
                      0 + fun( mid+1, 0, k, n, num, dp) );
    }
    return dp[mid][buy][k] = profit;
}
int maxprofit( int k, int n, vector<int>& nums){
    vector<vector<vector<int>>> dp( n+1, vector<vector<int>>( 2,
                                    vector<int>( k+1, -1)));
    return fun( 0, 1, k, n, nums, dp);
}
```

↓ convert memorization to tabulation.

① declare required size vector → yes / no

dp[N+1][2][K+1] → (0-k) for storing kth for base case

(0-n-1) for storing n+1 for base base

② convert base case into tabulation

(i) (mid==n) = 0

```
for(int i=0; i<2; i++){
    for( int j= 0; j<=k; j++){
        dp[n][i][j]=0;
    }
}
```

if(k==0) return 0;

```
for (int i=0; i<n; i++) {
    for (int j= 0; j<2; j++) {
        dp[i][j][0] =0;
    }
}
```

We can avoid writing this code just by declaring array with initialize value '0'. [∵ we are updating only 0]

(iii) convert recursion to loops

Memo		Tabulation	
ind → 0 → n	→	(n-1 → 0)	1st loop
buy → 1 → 0	→	(0 → 1)	2nd loop
trans → k → 0		(0 → k)	3rd loop

Code

```
int fun( int k, int n, vector<int> &nums) {
    vector<vector<vector<int>>> dp( n+1, vector<vector<int>> (2,
                                    vector<int> (k+1, 0)));
    // no need to write base case
    for ( int i = n-1 ; i>=0; i--){
        for (int j = 0; j<=1; j++){
            for (int m= 1; m <= k ; m++){
    if (j) {
        dp[i][j][m] = max (-nums[i] + dp[i+1][0][m], dp[i+1][1][m]);
    }
    else {
        dp[i][j][m] = max( nums[i]+ dp[i+1][1][m-1], dp[i+1][0][m]);
    }
} } }
    return dp[0][1][k];
```

→ Just write the 3 variable input you gave in memo fun(...)

Another approach

→ like previous method we can use dp(n+1)(2xk) → vector instead of dp(N+1)(2)(k) or

→ watch striver video for DP solution

DP→39 Best time to buy and sell stock with cooldown.

Problem statement →
- Given price of stock for each day [1, 2, 3, 0, 2]
- can do any no of transactions
- But after every sold we can't buy on just next day

idea/logic - We know the recursion part without cool down condition

```
fun( ind, buy){
    if (buy) {
        profit = max ( -nums[ind] + fun (ind+1, 0)
                       0 + fun (ind+1, 1) )
    }
    else
        profit = max( +nums[ind] + fun ( ind+1, 1)
                      0 +  fun (ind+1, 0)
```

this is sold part + add cooldown down condition

when we sold we can buy next to next day

num[ind] + fun(ind+2, 1)

Memorization + Recursion :-

```
int fun ( int mid, int buy , int n, vector<int> & nums,
          vector < vector <int>> & dp) {
    if ( mid >= n ) return 0;
    if ( dp[mid][buy] != -1) return dp[mid][buy];
    if ( buy) {
        dp[mid][buy] = max ( -nums[mid] + fun( mid+1, 0, n, nums, dp)
                             0 + fun( mid+1, 1, n, nums, dp)
    else {
        dp [mid][buy] = max [ +nums[mid] + fun( mid+2, 1, n, nums, dp),
                              0 + fun( mid+1, 0, n, nums, dp) ];
    return dp[mid][buy];
}
int maxprofit ( int n, vector<int> & nums){
    vector <vector <int>> dp( n+1, vector < int > (2,-1));
    return fun( 0, 1, n, nums, dp);
}
```

O(N) → time
O(N×2) → space + O(N)

* Convert memorization to tabulation

① convert base case (but no need here due to '0' as the values we have
② copy paste recursion to update)
③ convert recursion into loops in opposite fashion
in memo we go (0 → n) so in tabulation (n-1 → 0)

Code :-

```
int maxprofit ( int n, vector<int> & nums){
    vector < vector < int >> dp (n+2, vector<int> (2,0));
    for(int i = n-1; i>=0 ; i--){
        for (int j=0; j<2 ; j++){
    if ( j){
            dp(i)(j) = max( -nums(i) + dp(i+1)[0], dp(i+1)(1) );
    else {
            dp(i)(j) = max( nums(i) + dp(i+2)(1) , dp(i+1)[0] );
    } }
    return dp[0][1];
```

O(N) → time
O(N×2) → space

just copy returning format memo
dp(0)(1) ≈ fun(0, 1, n, nums, dp);

* here space optimization is not very good option just because we have to keep track of 2 dp[i+1], and dp[i+2]

DP-> 40 **Buy and sell stocks with transaction fees.**

Problem statement

Given → • a price list of per day [1, 3, 2, 8, 4, 9] and fees ⇒ k

Find max profit ? • Buy and sell can be done unlimited time, but each time we sold a transaction fee is charged.

idea | logic

> We know that the only difference is transaction fee so every time we sold we put extra fee on profit

Memorization + Recursion code

```
int fun (int ind, int buy, int k, int n, vector<int> & nums,
         vector<vector<int>> & dp ) {
    if ( ind == n) return 0;
    if (dp[ind][buy] != -1) return dp[ind][buy];
    int profit = 0;
    if (buy) {
    profit = max ( -nums[ind] + fun(ind+1, 0, k, n, nums, dp)
                   0 + fun( ind+1, 1, k, n, nums, dp) );
    else {
    profit = max ( (+nums[i] + fun (ind+1, 1, k, n, num, dp) - k)
                   0 + fun( ind+1, 0, k, n, nums, dp)
    )
    return dp[ind][buy] = profit )
}
    int maxprofit ( int k, vector<int> & nums) {
        int n = nums.size())
    vector< vector< int >> dp( n+1, vector <int> (2, -1));
        return fun( 0, 1, k, n, nums, dp);
    }
```

* can easily convert to tabulation
→ can refer to previous notes.

Time → O(N X 2)
Space → O(N X 2) + O(N)