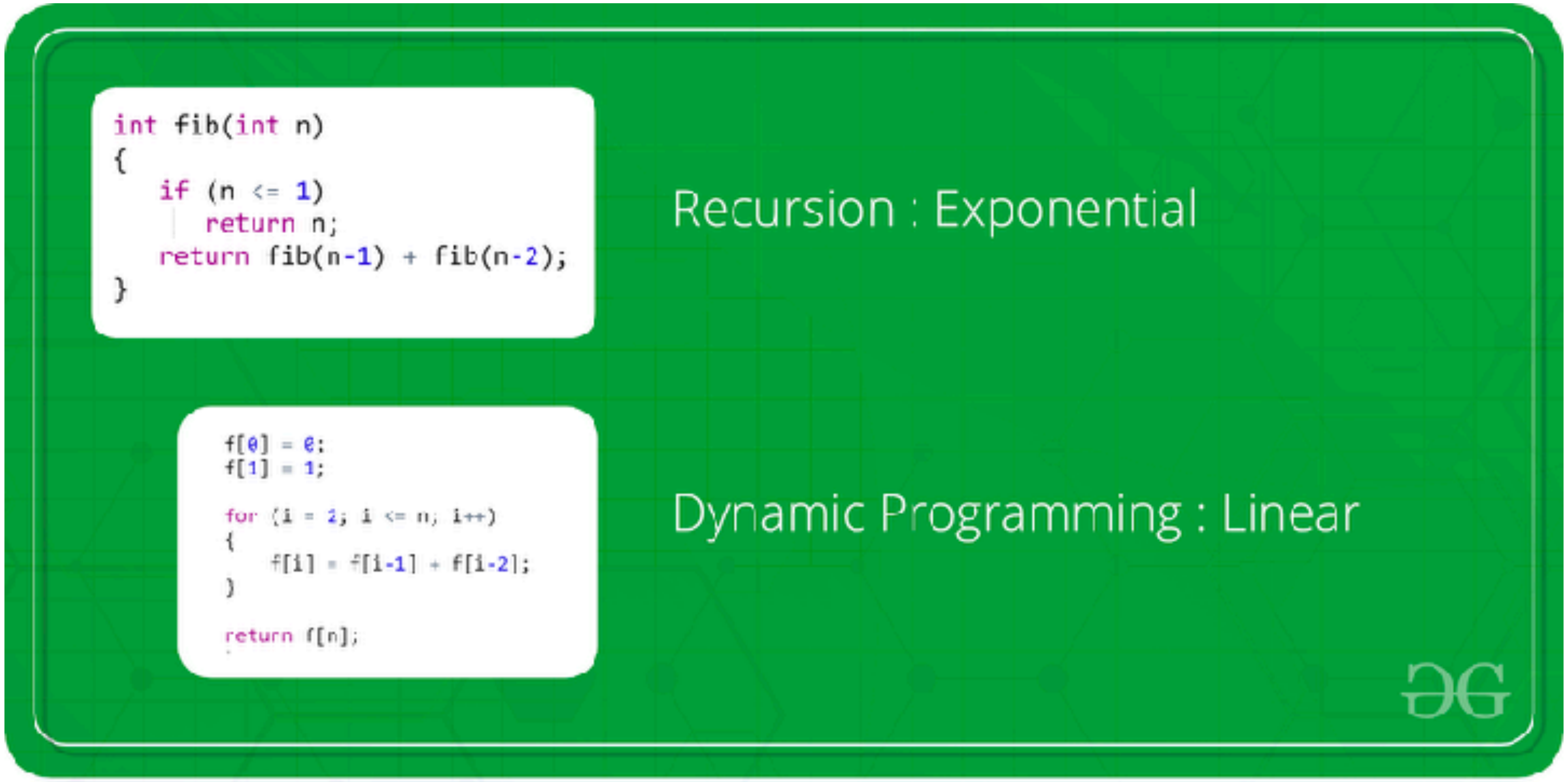
Topic 6: Dynamic Programing

Definition

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. (GeeksforGeeks)



```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Recursion : Exponential

```
f[0] = 0;
f[1] = 1;

for (i = 2; i <= n; i++)
{
    f[i] = f[i-1] + f[i-2];
}

return f[n];
```

Dynamic Programming : Linear

Methodology

- **How many different states we have?**
- **How many different independent variables we have?**
- **What is the recurrence formula?**

122. Best Time to Buy and Sell Stock II

Say you have an array for which the *i-th* element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell
on day 3 (price = 5), profit = 5-1 = 4.
             Then buy on day 4 (price = 3) and
sell on day 5 (price = 6), profit = 6-3 = 3.
```

Example 2:

```
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell
on day 5 (price = 5), profit = 5-1 = 4.
             Note that you cannot buy on day
1, buy on day 2 and sell them later, as you
are
             engaging multiple transactions at
the same time. You must sell before buying
again.
```

Method1

```
class Solution:
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if len(prices)<=1:
            return 0
        max_profit=0
        for i in range(1, len(prices)):
            diff = prices[i]-prices[i-1]
            if diff>=0:
                max_profit+=diff
        return max_profit
```

Method2

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        buy = [float('-inf')]+[0 for i in prices]
        sell = [0]+[0 for i in prices]

        for i in range(len(prices)):
            price = prices[i]
            buy[i+1] = max(buy[i], sell[i]-prices[i])
            sell[i+1] = max(buy[i]+prices[i], sell[i])
        return sell[-1]
```

121. Best Time to Buy and Sell Stock

Say you have an array for which the *i-th* element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

Example 1:

```
Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell
on day 5 (price = 6), profit = 6-1 = 5.
             Not 7-1 = 6, as selling price
needs to be larger than buying price.
```

Example 2:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is
done, i.e. max profit = 0.
```

Method1

```
class Solution:
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if len(prices)<=1:
            return 0
        low = prices[0]
        max_profit = prices[1]-prices[0]
        for i in range(len(prices)):
            if low>prices[i]:
                low = prices[i]
            if prices[i]-low>max_profit:
                max_profit = prices[i]-low
        return max_profit
```

Method2

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        buy = -float('inf')
        sell = 0

        for price in prices:
            buy, sell = max(buy, -price), max(sell, buy+price)
        return sell
```

123. Best Time to Buy and Sell Stock III

Say you have an array for which the *i-th* element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete at most *two* transactions.

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell
on day 6 (price = 3), profit = 3-0 = 3.
             Then buy on day 7 (price = 1) and
sell on day 8 (price = 4), profit = 4-1 = 3.
```

Example 2:

```
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell
on day 5 (price = 5), profit = 5-1 = 4.
             Note that you cannot buy on day
1, buy on day 2 and sell them later, as you
are
             engaging multiple transactions at
the same time. You must sell before buying
again.
```

Example 3:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is
done, i.e. max profit = 0.
```

- **Four states: first buy, first sell, second buy, second sell**
- **One independent variable: day**
- **Recurrence formula**

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if len(prices)==0:
            return 0
        profit1 = float('-inf')
        profit2 = 0
        profit3 = float('-inf')
        profit4 = 0
        for x in prices:
            profit1 = max(profit1, -x)
            profit2 = max(profit2, profit1+x)
            profit3 = max(profit3, profit2-x)
            profit4 = max(profit4, profit3+x)
        return profit4
```

188. Best Time to Buy and Sell Stock IV

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete at most **k** transactions.

Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example 1:

```
Input: [2,4,1], k = 2
Output: 2
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.
```

Example 2:

```
Input: [3,2,6,5,0,3], k = 2
Output: 7
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4.
             Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
```

```
class Solution(object):
    def maxProfit(self, k, prices):
        """
        :type k: int
        :type prices: List[int]
        :rtype: int
        """
        if len(prices)<=1:
            return 0
        if k==0:
            return 0

        if k*2>=len(prices):
            buy = float('-inf')          MemoryError
            sell = 0
            for price in prices:
                buy, sell = max(sell-price, buy), max(sell, buy+price)
            return sell

        buy = [float('-inf') for i in range(k)]
        sell = [0 for i in range(k)]

        for price in prices:
            for j in range(len(buy)):
                if j==0:
                    buy[j] = max(buy[j], -price)
                else:
                    buy[j] = max(buy[j], sell[j-1]-price)
                sell[j] = max(sell[j], buy[j]+price)
        return sell[-1]
```

309. Best Time to Buy and Sell Stock with Cooldown

Say you have an array for which the *i-th* element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

```
Input: [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell,
cooldown, buy, sell]
```

- **Three states: buy, sell, cooldown**
- **One independent variable: day**
- **Recurrence formula**

buy->cooldown, buy

sell->buy, sell

cooldown->sell

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        buy = float('-inf')
        sell = 0
        cooldown = 0

        for price in prices:
            buy, sell, cooldown = max(sell-price, buy), max(cooldown, sell), buy+price
        return max(sell, cooldown)
```

714. Best Time to Buy and Sell Stock with Transaction Fee

Your are given an array of integers `prices`, for which the `i`-th element is the price of a given stock on day `i`; and a non-negative integer `fee` representing a transaction fee.

You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction. You may not buy more than 1 share of a stock at a time (ie. you must sell the stock share before you buy again.)

Return the maximum profit you can make.

Example 1:

```
Input: prices = [1, 3, 2, 8, 4, 9], fee = 2
Output: 8
Explanation: The maximum profit can be
achieved by:
Buying at prices[0] = 1
Selling at prices[3] = 8
Buying at prices[4] = 4
Selling at prices[5] = 9
The total profit is ((8 - 1) - 2) + ((9 - 4) -
2) = 8.
```

Note:

`0 < prices.length <= 50000`.
`0 < prices[i] < 50000`.
`0 <= fee < 50000`.

Where should we deduct the fee?

```
class Solution(object):
    def maxProfit(self, prices, fee):
        """
        :type prices: List[int]
        :type fee: int
        :rtype: int
        """
        buy = float('-inf')
        sell = 0

        for price in prices:
            buy, sell = max(buy, sell-price), max(sell, buy+price-fee)
        return sell
```

322. Coin Change

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.

Example 1:

```
Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Note:
You may assume that you have an infinite number of each kind of coin.

```
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        res = [-1 for i in range(amount+1)]
        res[0] = 0
        for target in range(1, len(res)):
            for coin in coins:
                if target-coin<0:
                    continue
                if res[target-coin] > -1:
                    if res[target]==-1:
                        res[target] = res[target-coin]+1
                    else:
                        res[target] = min(res[target], res[target-coin]+1)
        return res[-1]
```

718. Maximum Length of Repeated Subarray

Given two integer arrays `A` and `B`, return the maximum length of an subarray that appears in both arrays.

Example 1:

```
Input:
A: [1,2,3,2,1]
B: [3,2,1,4,7]
Output: 3
Explanation:
The repeated subarray with maximum length is
[3, 2, 1].
```

Note:

1. 1 <= len(A), len(B) <= 1000
2. 0 <= A[i], B[i] < 100

```
class Solution(object):
    def findLength(self, A, B):
        """
        :type A: List[int]
        :type B: List[int]
        :rtype: int
        """

        max_len = 0
        result_list = [[0 for i in range(len(B)+1)] for j in range(len(A)+1)]
        for i in range(0,len(A)):
            for j in range(0,len(B)):
                if A[i]==B[j]:
                    result_list[i+1][j+1] = result_list[i][j]+1
                else:
                    result_list[i+1][j+1] = 0
                max_len = max(max_len, result_list[i+1][j+1])
        return max_len
```

Homework

Required:
121. Best Time to Buy and Sell Stock
122. Best Time to Buy and Sell Stock II
123. Best Time to Buy and Sell Stock III
714. Best Time to Buy and Sell Stock with Transaction Fee
322. Coin Change
718. Maximum Length of Repeated Subarray

Suggested:
188. Best Time to Buy and Sell Stock IV
309. Best Time to Buy and Sell Stock with Cooldown

Thank you