STOCK SELL BUY PROBLEMS

1-Best Time to Buy and Sell Stock

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int maxPro = 0;
        int minPrice = INT_MAX;
        for(int i = 0; i < prices.size(); i++){
            minPrice = min(minPrice, prices[i]);
            maxPro = max(maxPro, prices[i] - minPrice);
        }
        return maxPro;
    }
};
```

2-Best Time to Buy and Sell Stock II

```
class Solution {
public:
    unordered_map<string, int>hash;
    int solution(int ind, int cur_state, vector<int>&prices){
        if(ind>=prices.size())
            return 0;
        string key=to_string(ind)+"-"+to_string(cur_state);
        if(hash.count(key))
            return hash[key];
        int profit=0;
        if(cur_state==0){
            int buy=solution(ind+1, 1, prices)-prices[ind];
            int notBuy=solution(ind+1, 0, prices);
            profit=max(buy, notBuy);
        }else{
            int sell=solution(ind+1, 0, prices)+prices[ind];
            int notSell=solution(ind+1, 1, prices);
            profit=max(sell, notSell);
        }
        return hash[key]=profit;
    }
    int maxProfit(vector<int>& prices) {
        return solution(0, 0,  prices);
    }
};
```

3-Best Time to Buy and Sell Stock III

```
class Solution {
public:
    int dp[100001][2][2];
    int solution(int ind, int cur_state, int transactions, vector<int>&prices){
        if(ind>=prices.size() || transactions==2)
            return 0;
        if(dp[ind][cur_state][transactions]!=-1)
            return dp[ind][cur_state][transactions];
        int profit=0;
        if(cur_state==0){
            int buy=solution(ind+1, 1, transactions, prices)-prices[ind];
            int notBuy=solution(ind+1, 0, transactions, prices);
            profit=max(buy, notBuy);
        }else{
            int sell=solution(ind+1, 0, transactions+1, prices)+prices[ind];
            int notSell=solution(ind+1, 1, transactions, prices);
            profit=max(sell, notSell);
        }
        return dp[ind][cur_state][transactions]=profit;
    }
    int maxProfit(vector<int>& prices) {
        if(prices.size()<=1)
            return 0;
        memset(dp, -1, sizeof(dp));
        return solution(0, 0, 0, prices);
    }
};
```

4-Best Time to Buy and Sell Stock IV

```
class Solution {
public:
    vector<vector<vector<int>>> dp;
    int solution(int ind, int cur_state, int k, vector<int>&prices){
        if(ind>=prices.size() || k==0)
            return 0;
        if(dp[ind][cur_state][k]!=-1)
            return dp[ind][cur_state][k];
        int profit=0;
        if(cur_state==0){
            int buy=solution(ind+1, 1, k, prices)-prices[ind];
            int notBuy=solution(ind+1, 0, k, prices);
            profit=max(buy, notBuy);
        }else{
            int sell=solution(ind+1, 0, k-1, prices)+prices[ind];
            int notSell=solution(ind+1, 1, k, prices);
            profit=max(sell, notSell);
        }
        return dp[ind][cur_state][k]=profit;
    }
    int maxProfit(int k, vector<int>& prices) {
        if(prices.size()<=1)
            return 0;
        dp=vector<vector<vector<int>>>(prices.size()+1,vector<vector<int>>(2, vector<int>(k+1,-1)));
        return solution(0, 0, k, prices);
    }
};
```

5-Best Time to Buy and Sell Stock with Cooldown

```
class Solution {
public:
  unordered_map<string, int>hash;
  int solution(int ind, int cur_state, vector<int>&prices){
    if(ind>=prices.size())
      return 0;
    string key=to_string(ind)+"-"+to_string(cur_state);
    if(hash.count(key))
      return hash[key];
    int profit=0;
    if(cur_state==0){
      int buy=solution(ind+1, 1, prices)-prices[ind];
      int notBuy=solution(ind+1, 0, prices);
      profit=max(buy, notBuy);
    }else{
      int sell=solution(ind+2, 0, prices)+prices[ind];
      int notSell=solution(ind+1, 1, prices);
      profit=max(sell, notSell);
    }
    return hash[key]=profit;
  }
  int maxProfit(vector<int>& prices) {
    return solution(0, 0,  prices);
  }
};
```

6-Best Time to Buy and Sell Stock with Transaction Fee

```
class Solution {
public:
  int dp[100001][2];
  int solution(int ind, int cur_state, int fee, vector<int>&prices){
    if(ind>=prices.size())
      return 0;
    if(dp[ind][cur_state]!=-1)
      return dp[ind][cur_state];
    int profit=0;
    if(cur_state==0){
      int buy=solution(ind+1, 1, fee, prices)-prices[ind];
      int notBuy=solution(ind+1, 0, fee, prices);
      profit=max(buy, notBuy);
    }else{
      int sell=solution(ind+1, 0, fee, prices)+prices[ind]-fee;
      int notSell=solution(ind+1, 1, fee, prices);
      profit=max(sell, notSell);
    }
    return dp[ind][cur_state]=profit;
  }
  int maxProfit(vector<int>& prices, int fee) {
    if(prices.size()<=1)
      return 0;
    memset(dp, -1, sizeof(dp));
    return solution(0, 0, fee, prices);
  }
};
```