ARRAY

Arrays

Characteristics

- **Memory layout**: hold values in a **contiguous** block of memory.
- **Fixed Size**: the size of an array is defined when it is created and cannot be changed. However, high-level languages have different implementations, making it dynamic.
- **Homogeneous elements:** all elements are of the same data type (int, float, char…)
- **Efficiency**: accessing elements by index is very efficient *O(1)*, since each index maps directly to a memory location. Also, range scans benefit from CPU cache lines since arrays are stored in contiguous blocks of memory.

Arrays – Kadane's algorithm

- Kadane's algorithm is a dynamic programming algorithm to solve **maximum subarray sum**
- At every **index i**:

  start a new subarray at **i**

  extend the previous subarray to include **array[i]**
- **Algorithm**

1. Initialize:

```
int maxSoFar = array[0];
int maxEndingHere = array[0];
```

2. Loop through the array

```
for (int i = 1; i < array.size(); ++i) {
    maxEndingHere = max(array[i], maxEndingHere + array[i]);
    maxSoFar = max(maxSoFar, maxEndingHere);
}
```

3. Return maxSoFar;

Problem – Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Problem Statement

- Given an **array** of numbers and a **target**, example: **array [2,7,11,15]** and **target 9**
- Return indices of two numbers where they add up to **target**
- **Output**: [0,1]

```
array[0] + array[1] = 2 + 7 = 9
```

Solution – Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Solution

- Iterative over each number in the array
- Calculate the difference between target and each number, example:

  `array[0] = 2, target 9, then 9 - 2 = 7`
- Now we know we need the number **7** to sum up to **9**
- Check in a *hashmap* if we have 7 in some part of the array

  `hash[7] exists?`
- If yes, return the current index and the index of 7
- If not, store the index of the current number in the hashmap for future evaluation

  `hash[2] = 0`

Code – Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Code

Time: **O(n)** Space: **O(n)**

```
vector<int> twoSum(vector<int>& nums, int target) {
    std::unordered_map<int, int> numMap;
    // n being the size of nums
    for (int i = 0; i < nums.size(); i++) {
        // current number of the array
        int number = nums[i];
        int diff = target - number;

        // check if the difference is in some part of the array
        // by using a hashmap
        if (numMap.find(diff) != numMap.end()) {
            return { numMap[diff], i};
        }

        // register the current number index
        numMap[number] = i;
    }
    // no matches
    return {};
}
```

Problem - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Problem Statement

- Given an **array** of stock prices, choose a **price[i]** to buy and **price[i]** to sell where you achieve maximum profits
- **Example:**

  prices = [9, 1, 3, 4]
- **Output**: [1,3]

  `array[3] - array[1] = 4 + 1 = 3`

Solution - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Solution

- Initialize **profit = 0**
- Initialize **lowestBuyPrice = prices[0]**
- Loop through the prices
- Track the lowest buy price → **min(lowestBuyPrice, prices[i])**
- Check if selling "today" will make the maximum profit and update profit:

  max(prices[i] – buy > profit, profit)
- Update profit

  max(prices[i] - buy

Code - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Code (simplified)

Time: **O(n)** Space: **O(n)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        buy = min(buy, prices[i]);
        profit = max(profit, prices[i] - buy)
    }
    return profit;
}
```

Code - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Code (optimized)

Time: **O(n)** Space: **O(n)**

- Same logic, but with better branch prediction and less computation

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        if (prices[i] < buy) {
            buy = prices[i];
        } else if (prices[i] - buy > profit) {
              profit = prices[i] - buy;
        }
    }
    return profit;
}
```

Problem - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Problem Statement

- ...

Solution - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Solution

- ...

Code - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Code

Time: **O(n)** Space: **O(n)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        buy = min(buy, prices[i]);
        profit = max(profit, prices[i] - buy)
    }
    return profit;
}
```

Problem - Best Time to Buy and Sell Stock IV

Hard

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Problem Statement

- ...

Solution - Best Time to Buy and Sell Stock IV

Hard

leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Solution

- ...

Code - Best Time to Buy and Sell Stock IV

Hard

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Code (simplified)

Time: **O(n)** Space: **O(n)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        buy = min(buy, prices[i]);
        profit = max(profit, prices[i] - buy)
    }
    return profit;
}
```