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

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 O(n)

```
vector<int> twoSum(vector<int>& nums, int target) {
    std::unordered_map<int, int> numMap;
    // O(n)
    // 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

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

You are given an array **prices** where **prices[i]** is the price of a given stock on the **i**th day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1

Input: prices = [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.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

Solution - Best Time to Buy and Sell Stock

Easy

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

```
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;
}
```