**Interview Cake**

1. **Stock question:**

\_ Cannot just find the lowest and highest price because there is an order to it: must buy before sell

\_If stock value goes down all day, best profit will be negative

Math.min( one number, another number) : return smaller value of the 2

Greedy algorithm: iterates through the problem space taking the optimal solution, until it reaches the end

The max profit we can get by selling at the currentPrice is simply the difference between the currentPrice and the minPrice from earlier in the day. If this difference is greater than the current maxProfit, we have a new maxProfit.

(because unit of sale is constant (1))

Q:

// var stockPricesYesterday = [10, 7, 5, 8, 11, 9];

// getMaxProfit(stockPricesYesterday);

// // returns 6 (buying for $5 and selling for $11)

// how to do this: get the smallest and largest value of the array

// then sum them together

A:

function getMaxProfit(array){

if(array.length < 2){

throw 'This wont work unless we have at least 2 prices'

}

// Profit = difference between current & min price b/c unit of

// sale stays the same

// initialize them to the first possible instances

// not 0 because if the profit goes down, maxProfit won't

// decrease

var minPrice = array[0];

var maxProfit = array[1]- array[0];

// starts at i = 1 because we have to buy before sell, and we cannot

// do that at the same time

for(var i = 1; i< array.length ; i++){

//currentPrice

// current Profit

// maxProfit

// minPrice

let currentPrice = array[i];

let currentProfit = array[i] - minPrice;

// maxProfit comes first to ensure that we always buy

// before sell

maxProfit = Math.max(currentProfit, maxProfit);

minPrice = Math.min(currentPrice, minPrice)

}

return maxProfit;

}

console.log(getMaxProfit([10, 9, 8, 7, 1]))

Q: Greedy approach?

A: Get the local optimal solution in the hope that we’ll be able to find globally optimal solution

\_ write down the logic first, find the logic behind each question

1. **Array = products of all others but not at index**

Q:

// [1, 7, 3, 4]

// your function would return:

// [84, 12, 28, 21]

A: function getProductsOfAllIntsExceptAtIndex(array){

if(array.length < 2){

//edge case

throw ' Requires at least 2 numbers'

}

const productsOfEverythingExceptAtInteger = [];

// all the integers that come before

let productSoFar = 1;

for(let i = 0; i< array.length; i++){

//productSoFar at array[0] is 1

productsOfEverythingExceptAtInteger[i] = productSoFar;

// array[i], not array[i-1] because you update this after

// so when it's updated the line above will be i+1

productSoFar = productSoFar \* array[i];

}

productSoFar = 1;

for(let i = array.length-1; i >= 0; i--){

//Here we multiply because it's the stage where we're

//supposed to multiply things together

productsOfEverythingExceptAtInteger[i] \*= productSoFar;

// array[i], not array[i-1] because you update this after

// so when it's updated the line above will be i+1

productSoFar = productSoFar \* array[i];

}

return productsOfEverythingExceptAtInteger;

}

console.log( getProductsOfAllIntsExceptAtIndex( [1, 7, 3, 4]))

// Strategy: Draw on paper and find a pattern