Day 03

Thursday, December 31, 2020 7:44 PM

HomeWork Assignments:

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/
https://leetcode.com/problems/plus-one/
https://leetcode.com/problems/add-binary/
https://leetcode.com/problems/majority-element/

```
public static int maxWater(int[] arr, int n)
{

    // To store the maximum water
    // that can be stored
    int res = 0;

    // For every element of the array
    // except first and last element
    for(int i = 1; i < n - 1; i++)
    {

        // Find maximum element on its left
        int left = arr[i];
        for(int j = 0; j < i; j++)
        {
            left = Math.max(left, arr[j]);
        }

        // Find maximum element on its right
        int right = arr[i];
        for(int j = i + 1; j < n; j++)
        {
            right = Math.max(right, arr[j]);
        }

        // Update maximum water value
        res += Math.min(left, right) - arr[i];
    }
    return res;
}


  static int findWater(int n)
    {
        // left[i] contains height of tallest bar to the
        // left of i'th bar including itself
        int left[] = new int[n];

        // Right [i] contains height of tallest bar to
        // the right of ith bar including itself
        int right[] = new int[n];

        // Initialize result
        int water = 0;

        // Fill left array
        left[0] = arr[0];
        for (int i = 1; i < n; i++)
            left[i] = Math.max(left[i - 1], arr[i]);

        // Fill right array
        right[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--)
            right[i] = Math.max(right[i + 1], arr[i]);

        // Calculate the accumulated water element by element
        // consider the amount of water on i'th bar, the
        // amount of water accumulated on this particular
        // bar will be equal to min(left[i], right[i]) - arr[i] .
        for (int i = 0; i < n; i++)
            water += Math.min(left[i], right[i]) - arr[i];
        return water;
    }
```

Problem Statement : **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 th
stock), design an algorithm to find the maximum profit.

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

Input: [7,1,5,3,6,4]
Output : 5

```
public int maxProfit(int prices[])
{
  int maxProfit = 0 ;
  for(int I = 0; I < prices.length - 1; i++)
 {
    for(int j =i+1; I < prices.length; j++)
    {

      int profit = prices[j] - prices[i];
          if(profit > maxprofit)
         {
          maxprofit = profit ;
         }
    }
    return maxProfit;
  }
}
```

Input: [7,1,5,3,6,4]

```
public int maxProfit(int prices[])
{
 int minPrice = Integer.MAX_Value;
 int maxProfit = 0;
 for(int I = 0; I < prices.length; i++)
     {
```

```
                if(price[i] < minPrice )
                {
                    minPrice = prices[i];
                }
                else if( price[i] - minprice  >  maxprofit )
                {
                      maxprofit = price[i] - minPrice;
                }
             }
            return maxProfit;
        }




        }
```

https://leetcode.com/problems/plus-one/

Given a **non-empty** array of decimal digits representing a non-negative integer, increment one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element i array contains a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Input: digits = [1,2,3]

Input: digits = [4,3,2,1]

[9 , 9 , 9]

```
for()
{
  if(digits[i] < 9)
  {
             digits[i] ++;
            return digits;
  }

   digits[i]  = 0

}

int[] result =  new[digits.length +1]
result[0] = 1;
return result;

}
```

https://leetcode.com/problems/majority-element/

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

[3, 3 , ,2, 2]

HashTable & BruteForce Approach

Boyer-Moore Voting Algorithm

```
public int majorityElement(int[] nums)
{

        int count = 0;
         Integer candiadate = null;
         for(int num: nums)
        {

           if(count == 0)
          {
             candidate = num;
          }

          count + =  ( candidate == num)  ? 1 : -1;

        reurn candidate ;

}
```

```
for( int I = 0 ; I < arr.length; i++)
{

      if(a[i]  == candidate)
        count ++;

}

if(count > arr.length / 2)
{

return true  ;

}

return false;
```