Linked List

Mustafa Muhammad

12th November 2021

Checkout the other common questions like validating brackets as well .....

1 Two sum

Find index of two numbers that add upto the target sum

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hash_map = {}
        for i in range(0, len(nums)):
            targ = target - nums[i]
            if targ in hash_map:
                return hash_map[targ], i
            else:
                hash_map[nums[i]] = i

        return -1,-1
```

2 Best Time to Buy and Sell Stock

Simple 1 pass through an array Bane of my existance

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        profit = 0
        buy = float('inf')
        for i in range(0, len(prices)):
            buy = min(prices[i], buy)
            profit = max(profit, prices[i]-buy)

        return profit
```

3 Contains Duplicate

```
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        dup = set()
        for i in range(0, len(nums)):
            if nums[i] in dup:
                return True
            else:
                dup.add(nums[i])
        return False
```

4 Maximum Subarray

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        #Kadane's algorithm repeated work
        maxSum = nums[0]
        currSum = 0
        for i in range(0, len(nums)):
            if currSum<0:
                currSum = 0
            currSum += nums[i]
            maxSum = max(maxSum, currSum)

        return maxSum
```

5 Maximum Product Subarray

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        #Kadane's algorithm
        res = max(nums)
        curr_max = 1
        curr_min = 1

        for i in range(0, len(nums)):
            #Edge case, if 0 zero it makes eveything after it zero.
            if nums[i] == 0:
                curr_max = 1
                curr_min = 1
                continue

            temp = curr_max
            curr_max = max(nums[i]*curr_max, curr_min*nums[i], nums[i])
            curr_min = min(curr_min*nums[i], temp*nums[i], nums[i])
```

```
            res = max(res, curr_max)
        return res
```

6 Product of Array except self

Input: nums = [1,2,3,4] Output: [24,12,8,6]

Do it in 0(1) without extra space

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        res = [1] * len(nums)
        prefix = 1
        for i in range(0, len(nums)):
            res[i] = prefix
            prefix *= nums[i]

        postfix = 1
        for j in range(len(nums)-1, -1, -1):
            res[j] *= postfix
            postfix *= nums[j]

        return res
```

7 Find array in rotated sorted array

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        # binary search
        left = 0
        right = len(nums)-1

        while left <= right:
            if nums[left] <= nums[right]:
                return nums[left]

            mid = int(left + (right - left)/2)

            next = (mid+1) % len(nums)
            prev = (mid-1+len(nums)) % len(nums)

            if nums[mid] <= nums[prev] and nums[mid] <= nums[next]:
                return nums[mid]

            elif nums[left] <= nums[mid]:
```

```
                left = mid + 1

            elif nums[mid] <= nums[right]:
                right = mid -1
        return -1
```

8 Search in a rotated sorted array

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        def find_index(nums):
            left = 0
            right = len(nums)-1
            n = len(nums)

            while left <= right:
                mid = int(left + (right-left)/2)
                next = (mid + 1)%n
                prev = (mid-1+n)%n

                if nums[mid] <= nums[prev] and nums[mid] <= nums[next]:
                    return mid

                if nums[0] <= nums[mid]:
                    left = mid + 1

                if nums[mid] <= nums[-1]:
                    right = mid - 1

            return -1

        def binary_search(nums, target):
            left = 0
            right = len(nums)-1

            while left <= right:
                mid = int(left +(right-left)/2)

                if nums[mid] == target:
                    return mid

                elif nums[mid] < target:
                    left = mid+1

                elif nums[mid] > target:
                    right = mid -1
```

```
            return -1

        min_index = find_index(nums)
        left_arr = binary_search(nums[0:min_index], target)
        right_arr = binary_search(nums[min_index:], target)

        if right_arr != -1:
            return right_arr + len(nums[0:min_index])

        return left_arr
```

9 3 Sum

Find three numbers that add upto 0

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        if len(nums) < 3:
            return None

        res = []
        nums = sorted(nums)

        for i in range(0, len(nums)):
            first = nums[i]
            if i > 0 and first == nums[i-1]:
                continue
            left = i+1
            right = len(nums)-1

            while left < right:
                if first + nums[left] + nums[right] < 0:
                    left += 1
                elif first+nums[left]+nums[right] > 0:
                    right -=1

                else:
                    res.append((first, nums[left], nums[right]))
                    left += 1
                    while nums[left] == nums[left-1] and left < right:
                        left += 1
        return res
```

10 Container with most water

Input: height = [1,8,6,2,5,4,8,3,7] Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        left = 0
        right = len(height)-1

        max_area = 0

        while left < right:
            curr_height = min(height[left], height[right])
            max_area = max(max_area, (right-left)*curr_height)

            if height[left] < height[right]:
                left += 1
            else:
                right -=1

        return max_area
```

11 Merge Sort

Divide and conquer algorithm, to break down to a single element and recursively build up while sorting.

```
def sort(arr):
    def merge_sort(arr):
        if len(arr) > 1:
            mid = int(len(arr)/2)

            left_arr = arr[:mid]
            right_arr = arr[mid:]

            merge_sort(left_arr)
            merge_sort(right_arr)

            i, j, k = 0, 0, 0

            while i < len(left_arr) and j < len(right_arr):
                if left_arr[i] < right_arr[j]:
                    arr[k] = left_arr[i]
                    i+=1
```

```
            else:
                arr[k] = right_arr[j]
                j+=1
            k+=1

        while i < len(left_arr):
            arr[k] = left_arr[i]
            i+=1
            k+=1

        while j < len(right_arr):
            arr[k] = right_arr[j]
            j+=1
            k+=1
    merge_sort(arr)
    return arr
```

12 Quick Sort

Select a pivot point and arrange values according to the pivot.

```
def sort(arr):
    def quick_sort(arr):
        if len(arr) <= 1:
            return arr

        pivot = arr.pop()


        left, right = [], []

        for item in arr:
            if item < pivot:
                left.append(item)
            else:
                right.append(item)

        return quick_sort(left) + [pivot] + quick_sort(right)
    quick_sort(arr)
    return arr
```