ARRAYS-1

(with explanations)

Question:

Given a list of positive integers *nums* and an int *target*, return indices of the two numbers such that they add up to a target.

Conditions:

1. You will pick exactly 2 numbers
2. You cannot pick the same element twice
3. If you have multiple pairs select the pair with the largest number

Link : https://leetcode.com/problems/two-sum/

EXAMPLE 1 :

INPUT: *nums = [1 , 10 , 25 , 35 , 60]* and *target = 60*

OUTPUT: *[2, 3]*

EXPLANATION: *nums[2] + nums[3] = 25 + 35*

EXAMPLE 2 :

INPUT: *nums = [20 , 50 , 40, 25 , 30 , 10]* and *target = 60*

OUTPUT: *[1, 5]*

EXPLANATION: *nums[1] is the largest, nums[1] + nums[5] = 50 + 10*

SOLUTION1 - INEFFICIENT

```python
# Two for loops Two Sum

class Solution:
    def twoSum(self, nums: List[int], target: int) ->
        List[int]:
        res = {}
        result = []
        i = 0
        while i < len(nums):
            j=i+1
            while j < len(nums):
                if nums[i] + nums[j] == target:
                    result = [i,j]
                j+=1
            i+=1
        return result
```

TIME COMPLEXITY - $O(N^2)$

SOLUTION2 - INCORRECT

```python
# Two Sum w Sort

class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        nums.sort()
        l = 0
        r = len(nums) -1
        while l<r:
            if (nums[l] + nums[r] == target):
                return [l,r]
            elif (nums[l] + nums[r] < target):
                l += 1
            else:
                r -= 1
        return
```

INDEXES CHANGED

This approach doesn't work with constraints of this question. Because indexes must be returned. Indexes change after sorting.

SOLUTION3 EFFICIENT

```python
# Two Sum w Map

class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        res = {}
        result = []
        for key,value in enumerate(nums):
            if target - value in res:
                result.extend([res[target-value], key])
                return result
            else:
                res[value] = key

        return result
```

TIME COMPLEXITY - O(N) SPACE COMPLEXITY - O(N)

Question: Given an array of strings, group anagrams togethers.

EXAMPLE:

INPUT: [“eat” , ”tea” , ”tan” , ”ate” , ”nat” , ”bat”]

OUTPUT: [[“ate” ,”eat” ,”tea”], [“nat”,”tan”], [“bat”]]

Link : https://leetcode.com/problems/group-anagrams/

(Two strings are anagrams, if they have exactly the same characters in any order)

BUT FIRST -> VALID ANAGRAMS

```python
# Check two strings are anagrams by sort

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if (len(s) != len(t)):
            return False
        return sorted(s) == sorted(t)
```

TIME COMPLEXITY - O(KLOG(K)). Essentially the time complexity to sort the strings and then compare lexicographically. K = Size of the string s and t.

TIME COMPLEXITY - O(K). SPACE COMPLEXITY - O(K) K = Size of the string s and t.

VALID ANAGRAMS

```python
# Check two strings are anagrams using dict(map)
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if (len(s) != len(t)):
            return False
        s_dict = {}
        t_dict = {}
        for ch in s:
            s_dict[ch] = 1 if ch not in s_dict else
                s_dict[ch] + 1
        for ch in t:
            t_dict[ch] = 1 if ch not in t_dict else
                t_dict[ch] + 1
        return t_dict == s_dict
```

TIME COMPLEXITY - O(NKLOG(K))

GROUP ANAGRAMS BY SORT

```python
# Group anagrams using sorting
class Solution:
    def groupAnagrams(self, strs: List[str]) ->
        List[List[str]]:
        result = collections.defaultdict(list)
        for s in strs:
            result[tuple(sorted(s))].append(s)
        return result.values()
```

TIME COMPLEXITY - O(NK) SPACE COMPLEXITY - O(K)

GROUP ANAGRAMS BY COUNT MAP

```python
# Group anagrams using count map
class Solution:
    def groupAnagrams(self, strs: List[str]) ->
        List[List[str]]:
        result = collections.defaultdict(list)
        for s in strs:
            count = [0]*26
            for c in s:
                count[ord(c) - ord('a')] +=1
            result[tuple(count)].append(s)
        return result.values()
```

Question: Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

EXAMPLE:

INPUT: s: "cbaebabacd" p: "abc"

OUTPUT: [0, 6]

Link: https://leetcode.com/problems/find-all-anagrams-in-a-string/

The substring with start index = 0 is "cba", which is an anagram of "abc". The substring with start index = 6 is "bac", which is an anagram of "abc"

TIME COMPLEXITY - O(NK) SPACE COMPLEXITY - O(K)

SIMPLE SOLUTION

```
2  class Solution:
3      def findAnagrams(self, s: str, p: str) -> List[int]:
4          result = {}
5          cnt=0
6          patternLength = len(p)
7          for i in range( len(s) - patternLength + 1 ):
8              if self.isAnagram(s[i:i+patternLength],p):
9                  result[cnt] = i
10             cnt+=1
11         return result
```

SLIDING WINDOW VIDEO EXPLANATION:

https://youtu.be/-rcfE1Tj2E0

READING EXPLANATION:

https://www.geeksforgeeks.org/anagram-substring-search-search-permutations/

SLIDING WINDOW APPROACH

```python
# Sliding window
MAX = 256
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        result = {}
        cnt = 0
        patternLen = len(p)
        textLen = len(s)

        if patternLen == 0 or textLen == 0 or patternLen > textLen:
            return result

        countPattern = [0]*MAX
        countText = [0]*MAX

        for i in range(patternLen):
            (countPattern[ord(p[i]) ]) += 1
            (countText[ ord(s[i]) ]) += 1

        for i in range(patternLen, textLen):
            if self.compare(countPattern, countText):
                result[cnt] = i-patternLen
            cnt+=1

            (countText[ord(s[i])]) += 1
            (countText[ ord(s[i-patternLen]) ]) -= 1

        if self.compare(countPattern, countText):
                result[cnt] = textLen - patternLen
        return result
```

TIME COMPLEXITY - O(N+K) SPACE COMPLEXITY - O(K)

REFERENCE FOR SOLUTIONS IN OTHER LANGUAGES:

1. TWO SUM: https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/
2. GROUP ANAGRAMS: https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together/
3. FIND ALL ANAGRAMS: https://www.geeksforgeeks.org/anagram-substring-search-search-permutations/