1. Two Sum

```
class Solution {
   func twoSum(_ nums: [Int], _ target: Int) ->
[Int] {

      var numIndexMap = [Int : Int] ()
      for(index, number) in nums.enumerated()
      {
       let requiredNumber = target - number

       if let requiredIndex =
numIndexMap[requiredNumber]
       {
           return [requiredIndex, index]
       }

       numIndexMap[number] = index
   }
       return []


   }
}
```

242. Valid Anagram

```
class Solution {
   func isAnagram(_ s: String, _ t: String) ->
Bool {
       if s.count != t.count
       {
           return false
       }

       var characterCount = [Character : Int]()
       for char in s
       {
           characterCount[char, default: 0] +=
1
       }

       for char in t
       {
           characterCount[char, default: 0] -=
1

           if characterCount[char]!<0
```

```
            {
                return false
            }
        }
        return true



    }
}
```

217. Contain Duplicate

```
class Solution {
    func containsDuplicate(_ nums: [Int]) ->
Bool {
        var find = Set<Int>()
        for num in nums
        {
            if find.contains(num)
            {
                return true
            }
            find.insert(num)
        }
        return false
    }
}
```

49. Group Anagrams

```
class Solution {
    func groupAnagrams(_ strs: [String]) ->
[[String]] {
        var anagramDictionary = [String :
[String]]()


        for str in strs
        {
            var count = Array(repeating: 0,
count: 26)

            for char in str
            {
                let index = Int(char.asciiValue!
- Character("a").asciiValue!)
                count[index] += 1
            }
            let key = count.map
{String($0)}.joined(separator: "#")
            anagramDictionary[key, default:
[]].append(str)
        }
        return Array(anagramDictionary.values)
    }
}
```

347. Top K Frequent Elements

```
class Solution {
    func topKFrequent(_ nums: [Int], _ k: Int)
-> [Int] {
        var dictionary = [Int : Int]()
        for num in nums{
            dictionary[num, default: 0] += 1
        }

        let sortedKeys =
dictionary.keys.sorted{dictionary[$0]! >
dictionary[$1]!}
        return Array(sortedKeys.prefix(k))
    }
}
```

271. Encode and Decode String

```
class Codec {
    let encoder = JSONEncoder()
    let decoder = JSONDecoder()

    func encode(_ strs: [String]) -> String {
         if let data = try? encoder.encode(strs)
{
              return String(data: data, encoding:
.utf8) ?? ""
         }
         return ""
    }

    func decode(_ s: String) -> [String] {
         if let data = s.data(using: .utf8),
            let strs = try?
decoder.decode([String].self, from: data) {
              return strs
```

```
        }
        return []
    }
}
```

238. Product of Array Except Self

```
class Solution {
    func productExceptSelf(_ nums: [Int]) ->
[Int] {
        let n = nums.count
        var leftArray = Array(repeating: 1,
count: n)
        var rightArray = Array(repeating: 1,
count: n)

        for i in 1..<n {

            leftArray[i] = leftArray[i - 1] *
nums[i - 1]

        }
```

```
       for i in stride(from: n - 2, through: 0,
by: -1) {
         rightArray[i] = rightArray[i + 1] *
nums[i + 1]
         }

       var answerArray = Array(repeating: 1,
count: n)
       for i in 0..<n
       {
           answerArray[i] = leftArray[i] *
rightArray[i]
       }

       return answerArray




   }
}
```

128. LeetCode Consecutive Sequence

```
class Solution {
    func longestConsecutive(_ nums: [Int]) ->
Int {
        var longestLength = 0
        var dictionary : [Int : Bool] = [:]

        for num in nums
        {
            dictionary[num] = false
        }

        for num in nums{
            var currentLength = 1
            var nextNum = num + 1

        while let value = dictionary[nextNum],
value == false {
           currentLength += 1
           dictionary[nextNum] = true
           nextNum += 1
           }

        var prevNum = num - 1
        while let value = dictionary[prevNum],
value == false {
           currentLength += 1
           dictionary[prevNum] = true
           prevNum -= 1
           }

            longestLength = max(longestLength,
currentLength)

        }
        return longestLength

    }
}
```