1) Centre x Array
— Very Quick (Odd Cases)
— Two pointers (↑, ↓)
— Convex (↑, ↓)
— Binary Search (↓→↑)

2) Check if Array Array is Simplified Subsetting of Another Array
— Simple counter of Aggregate Checks
— Binary Search.
— Advanced array checking frequency counting

3) Check Whether Array is a Palindrome
— Two pointer

Palindrome Subsetting Checks.
- Check count of frequencies of palindrome in Array (and count every element).
- Find check, count of frequencies of characters in Array whether it forms a palindrome.
- Given input a long Array. (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
  - ↑ can check every odd, every odd, every odd, every odd.
  - ↑ can check every odd, every odd, every odd, every odd.
  - ↑ can check every odd, every odd, every odd, every odd.
- Minimum count of that elements, to palindrome number in Array(Array).
- The last element is odd divided by odd length, is definitely check it.

4) Check if Given Array was constructed of such strings
— Very simple.
- ↑ ↓ step of every odd signal, either odd, pair signal using to ↑↓ (pair)
- pair, pair, odd, ↑↓ ↑↓ (pair)

```
count = 0
while i < len(s):
    if s[i] == '↑':
        count += 1
    elif s[i] == '↓':
        count -= 1
    elif s[i:i+2] == '↑↓':
        count += 1
    elif s[i:i+2] == '↓↑':
        count -= 1
    i += 1
return True
```

5) Sort and Map
- 1, 11, 11, 111, 11111, 11111, 111111 . . . . .
Return the Number by counting (↑↓)* times

```
def helper(s):
    dp = [[-1]*(len(s)+1)
          for i in range(len(s))]
    count = 0
    i = 0
    while i < len(s):
        if s[i] == '↑':
            count += 1
        elif s[i:i+2] == '↑↓':
            count += 1
            i += 1
        elif s[i:i+2] == '↓↑':
            count -= 1
        elif s[i] == '↓':
            count -= 1
        i += 1
    return count
return -1
```

6) Find Longest Subsequence in Array (Longest Subsequence Subsets)
— Sort in the place, a subset count original to find subsequence by longest subsequence and keep track of longest subsequence subsetting counting.
Eg. 1) Input: a = [10, 22, 9, 33, 21, 50, 41, 60, 80]
Output: 6 (Longest subsequence is [10, 22, 33, 50, 60, 80])
1) Create Longest Subsequence Array with size 10 to the (Array)
2) ↑ using binary search find the index of the first element in the array greater than or equal to the current element.
3) ↑ keep increasing target, get (Array[mid] >= Array[i])
4) ↑ keep decreasing left mid: (Array[mid] >= Array[i])
5) Finally, keep increasing left mid decreasing left mid: (Array[mid] >= Array[i])
6) Calculate length(Longest) = ↑↓ length (Longest) + 1.
Update accordingly mid, find = left + 1.
7) Return maximum mid, (Array [mid] + 1)(Longest)

```
def lengthOfLIS(self, nums):
    n = len(nums)
    if n == 0:
        return 0
    dp = [1] * n
    for i in range(1, n):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)
    max_length = max(dp)
    return max_length
```

7) Find Longest Subsequence in Array
Longest Increasing Subsequence in Array
DP

8) Search for Subsequence Subsequence in a Array
DP

9) Longest a count in a ↑↓ pair of times.

10) Longest Time, Array in Pairs Longest

11) Longest all subsequence of a Array
— ↑↓ → ↑, ↑↓, ↑↓, ↑↓, ↑↓, ↑↓.
— Pick and Sort Pick.

```
def getSubsequences(self, nums):
    subsequences = []
    for i in range:
        current = [nums[i]]
        for j in range(i + 1, len(nums)):
            if nums[j] > current[-1]:
                current.append(nums[j])
        subsequences.append(current)
    return subsequences
```

12) Longest Array Array
Array Subsequence

13) Longest Array Array Subsequence with Pairs (↑↓ or ↓↑ and ↑↓)
— use count of consecutive subsequence odd, then find longest odd.
— take both odd, two subsequence are balanced.
— Example, count by odd and long count of ↑↓ and ↓↑.
↑↓ signal, balanced length.
Eg. 1) Input: [1, 17, 5, 10, 13, 15, 10, 5, 16, 8]
Output: 7
Longest subsequence: [1, 17, 5, 15, 13, 10, 16]
Eg. 2) Input: [44]
Output: 1
Longest subsequence: [44]
Eg. 3) Input: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Output: 2
Longest subsequence: [1, 2]

```
def longest(arr):
    n = len(arr)
    if n == 1:
        return 1
    if n == 2:
        return 2
    dp = [[0, 0] for _ in range(n)]
    dp[0][0] = 1
    dp[0][1] = 1
    for i in range(1, n):
        for j in range(0, i):
            if arr[i] > arr[j]:
                dp[i][0] = max(dp[i][0], dp[j][1] + 1)
            elif arr[i] < arr[j]:
                dp[i][1] = max(dp[i][1], dp[j][0] + 1)
    return max(dp[-1])
```

14) Find Longest Array Subsequence
— Find the longest subsequence in array, find longest subsequence, find longest subsequence, find longest subsequence.
— Find longest subsequence in array, find longest subsequence, find longest subsequence.
— Find longest subsequence in array, find longest subsequence, find longest subsequence.
— Find longest subsequence in array