1. Array journey
   We will traverse all previous k elements and pick the maximum one and add to the current position. TC is O(nk)

```
def arrayJourney(l, k):
    length = len(l)
    for i in range(length):
        if i < k:
            l[i] = max(l[:i] + [0]) + l[i]
        else:
            l[i] = max(l[i - k:i] + [0]) + l[i]
    return max(l[length - k:])

l = [10, 2, -10, 5, 20]
l = [3, 10, -20, -5, 2]
k = 2
print(arrayJourney(l, k))
```

2. Ascending Binary Sorting
   We will use sort directly. Set key as (len(1's present times), number)
3. Balanced or Not
   We will use a stack to memorize left (and compare its redundant right) with i. TC is O(n)
4. 1177. Can Make Palindrome from Substring

```
from collections import Counter
from collections import defaultdict
from bisect import *
class Solution:
    def canMakePaliQueries(self, s: str, queries: List[List[int]]) -> List[bool]:
        result = []
        self.memo = {}
        self.dict = defaultdict(list)
        for index, i in enumerate(s):
            self.dict[i].append(index)
        for start, end, k in queries:
            if (end - start + 1) // 2 <= k:
                result.append(True)
            else:
                if self.parDif(start, end, s) <= k:
                    result.append(True)
                else:
                    result.append(False)
        return result

    def parDif(self, start, end, s):
```

```
        if (start, end) in self.memo:
            return self.memo[(start, end)]
        count = 0
        for i in 'abcdefghijklmnopqrstuvwxyz':
            count += (bisect(self.dict[i], end) - bisect_left(self.dict[i], start)) % 2

        if (end - start + 1) % 2 == 0:
            self.memo[(start, end)] = count // 2
            return self.memo[(start, end)]
        else:
            self.memo[(start, end)] = (count - 1) // 2
            return self.memo[(start, end)]
```

5. Array Journey

```
def journey(path, k):
    ln = len(path)
    if ln == 0:
        return 0

    queue = []

    i = 0
    result = 0
    while i < ln:
        #remove the first element from the queue if it is outside the window
        if len(queue) > 0 and i - queue[0][1] > k:
            queue.pop(0)

        # also remove any elements that are less than the current num
        # as long as the current num is in the boundary I don't care about any other number
        # if this is the max, then be it.
        temp = path[i]
        if len(queue) > 0:
            temp = max(temp, queue[-1][0] + path[i])
        while queue and queue[-1][0] <= temp:
            queue.pop()
        queue.append((temp, i))
        i += 1

    return queue[-1][0]
```