JANUARY
18
THURSDAY
'24
03rd Week
018-348

l.reverse() — वही list को modify करेगा।
reversed(l) — ये नया list create करेगा। viz reversed version of it.

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Max element / min element

(a) return max(l)

(b)
```
def fun(l):
    m = l[0]
    for i in l:
        if i > m:        # for min '<'
            m = i
    return m
```

(c) return min(l)

second largest element in a list.

(a)
```
l.sort()
l[-2]
```

(b)
```
def largest(arr, arr_size):
    if (arr_size < 2):
        print("invalid input")
        return
    largest = second = -2147483648
    for i in range(0, arr_size):
        largest = max(largest, arr[i])
    for i in range(0, arr_size):
        if (arr[i] != largest):
            second = max(second, arr[i])
    if (second == -2147483648):
        print("No 2nd largest element")
    else:
        print("2nd largest element", second)
```

(c)
```
def largest(arr, arr_size):
    if (arr_size < 2):
        print("Invalid input")
        return
    first = second = -2147483648
    for i in range(arr_size):
        if (arr[i] > first):
            second = first
            first = arr[i]
        elif (arr[i] > second and
              arr[i] != first):
            second = arr[i]
    if (second == -2147483648):
        print("NO 2nd element")
    else:
        print("The 2nd element is", second)
```

FEB 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

→ l.sort → वही list को modify करेगा।
→ sorted(l) → ये new list create करेगा। '24
V12 sorted version of list
03rd Week
019-347

JANUARY 19 FRIDAY

list is sorted or not

(a) `l == l.sort()` — O(nlogn)

(b)
```
def sort(l):
    i = 1
    while i < len(l):
        if l[i] < l[i-1]:
            return False
        i = i+1
    return True
```
— O(n)

(c)
```
if sorted(l) == l:
    return True
else:
    return False
```

remove duplicates

(a)
```
def rd(arr,n):
    temp = [0]*n
    temp[0] = arr[0]
    res = 1
    for i in range(1,n):
        if temp[res-1] != arr[i]:
            temp[res] = arr[i]
            res += 1
    for i in range(0, res):
        arr[i] = temp[i]
    return res.
```

(b)
```
def remup(arr,n):
    res = 1
    for i in range(1,n):
        if (arr[res-1] !=
            arr[i]):
            arr[res]=arr[i]
            res += 1
    return res.
```

(c) `return set(l)`

Rotation array.

(Left)

m-1 `l = [1,2,3,4,5]` → `[2,3,4,5,1]`

```
l = l[1:] + l[0:1]
print(l)
```

m-2 `l.append(l.pop(0))`

(Right)

m-1 `l = [1,2,3,4,5]` → `[5,1,2,3,4]`

~~`l[-2:-1:-1]`~~

```
l = [-1:-2:] + [:len(l)-1]
```

Nothing is so infectious as an example

JANUARY
20
SATURDAY
'24
03rd Week
020-346

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JAN 2024

→ Using for loop.

```
def rotate(l):
    n = len(l)
    x = l[0]
    for i in range(1,n):
        l[i-1] = l[i]
    l[n-1] = x
```

→ left rotate by d position.

(a) O(n)

```
l = [1,2,3,4,5,6]
d = 2
l = l[d:] + l[:d]
```

(b) O(n)

```
d = 2
for i in range(0,d):
    l.append(l.pop(0))
```

→ Right rotate by d position.

(a)

```
d = 2
l = l[len(l)-d:] + l[:len(l)-d]
```

↳ `l = l[-d:] + l[:-d]`

(b)

```
d = 2
for i in range(0,d):
    l.insert(0,l.pop(-1))
print(l)
```

Majority Element. → The element that appear more than n/2 time

(a)

```
nums.sort()
return nums[len(nums)//2]
```

(b)

```
from collections import Counter
def fun(nums):
    counts = Counter(nums)
    return max(counts, key=counts.get)
```

Necessity is the mother of invention, but Laziness is the father

Left rotation by d Places. i/p = l = [1,2,3,4,5] d = 2
o/p = [3,4,5,1,2]

l = [10,20,30,40,50]

d = 2

dq = deque (l) ⇒ for using deque we must have to import it first.
dq.rotate(-d) from collections import deque.
l = list (dq)
Print(l)

↳ TC = O(k) ∴ No. of element shifted.

Reverse Array/List

M-1
nums = [5,4,3,2,1]
nums.reverse()

M-2
rv = nums[::-1]

M-3
n = reversed(nums)
print(n) : insert the item at index '0' effectively pushing all existing element one position right.

M-4
l = []
for i in nums:
    l.insert(0, i)
Print(l).

M-5
~~r = reversed(nums)~~
~~l = []~~
~~while r: l.append(r.__next__())~~

Missing NO:

arr = [1|3|4|5|6]
n = 6, so, 1,2,3,4,5,6

⊕ = 19
Sum निकाले
then ⊖ कर दो

⊕ = 21
Sum निकाले

M-1
def MissingNO:
    S = sum(arr)
    l = n*(n+1)//2
    return l - S

firstly find sum of given array
find sum of total element in array
return sum of total element − sum of given elem

List Function in Python.

(a) sum() → Sum up the numbers in list.
(b) ord() → return integer of unicode.
(c) cmp() → function return 1 if first list is greater than the second list.
(d) max() → return maximum element of list.
(e) min() → return minimum element of list.
(f) all() → return true if all element is true (or) if the list is empty.
(g) any() → return true if any element of the list is true, if the list is empty return false
(h) len() → return length of list (or) size of list.
(i) filter() → test if each element of a list is true (or) not.
(j) map() → return a list of the result after applying the given function to each item of given iterable.
(k) Lambda → this function can have only/any no. of argument but only one expression which is evalluated and returned.
(l) copy() → It is used to copy list to another list.
(m) extend() → It is used to append whole list to another list.
(n) reverse() → It is used to reverse the complete list. [List specific method]
(o) append() → It is used to append/add the element in list at the end.
(p) sort() → It is used to sort the list in ascending. [List specific method]
(q) count() → It is used to count the occurrence of element in list.
(r) index() → It is used to find the index of element in list.
(s) clear() → It is used to clear the list.
(t) remove() → It is used to remove the particular element from list.
(u) insert() → It is used to insert element in list at specific position.
(v) pop() → It is used to remove last element of list.
(w) reversed() → It is used to reverse the list but it will create new list.
(x) sorted() → It is used to sort the list but it will create new list.
(y) sorted(li, reverse=True) → It is used to sort the list in descending order.
(z) in → to check membership of element in list.
(a) not in → " " " " " " " " .
(b)
(c)

Tuple Function In Python

(a) len() → to return No of element present in the tuple.
(b) count() → to return No. of occurrences of given element in the tuple.
(c) index() → to return the index of first occurrence of the given element.
(d) sorted() → to sort element based on default natural sorting order.
(e) reversed() → to reversed element based on default Descending order.
(f) min() → to find minimum value of tuple.
(g) max() → to find maximum value of tuple.
(h) cmp() → to compare the element of both tuple
  ↳ if both tuple are equal then return 0.
  ↳ If first tuple is less than second tuple then it returns −1.
  ↳ If first tuple is greater than second tuple then it returns +1.

Set Function in Python

(a) add() → Add item x to the set.
(b) len() → return No. of element Present in set.
(c) copy() → return copy of set.
(d) pop() → It removes and return some random element from the set.
(e) remove() → It remove specified element from the set.
(f) discard() → It remove the specified element from the set.
(g) clear() → to remove all element from the set.
(h) union() → this function to return all element present in both sets.
(i) Intersection() → to return common element present in both x & y / sets.
(j) difference() → to return element present in x but not in y.
(k) isdisjoint → return True if the set has No element in common with other.
(l) issubset → test whether every element in the set is in other.
(m) issuperset → test whether every element in other is in the set.

JANUARY
22
'24
04th Week
022-344
MONDAY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JAN 2024

Cummulative Sum of each element in list / Prefix Sum

08 [1,2,3,4,5] → [1,3,6,10,15]

@ def fun(l):
09 l1 = []
S = 0
10 for i in l:
S = S+i
11 l1.append(S)
return l1
12

Sum of 1st, last and 2nd, second last and so on --

01 [1,4,2,7,3,5] → 6 7 9
def fun(l):
02 lindex = 0
rindex = len(l) - 1
03 while lindex <= rindex:
print(l[lindex] + l[rindex], end = ' ')
04 lindex = lindex + 1
rindex -= 1
05

Maximum difference b/w two elements such that larger element appears after the smaller element.

06 i/p — arr = {2,3,10,6,4,8,1}, o/p — 8, [2, 10]
07 ↗ arr size
def diff(arr, n):
max_dif = arr[1] - arr[0]
for i in range(n):
for j in range(i+1, n):
if (arr[j] - arr[i] > max_diff):
max_diff = arr[j] - arr[i]
return max_diff.

TC = O(n²)

Maximum Difference → such that $J > i$ → maximum Value of arr[J] - arr[i]

i/p — arr = [2, 3, 10, 6, 4, 8, 1]
o/p — 8

i/p — arr = [7, 9, 5, 6, 3, 2]
o/p — 2

M-1
```
def fn(arr, n):
    res = arr[1] - arr[0]
    for i in range(0, n-1):
        for J in range(i+1, n):
            res = max(res, arr[J] - arr[i])
    return res
```

M-2
```
def fun(arr, n):
    res = arr[1] - arr[0]
    minval = arr[0]
    for i in range(1, n):
        res = max(res, arr[i] - minval)
        minval = min(arr[i] - minval)
    return res
```

M-3
```
def fun(arr, n):
    minE = arr[0]
    maxE = arr[0]
    for i in range(1, n):
        minE = min(minE, arr[i])
        maxE = max(maxE, arr[i])
    return (maxE - minE)
```

M-4
```
def fun(arr):
    n = len(arr)
    suffix = [0]*n
    suffix[n-1] = arr[n-1]
    for i in range(n-2, -1, -1):
        suffix[i] = max(suffix[i+1], arr[i])
    ans = float('-inf')
    for i in range(n-1):
        ans = max(ans, suffix[i+1] - arr[i])
    return ans
```

JANUARY

24

'24

04th Week

024-342

WEDNESDAY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Equilibrium Point → an index such that sum of element at lower indexes is equal to the sum of element at higher indexes

i/p — l = [3, 4, 8, -9, 20, 6]

o/p — True

M-1

```
def fun(arr):
    n = len(arr)
    for i in range(n):
        rs, ls = 0, 0
        for j in range(i):
            ls = ls + arr[j]
        for k in range(i+1, n):
            rs = rs + arr[k]
        if (ls == rs):
            return True
    return False
```

O(n^2)

i = 0 : ls = 0 , rs = 11
i = 1 : ls = 3 , rs = 15
i = 2 : ls = 7 , rs = 7

→ A Point is called equilibrium point if sum of element before it and sum of element after it is same.

M-2

```
def epoint(arr):
    rs = sum(arr)
    ls = 0
    for i in range(len(arr)):
        rs = rs - arr[i]
        if (ls == rs):
            return True
        ls = ls + arr[i]
    return False
```

O(n)

The idea is to get the total sum of the array first. Then iterate through the array and keep updating the left sum which is initialized as zero. In the loop, we can get the right sum by subtracting the element one by one.

Minimum size Subarray Sum :-

```
def Subarray(target, nums):
    s, cs = 0, 0
    min_length = 0 || float('inf')
    for i in range(len(nums)):
        cs = cs + nums[i]
        while (cs >= target):
            min_length = min(min_length, i - s + 1)
            cs -= nums[s]
```

→ return 0 if min_length = float('inf') else min_length.

Prefix Sum Technique

Given an array arr[] of size n, its Prefix Sum array is another array prefixSum[] of the same size, such that the value of prefixSum[i] is arr[0] + arr[1] + arr[2] --- arr[i]

eg i/p - arr = [10, 20, 10, 5, 15] → o/p — [10, 30, 40, 45, 60]

prefixSum[0] = 10,
prefixSum[1] = prefixSum[0] + arr[1] = 30
prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.

M-1

```
def sum(l,r):
    res = 0
    for i in range(l,r+1):
        res += arr[i]
    return res.
```

TC - O(n)

M-2

Idea for getSum() in O(1) Time —

① Precompute Prefix Sum array

psum = [2, 10, 13, 22, 28, 33, 37]

$$psum[i] = \sum_{0}^{i} arr[i]$$

② $$getSum(l,r) = \begin{cases} psum[r], & \text{if } l == 0 \\ psum[r] - psum[l-1], & \text{otherwise} \end{cases}$$

getSum(0,2) = psum[2] = 13
getSum(1,3) = psum[3] - psum[0] = 22 - 2 = 20
getSum(2,6) = psum[6] - psum[1] = 37 - 10 = 27

JANUARY
26 '24
FRIDAY
04th Week
026-340

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JAN 2024

Preprocessing

```
n = len(arr)
psum = [None]*n
psum[0] = arr[0]
for i in range(1,n):
    psum[i] = psum[i-1] + arr[i]
```

O(1)

```
def getSum(l, r):
    if l == 0:
        return psum[r]
    else:
        return psum[r] - psum[l-1]
```

Array into 2 equal Sum Subarray :—

arr = [3, 4, -2, 5, 8, 20, -10, 8]
(18) (18)

```
def equal(arr):
    total_sum = sum(arr)
    if total_sum % 2 != 0:
        return False
```

} Sum odd होगा तो, divide नहीं कर सकते।

```
    left_sum = 0
    target_sum = total_sum // 2
```

→ 36//2 = 18

```
    for i in arr:
        left_sum = left_sum + i
        if left_sum == target_sum:
            return True
return False.
```

// 2nd 18 == 18

↙ else Not found.

Stock buy & Sell

The cost of a stock on each day is given in an array.
Find the maximum profit that you can make by buying and selling on those days.
If the given array of prices is sorted in decreasing order, then profit can not be earned at all.

B S B S
i/p = [1, 5, 3, 8, 12] , i/p = [30, 20, 10].
o/p = 13 o/p = 0

M-1 → arr

```
def Profit(Price, start, end):
    if (end <= start):
        return 0
    Profit = 0
    for i in range(start, end-1):
        for J in range(i+1, end+1):
            if (Price[J] > Price[i]):
                curr_pro = price[J] - price[i] +
                    Profit(price, start, i-1) +
                    Profit(price, J+1, end)
                Profit = max(profit, curr_pro)
    return Profit.
```

TC = O(n²)
SC = O(1)

M-2

```
def Profit(prices, days):
    Profit = 0
    for i in range(1, days):
        if prices[i] > Prices[i-1]:
            Profit += prices[i] - prices[i-1]
    return profit.
```

TC = O(n)
SC = O(1)

Approach - Pointers

```
def maxProfit(prices):
    left = 0
    Profit = 0
    for right in range(1, len(prices)):
        if prices[right] < prices[left]:
            left = right
        else: Profit = max(Profit, Prices[right] - Prices[left])
    return Profit.
```

JANUARY
28
SUNDAY
'24
05th Week
028-338

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Maximum appearing elements in Range:-

i/p: left = [1,2,5,15] , right = [5,8,7,18] ; o/p = 5

[1,2,3,4,(5)] , [2,3,4,(5),6,7,8] , [(5),6,7] , [15,16,17,18]

M-1

```
def maxA(left, right):
    freq = [0]*100
    for i in range(len(left)):
        for J in range(left[i], right[i]+1):
            freq[i] += 1
    return freq.index(max(freq))
```

TC = O(n × max), n = 100

= (5) ✓

M-2

```
def maxA(left, right):
    freq = [0]*101
    for i in range(len(left)):
        freq[left[i]] += 1
        freq[right[i]] -= 1
    for i in range(1,100):
        freq[i] = freq[i] + freq[i-1]
    return freq.index(max(freq))
```

TC = O(n + max)

Majority Elements :-

Find the majority element in array. A majority element in an array of size n is an element that appears more than n/2 times.

eg i/p: [3,3,4,2,4,4,2,4,4]

o/p = 4

The frequency of 4 is 5 which is greater than the half of the size of array size.

eg i/p: [3,3,4,2,4,4,2,4]

o/p = No elements

```
def majority(nums):
    e = 0
    cnt = 0
    for i in range(len(nums)):
        if cnt == 0:
            e = nums[i]
        if e == nums[i]:
            cnt += 1
        else:
            cnt -= 1
    return e.
```


M-1

```
def Maj(arr,n):
    maxC = 0
    index = -1
    for i in range(n):
        count = 0
        for J in range(n):
            if(arr[i] == arr[J]):
                count += 1
        if (count > maxC):
            maxC = count
            index = i
    if (max count > n//2):
        print(arr[index])
    else:
        Print('NO majority')
```

TC = $O(n^2)$

M-2

```
def maj(arr):
    arr.sort()
    return arr[len(arr)//2]
```

TC = $O(n \log n)$

M-3

```
def Maj(arr,n):
    for i in range(n):
        count = 1
        for J in range(i+1,n):
            if arr[i] == arr[J]:
                count += 1
        if cant > n/2:
            return i
    return -1
```

$O(n^2)$

Optimized way for majority element

Moore's Voting Algorithm:- [TC => O(n)]

- The first step gives the element that may be the majority element in the array. If there is a majority element in an array, then this step will definitely return majority element, otherwise, it will return candidate for majority element.
- check if the element obtained from the above step is the majority element. This step is necessary as there might be no majority element.

```
def findCandidate(A):
    maj_index = 0
    count = 1
    for i in range(len(A)):
        if A[maj_index] == A[i]:
            count += 1
        else: count -= 1
        if count == 0:
            maj_index = i
            count = 1
    return A[maj_index]
```

TC = O(n)

```
def isMaj(A, cand):
    count = 0
    for i in range(len(A)):
        if A[i] == cand: count += 1
    if count > len(A)/2:
        return True
    else: return False

def maj(A): cand = findCandidate(A)
    if (isMaj(A,cand) == True:
        Print(cand)
```

Where to use two pointers. —

- Works well when array is sorted.
- Search Problem like triplet, duplicates.
- Can Reduce Tc from O(n²) to O(n)

JANUARY 30 TUESDAY '24
05th Week 030-336

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JAN 2024

1. Segregate 0 & 1.

i/p = [1,0,1,0,1,0] , o/p = [0,0,0,1,1,1]

M1 use Insertion(or) Bubble sort → O(n²)

M2 use l.sort() → O(nlogn)

M3

```
n = [1,0,1,0,1,0]
count0 = 0 , count1 = 0
for i in n:
    if n[i] == 0:
        count0 += 1
    else:
        count1 += 1
for i in range(count0):
    n[i] = 0
for i in range(count1):
    n[i] = 1
print(n)
```

O(n)

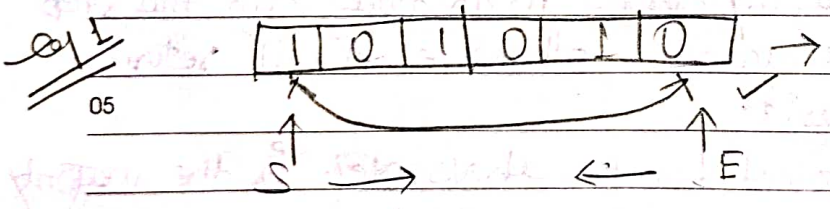
Use Two Pointer

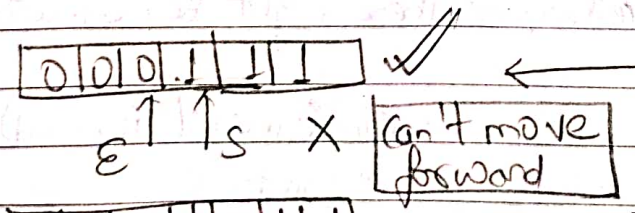
[1,0,1,0,1,0]

(Start pointer) → ← (End pointer)

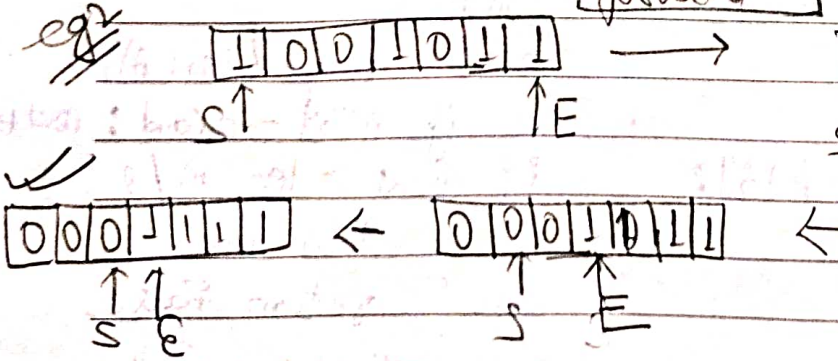
Start — End को cross करेगा तो both pointer को move नहीं करेंगे।

eg1





eg2



Obedience a alone gives the right to command

→ Two Pointer Pattern is used to efficiently traverse array, linked list — TC = O(n)

→ It is used to solve Problem that involves searching for a pair of element in sequence

Using Two pointer:

```
n= [1,0,1,0,0,1,0]
s=0
e= len(n)-1
while (s<e):
    if n[s]==0:
        s+=1
    else:
        if n[e]==0:
            n[s],n[e]=n[e],n[s]
            s+=1
            e-=1
        else:
            e-=1
print(n)
```

this Problem is solved

swaping →

Two Sum.

Sum of two NO = target.

n= [2|7|11|15|29]

Target = 22

```
for i in range(len(n)):
    for j in range(i+1, len(n)):
        if n[j]==target-n[i]:
            return i, j.
```

TC=O(n²)

If Array is sorted

Must →

two pointer Approach

```
n= [1,2,4,6,8,25]
t= 10
s=0
e= len(n)-1
while (s<e):
    if n[s]+n[e]==t:
        print(n[s],n[e])
    elif n[s]+n[e]<t:
        s=s+1
    else:
        e=e-1.
```

TC=O(n)
SC=O(1)

for leet code

```
return [s+1, e+1]
```

target > ans
right ↑ shift

target < ans
left ↑ shift.

```
while (l<r):
    sums=n[l]+n[r]
    if sums<target:
        l+=1
    elif sums>target:
        r-=1
    else:
        return [l+1, r+1]
return []
```

FEBRUARY
01
THURSDAY
'24
05th Week
032-334

S↑ E ↓ fixed < Diff

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

Pair with Given Difference

n = [2, 3, 5, 10, 50, 80] , Diff = 45

S=0 E=1

```
while (E < len(n)):
    if (n[S] - n[E] == target):
        return 1.
    elif (n[S] - n[E] < target):
        (E = E+1)
    else:
        (S = S+1)
return 0
```

जब value छोटा आया तो end को बढ़ाना है

O(n)

Fact

n[start] ⊛ n[end] → any operation

Start को → Increase

Step पर → Decrease

Overall Answer: Incr / Desr ; Inc / Decr

यहाँ increase हो रहा तो तो यहाँ Decrease करना पड़ेगा

यहाँ Decrease हो रहा तो तो यहाँ increase करना पड़ेगा

Prefix & Suffix

arr = | 6 | 4 | 5 | -3 | 2 | 8 |

| 6 | 10 | 15 | 12 | 14 | 22 |

→ Prefix sum

| 22 | 16 | 12 | 7 | 10 | 8 |

→ Suffix sum

Where NOT to use two pointers.
- Unsorted array.
- Complex Relationship.
- Graph / Tree.

```
def prefix(arr):
    pre_sum = [0]
    for i in arr:
        pre_sum.append(pre_sum[-1] + i)
    return pre_sum.
```

Prefix

```
def Prefix(arr):
    for i in range(1, len(arr)):
        arr[i] = arr[i] + arr[i-1]
    return arr.
```

Suffix

```
def suffix(arr):
    for i in range(len(arr) -2, -1, -1)):
        arr[i] = arr[i] + arr[i+1]
    return arr.
```

Largest Sum Contiguous Subarray :⟶ 'Kadane's Algo'

0	1	2	3	4	5	6	7
3	4	-5	8	-12	7	6	-2

— 13

→ Kadane's Algorithm is a simple yet powerful method to find the largest sum of contiguous subarray.

→ It uses two main variable to keep track of the current subarray sum and the maximum sum found so far, iterating through the array in linear time to produce the result.

→ The subarray with negative sum is discarded (by assigning max_ending_here = 0 in code)

→ We carry subarray till it gives positive sum.

→ It uses dynamic programming approach. It involves traversing the array and maintaining a running sum of current subarray.

FEBRUARY
03
'24
05th Week
034-332
SATURDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

```
def largest(arr):
    max_so_far = float('-inf')
    max_ending_here = 0
    for i in arr:
        max_ending_here += i
        if max_ending_here < i:
            max_ending_here = i
        if max_so_far < max_ending_here:
            max_so_far = max_ending_here.
    return max_so_far.
```

TC = O(n)
SC = O(1)

Maximum Subarray Sum :—

~~Given an array of N integers the task is to find the maximum difference b/w any two element~~

Given an array of integers and a number k, find maximum sum of a sub array of size k.

i/p — arr = {100, 200, 300, 400} , K = 2
o/p = 700

i/p — arr = {1, 4, 2, 10, 23, 3, 1, 0, 20} , K = 4
op = 39 — {4, 2, 10, 23}

M-1

```
def maxSum(arr, n):
    res = arr[0]
    for i in range(0, n):
        curr = 0
        for j in range(i, n):
            curr = curr + arr[j]
            res = max(res, current)
    return res
```

O(n²)

M-2

```
def maxSum(arr, n):
    res = arr[0]
    maxEnding = arr[0]
    for i in range(1, n):
        maxEnding = max(maxEnding + arr[i], arr[i])
        res = max(maxEnding, res)
    return res.
```

O(n)

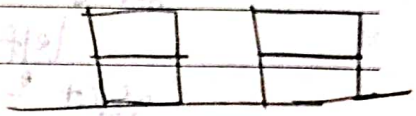
Prevention is better than cure

Trapping Rain water :—

→ Given an array of N non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

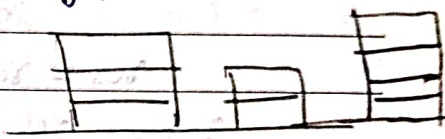
i/p — arr = {2,0,2} , o/p — 2

The structure is like →



we can trap 2 unit of water in the middle gap.

i/p — arr = {3,0,2,0,4} , o/p — 7.



we can trap 3 unit of water b/w 3 and 2. 1 unit on top of bar 2 and 3 units b/w 2 and 4.

→ An element of the array can store water if there are higher bars on the left and the right.

→ The amount of water to be stored in every position can be found by finding the height of bars on the left and right sides.

→ The total amount of water stored is the summation of the water stored in each index.

Brute force approach —

- Traverse the array from start to end.
  - for every element
    - traverse the array from start to that index and find the maximum height (a) and
    - traverse the array from the current index to the end, and find the maximum height (b).
  - The amount of water that will be stored in this column is min(a,b) − array[i], and this value to the total amount of water stored.
  - print the total amount of water stored.

FEBRUARY
05 '24
06th Week
036-330
MONDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

M-1

```
def water(arr,n):
    res = 0
    for i in range(1,n-1):
        left = arr[i]
        for J in range(i):
            left = max(left, arr[J])
        right = arr[i]
        for J in range(i+1,n):
            right = max(right, arr[J])
        res = res + (min(left,right) - arr[i])
    return res.
```

TC = O(n^2)
SC = O(1)

↳ for Every element we can Precalculate and store the highest bar on the left and on the right (say stored in array left[] and right[])
→ Then iterate the array and use the Pre calculated value to find the amount of water stored in this index
↳ which is same as (min(left[i], right[i]) - arr[i])

M-2

```
def water(arr,n):
    left = [0]*n
    right = [0]*n
    water = 0
    left[0] = arr[0]
    for i in range(1,n):
        left[i] = max(left[i-1], arr[i])
    right[n-1] = arr[n-1]
    for i in range(n-2,-1,-1):
        right[i] = max(right[i+1], arr[i])
    for i in range(0,n):
        water += min(left[i], right[i]) - arr[i]
    return water.
```

TC = O(n)
SC = O(n)

```
from datetime import date
print(date.today())
```

3 Sum

1	4	45	6	10	8

n = 13

Saying is there any three element whose sum is equal to 13.

```
def 3Sum(arr, n):
    for i in range(len(arr)-3):
        for J in range(i+1, len(arr)-2):
            for K in range(J+1, len(arr)-1):
                if (arr[i] + arr[J] + arr[K] == n):
                    return True
    return False.
```

TC = O(n³)

Reduce TC to O(n²) using sorting and two pointer.

```
def 3sum(arr, n):
    arr.sort()          → O(nlogn)
    for i in range(len(arr) - 3):
        ans = x = arr[i]
        s = i+1
        e = len(arr) - 1
        while (s < e):
            if (arr[s] + arr[e] == ans):
                return 1
            elif arr[s] + arr[e] > ans:
                end --
            else:
                s ++
```

1	4	6	8	10	45

पहली element को fix कर दिया → बाकी को इसके से end तक search करो

TC = O(n²)

two pointer approach.

यहाँ check कर रहे कि क्या दो element का sum उसके बराबर आ सकता है।

4-Sum भी ऐसा ही होगा →

→ res = [] , res.append([nums[i], nums[s], nums[e]]) for printing arrays.

→ Where to use Sliding window — TC: O(n)

FEBRUARY 07 '24 WEDNESDAY

06th Week 038-328

- Fixed window size
- Variable window size
- NON-Negative data — Work & best

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

→ Sliding window in Array ←

- It is suitable for processing Sequential data.
- We use dynamic window → slides through the array (or) string.

"Zero-Sum Subarray" — arr = [4, 2, -3, 1, 6], True ✓

- It is a contiguous portion of an array where the sum of its elements equals zero.
- If you take a subarray of element from the array and these sum is zero, that subarray is called zero-sum subarray.

M-1 Brute force

```
def zSub(arr):
    for i in range(len(arr)):
        sum = 0
        for j in range(i, arr):
            sum = sum + arr[j]
            if sum == 0:
                return True
    return False
```

TC: O(n²)
SC: O(1)

M-2 Prefix Sum with hash map.

```
def zSub(arr):
    seen_sum = set()
    prefix_sum = 0
    for i in arr:
        prefix_sum += i
        if prefix_sum == 0 or
           prefix_sum in seen_sum:
            return True
        seen_sum.add(prefix_sum)
    return False
```

TC: O(n)
SC: O(n)

Prefix sum : keeps track of the cummulative sum of the array element as we iterate through the array.

Seen-sum : Stores the prefix-sum that have been encountered. If the prefix sum occurs again, it means that the sum of subarray b/w the repeated indice is 0.

If the Prefix sum becomes 0 (or) repeat, it indicates that there is a zero sum subarray.

→ Where not to use sliding window

- Negative Number involving not works. ☹
- NON Contigous data ↑ Fail to work
  ↳ Works when Problem involves Contiguous Subarray.

eg

4	2	1	-3	1	6

Prefix Sum = 4 6 3 4 10

Rohit Negi explanation.

K → Sum = 0 ✓

M3 Kadane's Algo also used to find the maximum sum subarray but it can be modified to detect zero sum subarray as well. ✓ by tracking the sum of elements and checking for zero-sum, you can handle this problem in linear time.

```
def zSub(arr):                    TC = O(n)
    curr_sum = 0                  SC = O(1)
    for i in arr:
        curr_sum += i
        if curr_sum == 0:
            return True
    return False.
```

My TCS Question ☹

Sub array Sum equals K

We are given an array of integers and a target sum K. the task is to find how many continuous subarray have sum equal K.

K = 7, nums = [3, 4, -7, 1, 3, 3, 1, -4]

M1 Brute force approach TC = O(n²)

```
def Subarray(nums, K):
    count = 0                          ← for printing total possible ways.
    for i in range(len(nums)):
        current_sum = 0
        for j in range(i, len(nums)):
            current_sum = current_sum + nums[j]
            if current_sum == K: count = count + 1
    return count
```

This solution will works for negative integers as well.

People who live in glass houses should not throw stones

FEBRUARY 09 '24 06th Week 040-326

FRIDAY

If we have to print all subarray then instead of line 2, 8, 9 we have to write simply at line (5) Print(nums[i:j+1])

FEB 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

M-2 → Prefix Sum + hashmap approach.

```
def Subarray(nums, K):
    count = 0                                   TC = O(n)
    current_sum = 0                             SC = O(n)
    prefix_sum_map = {0:1}  # To handle case where subarray start from index 0.
    for i in nums:
        current_sum = current_sum + i
        if current_sum - K in prefix_sum_map:
            count = count + prefix_sum_map[current_sum - K]
        if current_sum in prefix_sum_map:
            prefix_sum_map[current_sum] += 1
        else:
            prefix_sum_map[current_sum] = 1
    return count
```

Check if (current_sum - K) exist in map

Update the prefix_sum map with current_sum

Sliding window Approach

In M-1 — If we have to print all subarray then just at line (7) use Print(nums[i:j+1])
and if we have to return all subarray at once then l = []
and l.append(nums[i:j+1]) → last element ✓

→ Don't use Print(fun(nums, K)) while calling because it will print NONE so just call fun(nums, K) ✓✓ ☺

Subarray sum divisible by K → change if (cs % k) == 0:

Subarray sum multiply by K → change in line 4, 6 & 7

(4) Product = 1
(6) product *= nums[j], (7) if product < K: cnt += 1 else break
return cnt

Find maximum Sum of K consecutive Elements.

i/p = [1, 8, 30, -5, 20, 7] , K=3 ; i/p = [5, -10, 6, 90, 3] , K=2

o/p = 45 ; o/p = 96

Naive Method

```
def maxsum(arr,k):
    n = len(arr)
    res, i = 0,0
    while (i+k-1<n):
        curr=0
        for j in range(k):
            curr += arr[i+j]
        res = max(curr, res)
        i += 1
    return res
```

O(n*k)

Sliding window

```
def slide(arr,k):
    curr=0
    for i in range(k):
        curr += arr[i]
    res = curr
    for i in range(k, len(arr)):
        curr = curr + arr[i] - arr[i-k]
        res = max(res, curr)
    return res
```

O(n) O(1)

Initially : curr = 1+8+30-5 = 34 , res = 34 , [1|8|30|-5|20|7] , K=4

1st slide : curr = 34+20-1 = 53 , res = 53 [1|8|30|-5|20|7]

Adding last of current window ↑ ↑ Removing first of previous window

2nd slide : curr = 57+7-8 = 52 , [1|8|30|-5|20|7]

```
def Sumn(arr,sum):
    s, curr = 0,0
    for i in range(len(arr)):
        curr += arr[i]
        while (curr > sum):
            curr = curr - arr[s]
            s += 1
        if (curr == sum):
            return True
    return False
```

O(n) O(1)

[4, 8, 12, 5] , sum = 17

s = 0, e = 0, curr = 4

e = 1, curr = 12

e = 2, curr = 24

s = 1, e = 2, curr = 20

s = 2, e = 2, curr = 12

s = 2, e = 3, curr = 17

return True

→ while curr is smaller than sum, extend the window by increasing e.

← while curr is greater than sum, shrink the window by increasing s.

FEBRUARY
11
'24
07th Week
042-324
SUNDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

FEB 2024

Longest Even odd Sub array.

→ contiguous elements

Given an array of integers the task is to find the length of the longest alternating Even odd Sub array Present in array.

i/p — [1,2,3,4,5,7,9] o/p-5

It has alternating even and odd element.

M-1 O(n²)

```
def EOS(arr,n):
    ans = 1
    for i in range(n):
        cnt = 1
        for J in range(i+1,n):
            if ((arr[J-1]%2==0 and arr[J]%2!=0) or
                (arr[J-1]%2!=0 and arr[J]%2==0)):
                cnt = cnt + 1
            else:
                break
        ans = max(ans, cnt)
    if (ans==1):
        return 0
    return ans
```

O(n)

```
def EOS(arr,n):
    longest = 1
    cnt = 1
    for i in range(n-1):
        if((arr[i]+arr[i+1])%2==1):
            cnt = cnt + 1
        else:
            longest = max(longest, cnt)
            cnt = 1
    if (longest==1):
        return 0
    return (max(cnt, longest))
```

Maximum circular Subarray Sum:

Given a circular array of size n, find the maximum Subarray Sum of the non-empty Subarray.

eg → l = [8,-8,9,-9,10,11,12]
o/p = 22

```
def circle(arr,n):
    res = arr[0]
    for i in range(0,n):
        curr_max = arr[i], curr_sum = arr[i]
        for J in range(1,n):
            index = (i+J)%n
            curr_sum = curr_sum + arr[index]
            curr_max = max(curr_max, curr_sum)
            res = max(res, curr_max)
    return res
```