Arrays - 1

Lists? Python

- collection of elements. (C++, Java: Homogeneous → Python: Heterogeneous.)
- Data structure → data → access
  → operations/method.

Initialize $l_1 = [1, 2, 5, -2, 4, 6]$

$l_2 = [\text{"akhil"}, 1, 2, 5, -2, 1.23]$

Declare an empty list

```
name_of_list = []
```

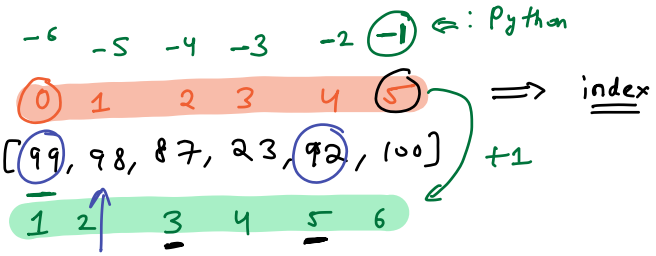
```
a=list()    ## declaration
```

Defining

Initial values

$a = [1, 2, 5, -3]$

For eg. ⇒



* First index = 0

Size = N

In this eg N = 6

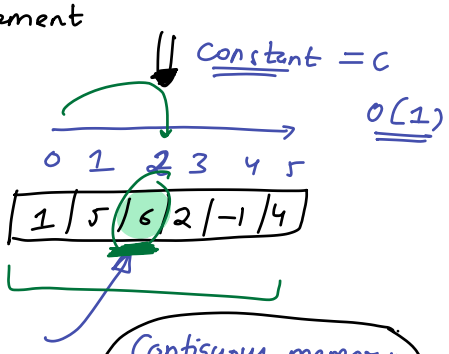
* Last Index = N-1

Sum of 1st & 5th elements = marks[0] + marks[4]

i^{th} elt ⇒ $(i-1)^{th}$ index.

NOTE: T.C. to access i^{th} index element

arr[i]



Index

i = → arr → arr[i]

arr[2]

Contiguous memory locations

Q) Print all the array elements.

```
          0  1  2  3  4  5  6  7        N = 8
arr = [1, 2, -3, 4, 6, 2, 7, 5]
```

Output

1
2
-3
4
6
2
7
5

=> N = len(arr) # range(N) => start = 0 (Default)

```
for i in range(0, N):
    print(arr[i])        # O(1)
```

No. of iterations = N - 0 = N

[0, N)

Time taken in each iteration = c (Constant)

Total time = c + c + c + c + ... + c = cN

(c added N times)

Big-O Notation = O(N)

```
for my_ele in my_list:
    print(my_ele)
```

Index

Q1

Given N elements in a list, count the no. of elements having atleast 1 element greater than itself.

i) 2, 5, 1, 4, 8, 0, 8, 1, 3, 8

res = 10 − 3 = 7

0
7
6
3

ii) −3, −2, 6, 8, 4, 8, 5

N = 7

res = 7 − 2 = 5

Observation: (1) Maximum element will not have any element greater than itself. (cnt)

(2) For all non-maximum elements, atleast 1 elt (max of array) is greater than it.

res = N − cnt of maximum

---

iii) arr = [1, 1, 1, 1, 1, 1]

maxi = 1

cnt of maxi = N

res = N − N = 0

Pseudo Code

1 — Iterate & Get max → O(N)

2 — Iterate & Cnt maxi → O(N)

3 → (N − cnt) as ans. → O(1)

2N + 1 = O(N)

Linear

1*10^9 → 1 sec

2*10^9 → 2 sec

10 min.

①

```
1  maxi = arr[0]   # -2 3
2  N = len(arri)
3  for i in range(1, N):
4      if arr[i] > maxi:
5          maxi = arr[i]
#  maxi
```

arr ← [-2, -8, 3, 2]

0 1 2 3

-2 -2 3 3

0 1 2 3 4 5 6 7 8 9 10 11

1, 3, 2, 4, 2, 5, 1, 6, 2, 7, 1, 2

1. no. of iterations

= N-1

i	0	1	2	3	4	5	6	7	8	9	10	11
arr[i]	1	3	2	4	2	5	1	6	2	7	1	2
maxi	1	3	3	4	4	5	5	6	6	7	7	7

2. Ignore any constants

TC: O(N)

---

②

```
cnt = 0
for i in range(0, n):
    if arr[i] == maxi:
        cnt += 1
```

TC: O(N)

③

```
return N - cnt
```

HW: Try to do ① & ② in the same loop.

Q2 Given N array elements, check if there exists a pair (i, j) such that

indices ⇐ (i, j)

Return True/False

arr[i] + arr[j] = K and (i) ≠ (j)

different indices.

```
def check_pair_sum(arr, K):
    —
    —
    —
    —
```

Eg

① [3, -2, 1, 4, 3, 6, 8] True

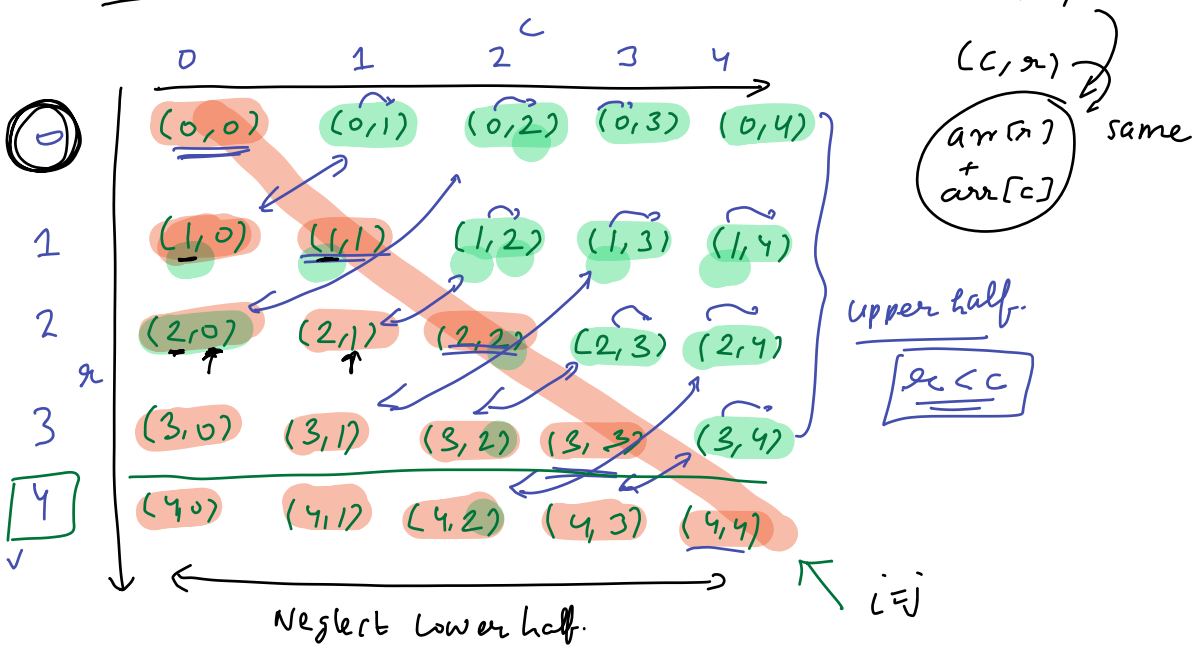
K = 10

② [6, 2, 3] K = 12 False

Brute Force

Idea: For every pair check if sum = K.

N = 5 (i, j) = (r, c)



Iterate on all the pairs:

```
arr = [6, 2, 3]
K = 12
```

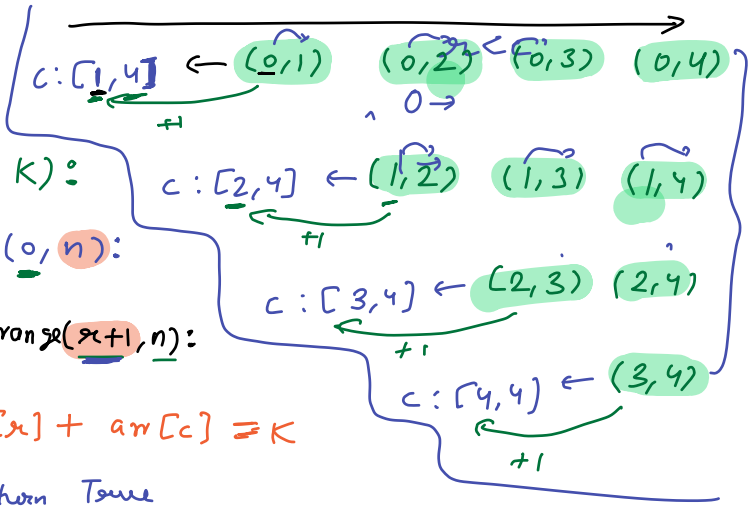
```
def check_pair_sum(arr, K):
    for r in range(0, n):
        for c in range(0, n):
            # (r, c)
            if r == c:  # edge cases
                continue
            if arr[r] + arr[c] == K:
                return True
    return False
```

r = 0 | c = 0 6 + 6 = 12

r	c	Total iterations
0	[0, N)	N
1	[0, N)	N
2	⋮	N
⋮	⋮	⋮
N-1	⋮	N
		N^2

T.C. $O(N^2)$

Iterate on Upper Half:




```
def check_pair_sum(arr, K):
    for r in range(0, n):
        for c in range(r+1, n):
            if arr[r] + arr[c] == K
                return True
    return False
```

r	c	Iterations
0	[1, N)	N-1
1	[2, N)	N-2
2	[3, N)	N-3
⋮	⋮	+
N-1	[N, N)	0

$$= 1+2+3+\dots+(N-1) \quad (1)$$

$$= \frac{(N-1)N}{2} \quad (2)$$

$$= \frac{N^2-N}{2} = \frac{1}{2}N^2 - \frac{1}{2}N$$

range(1,1) = []

TC: $O(N^2)$

Q3. Given a list, reverse this entire list.

in the same list

Constraint:

You have to reverse in the same input list.

In-place : O(1)

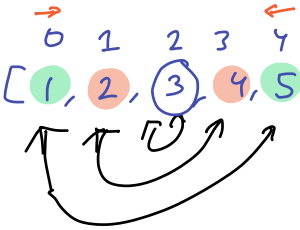
```
0  1  2  3  4  5  6  7
l = [-1, 4, 7, 6, -2, 7, 8, 10]
```

Output: [10, 8, 7, -2, 6, 7, 4, -1]

swap(i,j)

i, j = j, i

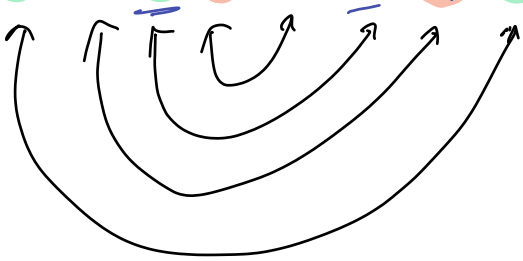
```
0  1  2  3  4
l2 = [1, 2, 3, 4, 5]
```



l2 = 5, 4, 3, 2, 1

i	j	
0	4	swap
1	3	swap
2	2	swap
3	1	

```
0  1  2  3  4  5  6  7
l = [-1, 4, 7, 6, -2, 7, 8, 10]
```



10, 8, 7, -2, 6, 7, 4, -1

i	j	
0	7 = N-1	swap
1	6	swap
2	5	swap
3	4	swap
4	3	

i > j => stopping condition

```
def reverse(arr):
    i = 0, j = len(arr) - 1
    while (i < j):          # (i != j) Fails:
        swap(arr[i], arr[j])   # arr[i], arr[j] = arr[j], arr[i]
        i += 1
        j -= 1
# arr is now reversed.
```

↑ ↑ ↑ ↑ ... (↑↑) ... ↑ ↑ ↑ ↑

N/2, N/2 + 1

Iterations ≅ N/2

Time = TC : O(N)

⇒ Space Complexity ⇒ not part of input.

Big-O Notation

⇒ 1→ How many extra variables

= 2 variables = constant. ⇒ O(1)

Precisely, Total Space = Input Space + Auxiliary/Extra Space

array of size N: integers = (4N) + 2*4

1 integers: 4 units. (bytes)

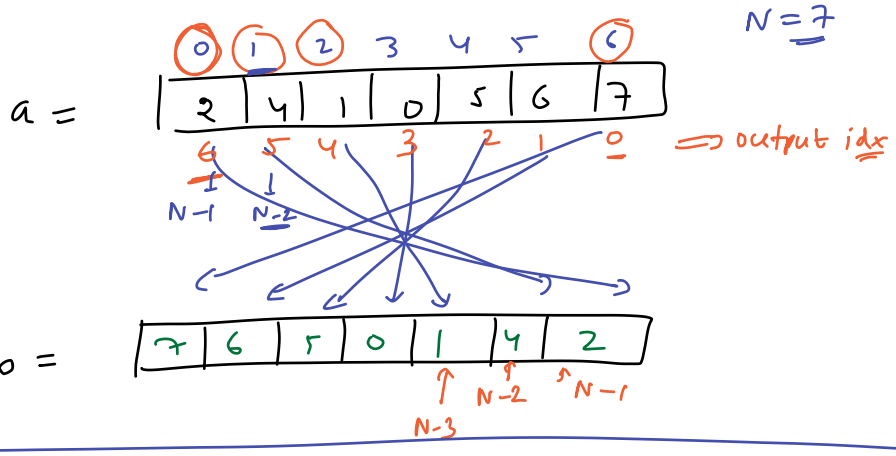
= 4N + 8

= O(N)

8 → O(1)

Space Complexity.

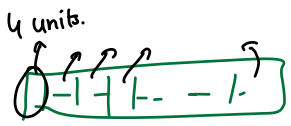
‼ Big-O.



```
def rev2(arr):
    i = 0
    N = len(arr)
    l2 = [0] * N
    for i in range(0, N):
        l2[i] = arr[N-i-1]
```

Try to do a dry run.

Space Complexity



N integers

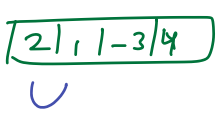
1 integer = 4 units.

Extra space = 4N

SC : O(N)

Quiz :

```
n = len(arr)        Input
sum = 0
for i in range(0, n):
    sum += arr[i]
```



2, 3, 0, 4

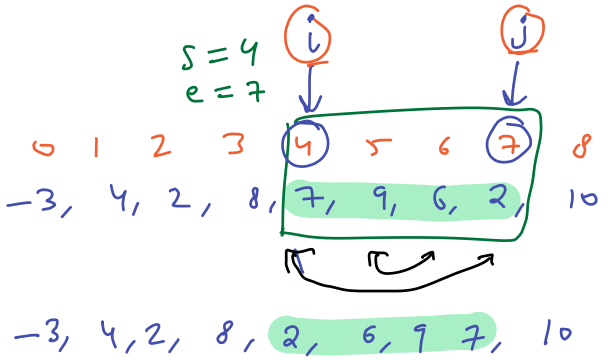
TC: Iterations = n

Bis-O: O(N)

SC: Extra variables? = 3 → 3 * c.

Bis-O = O(1)

Q4 Reverse part of the array. [s, e]



11.45- Doubts

TAs ≤ 30 min.

Slack. ⇒ 1 day

```
def reverse(arr, s, e):
    i = s, j = e
    while (i < j):
        swap(arr[i], arr[j])
        i += 1
        j -= 1
# arr is now reversed.
```

Same code as last problem works. With minor changes.

No. of iterations = $\frac{(e-s+1)}{2}$

No. of elements $N' = (e-s+1)$

$s = 0$, $e = N-1$

Worst case T.C. = $O(N)$

If we want to reverse the entire array, what is s & e?

$s = 0$, $e = n-1$ ⇒ $O(N)$

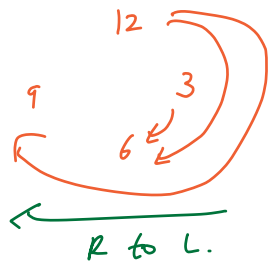
$e - s + 1 = n - 1 + 1 = n$

Q 5 Given an array of N elements, rotate the array in clockwise direction by K times.

i) Given. (K<N) ⟹ now.

ii) k can be anything → HW, discuss next class

Constraint: In-place SC: O(1)
(In the same array)



N=7

[3, 2, 1, 4, 6, 9, -8]

K=1

[-8, 3, 2, 1, 4, 6, 9]

1- Last element goes to 1st
2- Shift all other by 1 to right.

K=2

[9, -8, 3, 2, 1, 4, 6]

Brute Force: Repeat the rotation K times.

Try HW

1 rotation

[1, 2, 3]

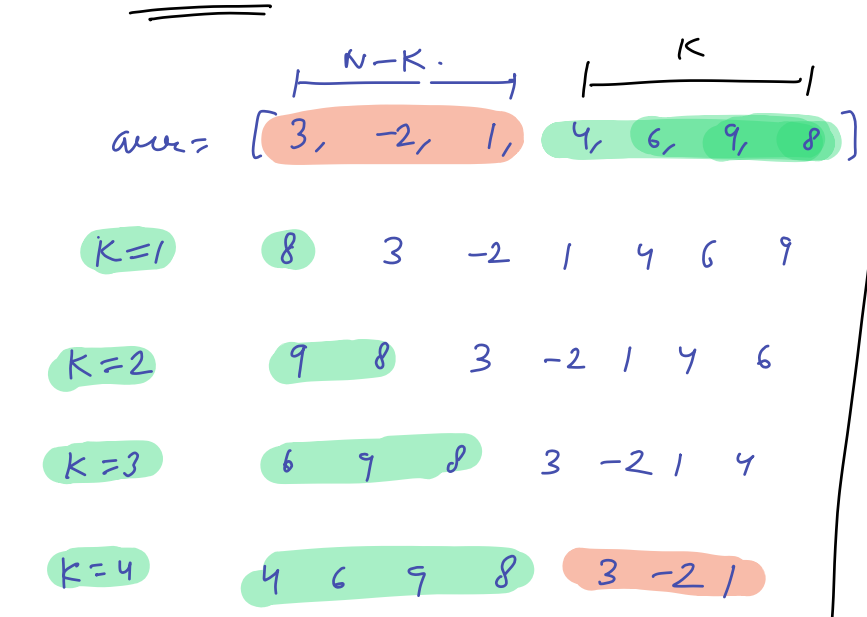
[1 0]

temp = arr[-1]
= 3

Shift 1st to 2nd last.

for i in range(___)

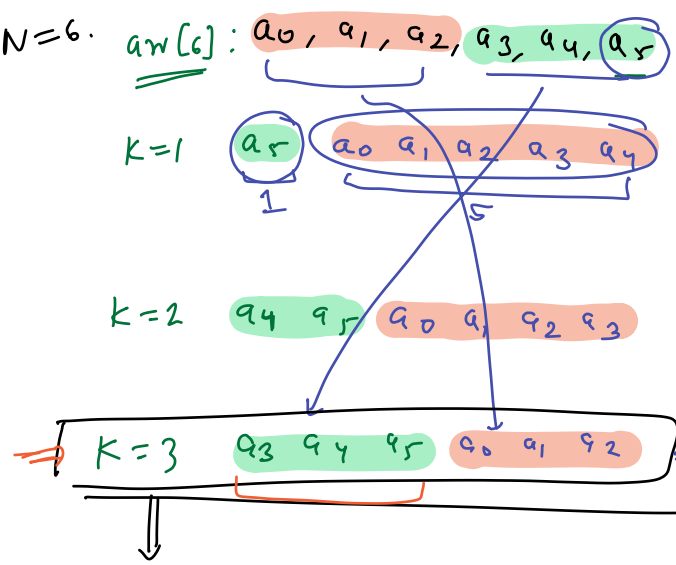
Observations



1 — Last K elements come to first K positions.

2 — First (N-K) elements come to last (N-K) positions.

N=6. arr[6] : $a_0, a_1, a_2, a_3, a_4, a_5$



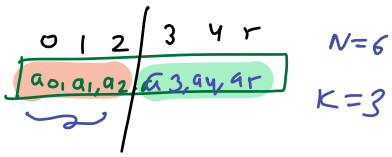
K	N-K
1	5
2	4
3	3

reverse output !

$a_2\ a_1\ a_0$ — reverse 1st K

$a_5\ a_4\ a_3$ — reverse Last N-K

wrt the original.



Iteration ✓ ① Reverse the first K elements

K ← rev(arr, 0, K-1) → a2, a1, a0, a3, a4, a5

✓ ② Reverse the last (N-K) elements

N-K ← rev(arr, K, N-1)

✓ ③ Reverse the entire array.

a3, a4, a5, a0, a1, a2

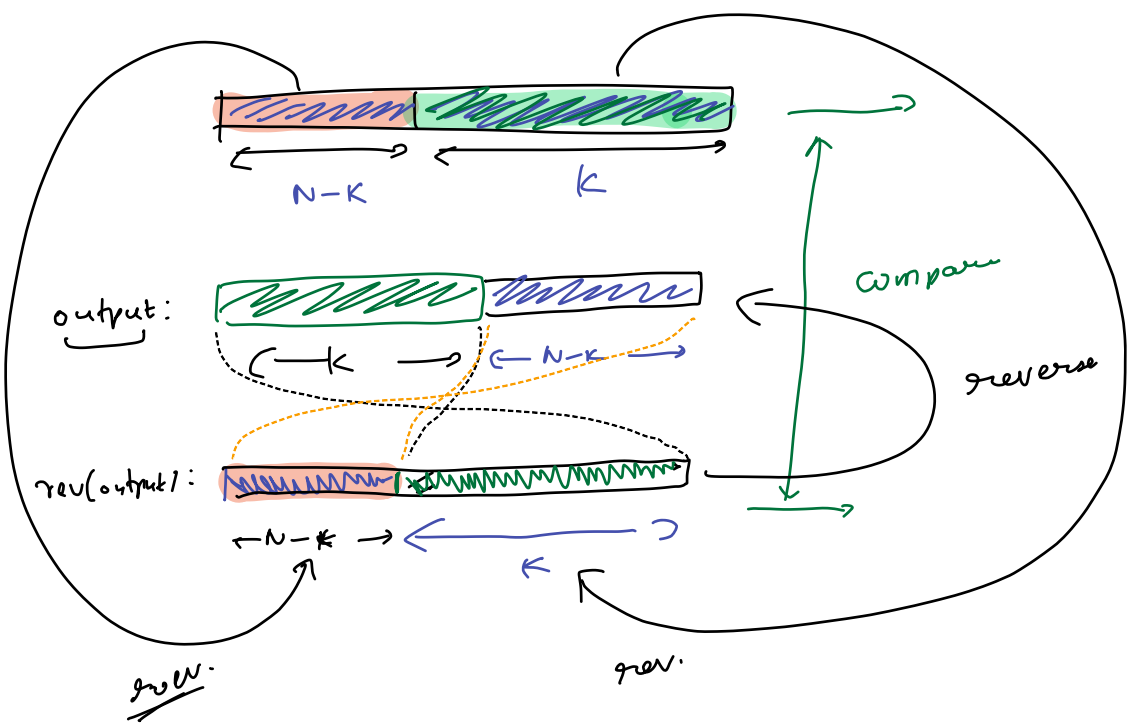
K / 2N ← rev(arr, 0, N-1)

rev(output) = rev(1st K) + rev(last N-K)

output = rev(. . .)

TC: O(N)

SC: O(1)



⇒

Doubts

TC, HW

message..

* Dictionary

arr = [1, 2, 3]

K = 5

① 3 1 2

② 2 3 1

③ 1 2 3

④ 3 1 2

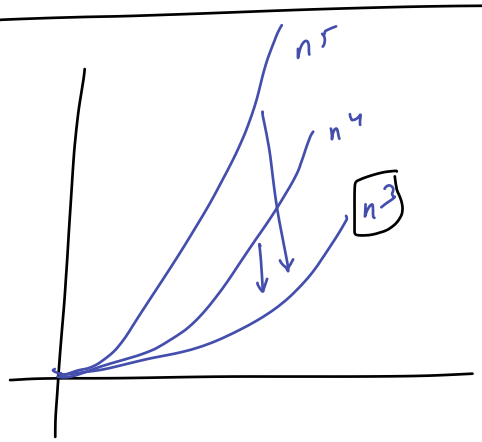
⑤ 2 3 1

```
for (i) in range(1, n):
    t = 5 + i
```

O(1) space.

1 variable.

All Classes → Optional Classes



$f(n) = O(n^3)$

$f(n) = O(n^4)$

$= O(n^5)$

$n^3 \leq c \cdot n^4 \quad c \geq 1, \; n_0 \geq 1$

```
i = N
while i > 0 :
    for j in range( 0, i ):
        # --
    i = i/2
```

i	j	Iterations
N	[0, N)	N - 0 = N
N/2	[0, N/2)	N/2 - 0 = N/2
N/4	[0, N/4)	N/4
⋮	⋮	⋮
1	[0, 1)	1
0		

$= O(N)$

$1 + 2 + 4 + \cdots + N$

$\simeq N.$

GP. $N + \frac{N}{2} + \frac{N}{4} + \cdots + 1 + \cdots$

$a = N$

$r = 1/2$

0.5
0.25
0.125
⋮

$S_\infty = \frac{a}{1-r} \quad |r| < 1$

$= \frac{N}{1 - \frac{1}{2}} = \frac{N}{1/2} = 2N$