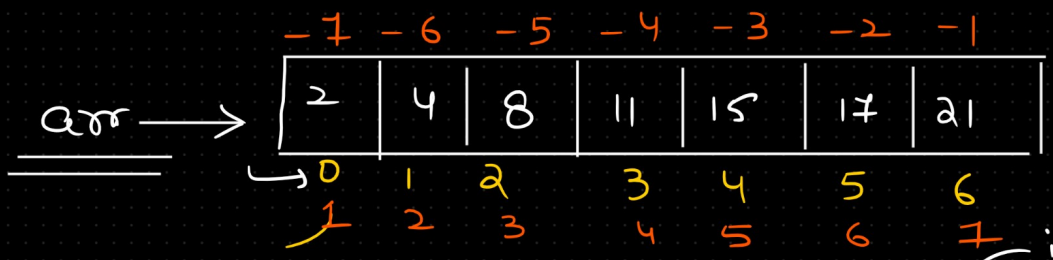
List



by default

arr[3] = 11

Random access

arr(-1) = 21

range(1, 8) → [1, n+1)

n = 7

[0, n)

O(1) → Constant time Complexity

```
for i in range(n):
    print(arr[i])
```

i = 0 →

i = 1 →

i = 2 →

|

i = 6 →

Sorting → Ascending or Descending order

Sorted manner

numbers = [2,7,11,15], target = 9

0 1 2 3

2, 7, 11, 15

2 + 7 = 9

[1, 2]

i = 0 to 3

J = i+1 to 3

i = 0 → J = 1, 2, 3

i = 1 → J = 2, 3

i = 2 → J = 3

Two Pointers

Sum = nums(low) + nums(high)

target = 9

0	1	2	3
2	7	11	15

low = 0

high = ~~3~~ 1

[1, 2]

(low+1, high+1)

low

high

1 2 3 4

if (sum == target(9)) → return (low+1, high+1)

13 > 9

17 9

elif sum > target :

high = high - 1

else :

low = low + 1

(-1, -1) → no pair is available

0 1 2 3 4 5 6

→ (7, 6, 5, 4, 3, 2, 1)

3 6

→ 5, 6, 7, 4, 3, 2, 1

→ 5, 6, 7, 1, 2, 3, 4

Reversed

X

(1st)

Problem 2

nums = [1,2,3,4,5,6,7], k = 3

0 1 2 3 4 5 6

0 k-1 → 1st 2

k n-1

0 1 2 3 4 5 6

7, 1, 2, 3, 4, 5, 6 — k=1

6, 7, 1, 2, 3, 4, 5 — k=2

5, 6, 7, 1, 2, 3, 4 — k=3

n = 7

k = k % n

= 3 % 7

= 3

Brute force approach

O(n*k)

```
for i in range(0, k):
    num = nums[-1]
    for j in range(n):
        nums[j], num = num, nums[j]
```

i = 0

num = 7

J = 0

1. Complete list reversal(0, n-1)
2. reversal(0, k-1)
3. reversal(k, n-1)

nums = [-1,-100,3,99], k = 2

nums → (99, 3, -100, -1) — ①

nums → (3, 99, -100, -1) — ②

nums → (3, 99, -1, -100) → output ③

k ≥ len(nums) → Logic not work

nums = [-1]

k = 2

0 ✓ → 1 — ✓

0 — 1

out of index error

common

remainder

$k = k \% n$

$= 2 \% 1$

$= 0$

$20 \% 3$

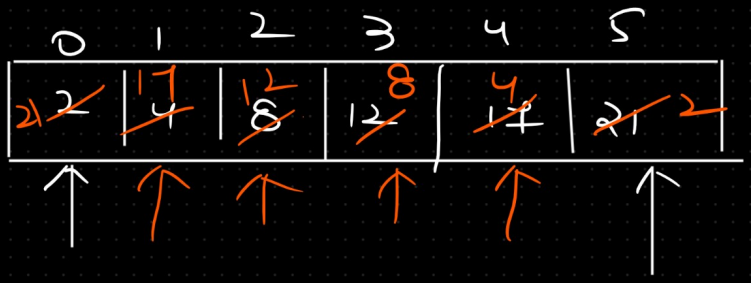
$\Rightarrow 2$

$1 \% 2$

$\Rightarrow 1$

$n > k$ → $k = k \% n$ → same

$n < k$ → $k = k \% n$ → avoid out of index



$O(n)$

no change

$k < n$

$k = k \% n$

$k = 3 \% 7$

$k = 3$

$k = 2$ ~~2~~ 0

$n = 1$

$k = 3$

$n = 7$

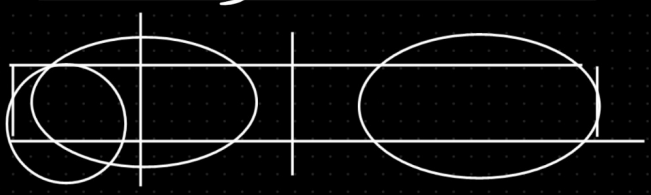
$k = k \% n$

$= 2 \% 1 = 0$

Logarithmic

→ dividing the array values by 2.

Binary Search



$n = 2$

→ 2 time

$n = 1000$

→ 1000 times

→ $O(n)$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$\log n = k \log 2$ → $O(\log n)$

$\log_2 16 = \log_2 2^4$

$= 4 \log_2 2$

$= 4$

$n = 16$

```
i = n
while i > 0:
    i = i // 2
```

$n = 64$

↳ 8

$\log_2 n$

↳ time complexity

$i = 16$

$i = 8$ — ✓ — $16/2^1$

$i = 4$ — ✓ — $16/2^2$

$i = 2$ — ✓ — $16/2^3$

$i = 1$ ✓

$i = 0$

$1 // 2$

Time Complexity (More Priority)
as Comparable to
Space Complexity