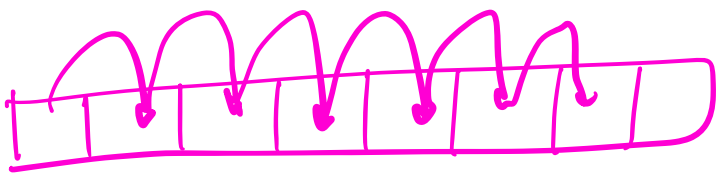
Arrays

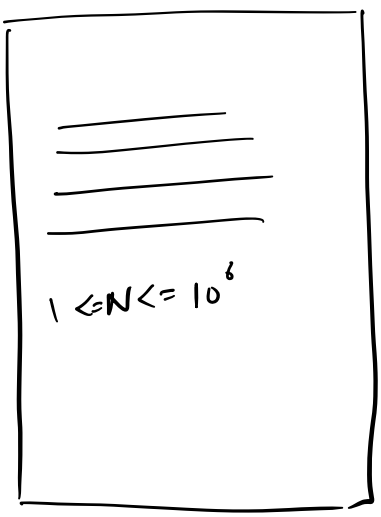
- Linear D.S. → logically!
- Collection of same datatype elements, stored contiguously inside memory!

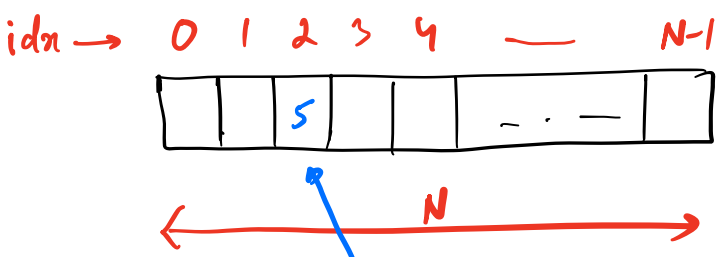


`int A[100];` → Creating an array of size 100

- datatype
- name
- size

`int A[N];`





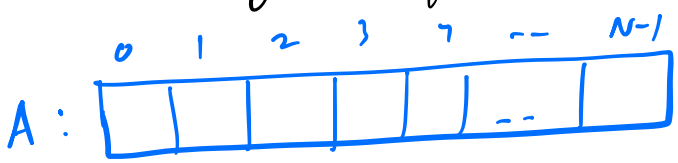
O(1) → `A[2] = 5`

↳ update

O(1) → `print(A[N-1]);`

↳ print the last element!

Q Given an Array A of size N. Print it!



print(A[0])
print(A[1])
print(A[2])
.
.
print(A[N-1])

0 ... N-1

for(i=0; i<N; i++){
print(A[i]);
}

N x O(1)

TC : O(N)

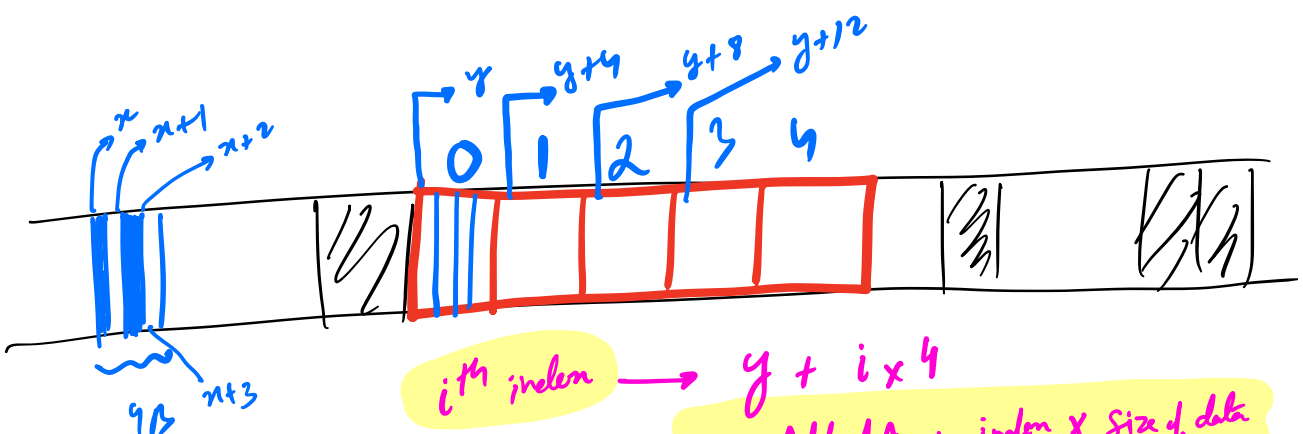
Why constant access time?

int A[5]; → 5 x 4 B : 20 B

1 int → 4B → 32 bits

print(A[2]);

A → base Address



i^{th} index → $y + i \times 4$

Base Add. of A + index x Size of data type

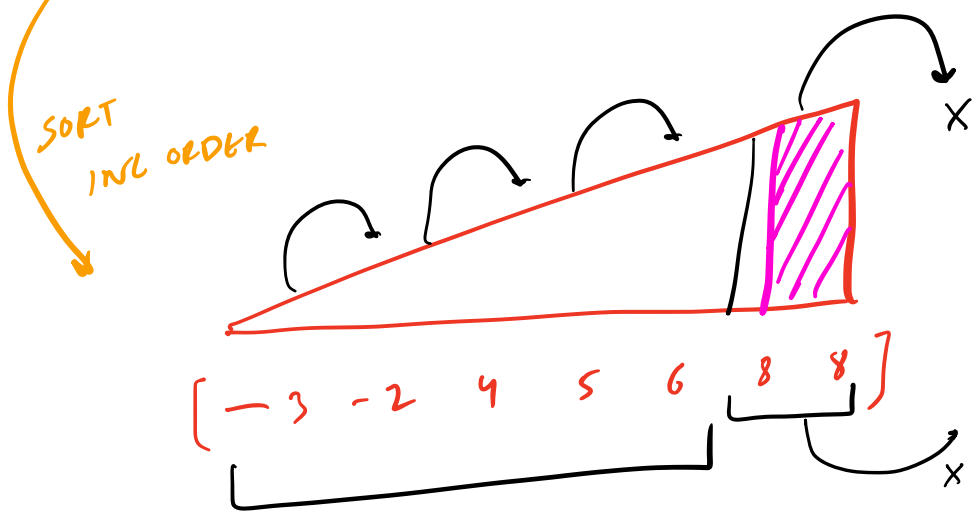
Given N elements → Array.

Count the no. of elements having atleast 1 element greater than itself!

A = [-3, -2, 6, 8, 4, 8, 5] → 5

✓ ✓ ✓ ✗ ✓ ✗ ✓

SORT INC ORDER



ANS = N − count(Max(A))

7 − cnt(8)

7 − 2 = 5

STEPS:

1. Find the MAX element!
2. Count the freq. of the MAX element!
3. ANS = N − cnt(MAX(A))

1. Find the MAX element!

A: [−3, −2, 6, 8, 4, −5, 8, 7]

mC = −∞ → −3 → −2 → 6 → 8 → 8 → 8 → 8 → 8

```
int mC = INT_MIN;
for( i=0; i<N; i++) {
    if( A[i] > mC) {
        mC = A[i];
    }
}
```

N × O(1) : O(N)

Constraints

$1 <= N <= 10^6$

$-10^9 <= A[i] <= 10^9$

mC = max(mC, A[i])

```
cnt = 0;
for(i=0; i<N; i++){
    if( mC == A[i]){
        cnt++;
    }
}
ret N - cnt;
```

N × O(1) : O(N)

O(N+N) = O(2N)

TC = O(N)

SC = O(1)

II)

A: [2, 3, 10, 7, 3, 10, 2, 10, 5]

mC = -∞ → 2 → 3 → 10 → 10 → 10 → 10 → 10 → 10 → 10

cnt = 0 → 1 → 1 → 1 → 1 → 1 → 2 → 2 → 3 → 3

```
mC = INT_MIN;
cnt = 0;
for(i=0; i<N; i++){
    if(A[i] > mC){
        mC = A[i];
        cnt = 1;
    }
    else if(A[i] == mC){
        cnt++;
    }
}
ret N-cnt;
```

N x O(1)

TC: O(N)

SC: O(1)

Doubt:

A: [2, 3, 10, 7, 4, 7, 5, 10, 7, 10, 15, 15]

mC → -∞ 2 3 10 10 10 10 10 10 10 10 15 15

cnt → 0 1 1 1 1 1 1 1→2 2→3 1→2

Q Given an array of size N & k.

Check if there exists a pair (i, j) : A[i] + A[j] = k && (i != j)

2-SUM

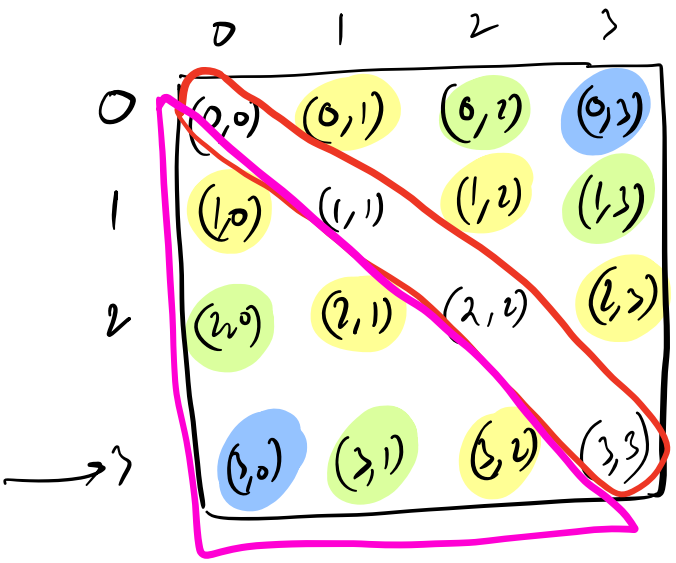
true / false

A : [3, -2, 1, 4, 3, 6, 8]
0 1 2 3 4 5 6

K = 10 → true

K = 8 → false

N = 4 [_____]



$A_i + A_j == k$

$A_j + A_i == k$

TC : $O(N^2)$

```
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        if(i != j){
            if(A[i]+A[j]==k){
                return true;
            }
        }
    }
}
return false;
```

i = 0 | j = X

i = 1 | j = 0

i = 2 | j = 0, 1

i = 3 | j = 0, 1, 2

i = 1 → N-1

i j : 0 → i-1

```
for(i=1; i<N; i++){ → ~N²/2
  for(j=0; j<i; j++){
    if(A[i]+A[j]==K){
      ret true;
    }
  }
}
ret false
```

TC : O(N²) ← N²/2

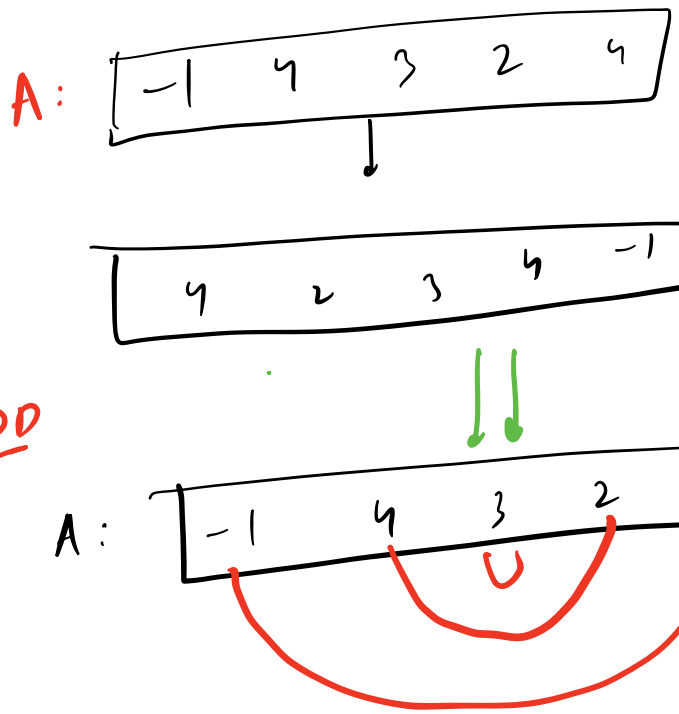
N = 10⁹ → N² → 10¹⁸

N²/2 → 10¹⁸/2 → 5 x 10¹⁷

N → 10⁹

Given an Array. Reverse it INPLACE!

↳ IN the same Array!



STOP

1. i == j
2. i > j

WORK

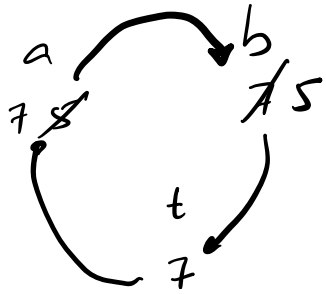
i < j

```
i = 0 , j = N-1 ;
while ( i < j ) {
    swap( A[i], A[j]);
    i++, j--;
}
```

$\frac{N}{2} \times O(1)$

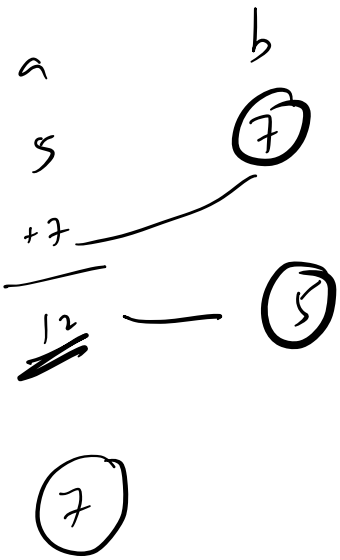
$TC = O(N)$

$SC = O(1)$



with temp

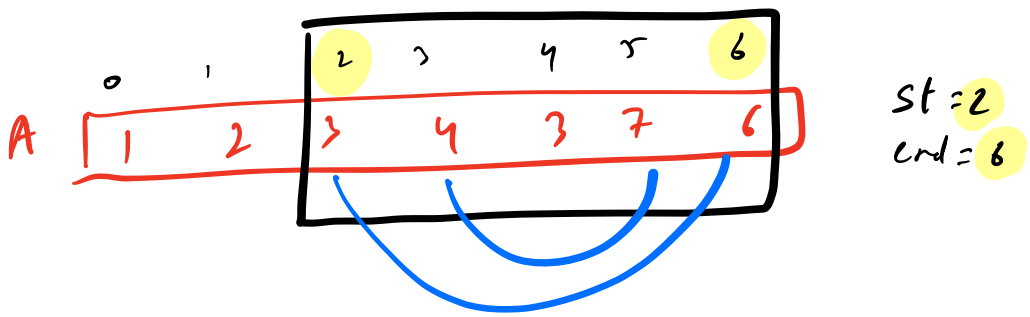
```
t = b
b = a
a = t
```



w/o temp

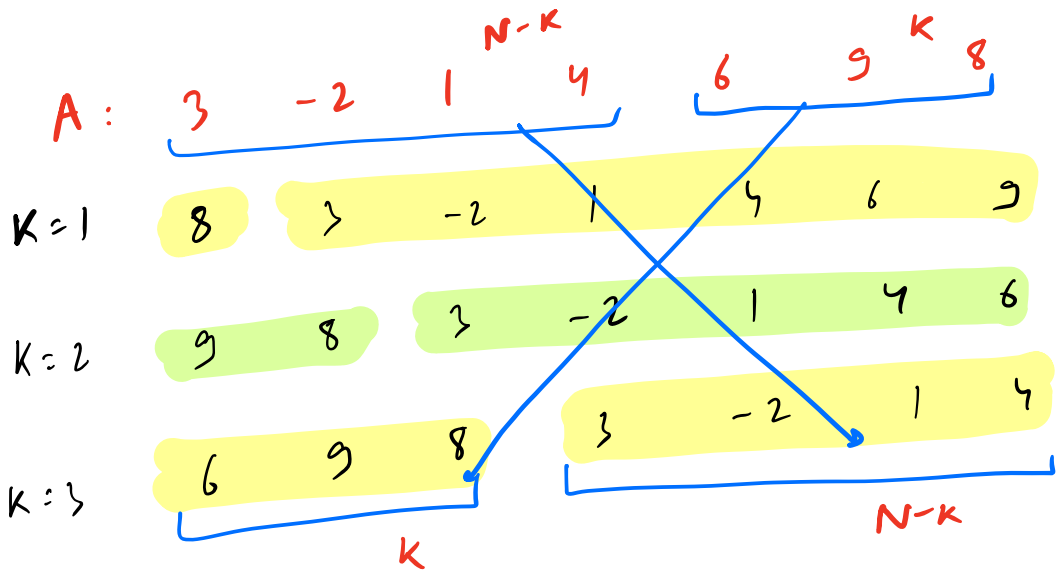
```
a = a + b
b = a - b
a = a - b
```

Q Given an Array & st & end indexs of a sub-array!
Reverse that subarray!




```
void reverse ( A[], st, end) {
    i = st, j = end;
    while ( i < j) {
        swap( A[i], A[j]);
        i++, j--;
    }
}
```

Q Given an array.
Rotate the array from last to first K times



I) BF

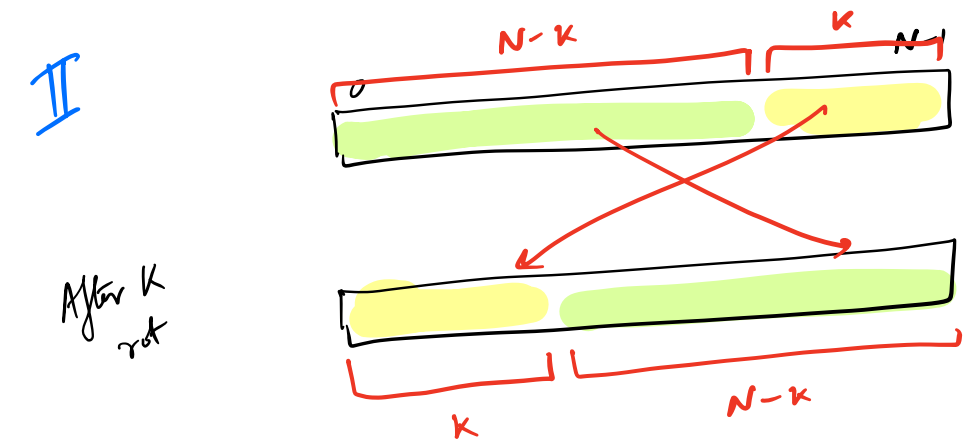
1. Write code for rot. 1 time.
2. Call K times !

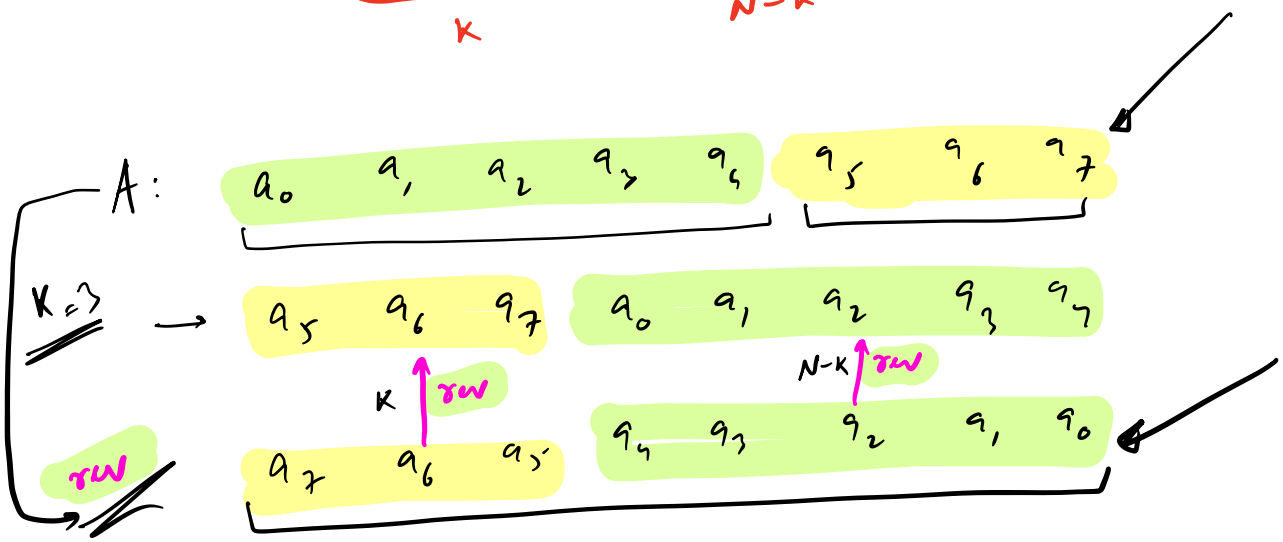
Rotating 1 time → O(N)
— K — → O(NK)

$1 <= N <= 10^6$

$1 <= K <= 10^9$

II

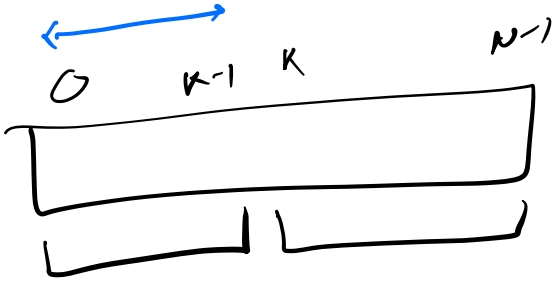




STEPS:

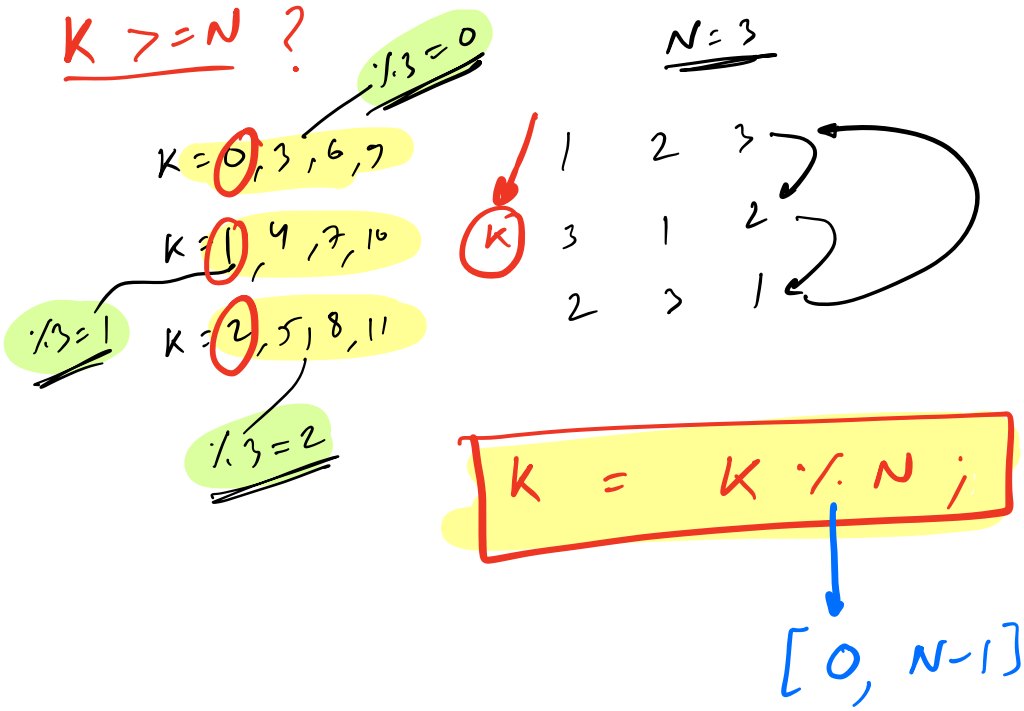
1. Reverse the Whole array! rev(A, 0, N-1) — O(N)
2. Reverse the first K elements! rev(A, 0, K-1) — O(K)
3. ——— last N-K ——— rev(A, K, N-1) — O(N-K)

TC = O(2N) = O(N)



TC = O(N)

SC = O(1)




```
K = K % N;
```

[0, N-1]