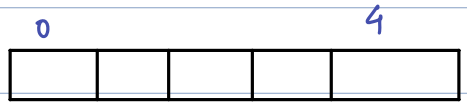
int ar[5] =



int ar[N] =


```
for (i=0; i<= N-1 ; i++) {
    print(A[i])
}
```

Q1) Given an array of size N, count number of elements having atleast one element greater than itself?

	0	1	2	3	4	5	6
A[7]	-3	-2	6	8	4	8	5

Ans: 5

Ans = N - freq(8)

= 7 - 2 = 5

A[8] { 2 3 10 7 3 2 10 8 }

max = 10

ans = N - freq(max)

= 8 - 2 = 6

A[10] { 2 5 1 4 8 0 8 1 3 8 }

Max: 8

Ans: 10 - freq(8) = 10 - 3 = 7

[7, 7, 7, 7, 7]

Ans: 0

Obs: No element can be greater than max element

obs2: All other (non max) elements will have atleast one element greater than themselves

ans = N - freq(max)

5 3 7 2 4

↑ i max = 7

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

TC: O(N)

SC: O(1)

```
// freq of max
count = 0
for(i=0; i<N; i++){
    if(A[i] == max){
        count++
    }
}
return N - count
```

HW: do it in a single loop

bool

(Q2) Given N array elements, check if there exists a pair (i, j) such that A[i] + A[j] == k && i != j

Note: i, j are index values and k is given sum

	0	1	2	3	4	5	6	
A:	3	2	1	4	3	6	8	True

i = 4 j = 5 k = 9

A[i] + A[j] = 9

	0	1	2	3	4
A:	3	5	2	7	3

k = 6

k = 5

i = 0 , j = 4

A[i] + A[j] = 6

	0	1	2	3	4
A:	3	5	2	7	3

k = 14

i = 3 j = 3

7 + 7 = 14

i != j fails

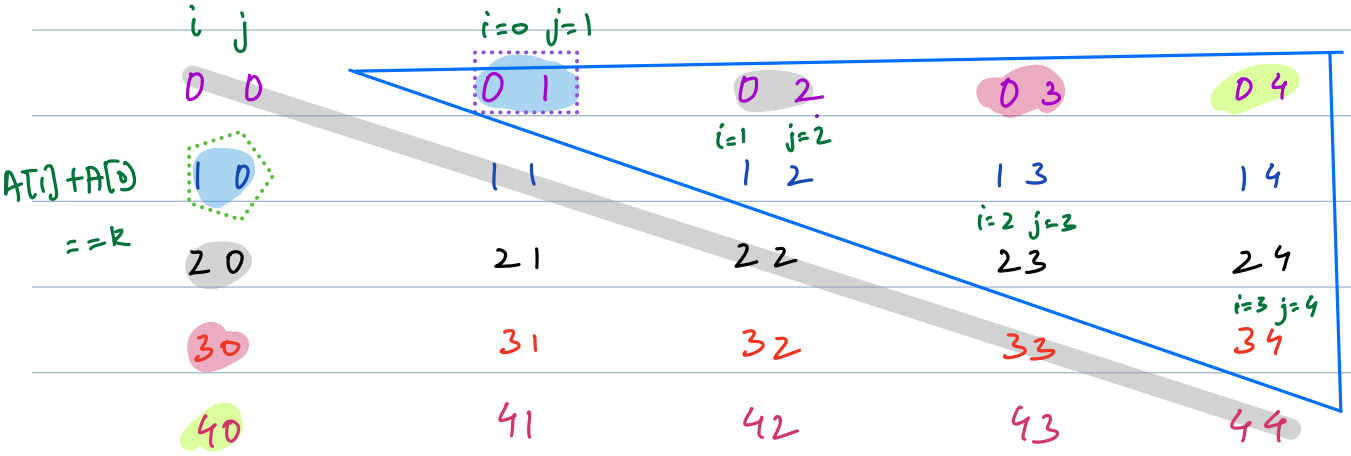
	0	1	2	3	4
A:	5	7	9	3	2

k = 12

Index:	i	j	
	2	3	True
	0	1	

Check all pairs

A[0] + A[i] == k N=5



Lower Triangl HW

```
for (i = 0;  i < N ; i++) {                    TC: O(N^2)
    for(j=0; j<N; j++) {                       SC: O(1)
        if (A[i] + A[j] == k && i != j)
            return true
    }
}
return false
```

Correct

```
for (i=0;  i<=N-2 ; i++) {
    for (j= i+1;  j<N;  j++) {
        if (A[i] + A[j] == k) {
            return true
        }
    }
}
return false
```

i	j [i+1, N-1]		
0	[1, N-1]	N-1	+ 1 (outer)
1	[2, N-1]	N-2	+ 1 (outer)
2	[3, N-1]	N-3	+ 1 (outer)
⋮			
N-2	[N-1, N-1]	1	+ 1

$1 + 2 + 3 + 4 \ldots N-1$ $\quad N-1$

AP

$$\frac{N \times (N-1)}{2} + N - 1$$

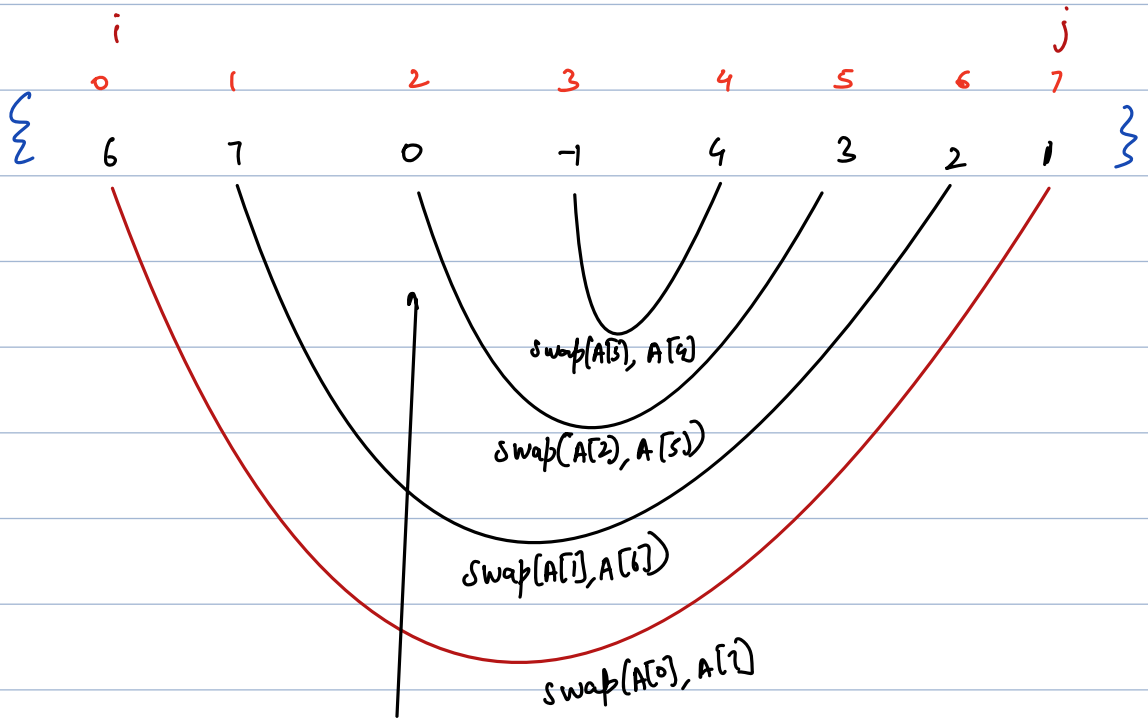
TC: $O(N^2)$

Q3) Given an array, reverse it

SC: O(1)

	0	1	2	3	4	5	6	7
A:	1	2	3	4	-1	0	7	6

rev(A): { 6, 7, 0, -1, 4, 3, 2, 1 }



i = 0 j = N-1

TC: O(N)

```
while ( i < j ) {
    swap (A[i], A[j])
    i++
    j--
}
```

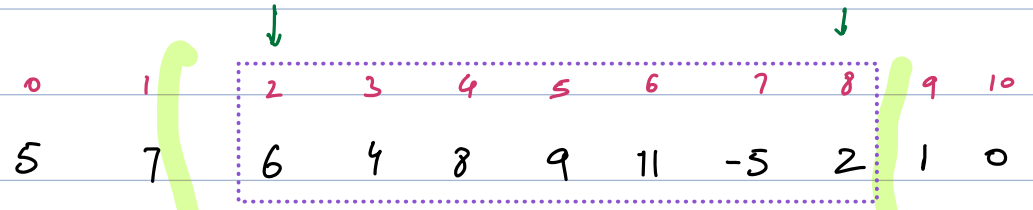
i != j

0	1	2	3
4	1	2	3
	↑ i	↑ j	

<=

Break (10:25 - 10:35)

Q4) Reverse a part of an array between [s, e]



output: 5, 7, 2, -5, 11, 9, 8, 4, 6, 10

```
void reverse (A, s, e) {
    i = s      j = e                    TC: O(N)
    while ( i < j ) {
        swap (A[i], A[j])
        i++
        j--
    }
}
```

Google / Microsoft

(Q5) Rotate an array from last to first by k times

A = { 3 -2 1 4 6 9 8 } k = 3

k = 1 { 8 3 -2 1 4 6 9 }

k = 2 { 9 8 3 -2 1 4 6 }

k = 3 { 6 9 8 3 -2 1 4 } Ans

0 1 2 3 4 5 6 7 8

{ 4, 1, 6, 9, 2, 14, 7, 8, 3 } k = 4

k = 1 { 3 4, 1, 6, 9, 2, 14, 7, 8 }

k = 2 { 8 3 4, 1, 6, 9, 2, 14 7 3

k = 3 { 7 8 3 4, 1, 6, 9, 2, 14 3

k = 4 { 14 7 8 3 4, 1, 6, 9, 2 3

k < N

{ 8 7 9 3 -2 3 1 4 6 2 3 k = 4

0 1 2 3 4 5 6 7 8 9

{ -2 3 1 4 6 2 8 7 9 3 } K = 4

{ 3 9 7 8 2 6 4 1 3 -2 } rev(A, 0, N-1)

{ 8 7 9 3 2 6 4 1 3 -2 } rev(A, 0, k-1)

{ 8 7 9 3 -2 3 1 4 6 2 } rev(A, k, N-1)

Input

k = 5

A = { a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 | a_8 a_9 a_{10} a_{11} a_{12} }

k = 5

1) Reverse array reverse(A, 0, N-1)

0 1 2 3 4

{ a_{12} a_{11} a_{10} a_9 a_8 | a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 }

[0, k-1]

reverse(A, 0, k-1)

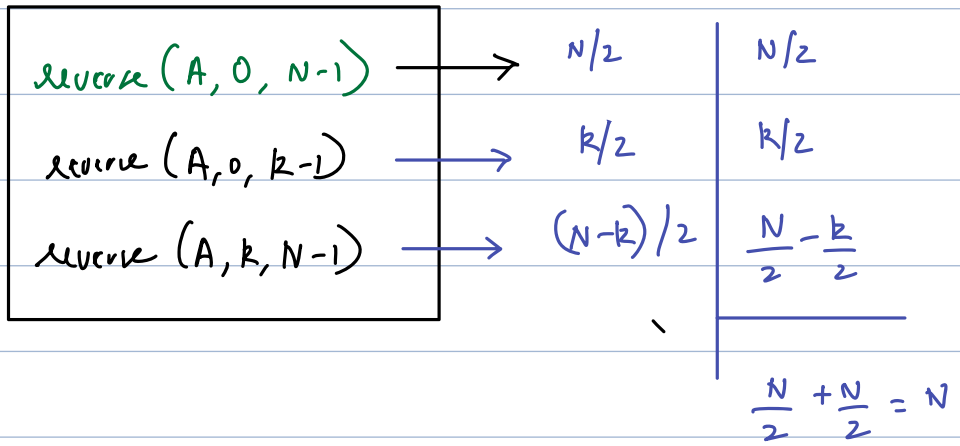
reverse(A, k, N-1)

{ a_8 a_9 a_{10} a_{11} a_{12} | a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 }

↓

$\{ a_8 \; a_9 \; a_{10} \; a_{11} \; a_{12} \; a_0 \; a_1 \; a_2 \; a_3 \; a_4 \; a_5 \; a_6 \; a_7 \}$

Output



TC: O(N)

SC: O(1)

k > N

k=0 1 2 3 4 5

k=1 5 1 2 3 4

k=2 4 5 1 2 3

k=3 3 4 5 1 2

k=4 2 3 4 5 1

k=5 1 2 3 4 5

k=6 5 1 2 3 4

k=7 4 5 1 2 3

k > N

HW

Division : Repeated subtraction

Multiplication : Repeated addition

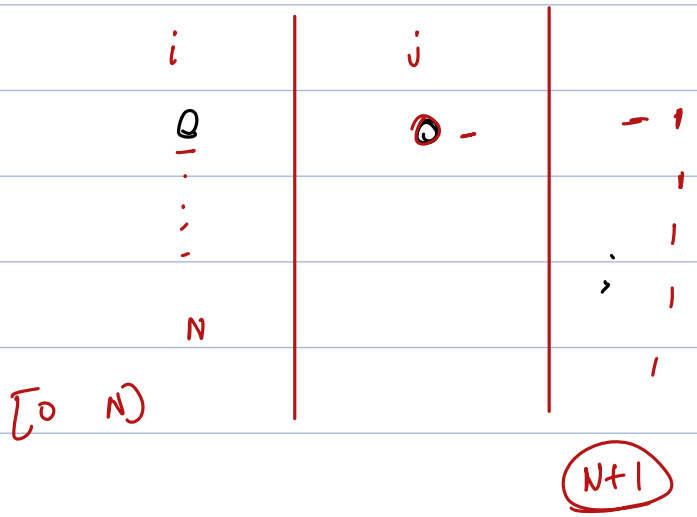
$3 = \frac{15}{5} \Rightarrow$ $15 - 5 - 5 - 5 = 0$

$\frac{15}{7} \Rightarrow$ $15 - 7 - 7 = 1$

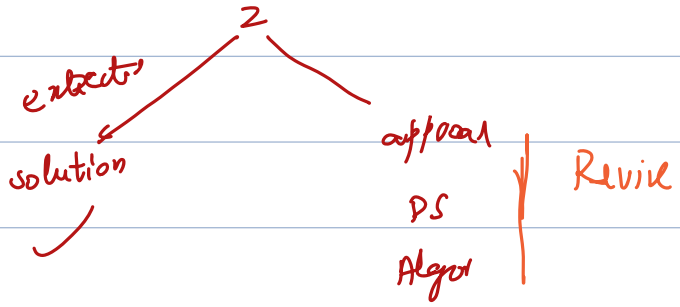
$15 \% 7 =$

18 → 13 → 8 → 3

$18 \% 5 =>$



for



big O ✓

for (i=0; i< N ; i+=c) → O(N)

for (i=1; i<N; i*=c) $O(\log_c N)$