Lecture 4 :- Introduction to Arrays

Agenda

- Arrays Revision.
- Problems on Arrays

Class starts at 7:05 AM.

You didn't come this far only to come this far

Arrays → collection of **similar data** type.

→ contiguous memory allocation.

Initialization

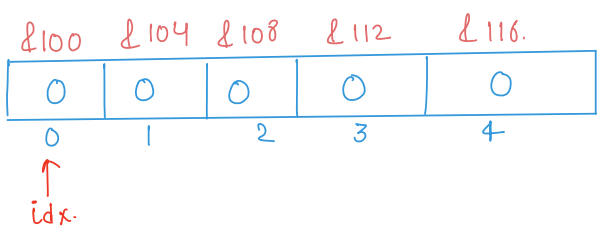
`int[] arr = new int[10]` [Java]

`int arr[10];` [C++]

`int[5] arr = { 1, 2, 8, 9, 5 }` → valid.

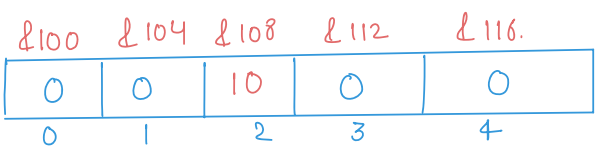
`arr[] = { 2, 8, "Ayush", c, 1, 5 }` — invalid.

`int[] arr = new int[5];`



[arr[n] — idx[0, n-1]]

`arr[2] = 10.`



[idx 0 - 4]

`arr[-1]`
`arr[5]`
} — out of bound exception.

(1) Print all array el.

```
arr[5] = { 2, 8, 2, 3, 5 }
          0  1  2  3  4
```

```
void print(int[] arr) {
    for(i=0; i<arr.length; i++) {
        print(arr[i]);
    }
}
```

Qu! Given arr[n], count no. of el. having at least 1 el greater than itself.

arr[7] = { -3 -2 6 8 4 8 5 } count = 5.

(0 1 2 3 4 5 6)

-3 < 6 -2 < 8

arr[8] = { 2 3 10 7 3 2 10 8 } count = 6

arr[5] = { 8 8 8 8 8 } count = 0.

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

Observation :- max el of arr can never be part of my answer.

Logic arr[10] = { 2 5 1 4 8 0 8 1 3 8 }

count max el = 3.

ans = 10 − 3 = 7 Ans.

1.) find max el.

2.) find occ of max el.

3.) arr.length − occ of max el.

```
int countGreaterThanItself(int[] arr) {
    int n = arr.length;
    // find max el.
    int max = arr[0];
    for(i=1; i<n; i++) {
        if(arr[i] > max) {
            max = arr[i];
        }
    }
    // occ of max el
    int count = 0;
    for(i=0; i<n; i++) {
        if(arr[i] == max) {
            count++;
        }
    }
    return arr.length - count;
}
```

ip

O(n) —— O(n)

TC :- $O(n+n) = O(2n) \simeq O(n)$.

SC :- $O(1)$

Qu2 Given arr[n], check if there exists a pair (i, j) such that arr(i) + arr[j] = K & i != j.

Ex1. arr[7] = { 0: 3, 1: -2, 2: 1, 3: 4, 4: 3, 5: 6, 6: 8 }

k = 10. true.

Ex2. arr[4] = { 2, 4, -3, 7 }

k = 5. false

Ex3. arr[4] = { 0: 2, 1: 4, 2: -3, 3: 7 }

k = 8. false

Ex4. arr[5] = { 0: 2, 1: 4, 2: 5, 3: 4, 4: 7 }

k = 8. true

Approach: Go to all pairs of arr.

arr[5] = { 3 5 2 7 4 } k = 11.

(0,0) i=0 i==j

(1,0) (0,1) i=1 Already calculated

(2,0) i=2 0,2 calculated

(3,0) i=3

(4,0) i=4

0,1	1,1	2,1	3,1	4,1
3+5=8	i==j	1,2 calculated		
0,2	1,2	2,2	3,2	4,2
3+2=5	5+2=7	i==j		
0,3	1,3	2,3	3,3	4,3
3+7=10	5+7=12	2+7=9		
0,4	1,4	2,4	3,4	4,4
3+4=7	5+4=9	2+4=6	7+4=11	

Logic:

$i \in [0, n-2]$ ← second last idx.

$j \in [i+1, n-1]$ ← last idx.

i=0: 0 0, 0 1, 0 2, 0 3, 0 4

i=1: 1 0, 1 1, 1 2, 1 3, 1 4

j → starting bt = i+1.

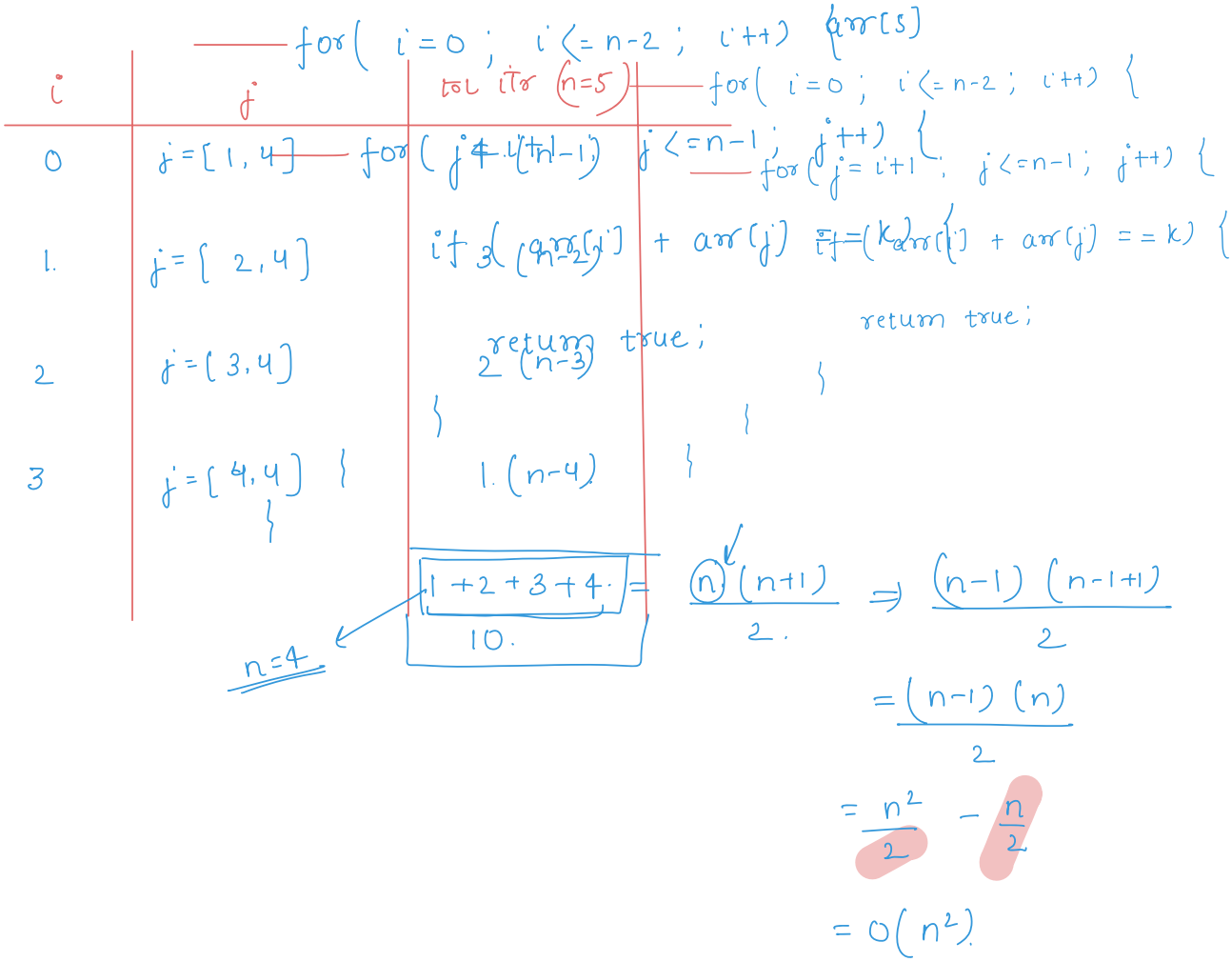
```
boolean checkSumPair(int[] arr, int k) {
    int n = arr.length;                          → O(1)
O(n).  ——— for(i=0; i<=n-2; i++) {
O(n).  ——— for(j=i+1; j<=n-1; j++) {
            if(arr[i] + arr[j] == k) {
                return true;
            }
        }
    }
    return false;
}
```

TC: $O(n^2)$

SC: $O(1)$

Optimised: HashMap

Sorting


for(i=0 ; i<=n-2 ; i++) {arr[5]
i
j
tot itr (n=5)
for(i=0 ; i<=n-2 ; i++) {
0
j=[1,4] for(j=i+1 ; j<=n-1 ; j++) {
for(j=i+1 ; j<=n-1 ; j++) {
1.
j=[2,4]
if(arr[i] + arr[j] == k) {
if(arr[i] + arr[j] == k) {
return true;
2
j=[3,4]
2 (n-3)
return true;
}
}
}
3
j=[4,4]
}
1.(n-4)
}
1+2+3+4 = n(n+1)/2 ⇒ (n-1)(n-1+1)/2
10.
n=4
= (n-1)(n)/2
= n²/2 - n/2
= O(n²)

Qu Given arr[n], reverse the entire array

arr[5] = { 2, 3, 1, 5, 8 } ⟷ (reverse) { 8, 5, 1, 3, 2 }

constraint:- SC = O(1), Array should be reversed internally

	0	1	2	3	4	5	6	7
arr[8] =	-1	4	7	6	-2	17	8	10
reverse(arr) =	10	8	17	-2	6	7	4	-1

observation:- swapping

	0	1	2	3	4	5	6	7
arr[8] =	-1	4	7	6	-2	17	8	10

i		j		
0	swap	7	[10	4 7 6 -2 17 8 -1]
1	swap	6	[10	8 7 6 -2 17 4 -1]
2	swap	5	[10	8 17 6 -2 7 4 -1]
3	swap	4	[10	8 17 -2 6 7 4 -1]

i < j

4 3 [stop]

arr[7] = { 0: 1, 1: 2, 2: 8, 3: 3, 4: 4, 5: 6, 6: 7 }

i		j	
0	swap	6	[7 2 8 3 4 6 1]
1	swap	5	[7 6 8 3 4 2 1]
2	swap	4	[7 6 4 3 8 2 1]
3	swap	3	stop

Condition: $i < j$

```
void reverse(int[] arr) {
    int i = 0;                    // 4B
    int j = arr.length - 1;       // 4B
    while(i < j) {
        swap(arr, i, j);          // → h/w
        i++;
        j--;
    }
}
```

SC:- 4B + 4B = 8B

O(8B) = O(1)

TC:- O(n).

SC:- O(1).

Break:- 8:45 AM.

Qu Given arr[n], si & ei, reverse arr from si to ei.

	0	1	2	3	4	5	6	7	8
arr[ ] = {	-3	4	2	8	7	9	6	2	10 }

si = 2, ei = 6 [-3 4 6 9 7 8 2 2 10.]

```
void reverse(int[] arr, int si, int ei)
    int i = si;
    int j = ei;
    while(i < j) {
        swap(arr, i, j);   → h/w
        i++;
        j--;
    }
}
```

TC: O(n)
SC: O(1)

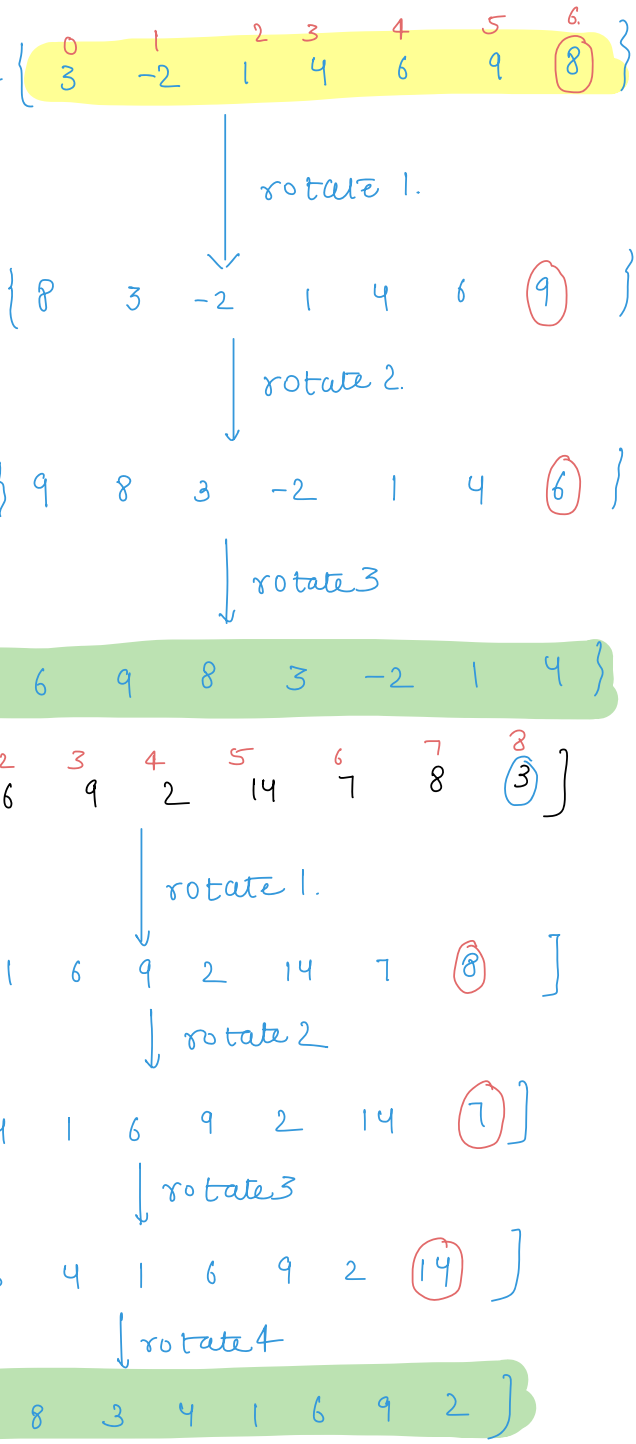
Reverse of entire arr: reverse(arr, 0, arr.length-1);

Amazon, Meta, Google ----

Qu Given arr[n], rotate array from last to first by k times.

Examples arr[7] = { 3 -2 1 4 6 9 8 }

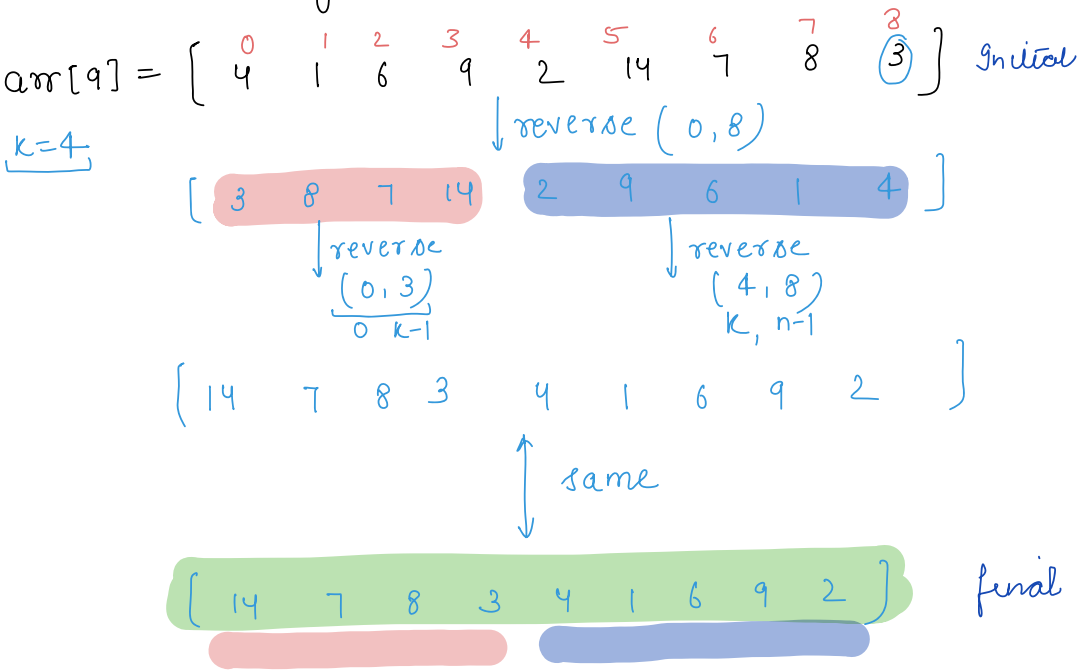
k = 3.



arr[9] = [4 1 6 9 2 14 7 8 3]

k = 4

case 1: k < arr.length.



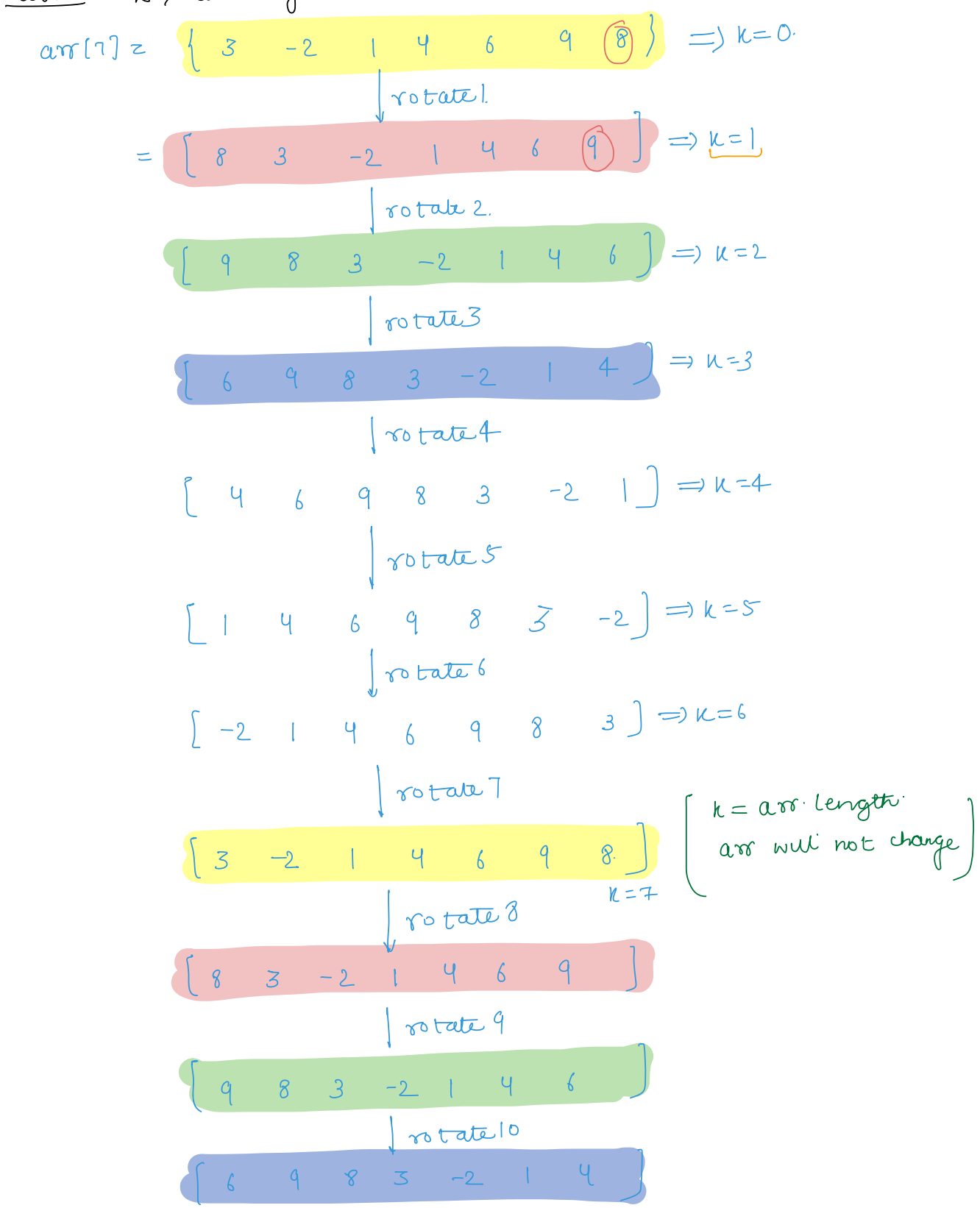
Steps involved

1. reverse (0, n-1) = reverse the entire array
2. reverse first k el ⇒ reverse(0, k-1)
3. reverse remaining el ⇒ reverse(k, n-1)

```
void rotate(int[] arr, int k) {
    reverse(arr, 0, arr.length-1); — O(n)
    reverse(arr, 0, k-1); — O(n)
    reverse(arr, k, arr.length-1); — O(n)
}
```

TC: O(3n) ≃ O(n)

SC: O(1).


case2: k > arr.length
arr[7] = { 3 -2 1 4 6 9 8 } ⇒ k=0.
rotate 1.
= [8 3 -2 1 4 6 9] ⇒ k=1
rotate 2.
[9 8 3 -2 1 4 6] ⇒ k=2
rotate 3
[6 9 8 3 -2 1 4] ⇒ k=3
rotate 4
[4 6 9 8 3 -2 1] ⇒ k=4
rotate 5
[1 4 6 9 8 3 -2] ⇒ k=5
rotate 6
[-2 1 4 6 9 8 3] ⇒ k=6
rotate 7
[3 -2 1 4 6 9 8]
k=7
k = arr.length
arr will not change
rotate 8
[8 3 -2 1 4 6 9]
rotate 9
[9 8 3 -2 1 4 6]
rotate 10
[6 9 8 3 -2 1 4]

Observation

arr[7] =

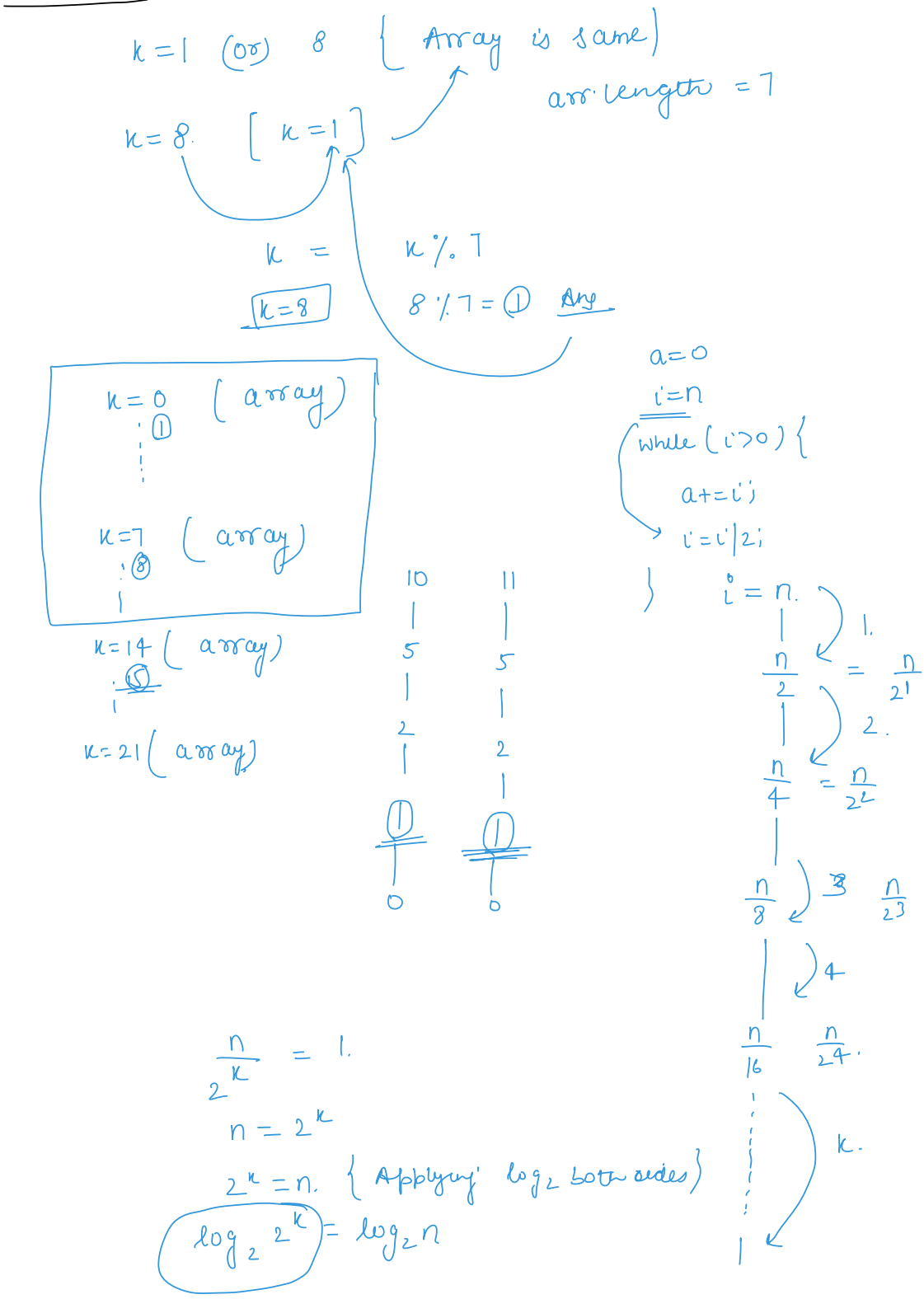
k=0	k= n.(7)	(Arrays same)
k=1.	k=n+1. (8)	(Arrays same)
k=2	k=9	(" ")
k=3	k=10	(" ")

k = k %. arr.length;

```
void rotate( int[] arr, int k) {
    k = k %. arr.length;

    reverse( arr, 0, arr.length-1); — O(n)

    reverse( arr, 0, k-1); — O(n)

    reverse( arr, k, arr.length-1); —O(n)
}
```

Thankyou ☺

Doubt


k = 1 (or) 8 { Array is same }
arr.length = 7
k = 8 [k = 1]
k = k % 7
k = 8
8 % 7 = 1 Any
k = 0 (array)
1
k = 7 (array)
8
k = 14 (array)
15
k = 21 (array)
a = 0
i = n
while (i > 0) {
a += i;
i = i / 2;
}
10
5
2
1
0
11
5
2
1
0
i = n.
n/2 = n/2^1
n/4 = n/2^2
n/8 = n/2^3
n/16 = n/2^4.
1.
2.
3
4
k.
n/2^k = 1.
n = 2^k
2^k = n. { Applying log2 both sides }
log2 2^k = log2 n

```
for(i=0; i<n; i++){
    for(j=i-1; j>=0; j++)
    }
}
```

i	-i-1,0)
0	0
1	j = 0, 1, 2, 3, . . . ———
2	
3	
⋮	
n