CS & IT ENGINEERING

Data structure & Programming

Arrays

Lec- 04

By- Pankaj Sharma sir

Arrays-4

Advantages :

① Random access: → access any element in constant time

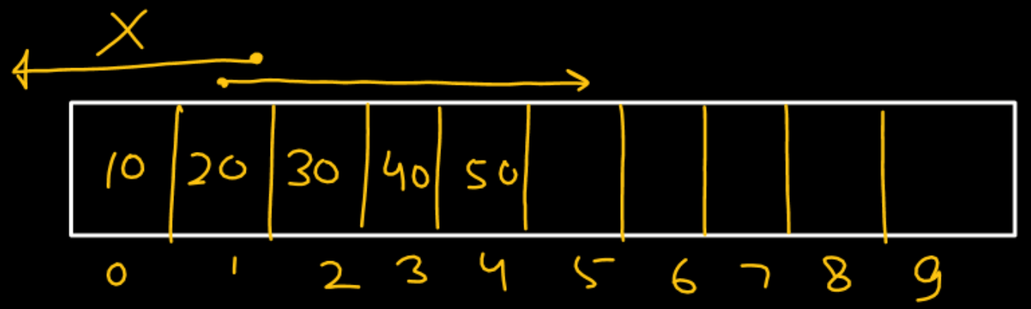
② Cache Friendly: performance ↑

disadvantage :

① pre-allocate size

② ↓ Insertion/deletion are expensive

```
#define SIZE 10
void main(){
    int n;
    int arr[SIZE];
```



$n = 5$

SIZE = 10

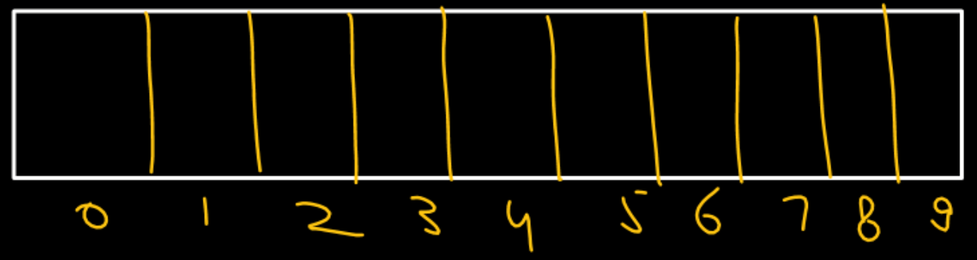
① Insert :

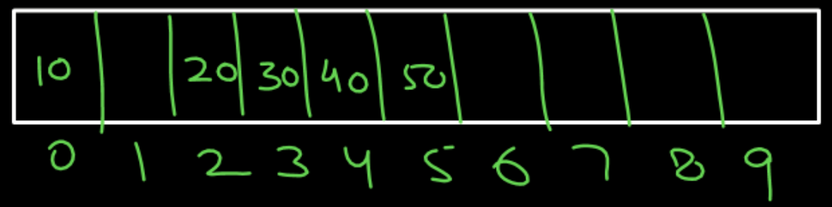
a) Insert 100 at End of array

constant time

```
arr[n] = 100;
n = n+1;
```

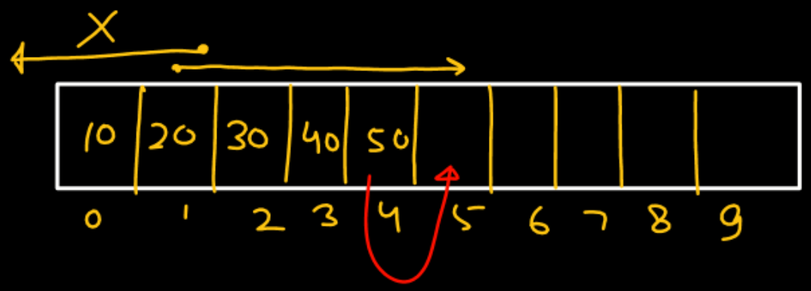
b) Insert 100 at index 1





$n = 5$

index => 1



$i+1 = n$

$A[5] = A[4]$

$A[4] = A[3]$

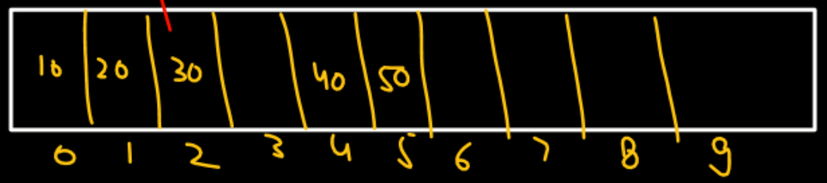
$A[3] = A[2]$

$A[2] = A[1]$

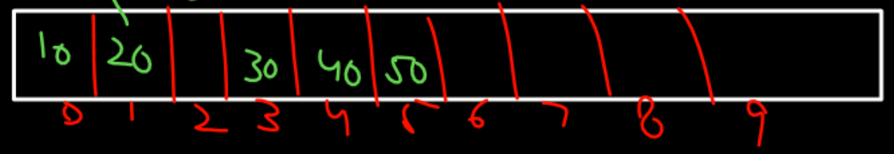
$A[5] = A[4]$


```
for(i=n-1; i>=index; i--)
    A[i+1] = A[i];
```

$A[4] = A[3]$



$A[3] = A[2]$



$A[2] = A[1]$

$i+1 = n$

index

Time complexity: → no. of operation

$A[5] = A[4]$

$A[4] = A[3]$

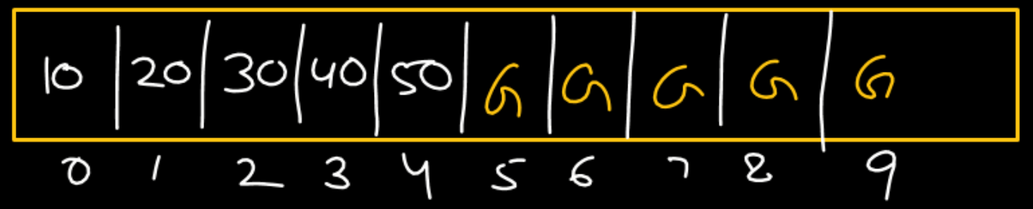
$A[3] = A[2]$

$A[2] = A[1]$

```
for(i=n-1; i>=index; i--)
    A[i+1] = A[i];
```

```
A[index] = element;
```

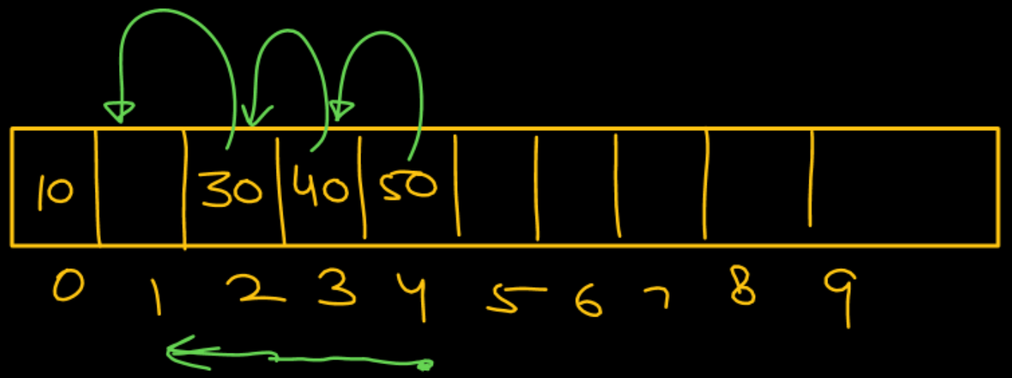
$n = 5$



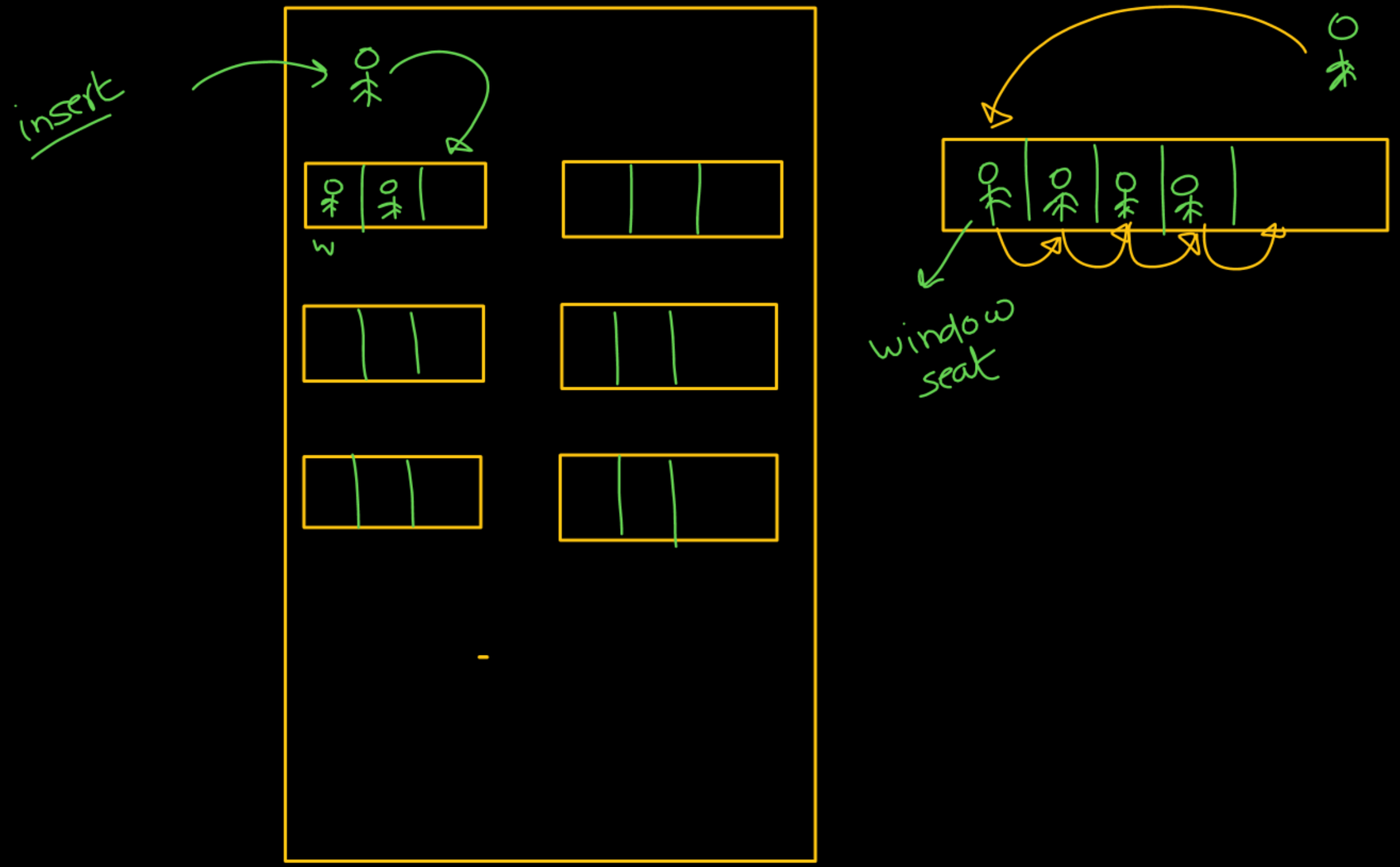
delete

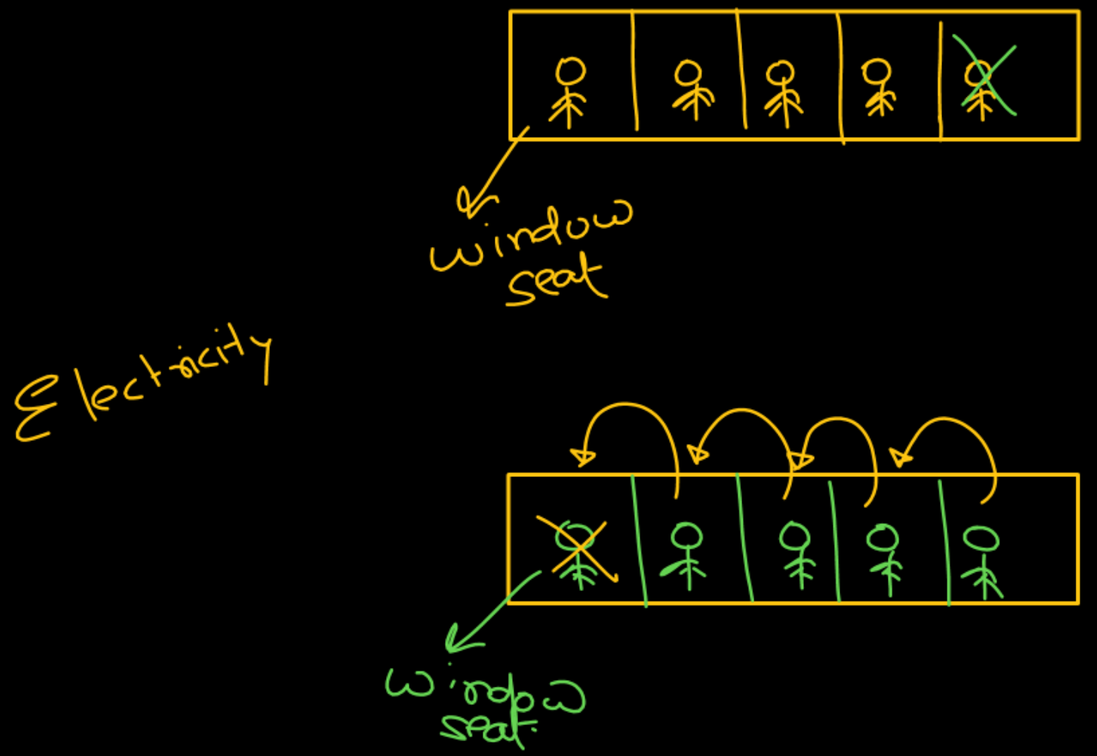
$n = n-1;$ ✗

index => 1



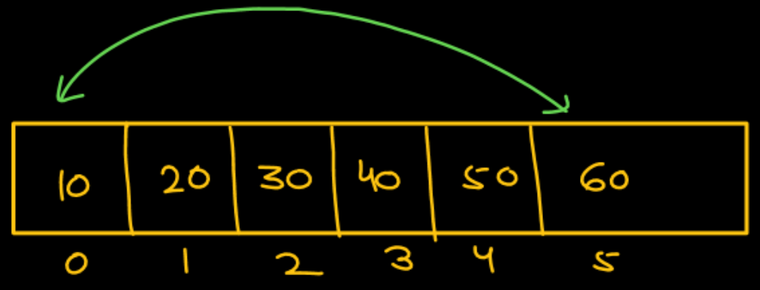
$\Rightarrow O(n)$


insert
w
window seat



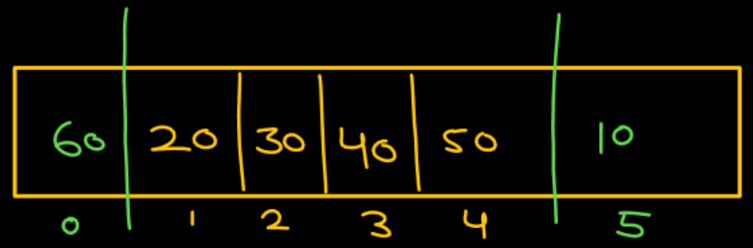
C Prog.
+
DS

Reverse



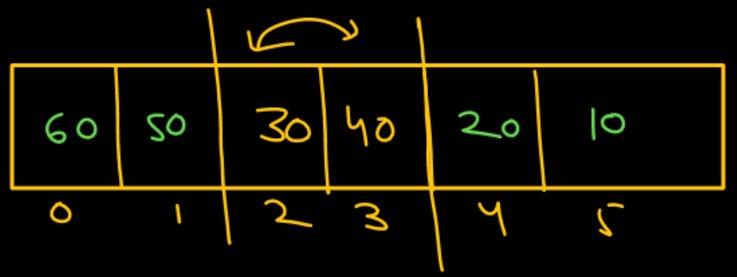
step 1

Swap(A[0], A[5])



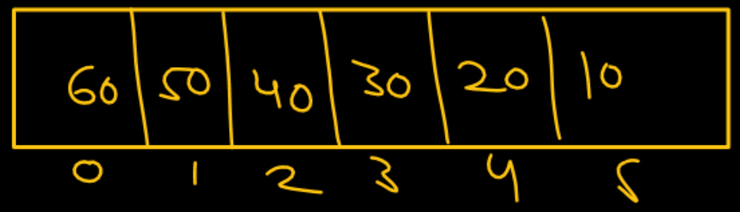
2 elem ✓

swap(A[1], A[4])



2 elem ✓

swap(A[2], A[3])



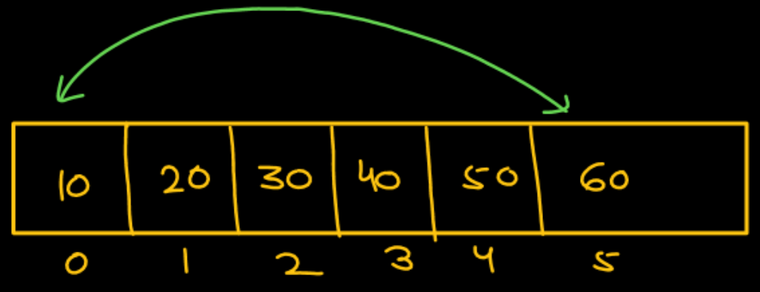
2 elem ✓

n/2 swaps

Reverse

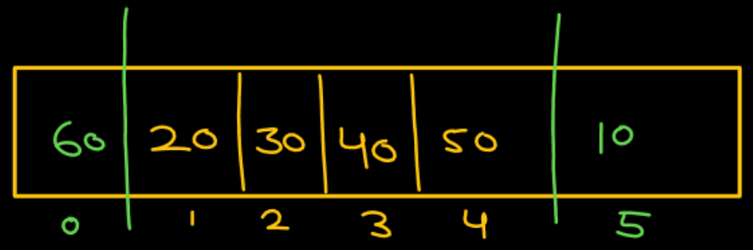
$i + j = n-1$

$j = n-i-1$

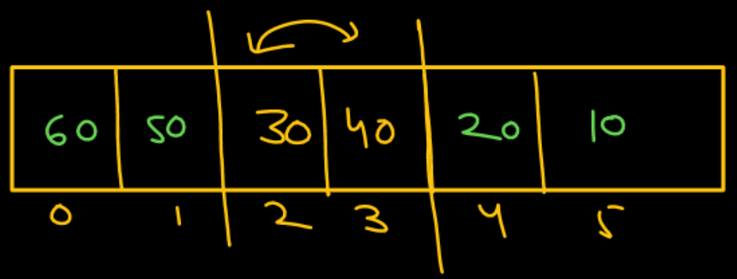


step1

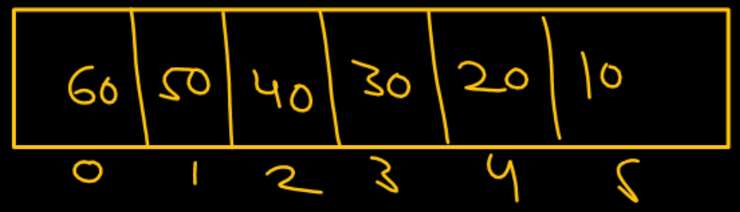
swap(A[0],A[5])



swap(A[1],A[4])



swap(A[2],A[3])



swap(A[i],A[j])

i	j
0	5
1	4
2	3

for(i=0; i<~~n~~ $n/2$; i++)

swap(A[i],A[n-i-1]);

7/2 = 3

odd no.

n = 7

i = 0, 1, 2

n-1-i

7-1-0

⇒ 6

7-1-1

= 7-2

= 5

0, 6

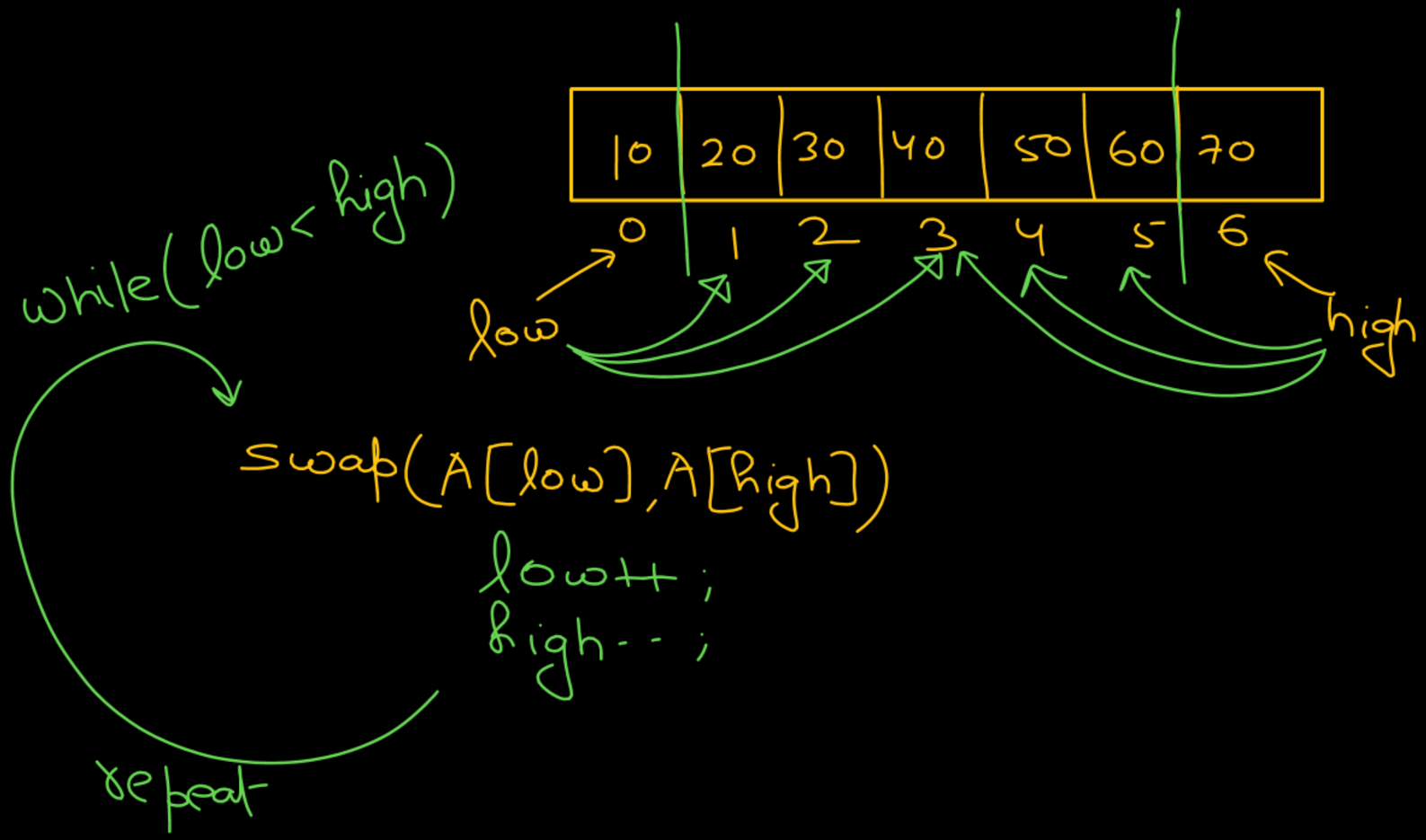
1, 5

2, 4

n-1-2

7-1-2

= 4

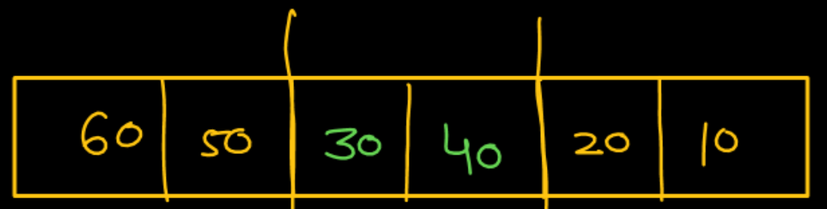

10
20
30
40
50
60
70
0
1
2
3
4
5
6
low
high
while(low<high)
swap(A[low],A[high])
low++;
high--;
repeat

first, last ⇒ swap

(i) swap(A[low], A[high])

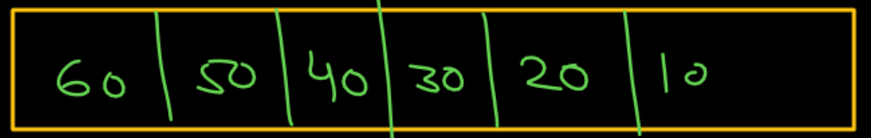
low++
high--;

(ii) swap(A[low], A[high])



low++
high--

(iii) swap(A[low], A[high])

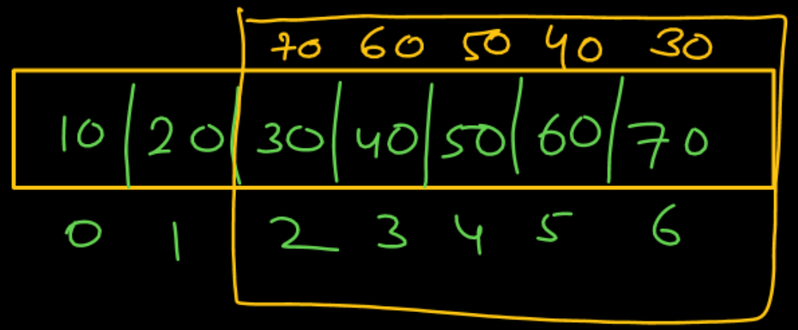


low++;
high--;

high
low

```
void reverse(int arr[], int low, int high)
    {
        while(low<high){
            swap(A[low],A[high]);
            low++;
            high--;
        }

}
```



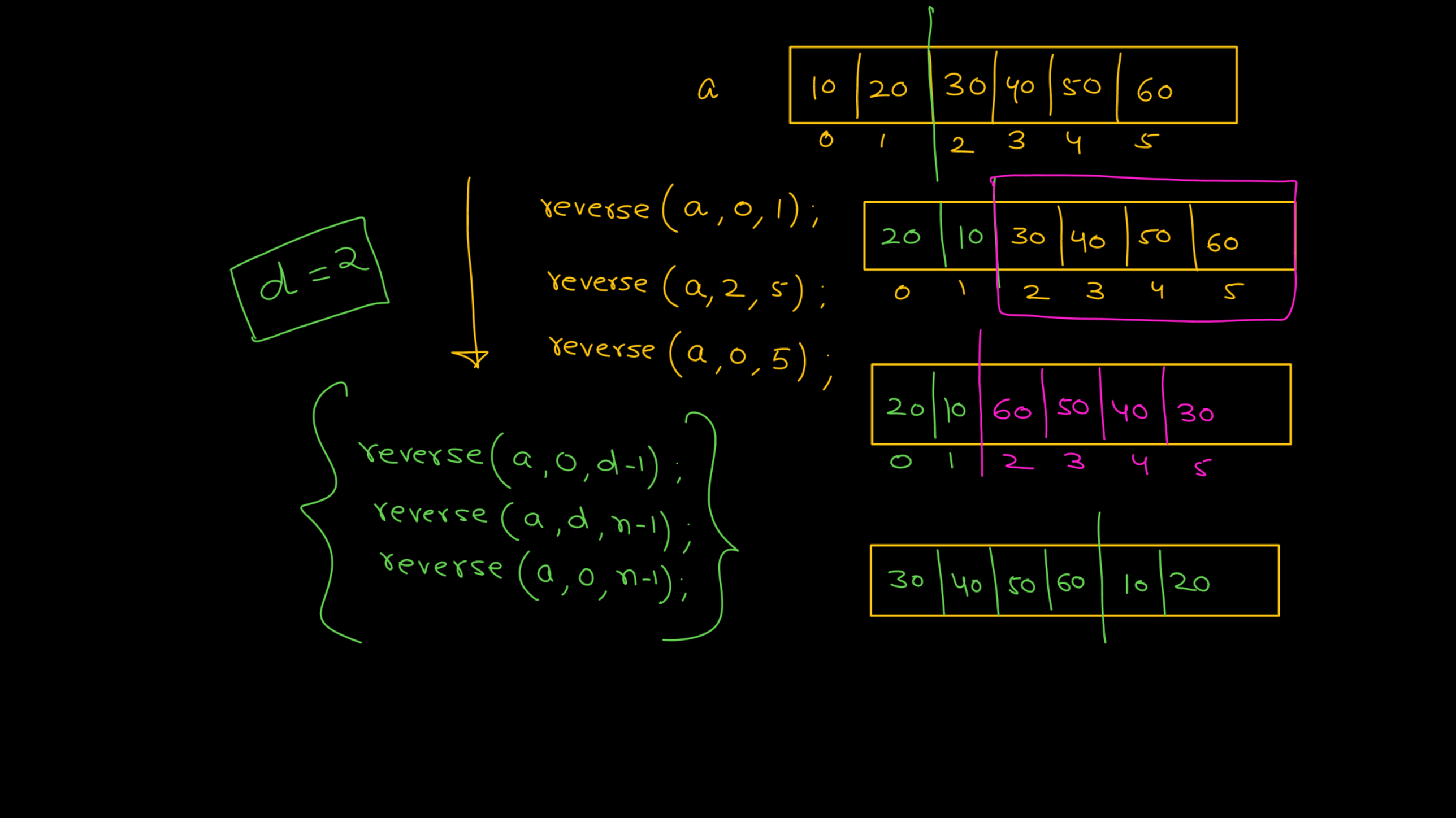

```
void main(){
    int a[] = {10,20,30,40,50,60
                70};

    reverse(a, 2, 6);

    ==

    ==

}
```


a
10 20 30 40 50 60
0 1 2 3 4 5
d = 2
reverse(a, 0, 1);
reverse(a, 2, 5);
reverse(a, 0, 5);
20 10 30 40 50 60
0 1 2 3 4 5
20 10 60 50 40 30
0 1 2 3 4 5
30 40 50 60 10 20
reverse(a, 0, d-1);
reverse(a, d, n-1);
reverse(a, 0, n-1);

1st approach

largest, second_largest — 3 min

Easy

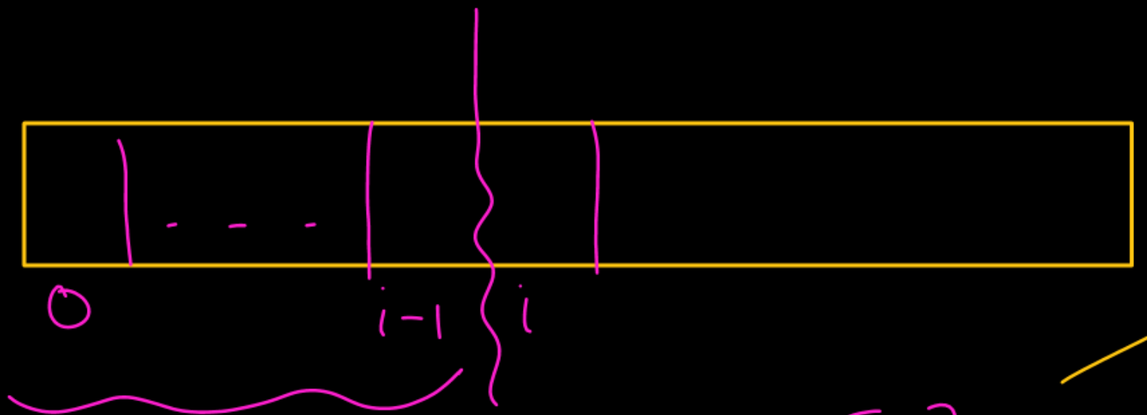
2 traverse

Can we do it in 1 traversal

```
max = -∞;
max2 = -∞;
for(i=0; i<n; i++)
{
    if (arr[i] > max2 && arr[i] != max)
        max2 = arr[i];
}
```

①

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



we know

max1 & max2

arr[i]

if ( )

1) if it is larger than max. till now

max2 = max1;

max1 = arr[i];

else if

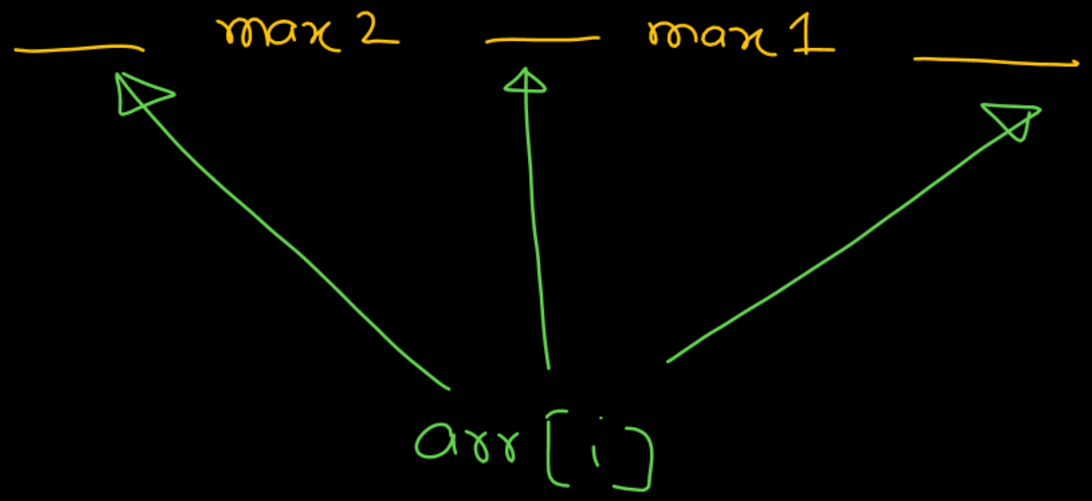
2)

arr[i] > max2

then

max2 = arr[i];


1
2
3
Linked list


max 2
max 1
arr[i]

Linked list

08 : 30 PM

- structures


THANK YOU GW
SOLDIERS !