★ 2nd Largest Element : ↷ asked In many Interview

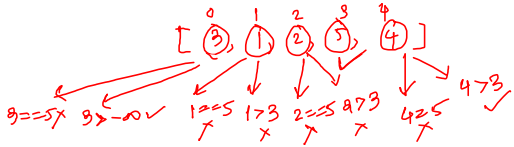
eg: [3, 1, 2, 5, 4]
(0 1 2 3 4)

op: 4

(3, 1, 2, 5, 4)
↓ remove 1st max
(3, 1, 2, 4)
↓ 1st max = 2nd max overall

① find the 1st Largest, firstMax = 5

② Find the 1st Largest, But this avoid the elements which are equal to firstMax.



maxEle = -∞ 5 4

① i = 0,
a. arr[i] == firstMax == 5
3 == 5 ⇒ false hence consider
b. arr[i] > maxEle,
maxEle = arr[i];
= 3

i = 3,
a. arr[1] == 5
5 == 5
⇒ avoid this element do not consider for maxEle calculation

★ follow up third max : repeat same process avoid 1st max, 2nd max

★ follow up Kth max → sorting
↘ min Heap Datastructure ★

★ Reverse an Array :

```
          0  1  2  3  4
Eg:   [10, 20, 30, 40, 50]

Op:   [50, 40, 30, 20, 10]
```

```
function reverse (arr) {

    /*
       Code
    */

}
```

↘ this will not return anything

```
const arr = [10, 20, 30, 40, 50];
reverse (arr);
console.log (arr); // [50, 40, 30, 20, 10]
```

#1 : Iterate from last ele to 1st ele

arr = [10, 20, 30, 40, 50]

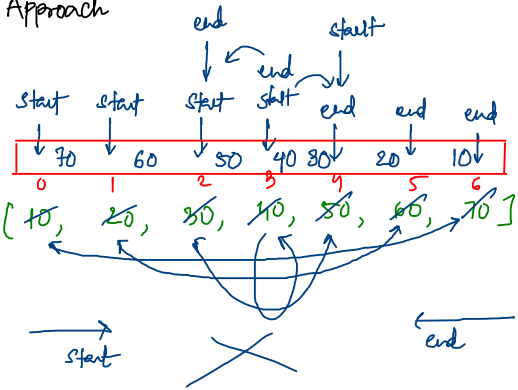
```
const rev_arr = [];
for (let i = n-1; i >= 0; i--) {
    rev_arr.push (arr[i]);
}
```

rev_arr = ~~[]~~ ~~[50]~~ ~~[50, 40]~~ ~~[50, 40, 30]~~ ~~[50, 40, 30, 20]~~ [50, 40, 30, 20, 10]

★ we need the reversed version in original arr itself, hence copy rev_arr → arr

```
for (let i=0; i<n; i++) {
    arr[i] = rev_arr[i];
}
```

#2 : Two pointer Approach




```
let start = 0;
let end = n-1  = 7-1 = 6;
while ( s <= e ) {
    swap( arr(s), arr(e) );
    s++;
    e--;
}
```

You should Implement this.

In-place solution ↓ make changes within given array instead of taking an extra array.

Crossing each other stop the process

end = 2, start = 4

Cross ⇒ start > end

all time ⇒ start <= end

★ Sum of Arrays Except Self :

0 1 2 3

ip : [4, 3, 2, 10]

op : [15, 16, 17, 9]

19 − 4 = 15 → [15]

19 − 3 = 16 → [16, 16]

19 − 2 = 17 → [15, 16, 17]

19 − 10 = 9 → [15, 16, 17, 9]

```
const ans = [];
for (let i = 0; i < n; i++) {
  ans.push(total - arr[i]);
}
```

ans[0] = arr(1) + arr(2) + arr(3) = 3 + 2 + 10 = 15

ans[1] = arr(0) + arr(2) + arr(3) = 4 + 2 + 10 = 16

ans[2] = arr(0) + arr(1) + arr(3) = 4 + 3 + 10 = 17

ans[3] = arr(0) + arr(1) + arr(2) = 4 + 3 + 2 = 9

ans = []

= [19 − 4] ⇒ ~~[15]~~

= [15, 19 − 3] ⇒ ~~[15, 16]~~

= [15, 16, 19 − 2] ⇒ ~~[15, 16, 17]~~

= [15, 16, 17, 19 − 10]

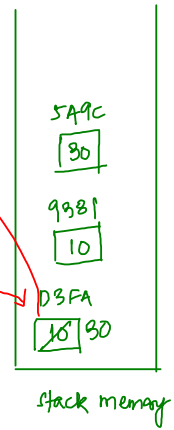
⇒ [15, 16, 17, 9]

① total sum = 4 + 3 + 2 + 10 = 19

② ans[i] = total sum − arr[i]

★ How Arrays work Internally :

```
let a = 10;
console.log(a);
let b = a;
let c = 30;
a = c;
```

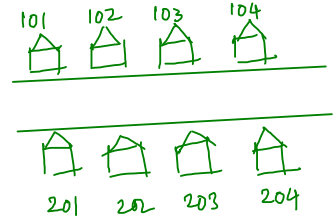


Variable	address	
a	D3FA	~~10~~ 30
b	9981	10
c	5A9C	30

→ randomly assumed

lookup table

1. memory space created in stack
2. entry made in lookup table



★ address is in Hexadecimal System/format, (base-16)

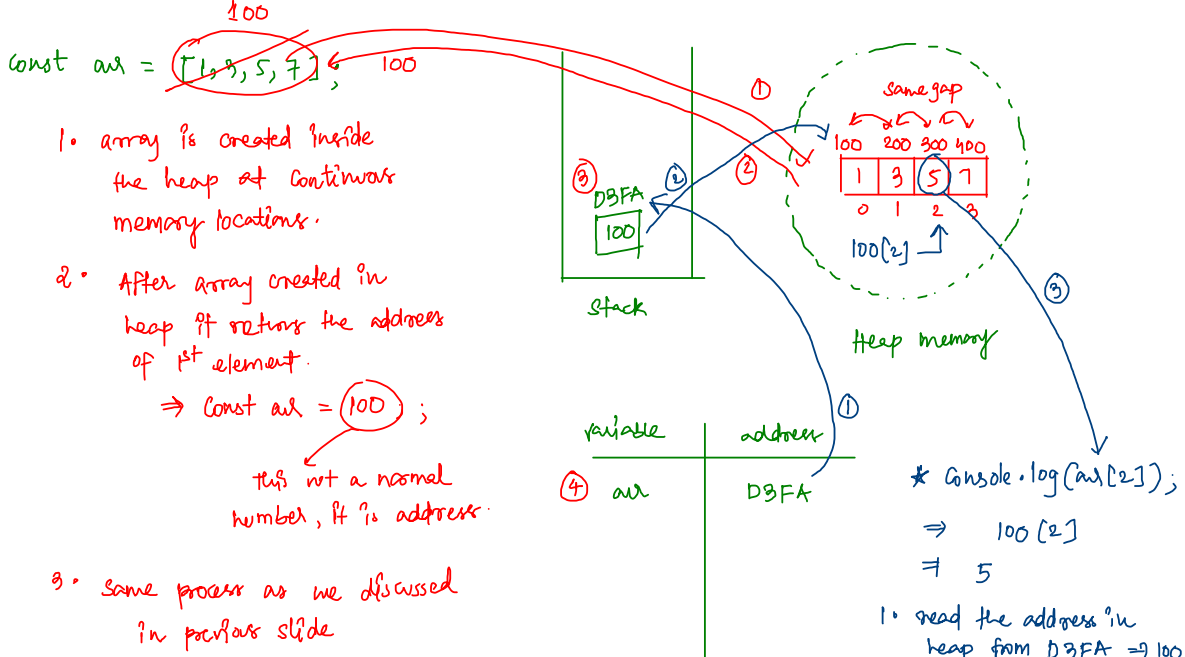
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (10), B (11), C (12), D (13), E (14), F (15)

3 2 1 0

D3FA

$\Rightarrow 16^0 \times A + 16^1 \times F + 16^2 \times 3 + 16^3 \times D$

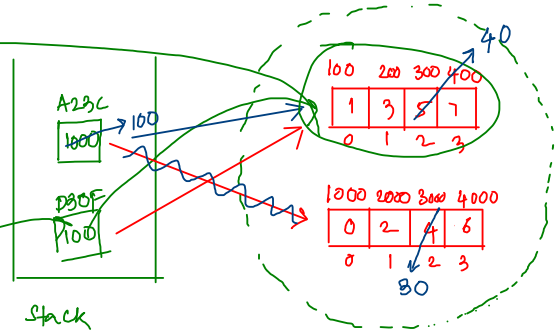
$= 16^0 \times 10 + 16^1 \times 15 + 16^2 \times 3 + 16^3 \times 13$


100
const arr = [1, 3, 5, 7];
100
1. array is created inside the heap at continuous memory locations.
2. After array created in heap it returns the address of 1st element.
⇒ const arr = 100 ;
this not a normal number, it is address.
3. same process as we discussed in previous slide
same gap
100 200 300 400
1 3 5 7
0 1 2 3
100[2]
① ② ③
D3FA
100
stack
Heap memory
variable
address
④ arr
D3FA
* console.log(arr[2]);
⇒ 100[2]
⇒ 5
1. read the address in heap from D3FA ⇒ 100
2. go to 100 in heap
3. 100[2] ⇒ 5

* Example :

```
775  let arr = [1, 3, 5, 7];
776  console.log(arr);
777
778  let brr = [0, 2, 4, 6];
779  brr[2] = 30;
780
781  brr = arr;
782  arr[2] = 40;
783  console.log(arr);
784  console.log(brr);
```

Vaiable	address
arr	D30F
brr	A23C



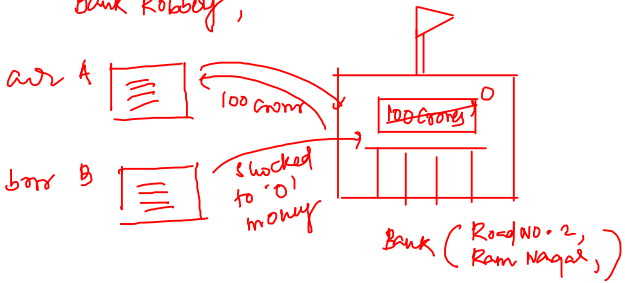
* Even though arr, brr are different variables they are looking at same array in the heap.

* 779. brr[2] = 30
  ⇒ 1000[2] = 30
* 781. brr = arr
  ⇒ brr = 100
  arr ⟶ 100 [1,3,5,7]
  brr ⟶ 1000 [0,2,4,6]

* 782. arr[2] = 40
  ⇒ 100[2] = 40
* 783. [1, 3, 40, 7] ✓
* 784. [1, 3, 40, 7] ✓

⇒ arr ⟶ 100 [1,3,5,7]
brr ⟶ 100

Bank Robbery,



1. A says to B

→ let's Rob Road no. 2 Ram Nagar at 2:00AM

2. A is cunning / greedy,

→ He alone rob's bank at 1:00 AM

3. Now when B goes to bank at 2:00 AM

→ the Bank will be empty

★ same name doesn't imply same variable

Pass By Value

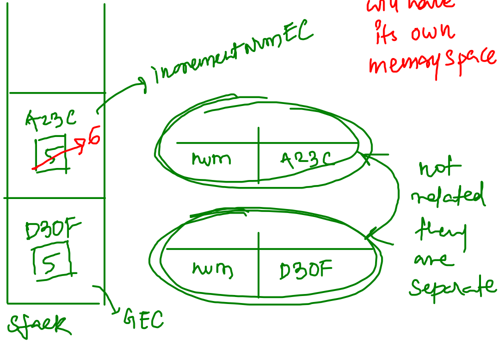
num = 5

```javascript
function incrementNum(num) {
  num++;
}

let num = 5;
incrementNum(num);
console.log(num);
```

5 → simple number value

op: 5

★ Every function / EC will have its own memory space



Pass By Reference

arr = 100

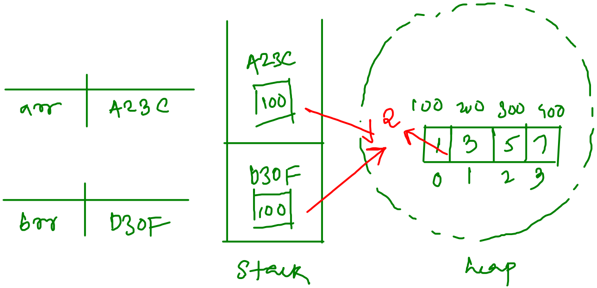
```javascript
function incrementArr(arr) {
  arr[0]++;
}

let brr = [1, 3, 5, 7];
incrementArr(brr);
console.log(brr);
```

100 address

[1, 3, 5, 7] — wrong assumption

arr[0]++ ⇒ 100[0]++

op: [2, 3, 5, 7]



★ Changes are reflected after function call.