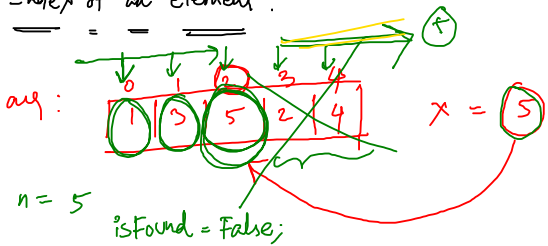
Q: Index of an element:

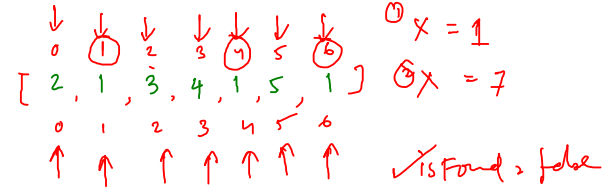


op: index

-1 → not found

```
for (let i = 0; i<n; i++) {
    if (arr[i] == x) {
        isFound = True;
        console.log("found at " + i);
        break; → only 1st occurrence
    }
}
if (isFound = False) {
    console.log(-1);
}
```

```
function indexOfElement(N, X, arr) {
  let isFound = false;
  for (let i = 0; i < N; i++) {
    if (arr[i] == X) {
      isFound = true;
      process.stdout.write(i + 1 + " ");
    }
  }

  if (isFound == false) {
    console.log(-1);
  }
}
```



* Linear search (n → log of all)
→ n iterations
n checks

① 2 5 7

② -1

i = 0
i = 1
i = n
.
.
.

Q: Reverse an Array:

	0	1	2	3	4
arr:	1	2	3	4	5
op:	0 5	1 4	2 3	3 2	4 1

#1:

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

-1>=0 ✗

↓	↓	↓	↓	↓	↓
-1	0	1	2	3	4
	1	2	3	4	5

n = 5

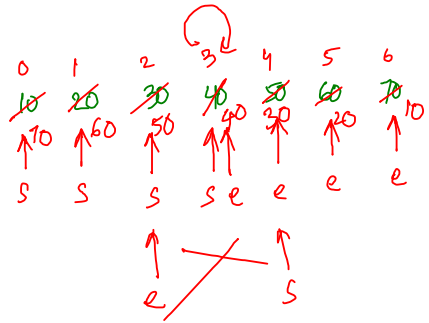
i = ~~4~~ ~~3~~ ~~2~~ ~~1~~ ~~0~~ -1

~~[5, 4, 3, 2, 1]~~

→ n iterations

⇒ using an additional array

#8: Two pointer Approach



start = 0

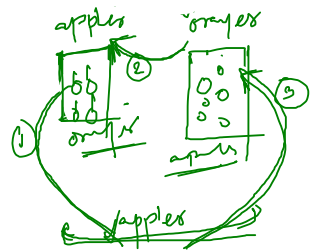
end = 6

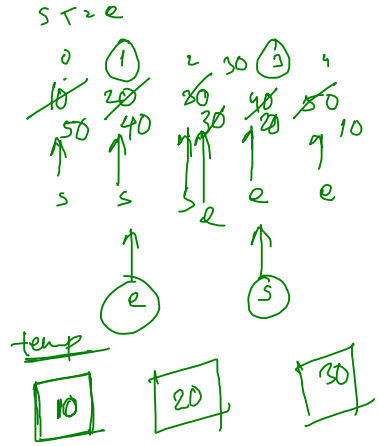
swap (arr[s], arr[end])

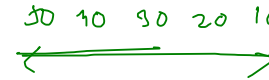
s++

e--

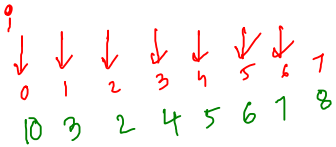
```
function reverseArray(arr, start, end) {
  while (start <= end) {
    let temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;
    start++;
    end--;
  }

  return arr;
}
```







9:30 pm
- 9:45 pm
Break




```
if (arr[i] % 2 == 0) {
    if (prev_even_idx != -1) {
        diff = i - prev_even_idx
        ans = min(ans, diff)
    }
    prev_even_idx = i
}
```

p_even_idx = -1 ∅ 2 ∅ 5

ans = ∅ 2 1

Q: 2nd largest element :

(Mod 2 sorting revisit)

0	1	2	3	4	5	6	7
3	1	2	5	5	4	5	3

→ arr

Op: 4

```
function SecondLargest(arr, n) {
  let firstMax = -Infinity;
  for (let i = 0; i < n; i++) {
    if (arr[i] > firstMax) {
      firstMax = arr[i];
    }
  }

  let secondMax = -Infinity;
  for (let i = 0; i < n; i++) {
    if (arr[i] != firstMax && arr[i] > secondMax) {
      secondMax = arr[i];
    }
  }

  console.log(secondMax);
}
```

① find 1st Maximum → 5

② run the find max by ignoring 1st Max

[3, 1, 2, ~~5~~, ~~5~~, 4, ~~5~~, 3]

max_1 = 5

max_ele = -∞, 3, 4

0	1	2	3	4	5	6	7
3	1	2	5	5	4	5	3

fM = 5

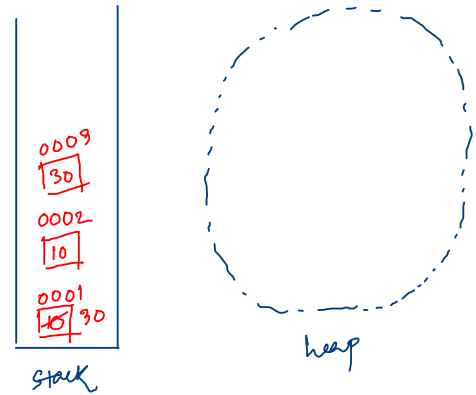
SM = -∞, 3, 4

max after removing 1st max will give you 2nd max

: How arrays work internally :

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

variable name	address	value
a	0001	~~10~~ 30
b	0002	10
c	0003	30



* primitives use stack space

* objects / arrays are reference variables and use heap space

address → hexadecimal

arr[0]
0001[0] → 1

let
~~Const~~ arr = [1, 3, 5, 7];
(reference variables)

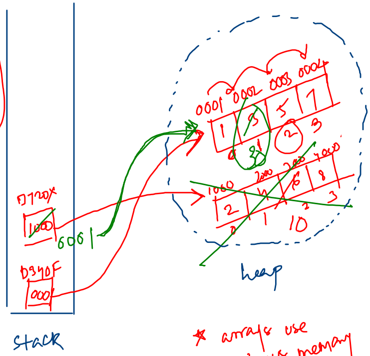
arr will not have the entire array instead it will store address of 1st element.

let
~~const~~ brr = [2, 4, 6, 8];

brr[2] = 10;   1000[2] = 10

✓ brr = arr;
✓ arr[2] = 30; ⇒ 0001[2] = 30;
✓ console.log(brr[2]); // 30;

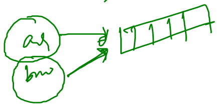
a = 10
b = 20
b = a

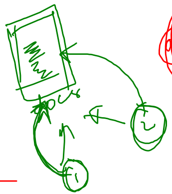


* arrays use continuous memory locations.

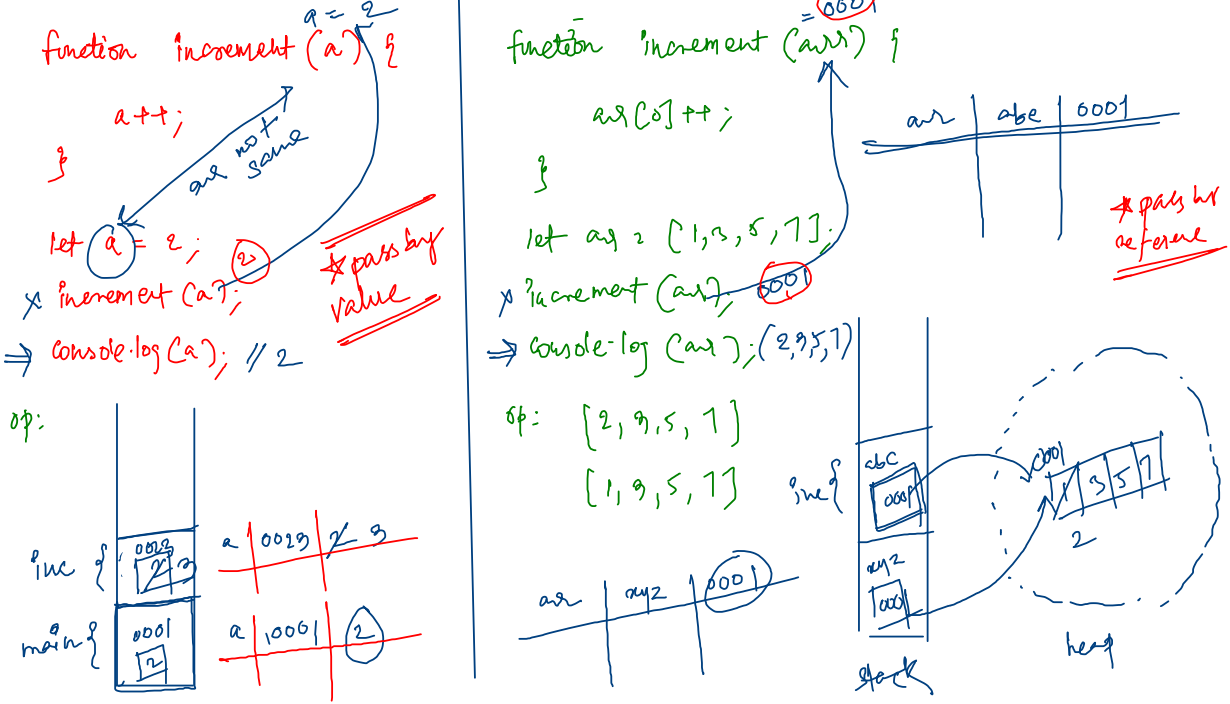
* the array can be changed only the reference is const

arr[2]
0001[2] → 5





Variable	address	value
arr	D340F	0001
brr	D720X	~~1000~~ 0001


function increment (a) {
a++;
}
let a = 2;
increment (a);
console.log (a); // 2
a = 2
are not same
2
* pass by value
op:
inc
main
0002
2
3
0001
2
a | 0002 | 2 | 3
a | 1000 | 2
function increment (arr) {
arr[0]++;
}
let arr = [1,3,5,7];
increment (arr);
console.log (arr); (2,3,5,7)
op: [2,3,5,7]
[1,3,5,7]
=0001
0001
arr | abc | 0001
* pass by reference
arr | xyz | 0001
inc
abc
0001
xyz
0001
1
3
5
7
2
heap
stack