★ Max diff b/w any 2 elements : [optimal Approach]

```
617  function ArrayProblem(arr) {
618    // Write your code here
619    let maxDiff = -Infinity;
620    for (let i = 0; i < n; i++) {
621      for (let j = i + 1; j < n; j++) {
622        const diff = Math.abs(arr[i] - arr[j]);
623        if (diff > maxDiff) {
624          maxDiff = diff;
625        }
626      }
627    }
628
629    console.log(maxDiff);
630  }
```

$n = 4$

① $i=0$, $j = 1, 2, 3, \cancel{x} \rightarrow 3$

② $i=1$, $j = 2, 3, \cancel{x} \rightarrow 2$

③ $i=2$, $j = 3, \cancel{x} \rightarrow 1$

④ $i=3$, $j = \cancel{x} \rightarrow 0$

Total iterations = Total pairs

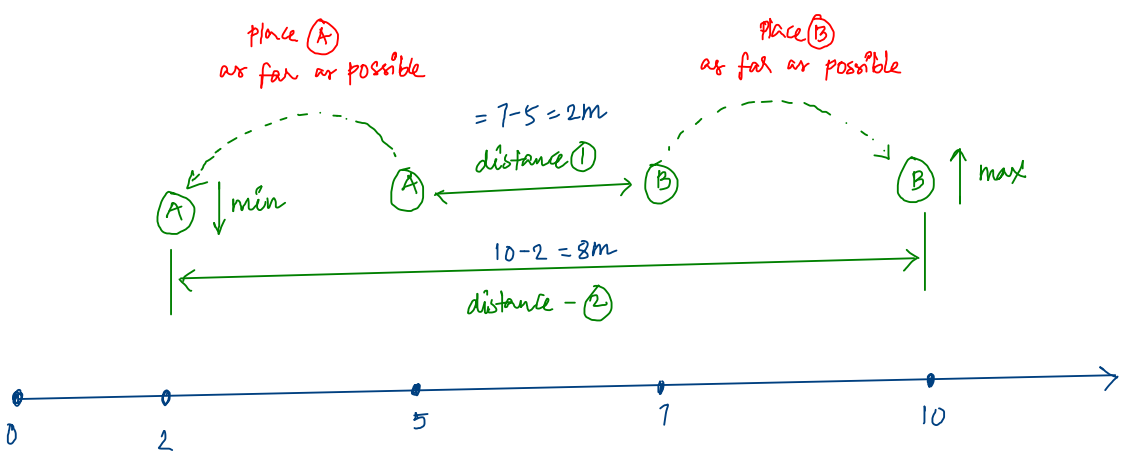
$= 3+2+1+0$

$= 6 = \frac{n(n-1)}{2} = \frac{n^2-n}{2} \simeq n^2$

→ generated all pairs

→ Calculated diff of every pair

→ keep track of max diff encountered so far.



$\Rightarrow$ distance - ② > distance - ①

diff b/w two elements = distance B/w them

★ max Diff = max Ele − min Ele

$\Rightarrow$ n iterations

* 2nd Largest Element :

Eg: [3, 1, 2, 5, 5, 4, 5, 3] (0 1 2 3 4 5 6 7)

Op: 4

① removed 1st largest
② find largest among remaining ⇒ 2nd largest

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

1st largest = 5

⇒ [3, 1, 2, 4, 3]

⇒ largest = 4 ⇒ 2nd largest

Now how do I remove 1st largest?

→ we will not remove we will just ignore that

1. find firstMax
2. find Maximum again by ignoring ele = firstMax

① firstMax = 5

[3, 1, 2, 5, 5, 4, 5, 3] (0 1 2 3 4 5 6 7)

skip skip

3 > -∞ | 1 > 3 | 2 > 3 | 4 > 3 | 3 > 4

maxEle = -∞ & 4

= 2nd largest

* Reverse an Array :

Eg: [1, 2, 3, 4, 5]

Op: [5, 4, 3, 2, 1]

```
function reverse (arr) {
    /* code */
}

const arr = [1, 2, 3, 4, 5];
reverse (arr);
console.log (arr) // [5, 4, 3, 2, 1]
```

#1 :

* this approach uses an extra array.

extra space

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

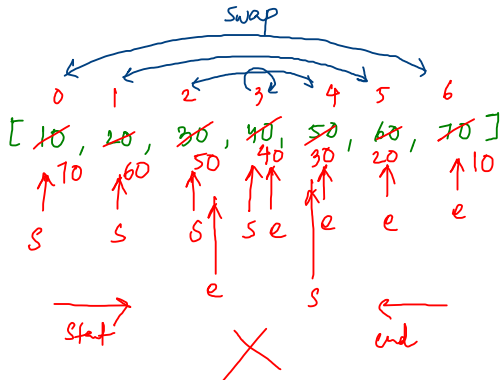
[ ] ~~[5]~~ ~~[5,4]~~ ~~[5,4,3]~~ ~~[5,4,3,2]~~ [5, 4, 3, 2, 1]

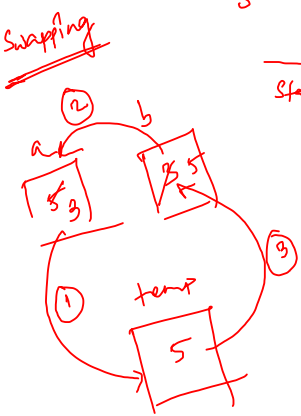
```
for (let i = 0; i < n; i++) {
    arr[i] = rev_arr[i];
}
```

* we need change in original given array hence copy rev_arr to arr.

#2 : Two pointer Approach

In-place solution






```
let start = 0;
let end = n-1 = 6

while (start <= end) {
    swap (arr[start], arr[end]);
    start++;
    end--;
}

let temp = arr[start];
arr[start] = arr[end];
arr[end] = temp;
```

★ How Arrays Work Internally ?

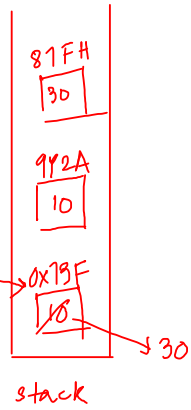
```
let a = 10;
console.log(a);
let b = a;
let c = 30;
a = c;
```

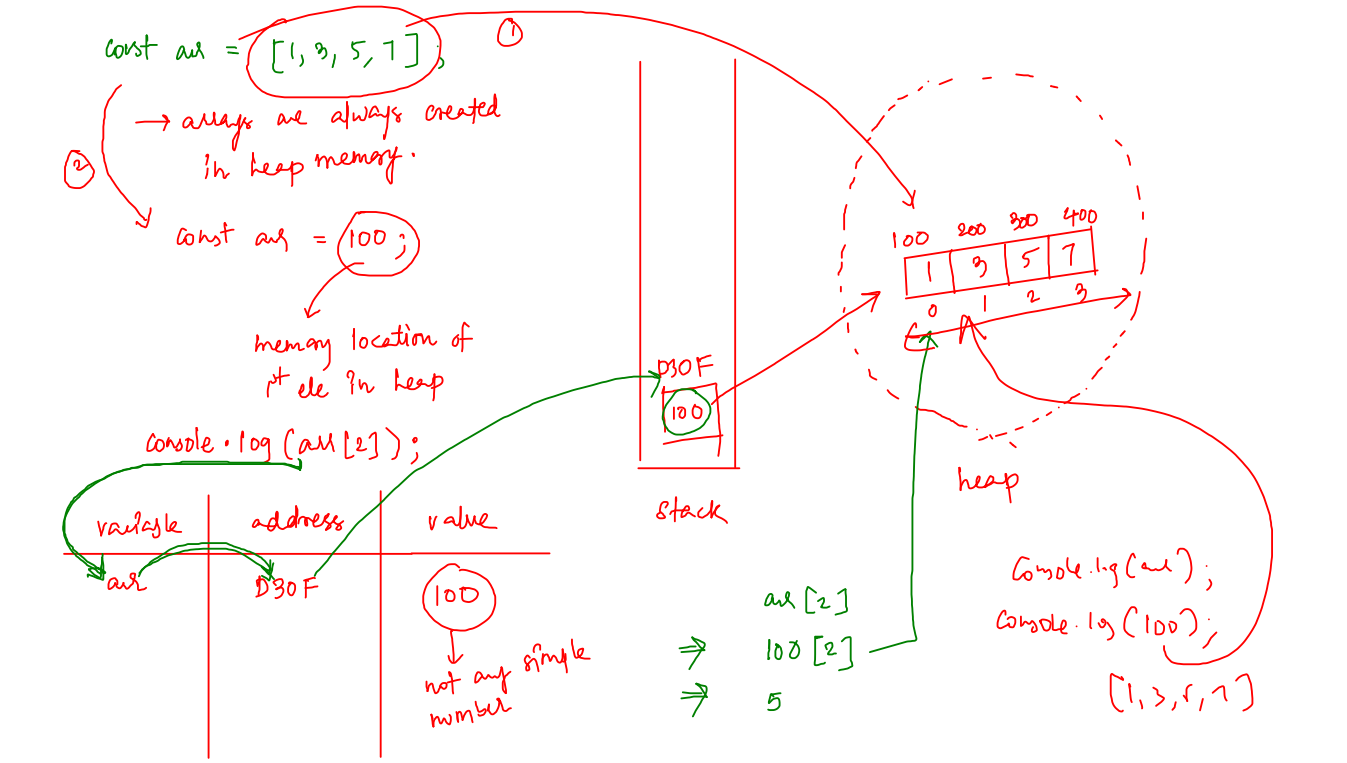
variable	address
a	0X73F
b	9Y2A
c	87FH

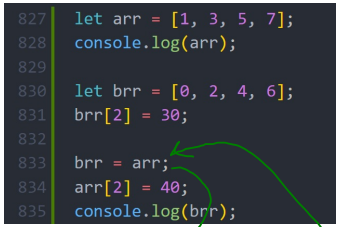
random assumption

address ⇒ hexadecimal system (base-16)

(0, 1, 2, ... 9, A, B, C, D, E, F)

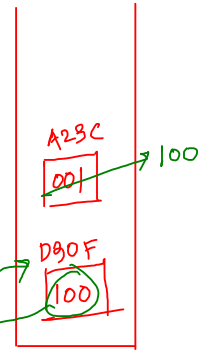


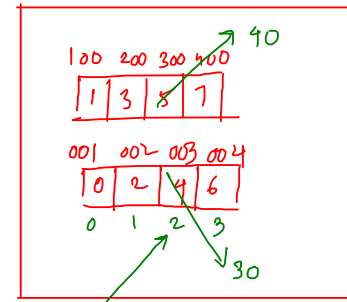

const arr = [1, 3, 5, 7];
1
2
→ arrays are always created in heap memory.
const arr = 100;
memory location of 1st ele in heap
console.log(arr[2]);
variable
address
value
arr
D30F
100
not any single number
D30F
100
stack
100
200
300
400
1
3
5
7
0
1
2
3
heap
arr[2]
⇒ 100[2]
⇒ 5
Console.log(arr);
Console.log(100);
[1,3,5,7]



100

⇒ [1, 3, 40, 7]





Variable	address
arr	D30F
brr	A23C

brr[2] = 30

⇒ 001[2] = 30

* brr = arr

⇒ brr = 100

* arr[2] = 40

⇒ 100[2] = 40

Pass By Value

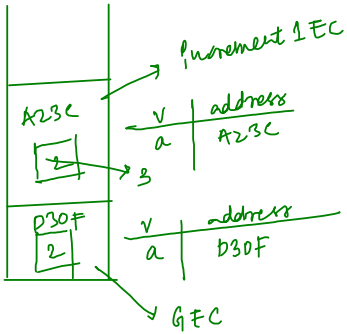
```
function increment1(a) {
  a++;
}

let a = 2;
increment1(a);
console.log(a);
```

= 2

changes are not reflected

2

number/value



* Same names doesn't imply they are same locations

Pass by Reference

```
function increment2(arr) {
  arr[0]++;
}

let arr = [1, 3, 5, 7];
increment2(arr);
console.log(arr);
```

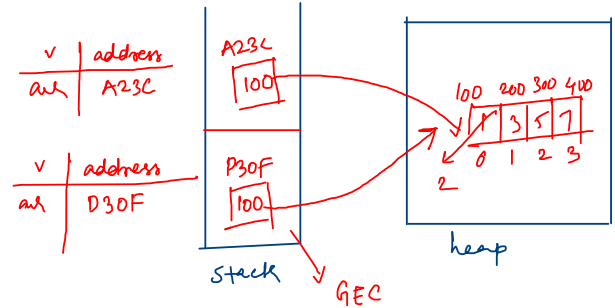
= 100

changes are reflected

100

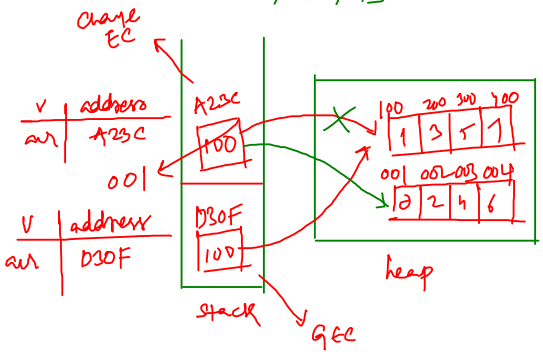
reference/address

→ [2, 3, 5, 7]

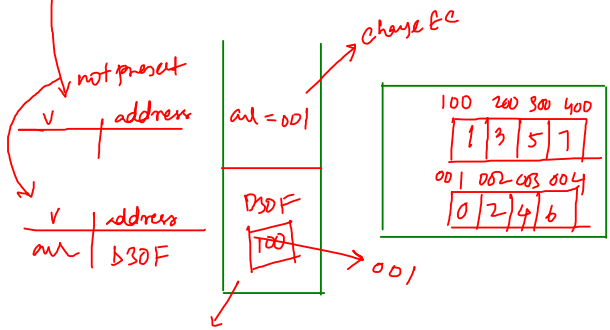


```
857  function change(arr) {
858    arr = [0, 2, 4, 6];
859  }
860
861  let arr = [1, 3, 5, 7];
862  change(arr);
863  console.log(arr);
```

⇒ [1, 3, 5, 7]




```
857  function change() {
858    arr = [0, 2, 4, 6];
859  }
860
861  let arr = [1, 3, 5, 7];
862  change(arr); change();
863  console.log(arr);
```



★ Due to Scope Chain.