* Min distance B/w two positive Even :

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

⇒ 2, 4 ⇒ (0, 1) ⇒ 1 ✓

2, 6 ⇒ (0, 3) ⇒ 3

4, 6 ⇒ (1, 3) ⇒ 2

```
function ArrayProblem6(n, arr) {
  // Write code here
  let minDis = Infinity;
  for (let i = 0; i < n; i++) {
    if (arr[i] % 2 != 0) {
      continue;
    }
    for (let j = i + 1; j < n; j++) {
      if (arr[i] > 0 && arr[i] % 2 == 0 && arr[j] > 0 && arr[j] % 2 == 0) {
        minDis = Math.min(minDis, j - i);
      }
    }
  }

  // if none of the pairs are positive even
  if (minDis == Infinity) {
    return -1;
  }
  return minDis;
}
```

1. I am thinking to use generate pairs,

2, 4 4, 1 1, 6
2, 1 4, 6 1, 7
2, 6 4, 7
2, 7

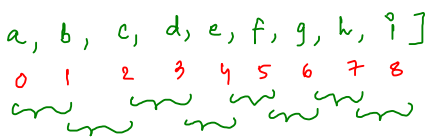
2. But I only need even pairs, How to handle

→ only pick the pair if both ele are even and positive.

if (arr[i] % 2 == 0 && arr[j] % 2 == 0 && arr[i] > 0 && arr[j] > 0)

efficient/optimal

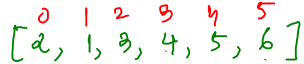
→ What kind of pairs will give you minimum distance ? ⇒ adjacent pairs (side-by-side)




```
let prevEvenNumIdx = -1;
let minDis = Infinity;
for (let i = 0; i < n; i++) {
  if (arr[i] > 0 && arr[i] % 2 == 0) {
    // check if prev even number exists and then update dis
    if (prevEvenNumIdx != -1) {
      minDis = Math.min(minDis, i - prevEvenNumIdx);
    }
    prevEvenNumIdx = i;
  }
}

// if none of the pairs are positive even
if (minDis == Infinity) {
  return -1;
}
return minDis;
```

→ Now you want min dist +ve even pairs ?



(2, 6) can never be min distance because nearest even number for 2 ⇒ 4

→ how to find nearest even efficiently ?

⇒ keep track previously seen even positive element

★ Sorted Inserted position :

eg: [1, 3, 5, 7, 9] , target = 6

(0 1 2 3 4)

If you insert (6)

⇒ [1, 3, 5, (6), 7, 9]

(0 1 2 3 4 5)

op: 3

or eaval

⇒ Just find the 1st greater element of target

7, 9 > 6

But 7 is the 1st greater

i=0 1≥6 ×

i=1 3≥6 ×

i=2 5≥6 ×

i=3 7≥6 ✓ ⇒ ans = (3)

```
for(let i=0; i<n; i++) {
    if (arr[i] >= target) {
        return i;
    }
}
return n;  handles 2nd case
```

① [1, 2, 3, 4, 5, 6] ⇒ insert(0)

⇒ [0, 1, 2, 3, 4, 5, 6] ⇒ op: 0

(0)

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

⇒ [1, 2, 3, 4, 5, (6)] ⇒ op: 5 (N)

(0 1 2 3 4 5)

★ 2nd Largest !

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

op: 4

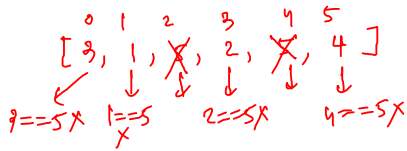
⇒ (3, 1, 5, 2, 5, 4)

⇒ remove 1st max ⇒ (3, 1, 2, 4)

⇒ max ele ⇒ 4 (second max)

① find the firstmax = 5

② find the max ele but avoid the elements which are equal firstmax



max ele = ~~-∞~~ ~~3~~ 4

```
1100  function SecondLargest(arr, n) {
1101    // Write code here
1102    let firstMax = -Infinity;
1103    for (let i = 0; i < n; i++) {
1104      firstMax = Math.max(firstMax, arr[i]);
1105    }
1106
1107    let secondMax = -Infinity;
1108    for (let i = 0; i < n; i++) {
1109      if (arr[i] != firstMax) {
1110        secondMax = Math.max(secondMax, arr[i]);
1111      }
1112    }
1113
1114    console.log(secondMax);
1115  }
```

★ follow-up qs : 3rd Max - repeat same process by avoiding 1st max, 2nd max

kth max → sorting

→ min heap Datastructure ★

★ Reverse an array :

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

op: [50, 40, 30, 20, 10]

```
const arr = [10, 20, 30, 40, 50]
console.log(arr); ⟶ [10, 20, 30, 40, 50]
/* code } solution
*/
console.log(arr); ⟶ [50, 40, 30, 20, 10]
```

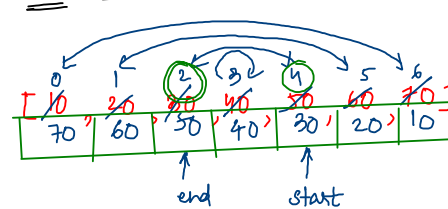
★★ ⇒ the value / addr of elements in original array must be changed.

#1: Iterate in reverse :

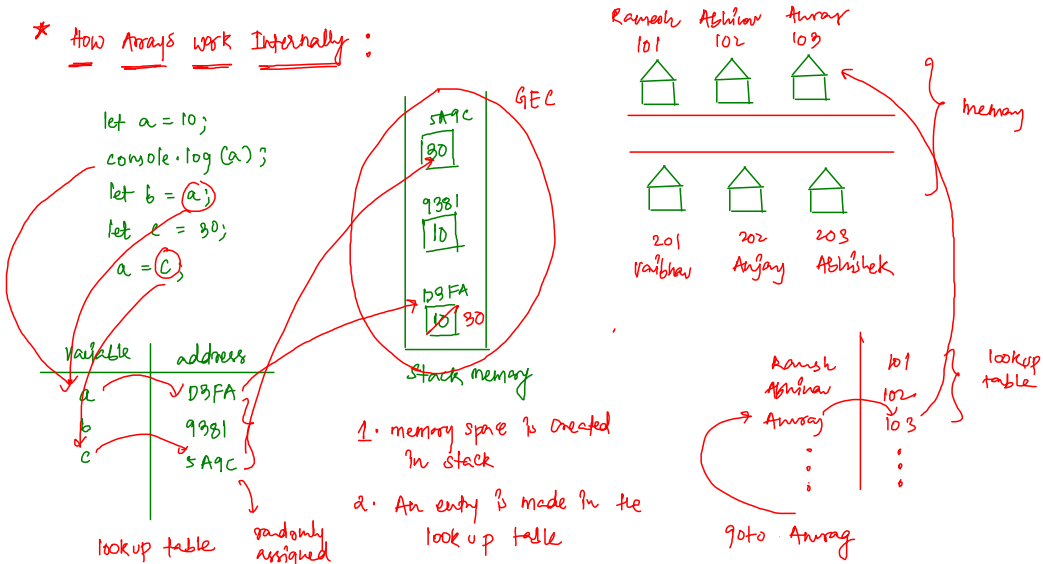
```
for(let i=n-1; i>=0; i--) {
    cl(arr[i]);
}
```

→ It is printing in reverse
But doesn't change the given original array.

#2: Two pointer approach :



* How Arrays work Internally :



1. memory space is created in stack

2. An entry is made in the look up table

randomly assigned by computer

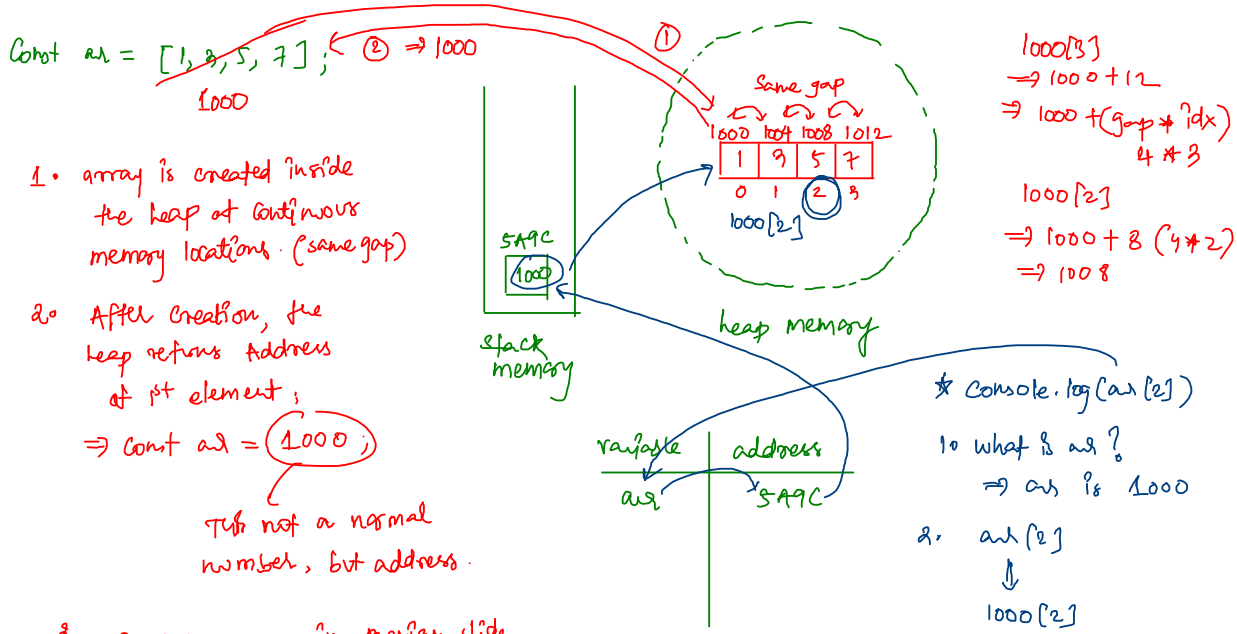
(Hexadecimal base - 16) ⇒ 0, 1, 2, 3, . . . , 9, A, B, C, D, E, F (10 11 12 13 14 15)

Eg: 5A9C

decimal ⇒ $16^0 \times C + 16^1 \times 9 + 16^2 \times A + 16^3 \times 5$

⇒ $16^0 \times 12 + 16^1 \times 9 + 16^2 \times 10 + 16^3 \times 5$

⇒ ____


Const ar = [1, 9, 5, 7];
1000
② ⇒ 1000
①
same gap
1000 1004 1008 1012
1 9 5 7
0 1 2 3
1000[2]
1. array is created inside the heap at continuous memory locations. (same gap)
2. After creation, the heap returns Address of 1st element;
⇒ Const ar = 1000;
This not a normal number, but address.
3. Same process as in previous slide (stack, look up table)
stack memory
5A9C
1000
heap memory
variable
address
ar
5A9C
1000[3]
⇒ 1000 + 12
⇒ 1000 + (gap * idx)
4 * 3
1000[2]
⇒ 1000 + 8 (4*2)
⇒ 1008
* console.log(ar[2])
1. what is ar?
⇒ ar is 1000
2. ar[2]
1000[2]
5

* Why heap only sends 1st ele address ?

4 close friends,
they are buying apartments

You are a mutual friend to all those 4 friends,
one of them invited to their house,
⇒ flat NO: 101, 1st first floor

If you wat to visit all friends,
you being smart, call to 1st friend ask his flat NO:
⇒ reply: 101.

* 1st address + gap * idx

* array is a datastructure where values are stored in continuous memory locations.

Example :

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

let brr = [0, 2, 3, 6];
brr[2] = 30;

brr = arr;
arr[2] = 40;
console.log(arr);
console.log(brr);
```