31   0

1010101010....10

n=0

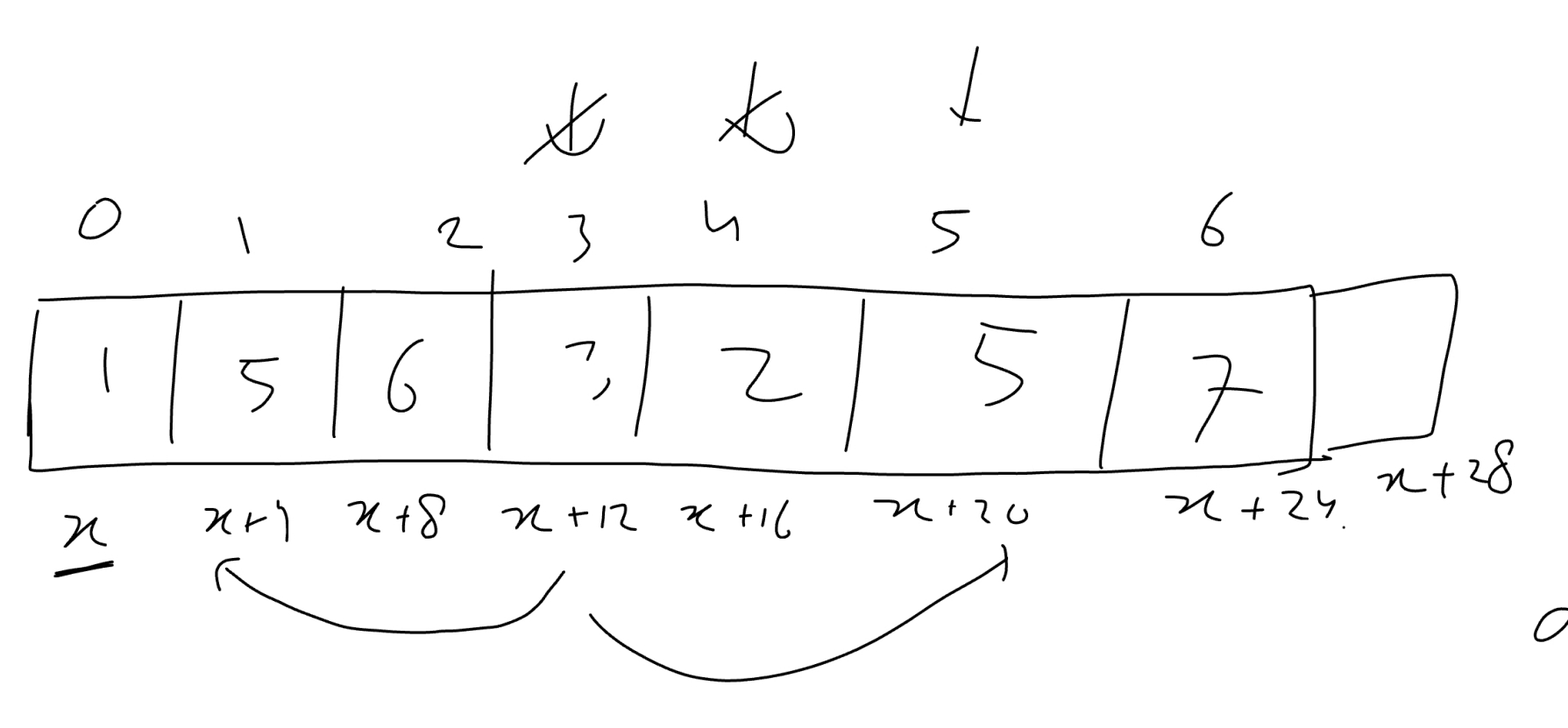
(i=1 ; i ≤ 31 ; i+=2)

n = (n | (1 << i))

Array

[1, 5, 6, 3, 2, 5, 7]

Contiguous Memory Allocation



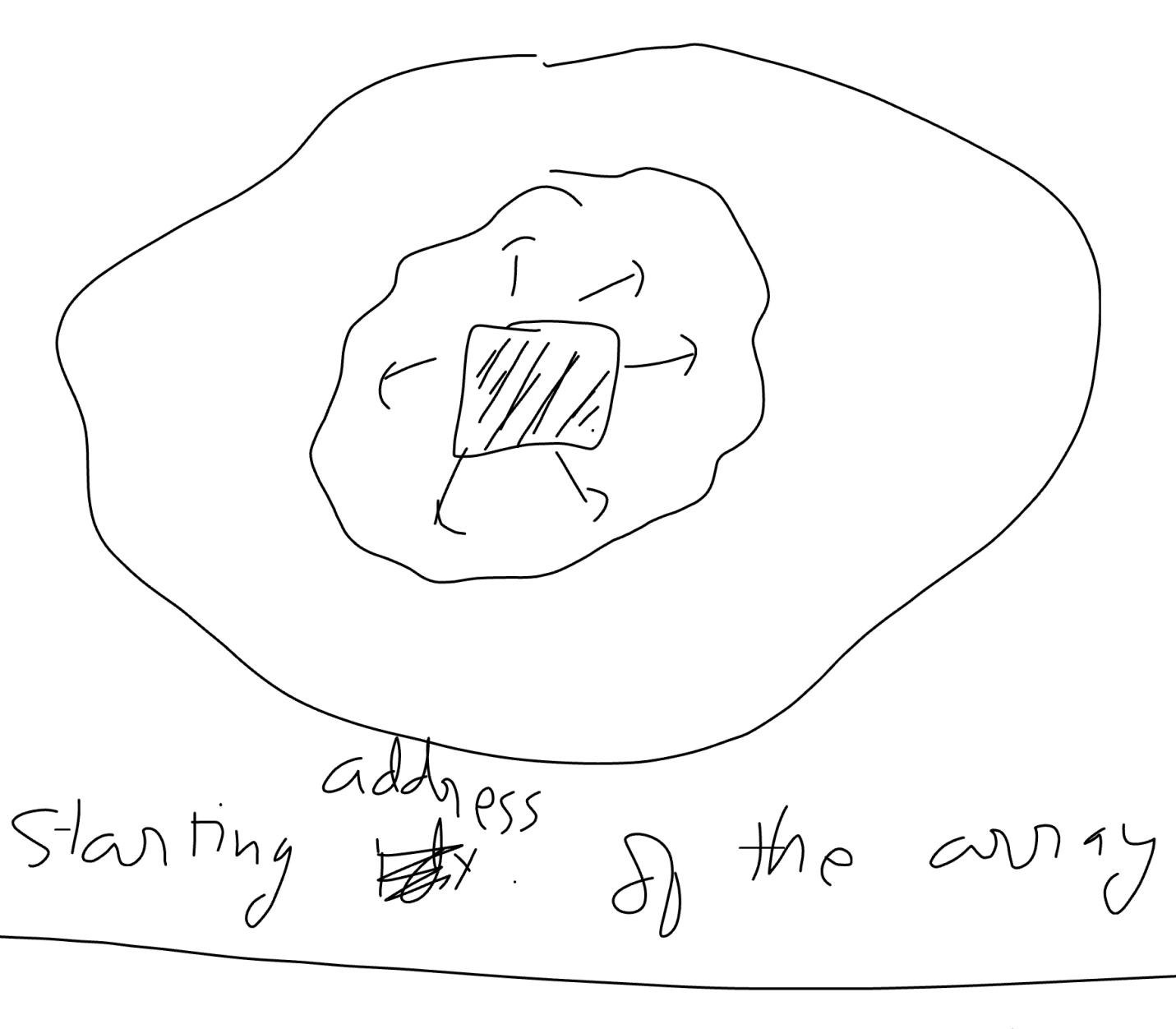
0
1
2
3
4
5
6
1
5
6
3
2
5
7
x
x+4
x+8
x+12
x+16
x+20
x+24
x+28

arr[100]

Adv.

1) Random Access → O(1) time
2) It makes arrays cache-friendly

Locality of References → IrP



NOTE: Starting ~~idx~~ address of the array always remains constant.

Insert at the end: O(1) on avg
O(n) in the worst-case

Insert at the begin: O(n)

Insert at the middle: O(n)

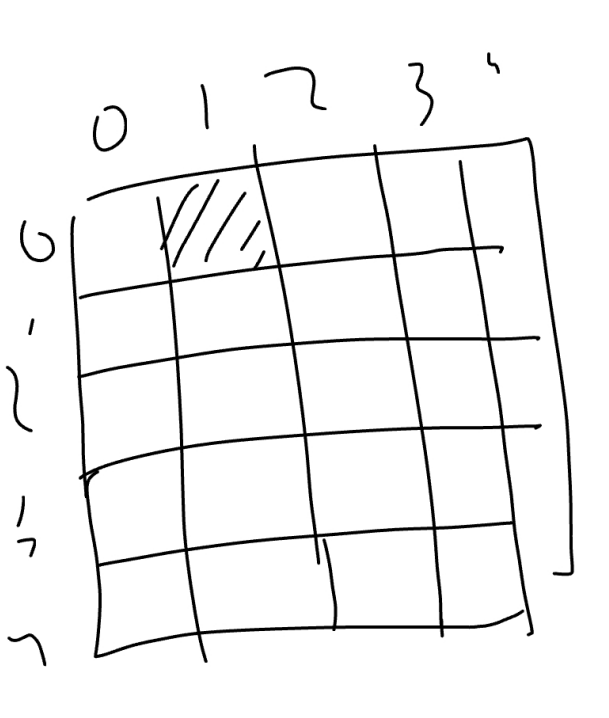
Delete at the end: O(1).

Delete at the begin: O(n).

Delete at the middle: O(n).

Random Access: O(1)

arr = [[[. .])
[ ]
[ ]
:
[ ]]



arr[0][1]

[3, 5, 10, 4, 7, 3]

o/p = 1

```
to_search = 1
while (true) {
    search for to_search;
    if found:
        to_search++;
    else:
        STOP
}
```

TC: $O(n^2)$
AS: $O(1)$

Binary-Search-Based AP

TC: $O(n \log n + n * \log n)$
$= O(n \log n)$
AS: $O(1)$

HashMap-Based AP

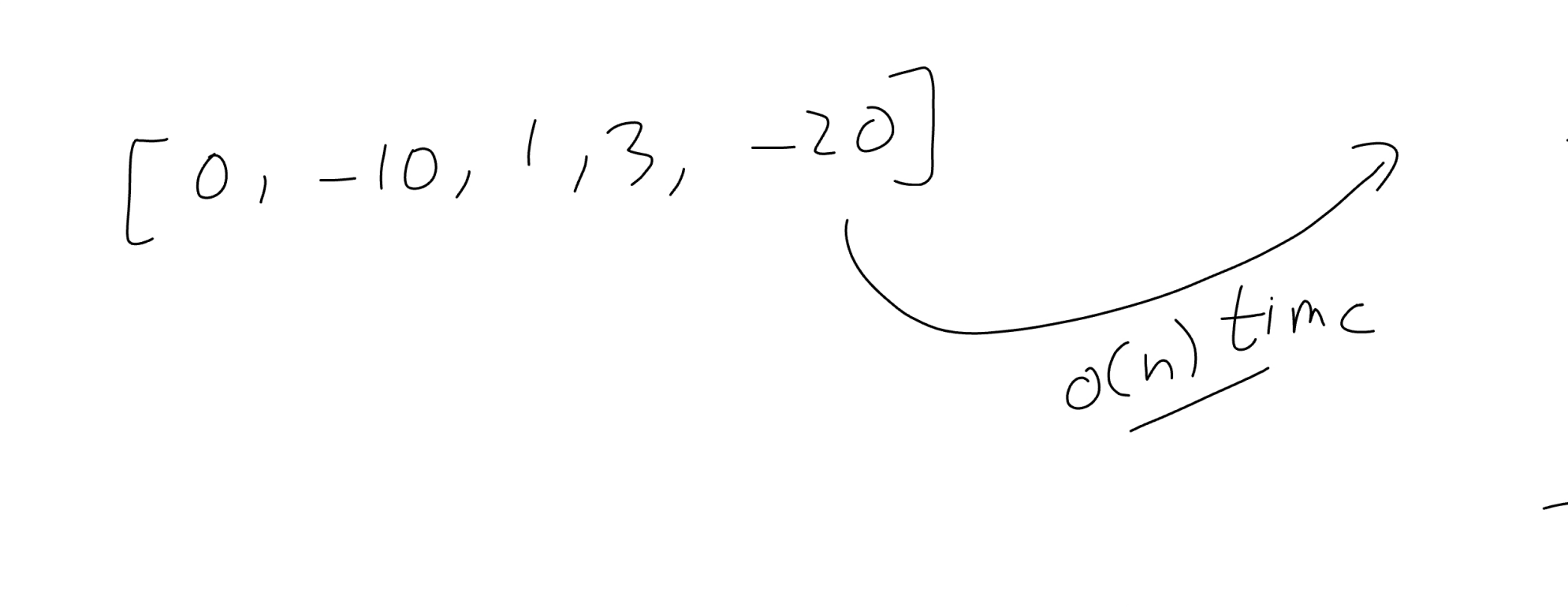
isPresent {int → bool}

[0, -10, 1, 3, -20] → O(n) time

0 → True
-10 → True
1 → True
3 → True
-20 → True

Get-op also takes O(1) time

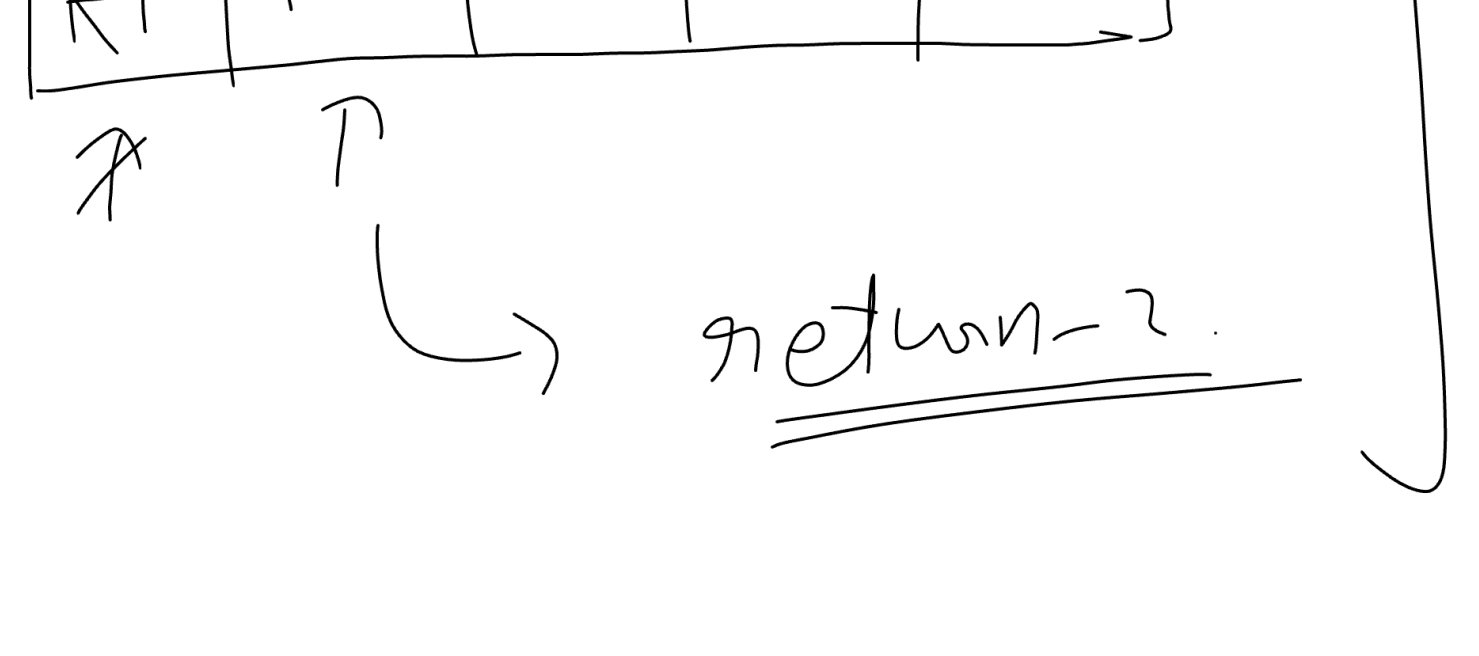
TC: O(n)
AS: O(n)



0
1
2
3
4
arr:
100
+10
1
3
+20
isPresent:
~~F~~T
F
~~F~~T
F
F
return 2

TC = O(n)
AS = O(n)

Bucketing-Based AP



0
1
2
3
4
-100
10
-1
3
20
n = 5
return 2

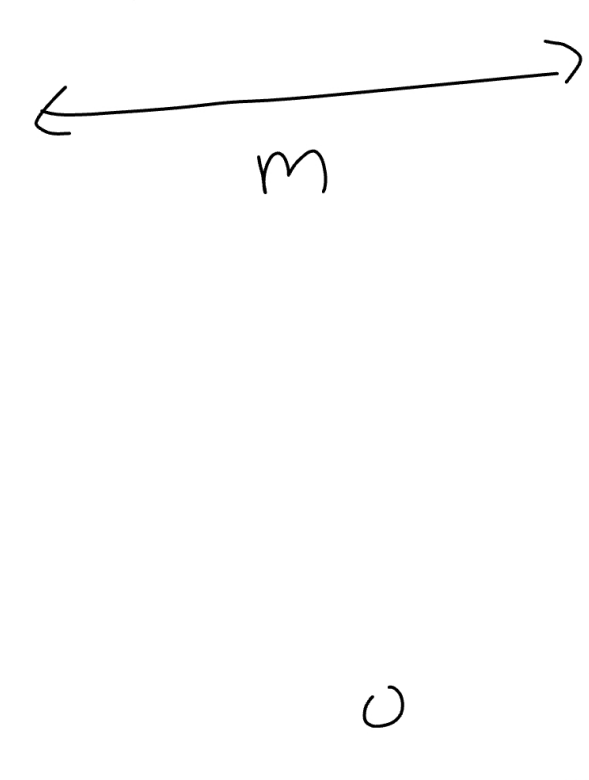
Bucket index of x = x-1

TC = O(n)
AS = O(1)

n=5

[0, -10, 1, 3, -20] → [-6, 6, -1, 3, 6].

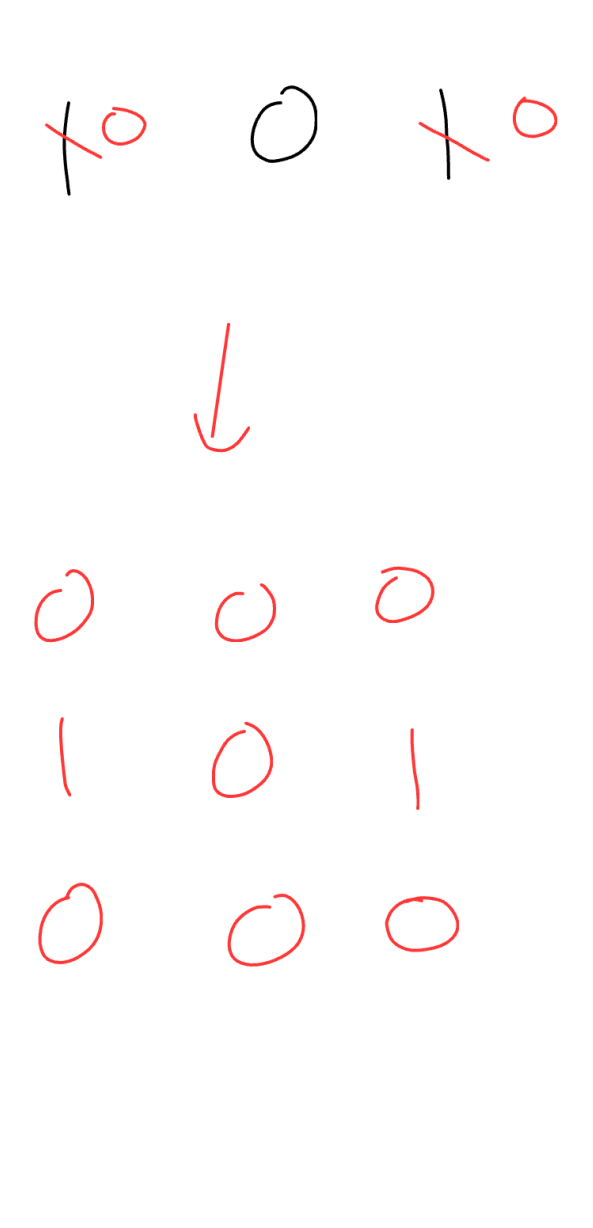
[-6, -5, -1, 3, -2, -2] n=6



0
1
2
0
1
2
1 0 1
1 1 1
1 0 1
n
m

(0, 1)
(2, 1)

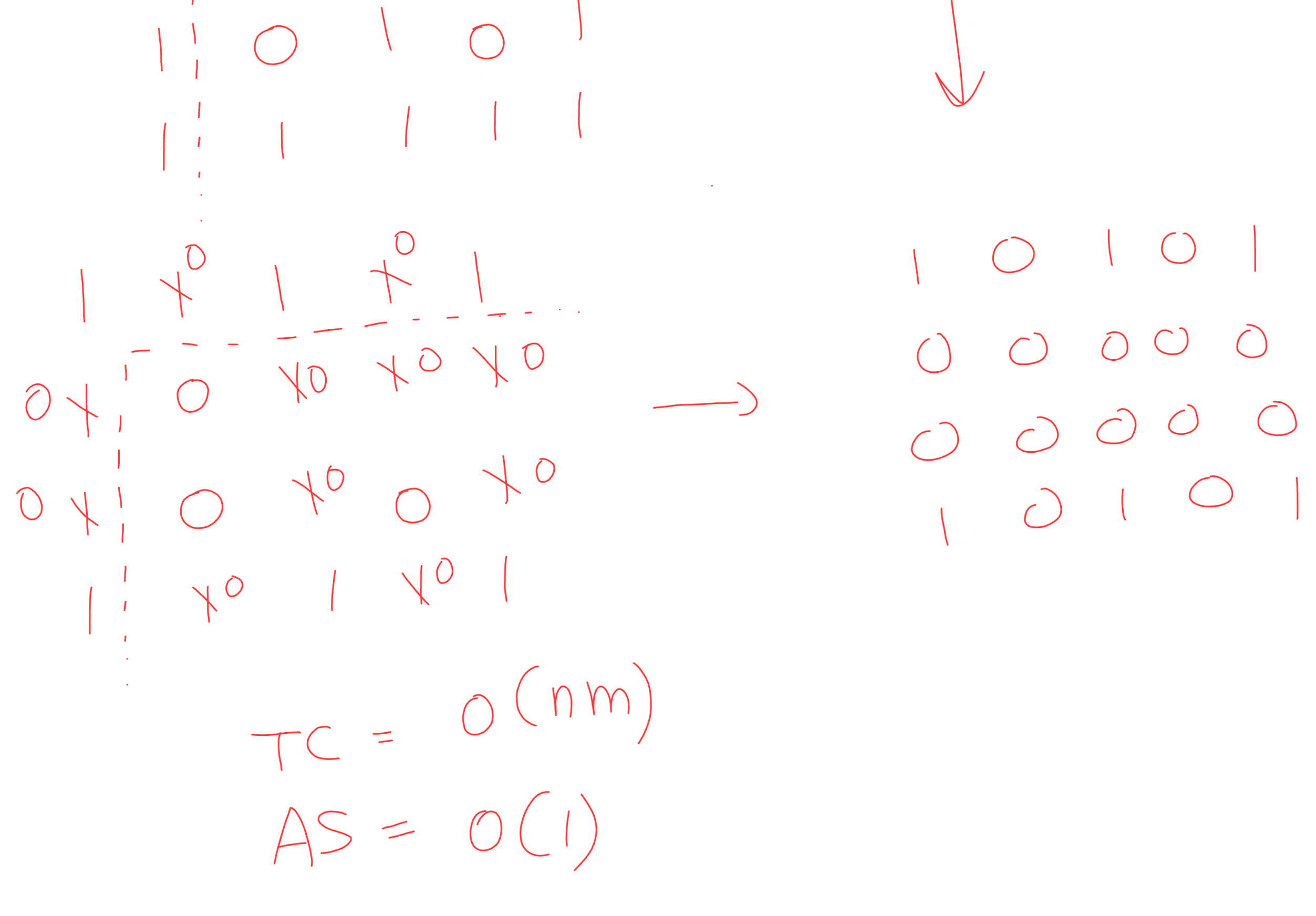
TC = $O(nm(n+m))$
AS = $O(nm)$



0
1
2
0
1
2
0X O X0
1 X0 1
X0 O X0
0 0 0
1 0 1
0 0 0

	0	1	2
rows:	T	F	T
cols:	F	T	F

TC = O(nm)
AS = O(n+m)



1 1 1 1 1
1 0 1 1 1
1 0 1 0 1
1 1 1 1 1
1 X0 1 X0 1
0X 0 X0 X0 X0
0X 0 X0 0 X0
1 X0 1 X0 1
1 0 1 0 1
0 0 0 0 0
0 0 0 0 0
1 0 1 0 1

TC = O(nm)
AS = O(1)