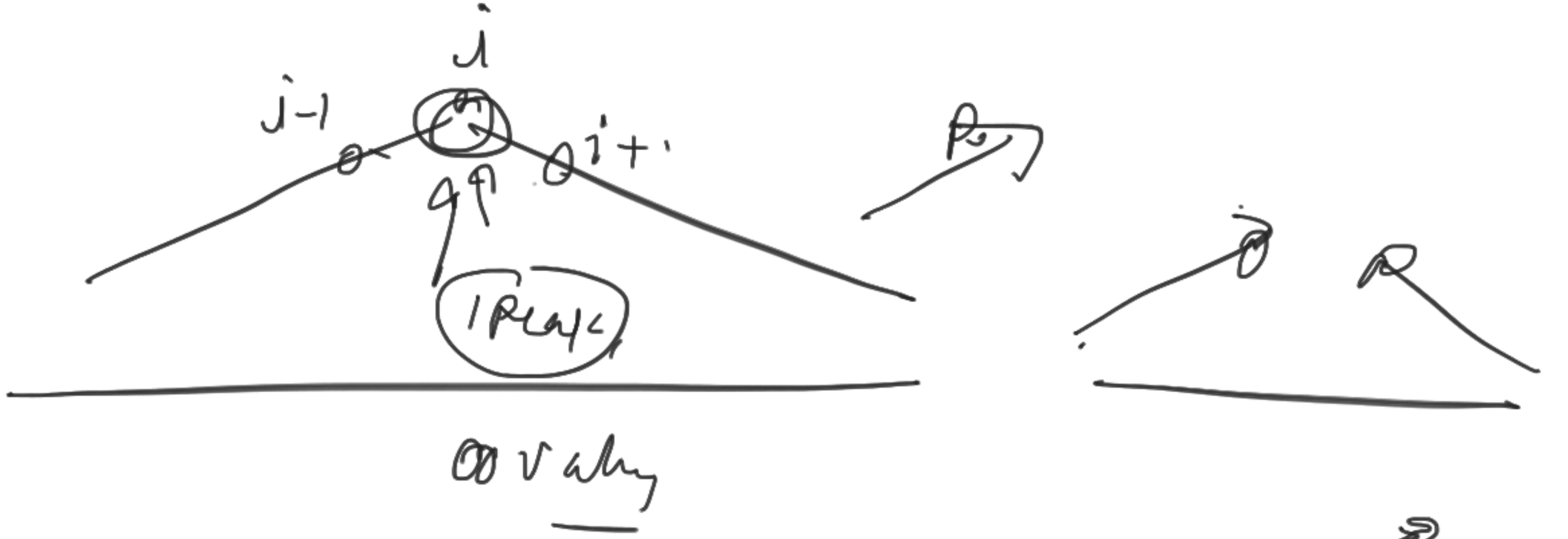
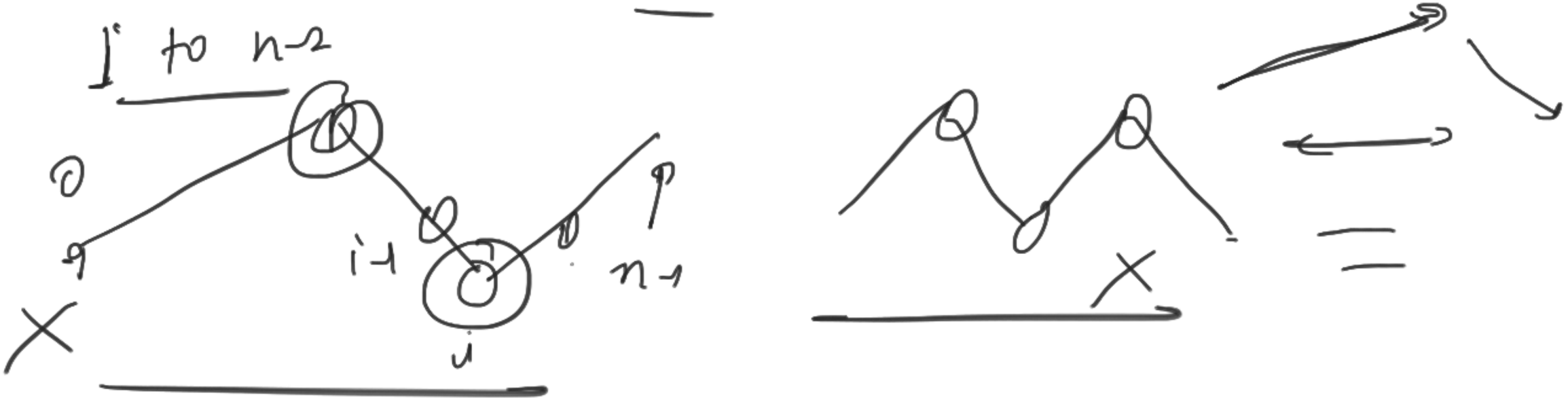
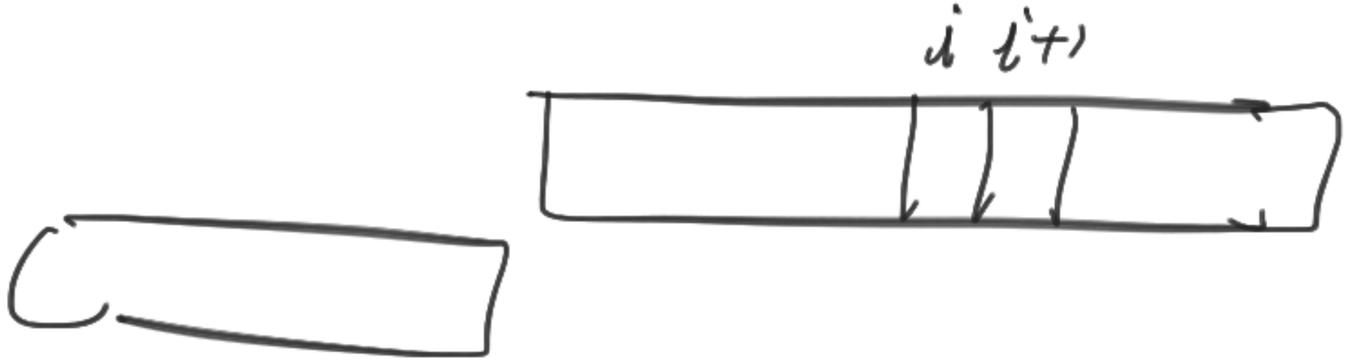

i
i-1
i+1
Replace


i to n-2
i-1
i
n-1
X
X

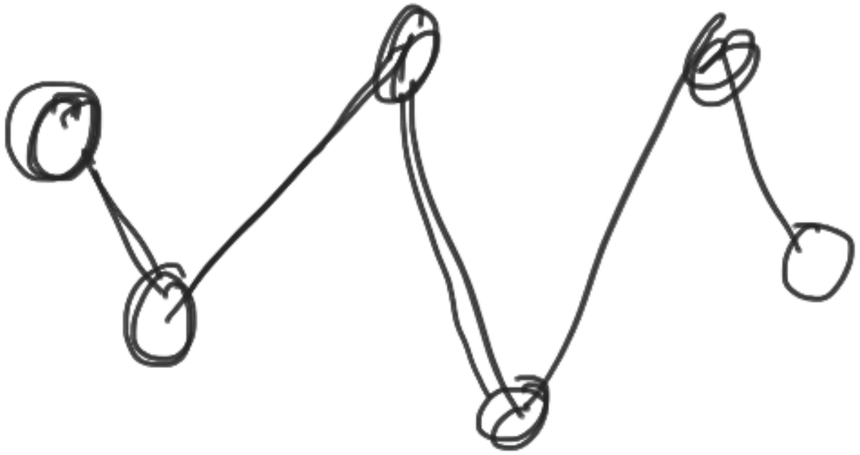
⇒ max and min

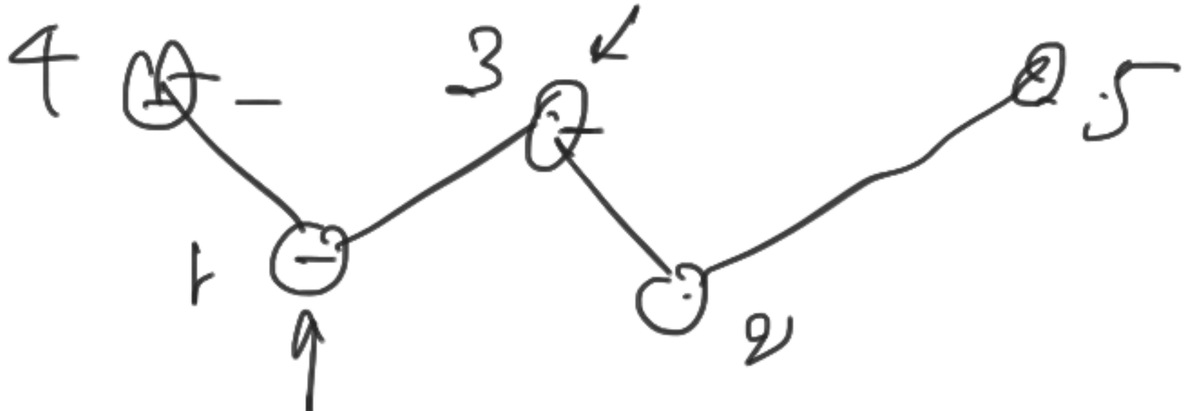
$i = i + 2$

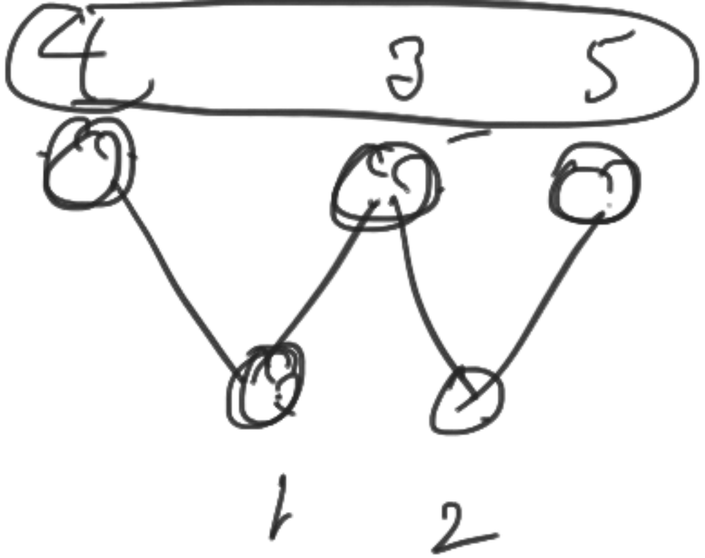
⇒ wave array

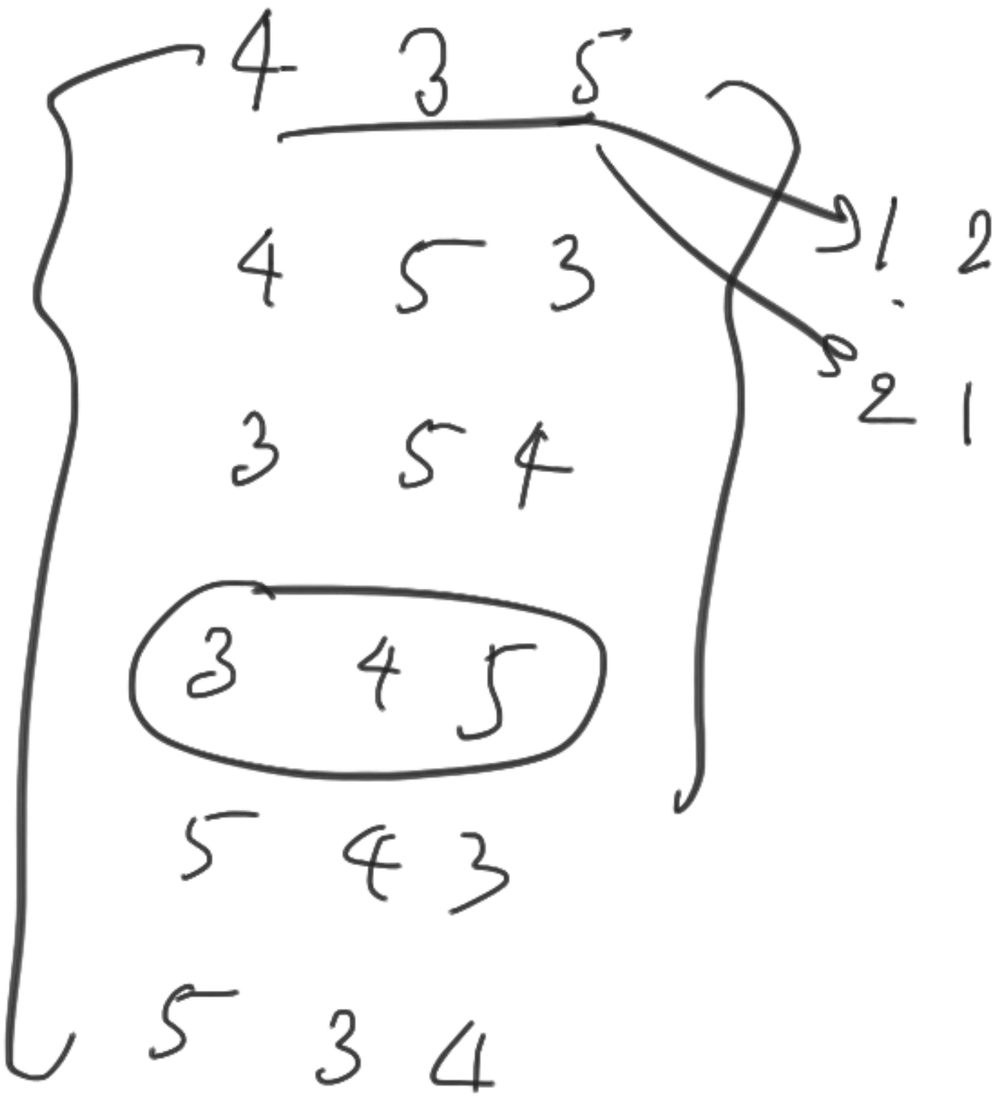


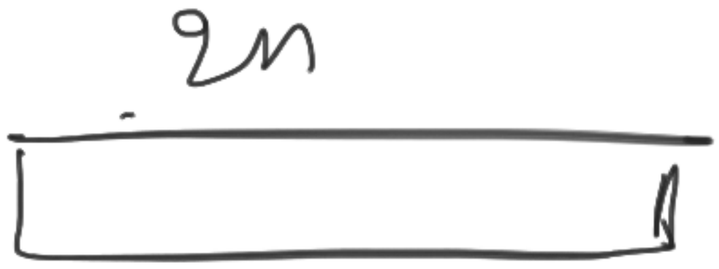
A[] 4 1 3 2 5 ⇒





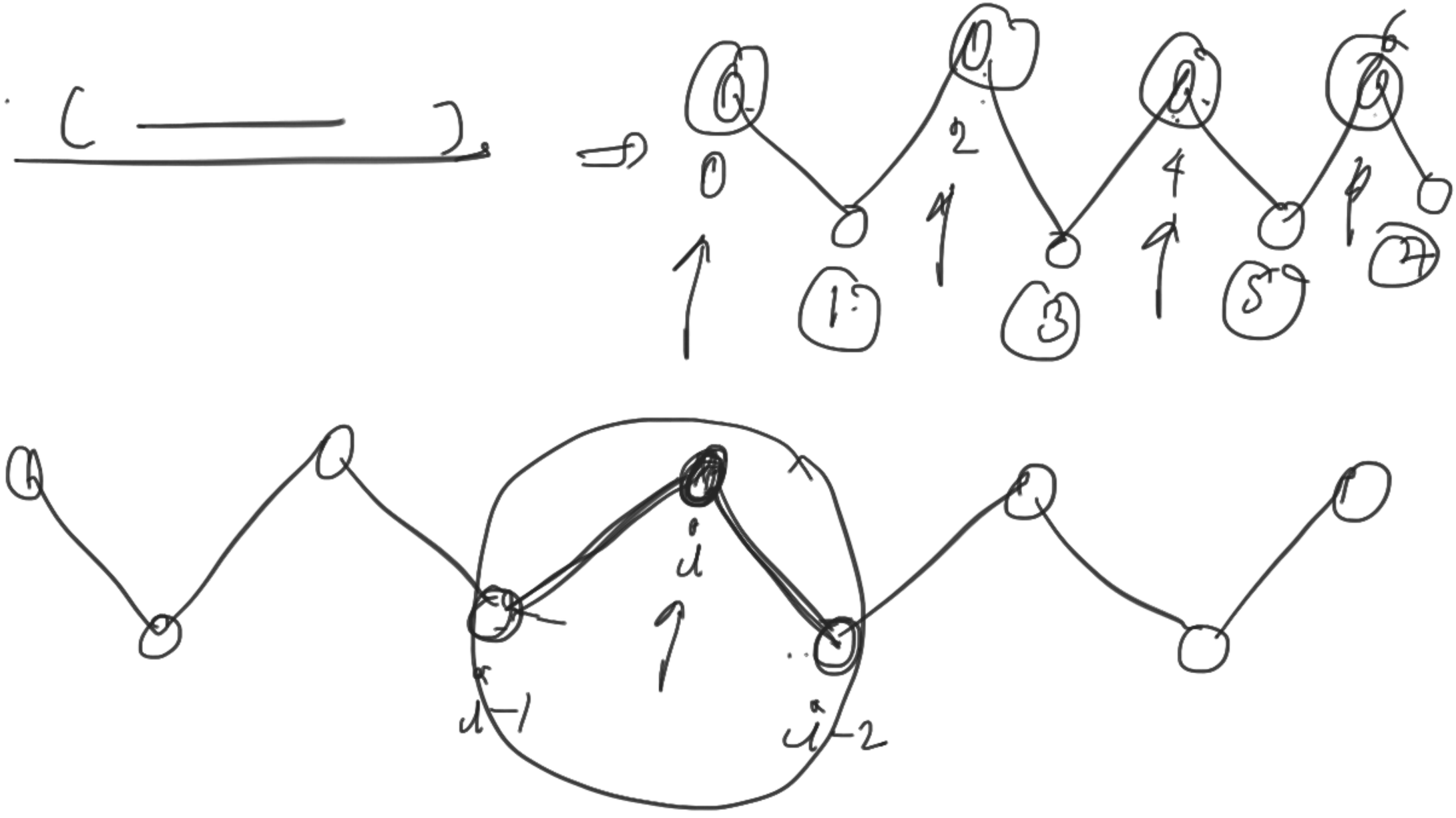


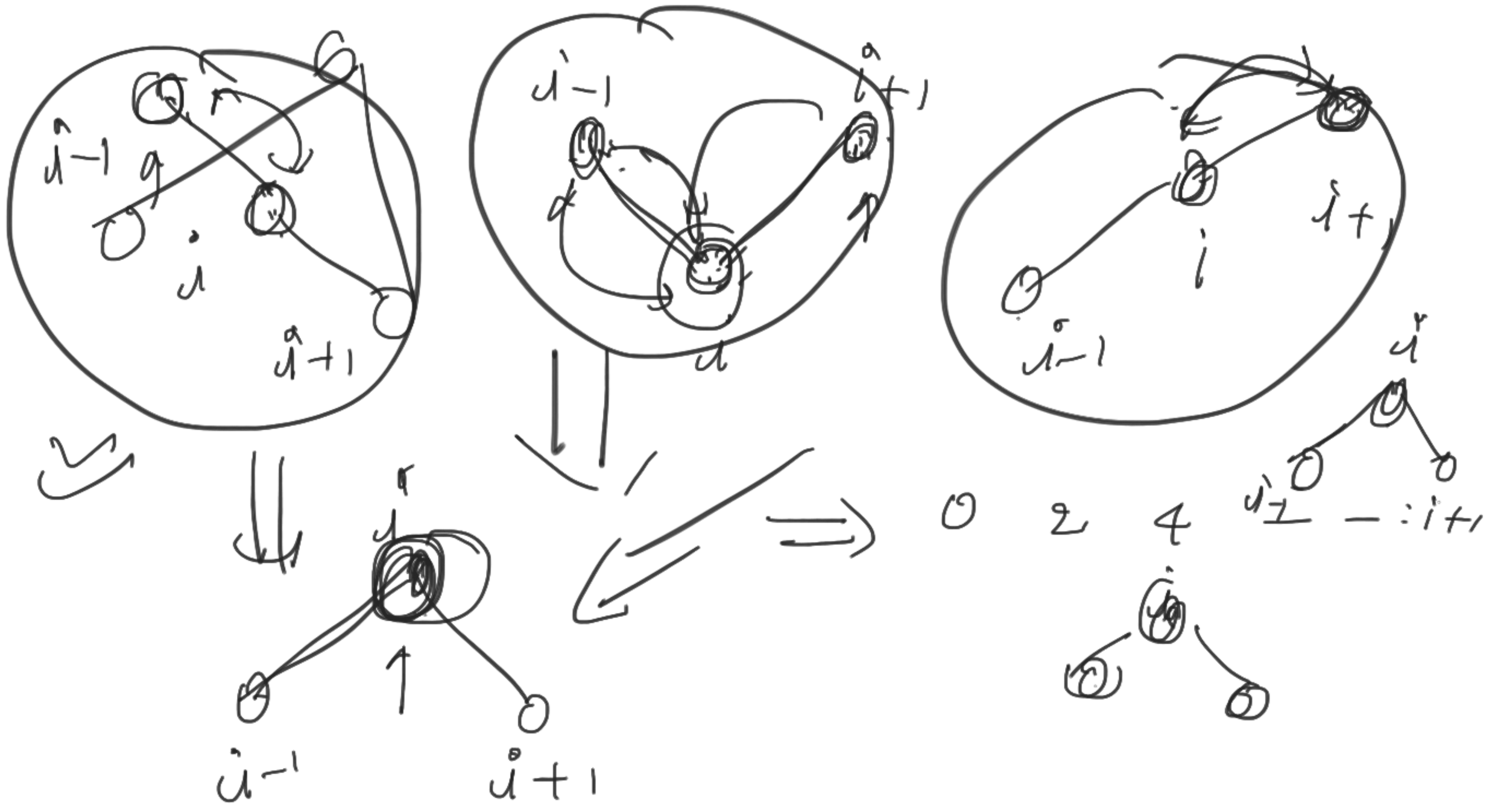


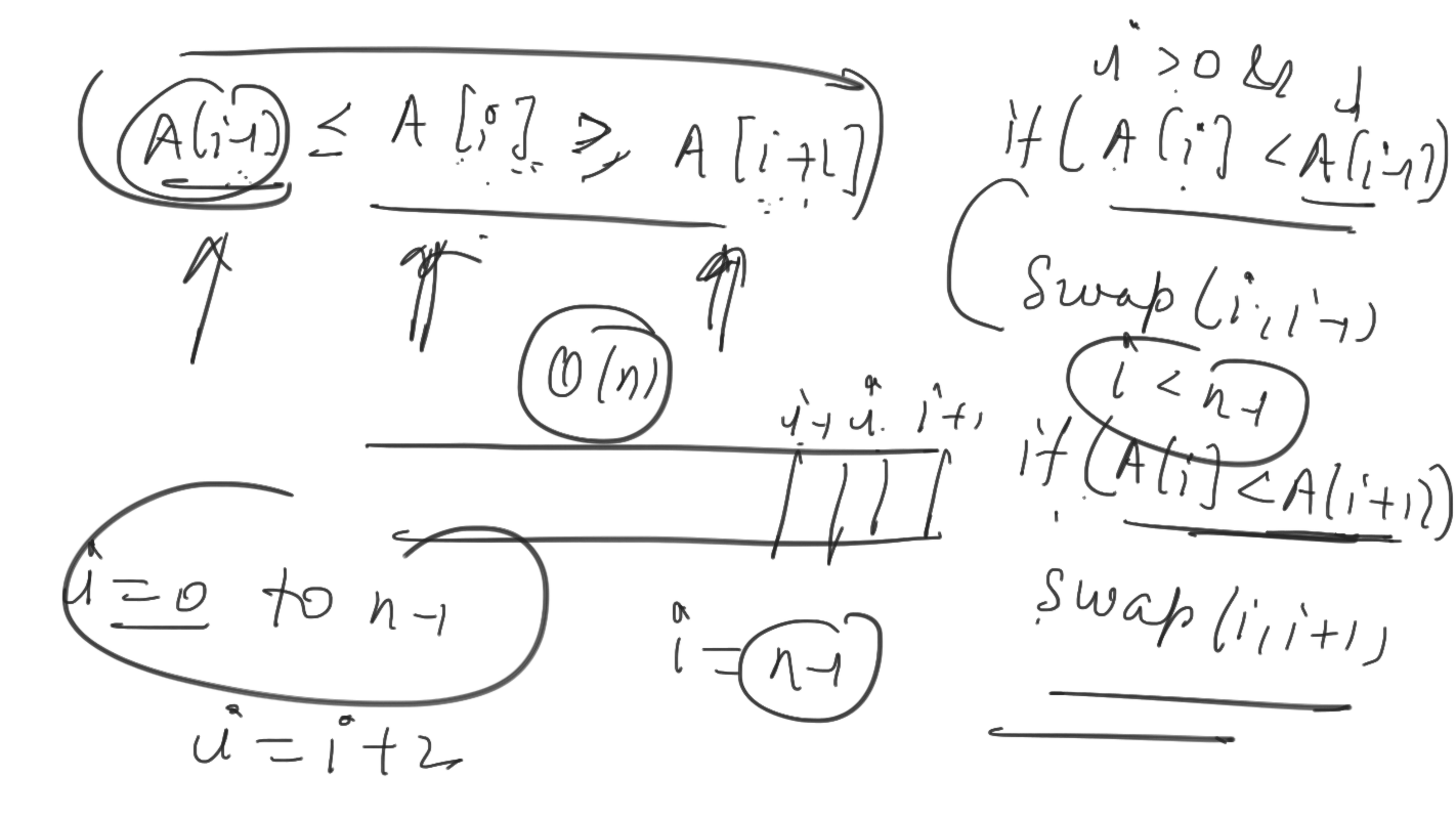


n Perm 1 $\Rightarrow$ $\lfloor n \rfloor$

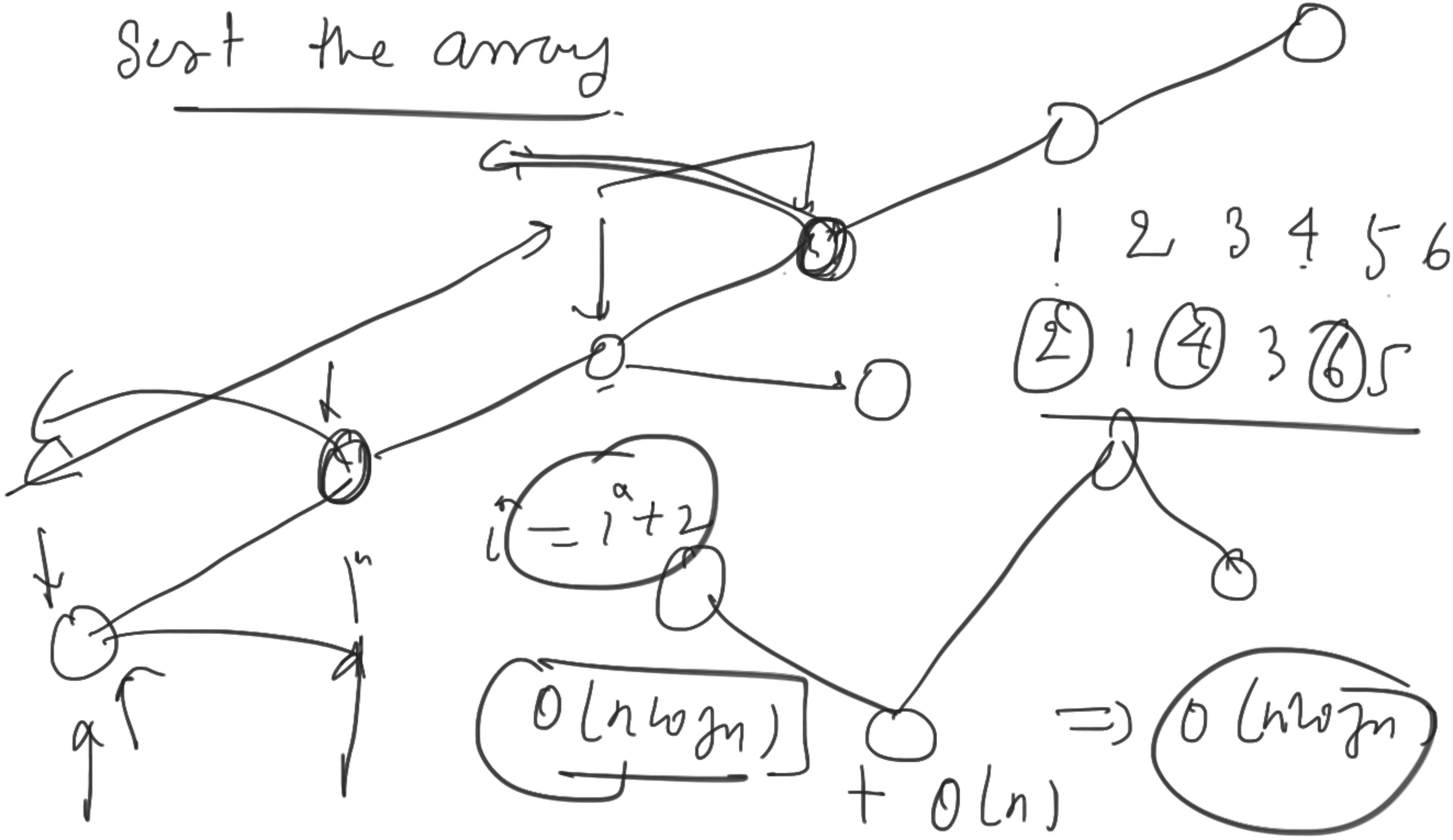
n valley $\Rightarrow$ $\lfloor n \rfloor$ $\Rightarrow$ $(\lfloor n \rfloor)^2$


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



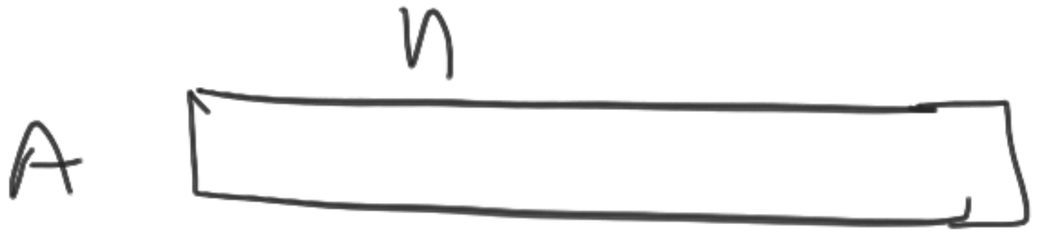

A[i-1] ≤ A[i] ≥ A[i+1]
O(n)
i=0 to n-1
i=i+2
i=n-1
i>0 &
if(A[i]<A[i-1])
Swap(i,i-1)
i<n-1
if(A[i]<A[i+1])
Swap(i,i+1)

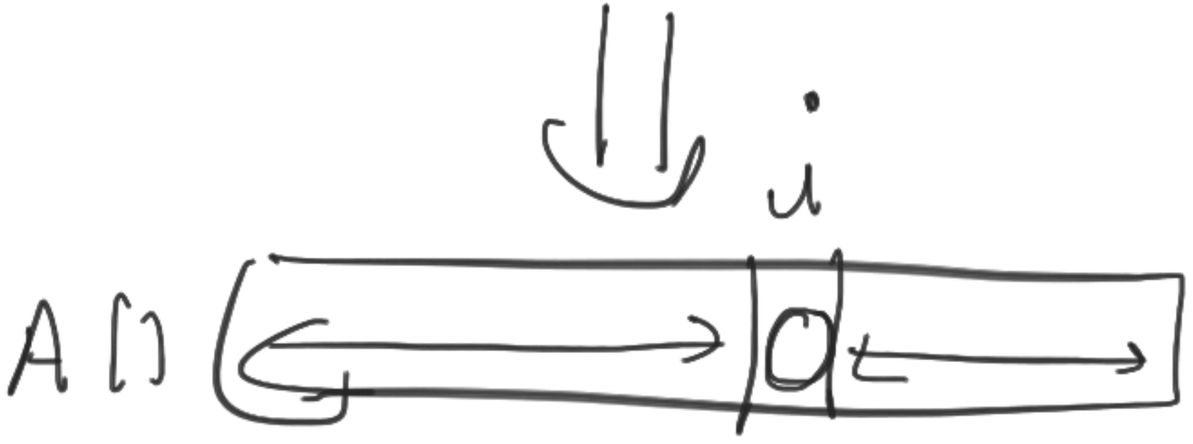
Sort the array



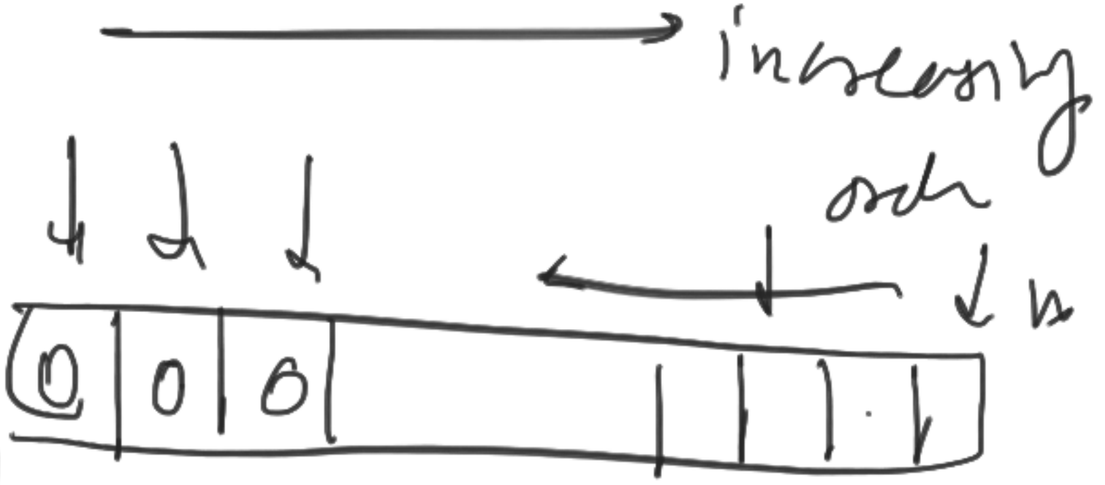
Selection sort

insertion sort

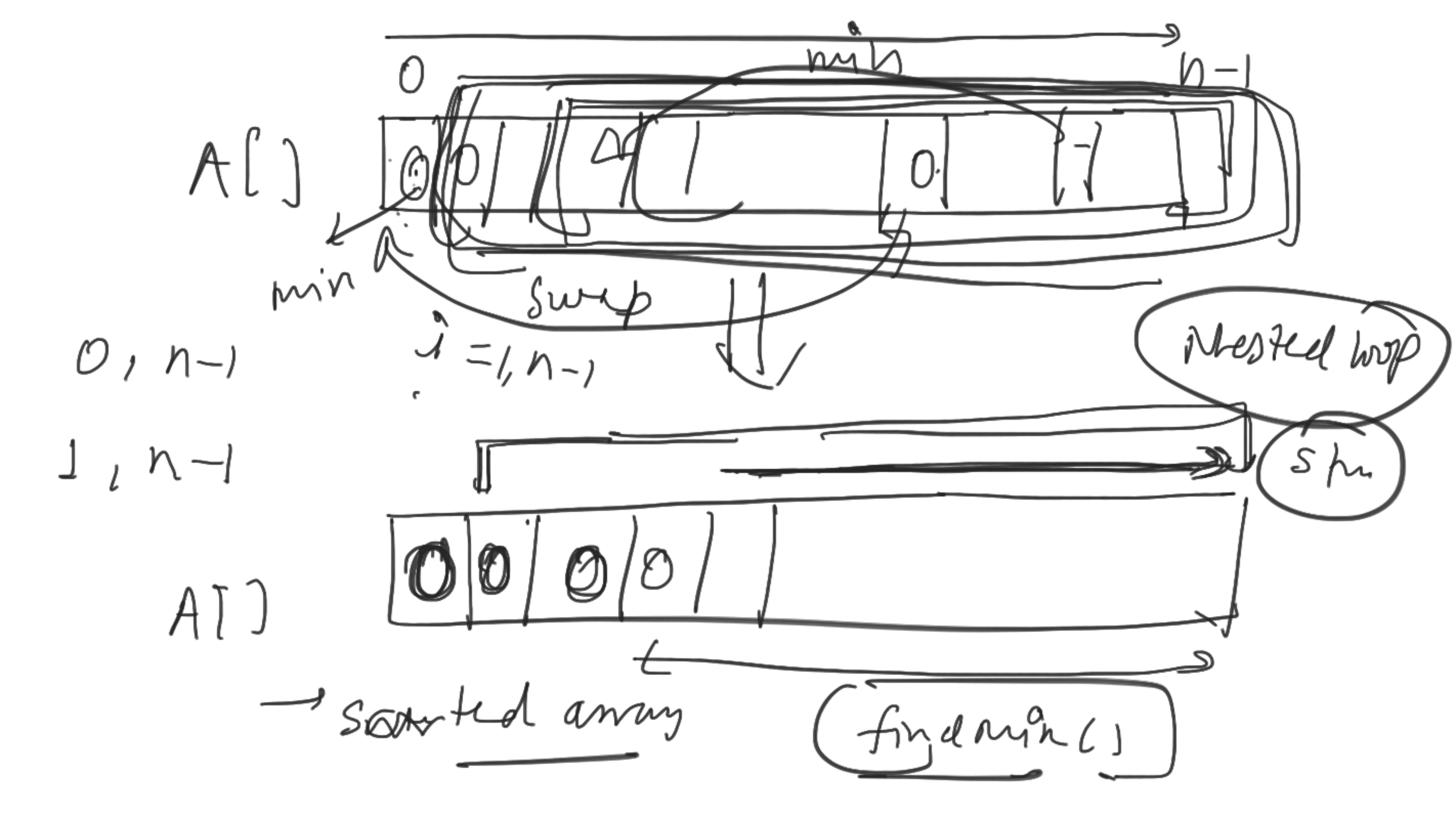




$A[i-1] \leq A[j] \leq A[i+1]$



$A[0] \leq A[1] \leq \quad \leq A[n-1]$

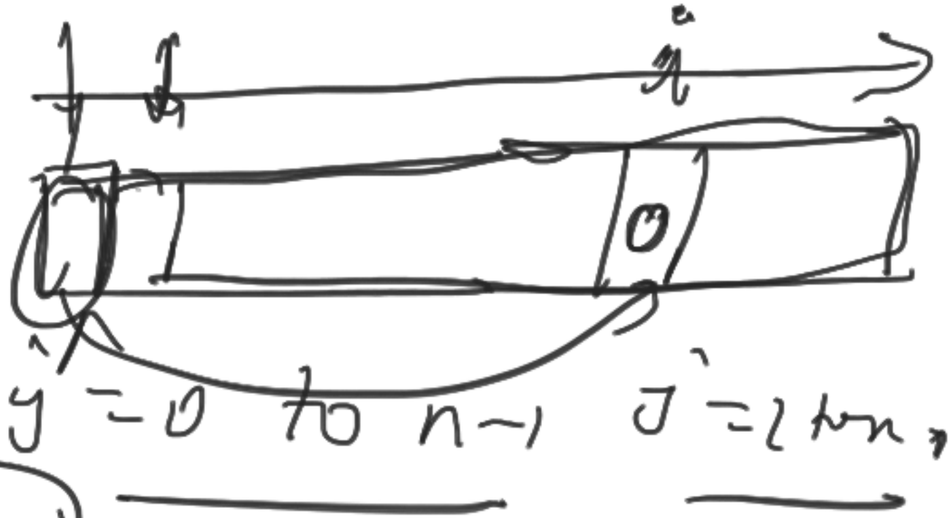

0
min
n-1
A[]
min
swap
0, n-1
i = 1, n-1
Nested loop
1, n-1
sh
A[]
sorted array
findmin()

for (i = 0 to n-2)

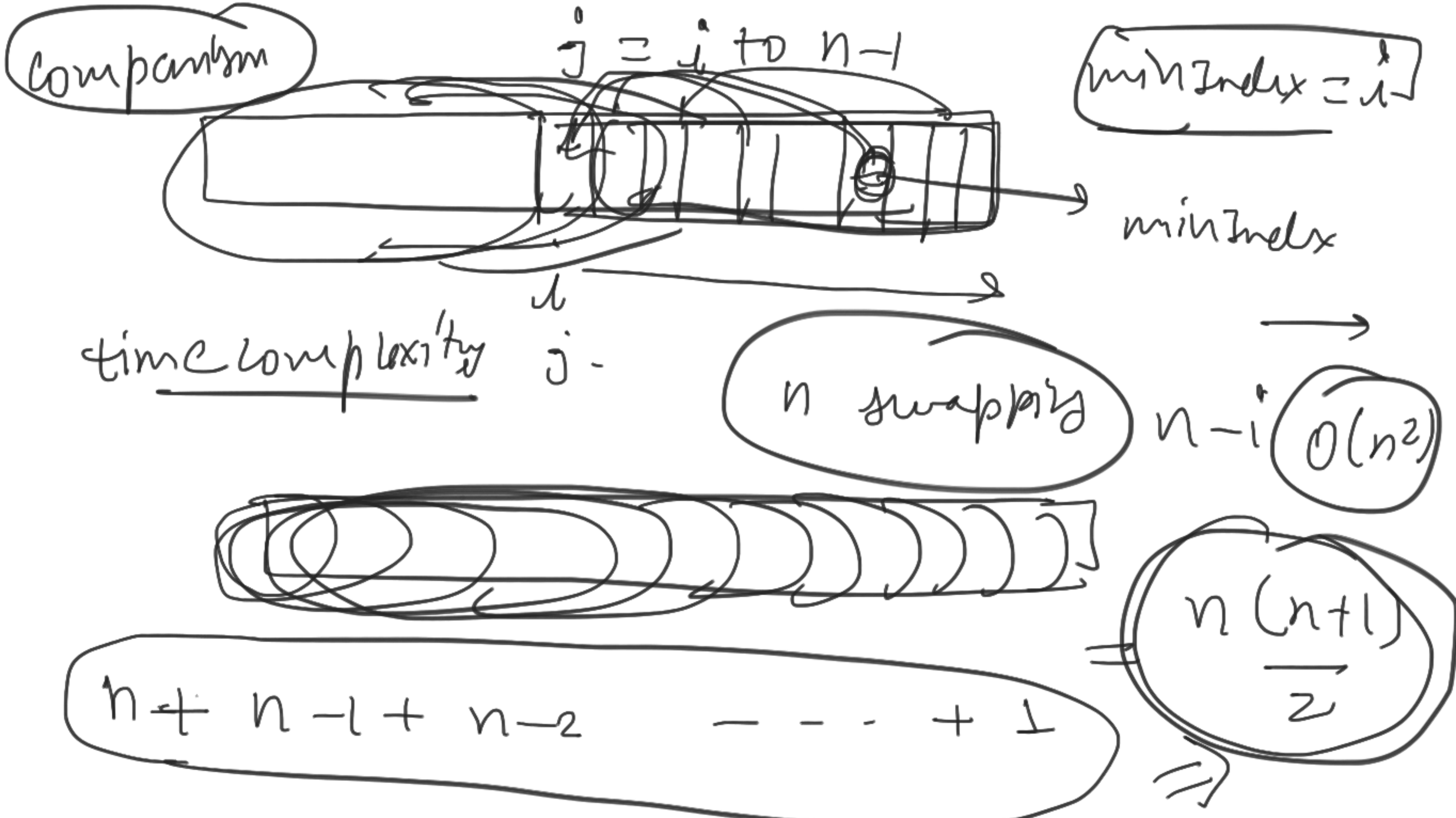
→ n

for (j = i to n-1)

i = 0 to n-1



n-1 comp


comparison
j = i to n-1
minIndex = i
minIndex
i
time complexity
j -
n swapping
n - i
O(n^2)
n + n-1 + n-2 - - - + 1
n (n+1)
2

Why sorting? ⇒ in-depth

Across the desk

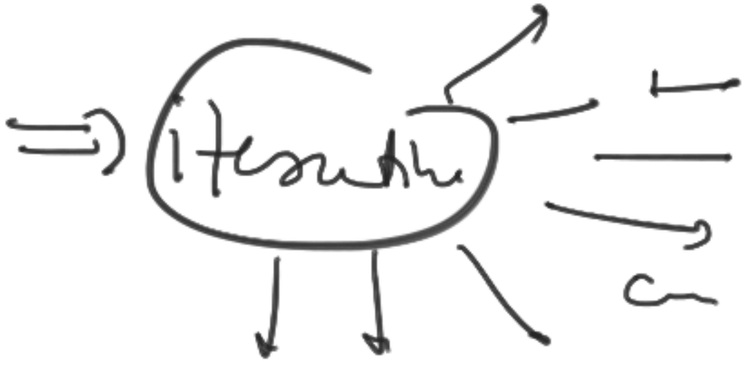
Sorted → binary search

O(log n)

unsorted

⇒ Recursive

⇒ Iterative

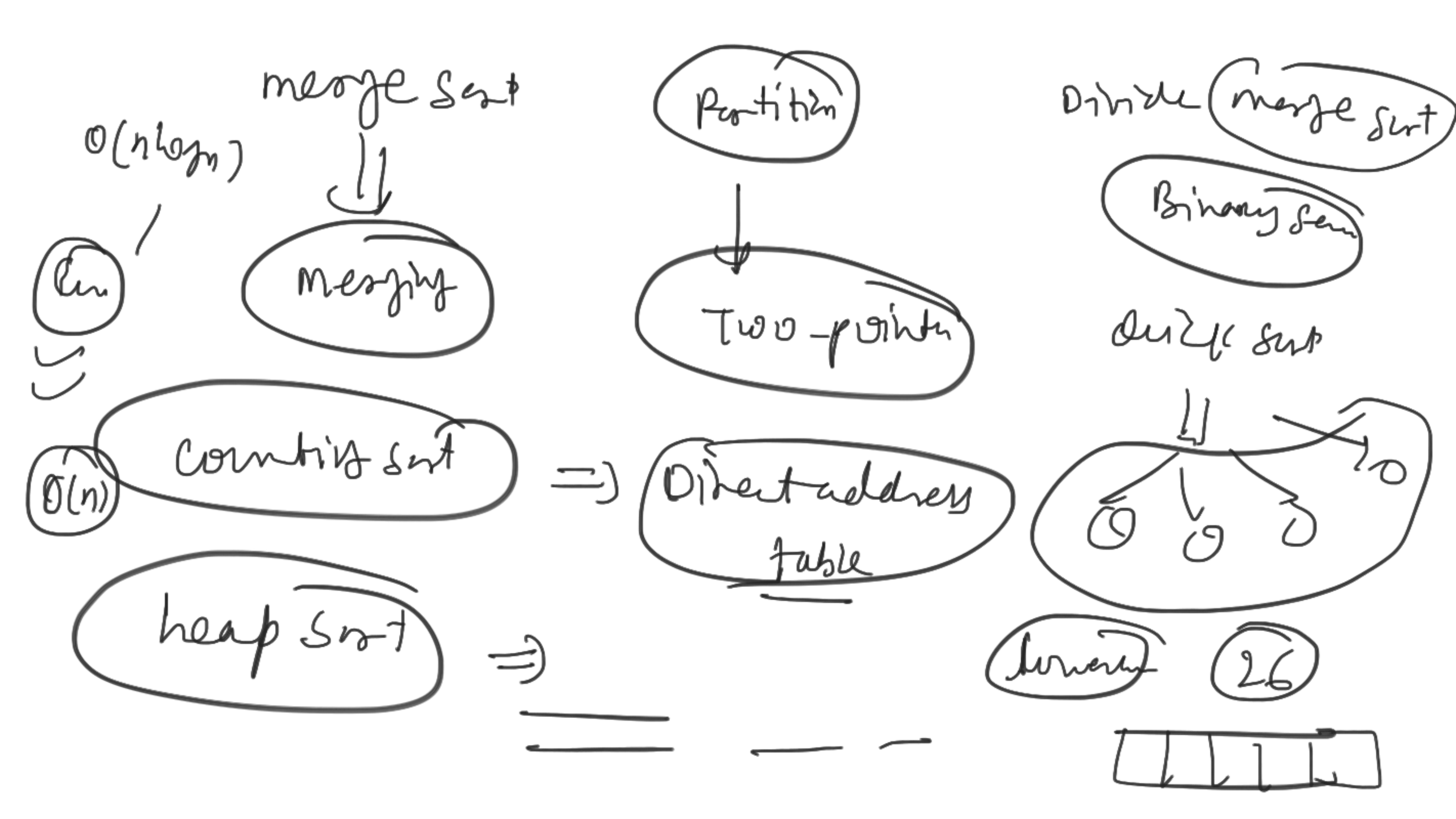


O(n log n)

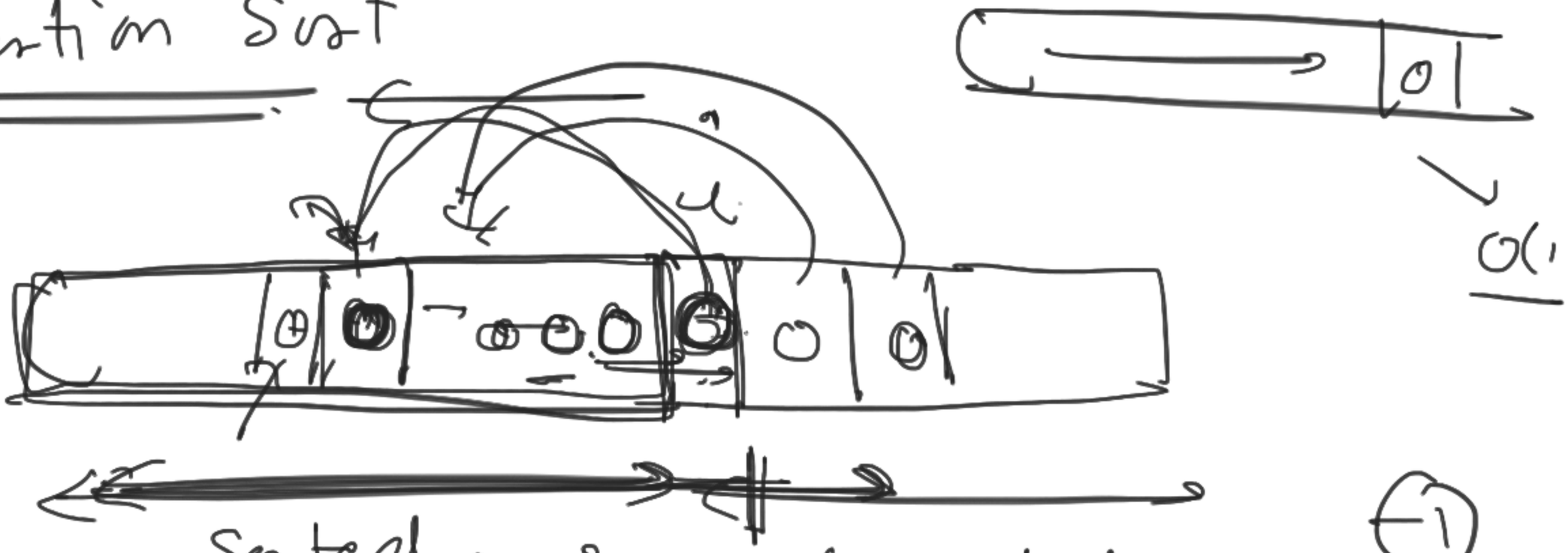
Sorted

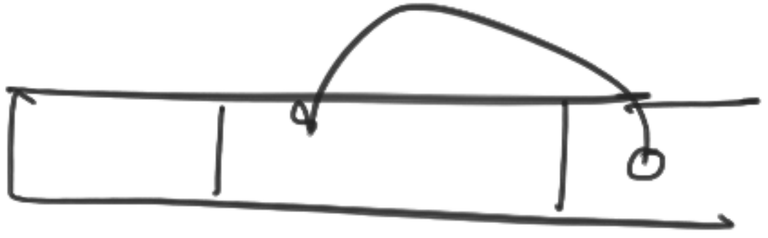
Sliding window

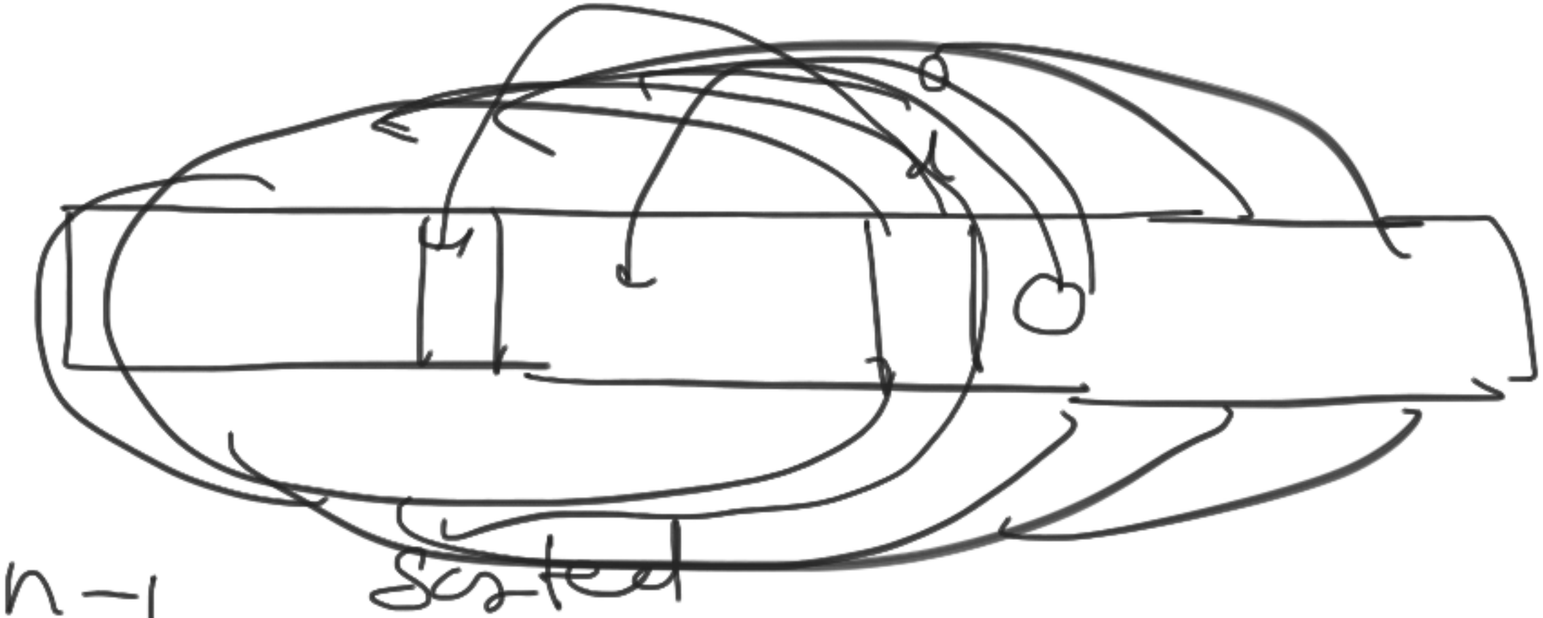
Two Pointers

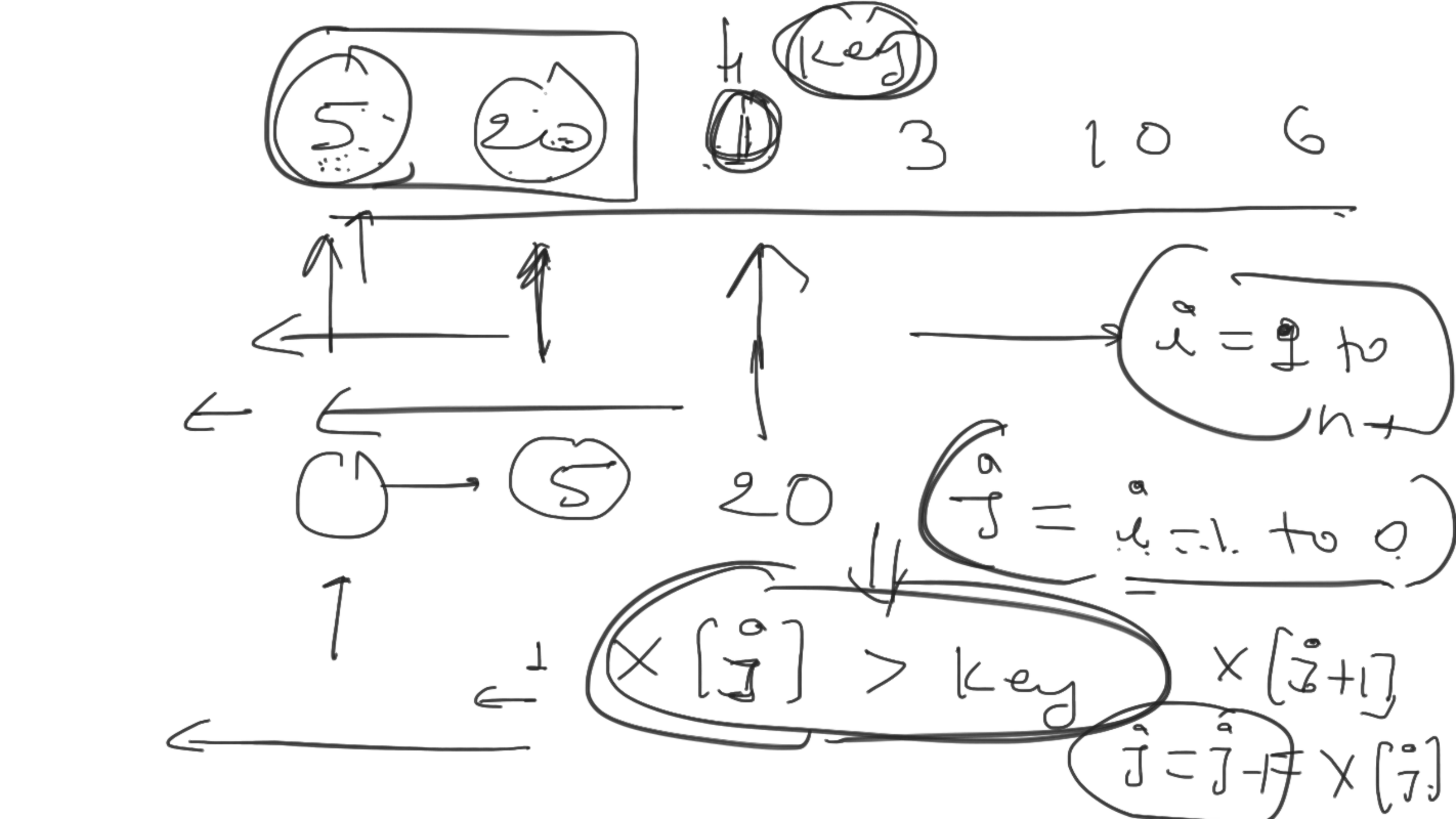

merge sort
O(nlogn)
Merging
Partition
Two-pointer
Divide
merge sort
Binary sea
Quick sort
Counting sort
O(n)
Direct address table
heap sort
Answer
26

Insertion Sort






d=1 to n-1
0
n-1


5
20
key
3
10
6
i = 1 to n
5
20
j = i-1 to 0
x[j] > key
x[j+1]
j = j-1
x[j]

for (i = 1 to n-1)

{

key = x [i]

j = i - 1

while (j >= 0 && x [j] > key)

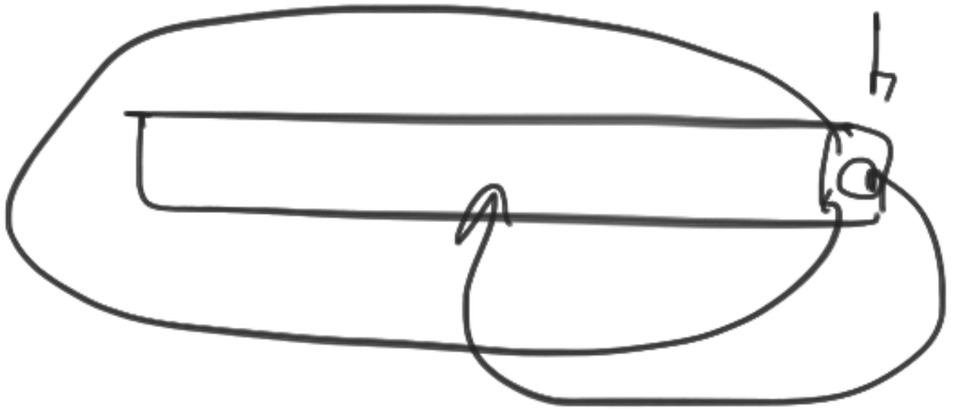
{

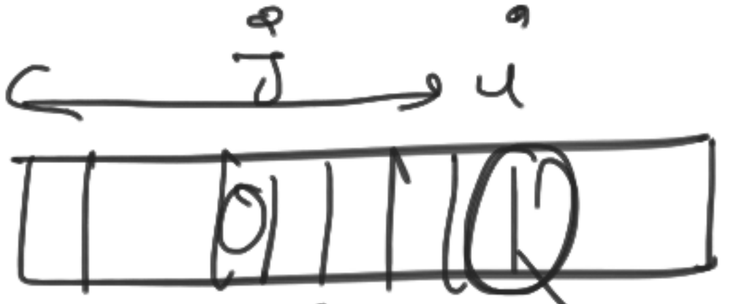
x [j + 1] = x [j]

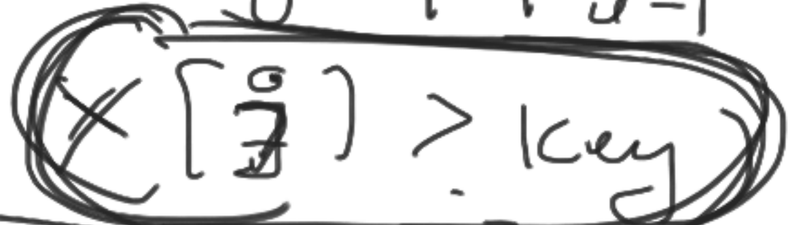
j = j - 1

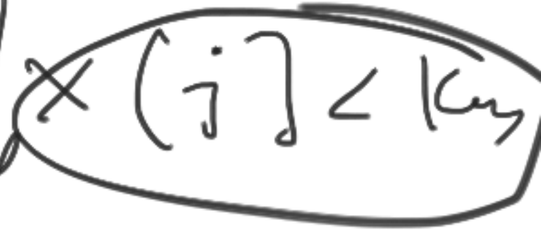
}

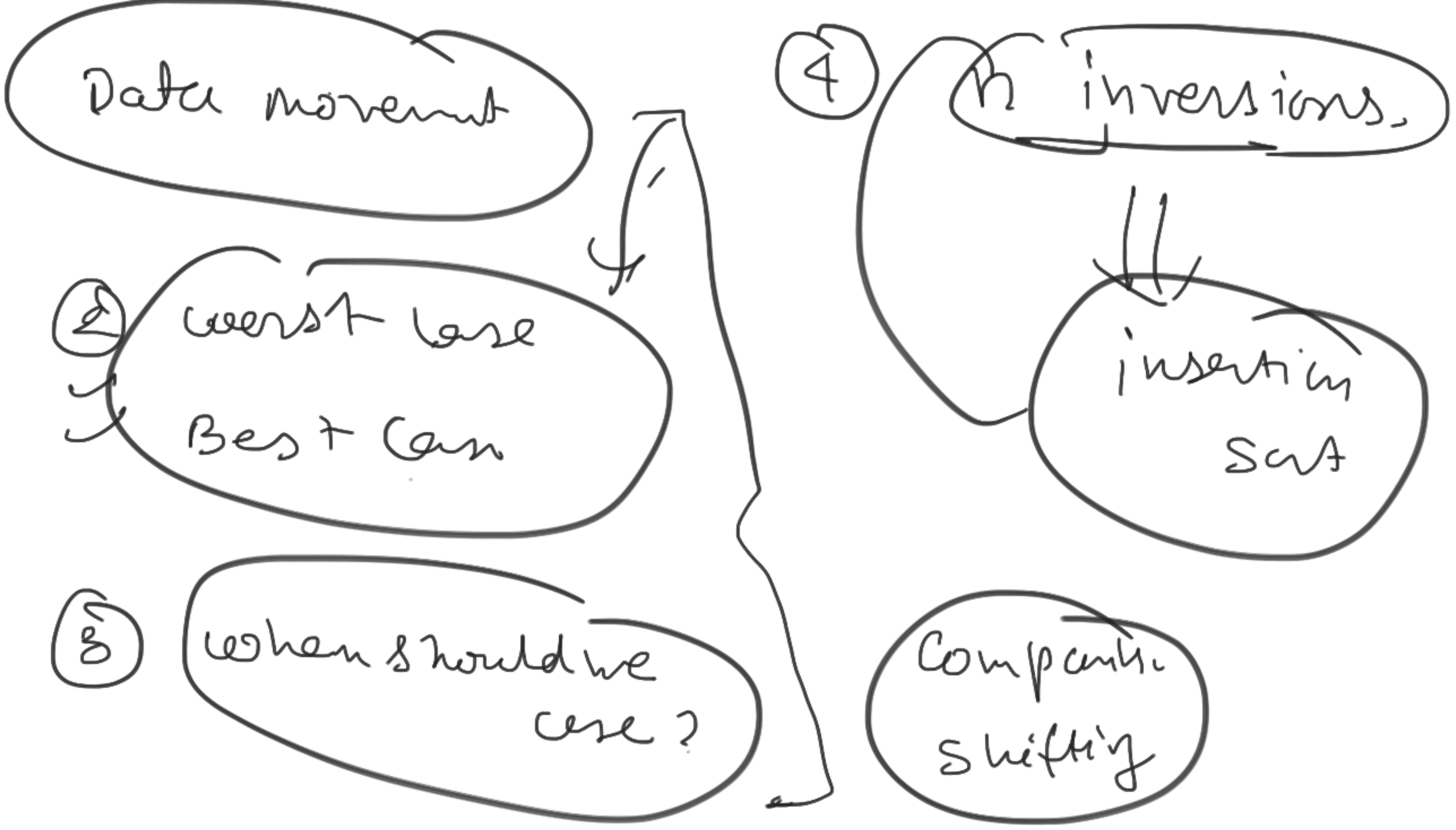
x [j + 1] = key

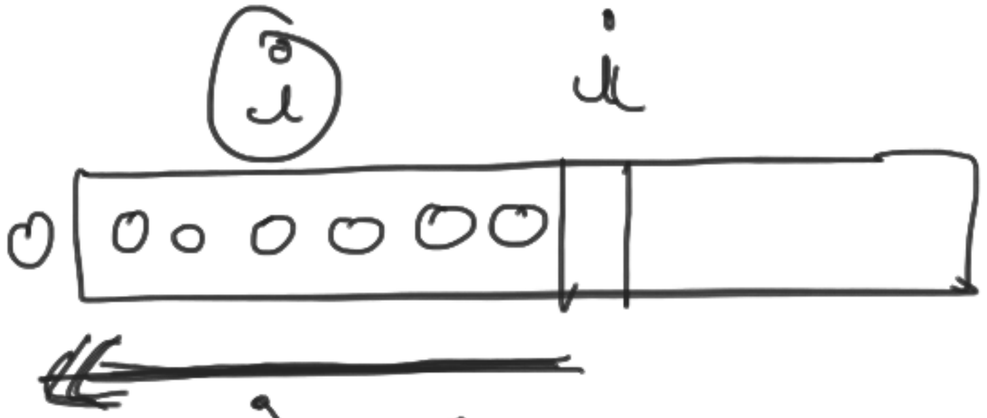









Data movement
4
n inversions.
2
worst case
Best Can
insertion sort
3
when should we use?
Comparison shifting



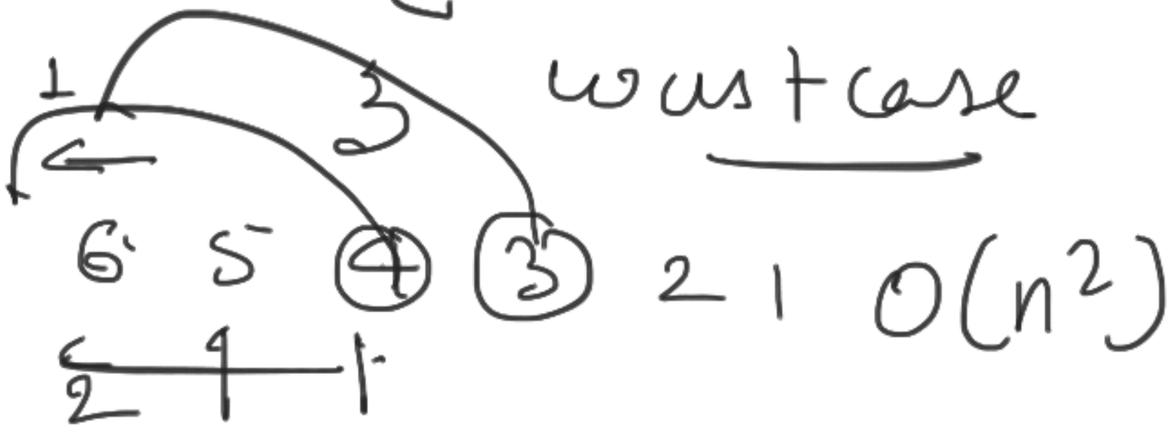
$\Rightarrow 1 + 2 \quad \cdots \quad n-1$

$= \frac{n(n-1)}{2} \Rightarrow O(n^2)$

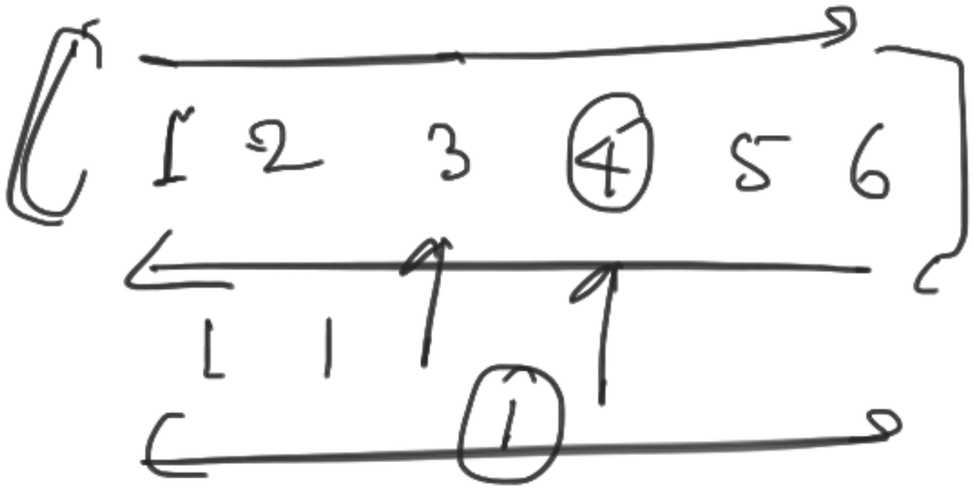
$j = i-1$ to 0 (worst case)

{ i comp
i shifting }

worst case



$O(n^2)$

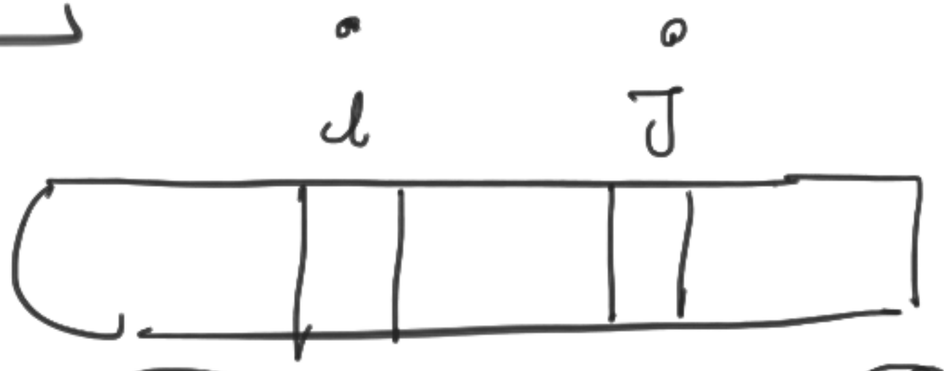


$O(n)$ n comp

0 shift

almost Sorted / Partially Sorted. ⇒ O(n)

inversion



(0, 0) (i, j)

if A[i] > A[j]

(0, 0) i < j

one inversion

3 | 2 | 4 | 1

3 1 4

(3, 2) (3, 1) (4, 1) (2, 1)