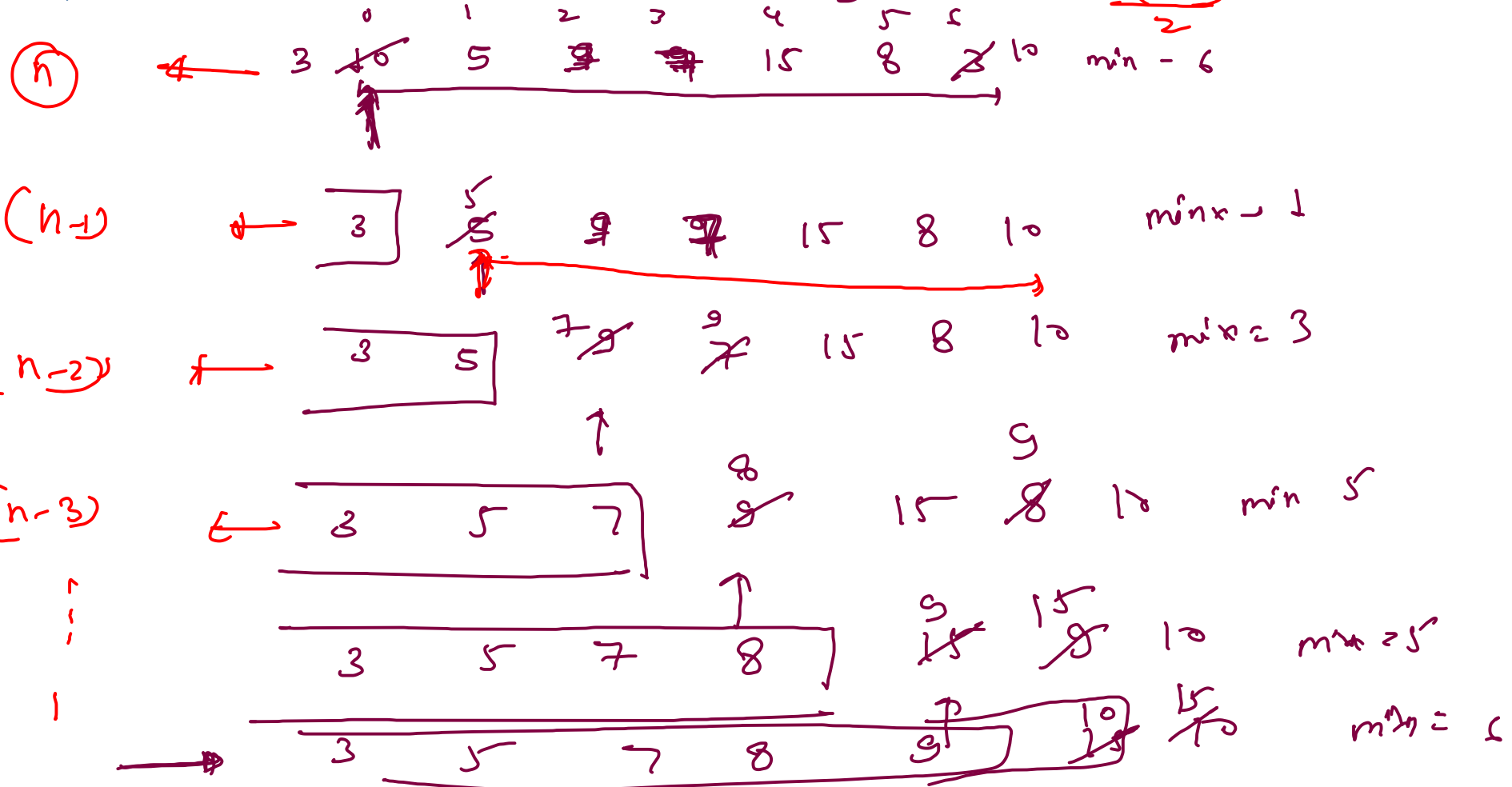
Bubble Sort → Adjacent Shifting

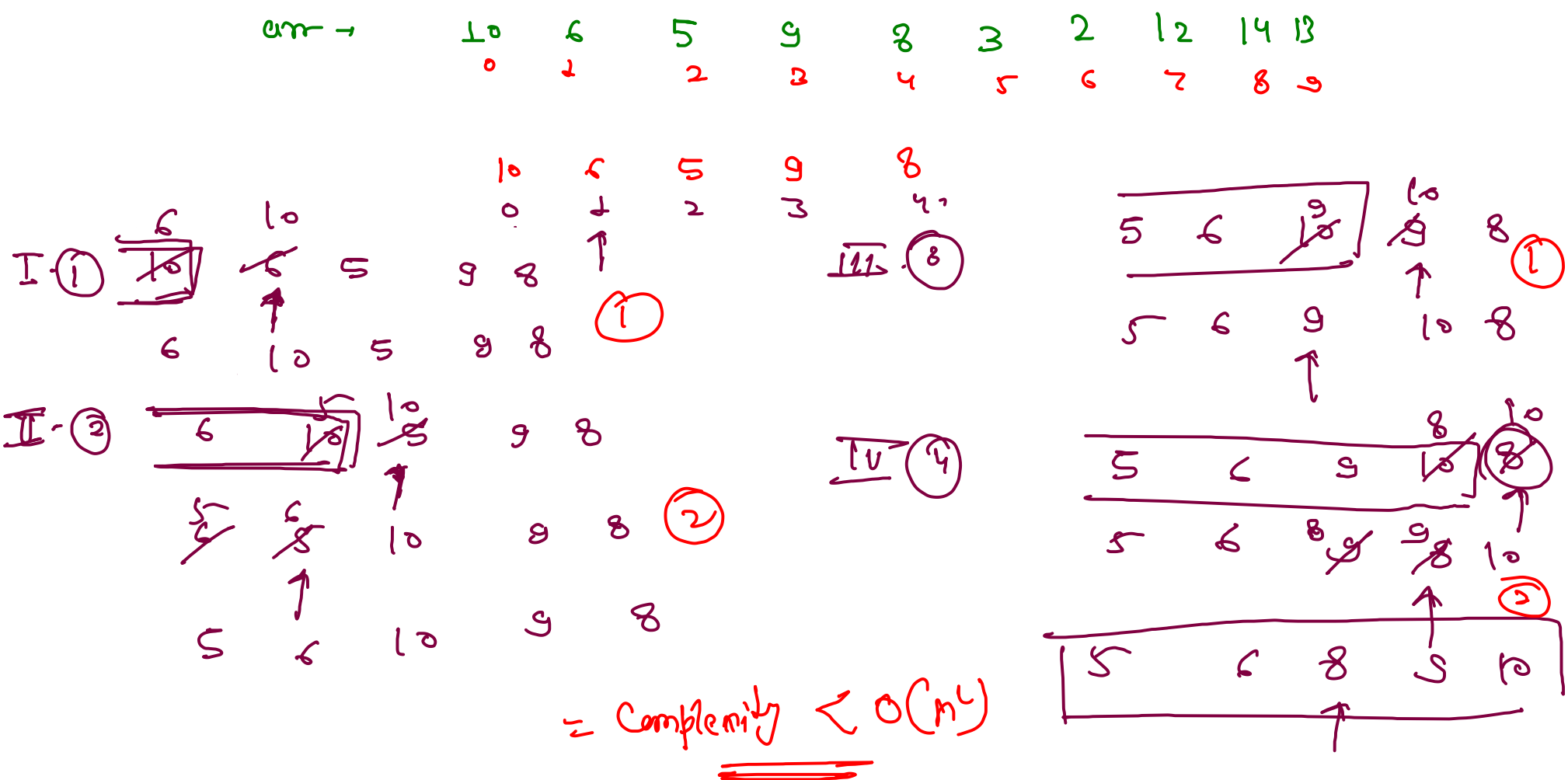
Selection Sort → Select Min. Element Index

Cost = $1 + 2 + 3 + \dots n$

$= \frac{n(n+1)}{2}$



Insertion Sort



50 40 30 20 10 } worst case - array - Decreasing order

0 1 2 3 4

→ $O(n^2)$

$i = 1$, $j = 1, j-1$ = (1)

$i = 2$, $j = i$; $j > 0$ = (2)

$i = 3$, = (3)

$i = 4$, = (4)

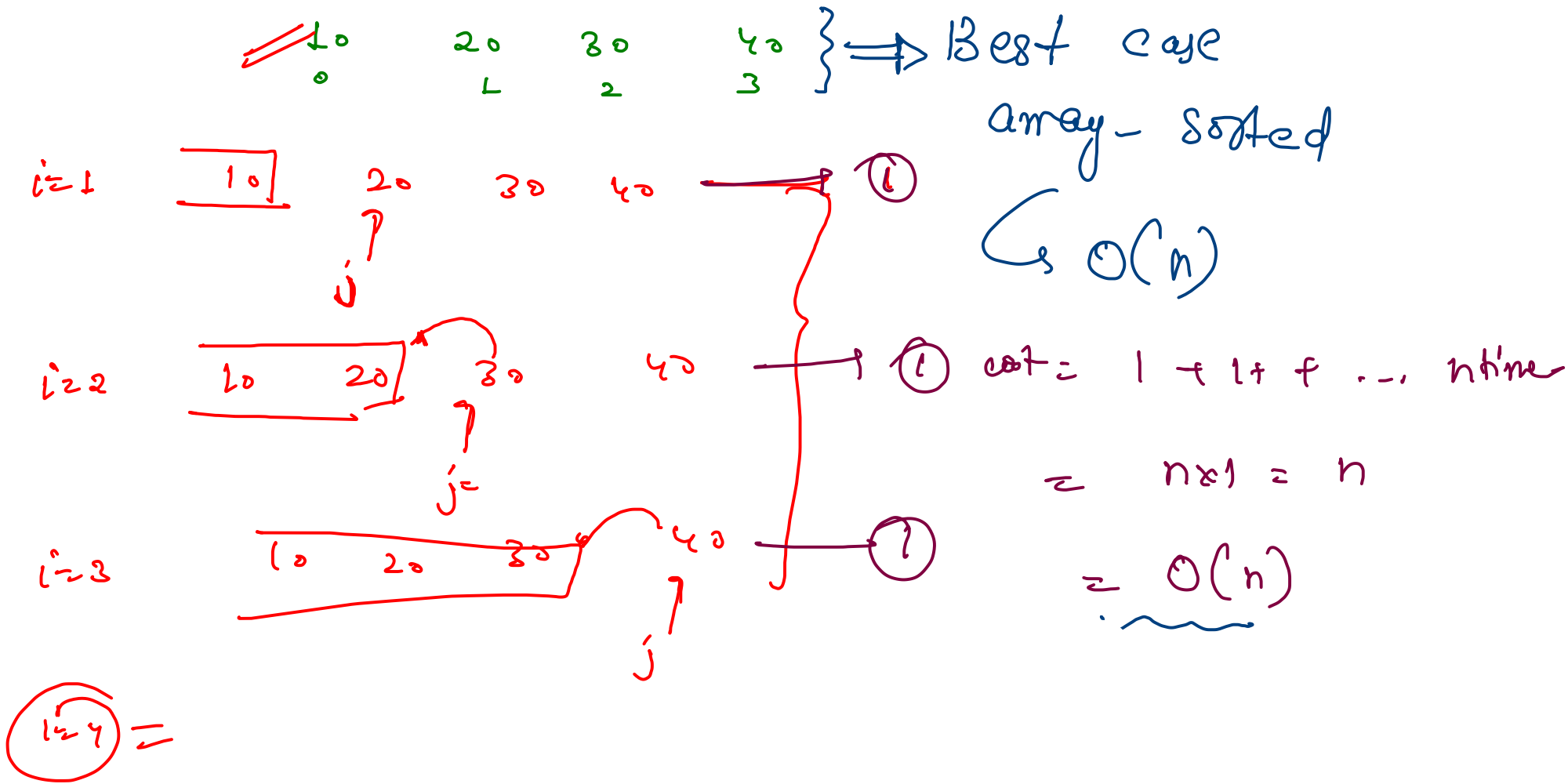
$\vdots$

(n)

$$\text{cost} = 1 + 2 + 3 + 4 + \cdots + n$$

$$= \frac{n(n+1)}{2} \approx \frac{n^2}{2} + \frac{n}{2}$$

$$= O(n^2)$$

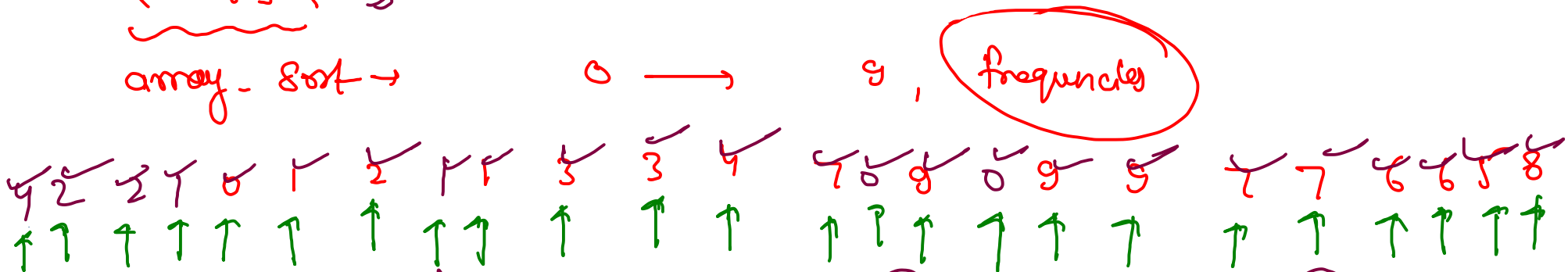

10 20 30 40
0 1 2 3
Best case
array - sorted
O(n)
i=1
10 20 30 40
j
1
i=2
10 20 30 40
j=
1
cost = 1 + 1 + ... n time
i=3
10 20 30 40
j
1
= n × 1 = n
= O(n)
i=4 =

Count sort →

$0 \le arr.length \le 100$

$0 \le arr[i] \le 9$

array sort → 0 ⟶ 9, frequency



0 → 3

1 → 4

2 → 3

3 → 2

4 → 2

5 → 1

6 → 2

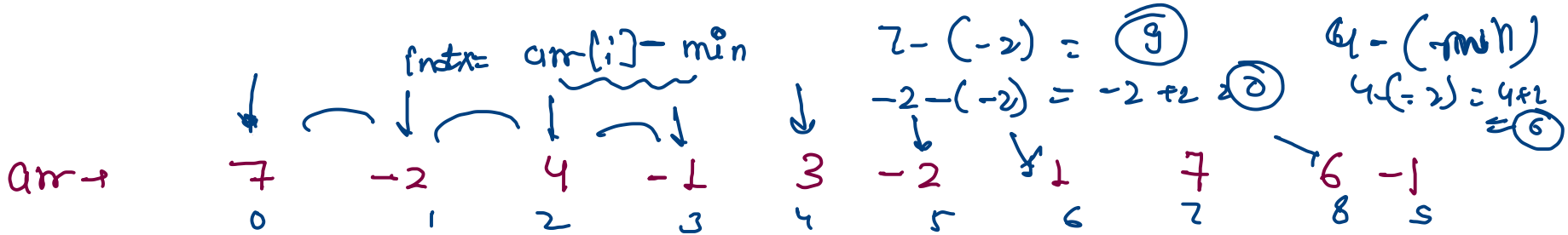
7 → 3

8 → 1

9 → 2

$\Rightarrow O(n+n) = O(2n) = O(n)$

[0 0 0 1 1 1 1 2 2 2 3 3 4 4 5 6 6 7 7 7 8 9 9 9



min = -2

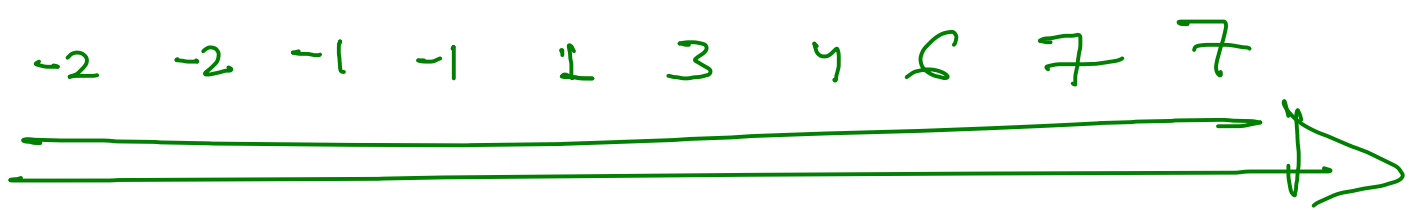
val = 0 + 3 max → 7

fmap = new int[max - min + 1]

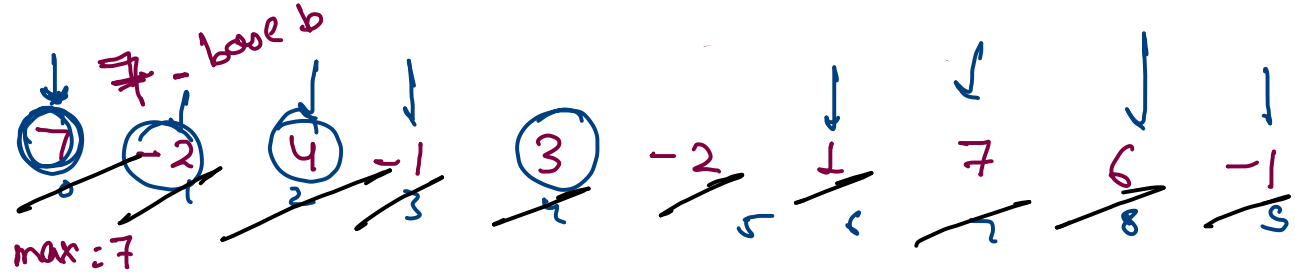
Radix { Stable sorting

indx	val	freq.	
0 →	-2	~~2~~	2
1 →	-1	~~1~~	2
2 →	0	= 0	
3 →	1	= ~~2~~	1
4 →	2	= 0	
5 →	3	= ~~2~~	1
6 →	4	= ~~2~~	1
7 →	5	= 0	
8 →	6	= ~~2~~	1
9 →	7	= ~~2~~	2

val = indx + min



```java
public static void countSort2(int[] arr, int min, int max) {
    int[] fmap = new int[max - min + 1];

    for(int i = 0; i < arr.length; i++) {
        int indx = arr[i] - min; // val - base value
        fmap[indx]++;
    }

    int indx = 0;
    for(int i = 0; i < fmap.length; i++) {
        int val = i + min; // indx + base value
        int freq = fmap[i];

        for(int j = 0; j < freq; j++) {
            arr[indx] = val;
            indx++;
        }
    }
}
```



7 - (-2) = 9

-2 - (-2) = -2 + 2 = 0

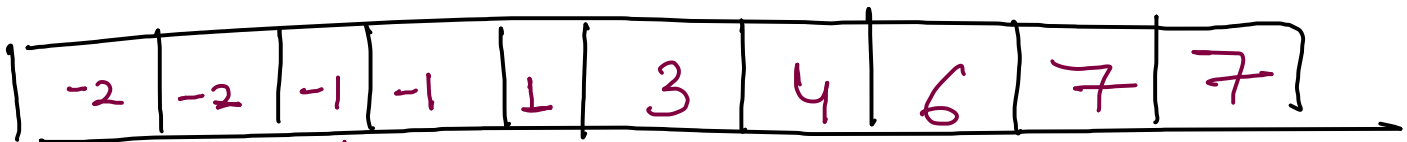
4 - (-2) = 6

-1 - (-2) = -1 + 2 = 1

val: -2 -1 0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7 8 9

2 2 0 1 0 1 1 0 1 2

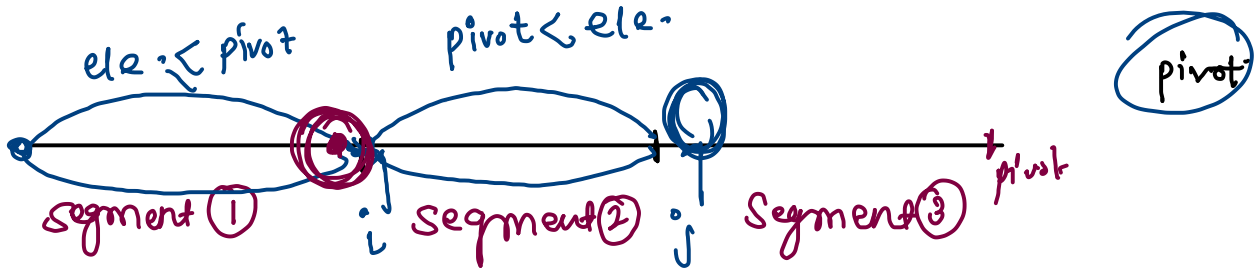


indx + baseval.

0 - 2 = -2 1 - 2 = -1 3 - 2 = 1 5 - 2 = 3

8 - 2 = 6 6 - 2 = 4 5 - 2 = 3

Partition of Array-



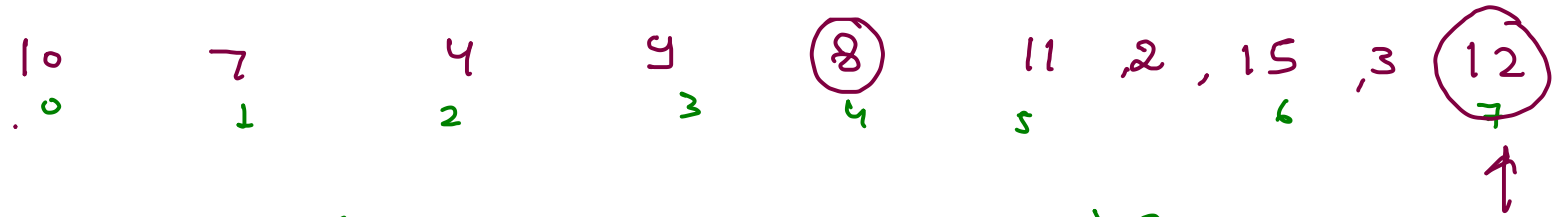
Segment ① → value <= pivot

Segment ② → value > pivot } i is first element of Segment 2

Segment ③ → value → unsolved → j is first element of Segment ②

arr[j] > pivot	arr[j] <= pivot

j++;	Swap(i, j)
	i++;
	j++;



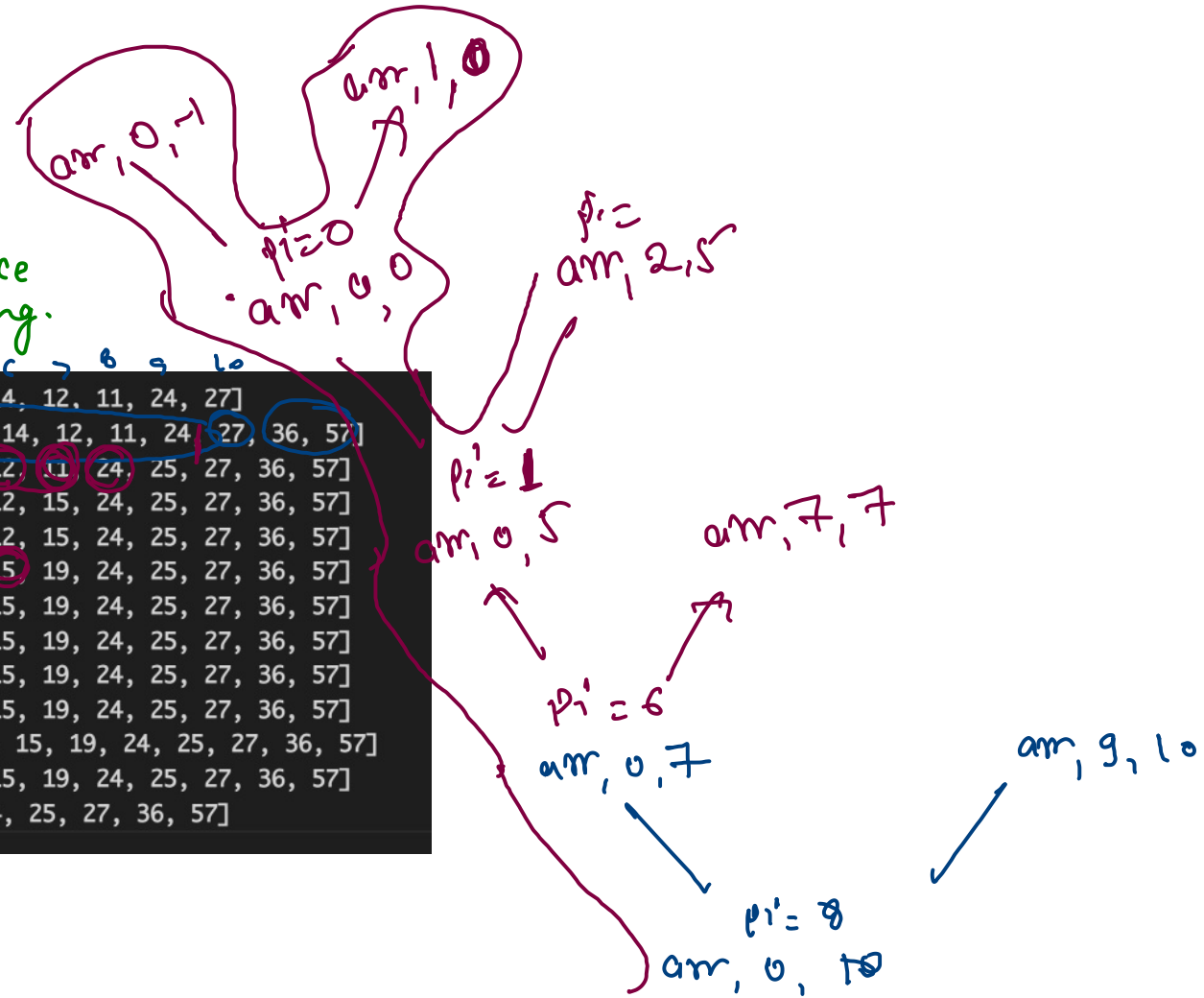

```
quicksort ( int[] arr, int lo, int hi) {

    int pivot = arr[hi];
    int pi = partitionIndex (arr, lo, hi, pivot);

    quickSort (arr, lo, pi-1);
    quicksort( arr, pi+1, hi);

}
```

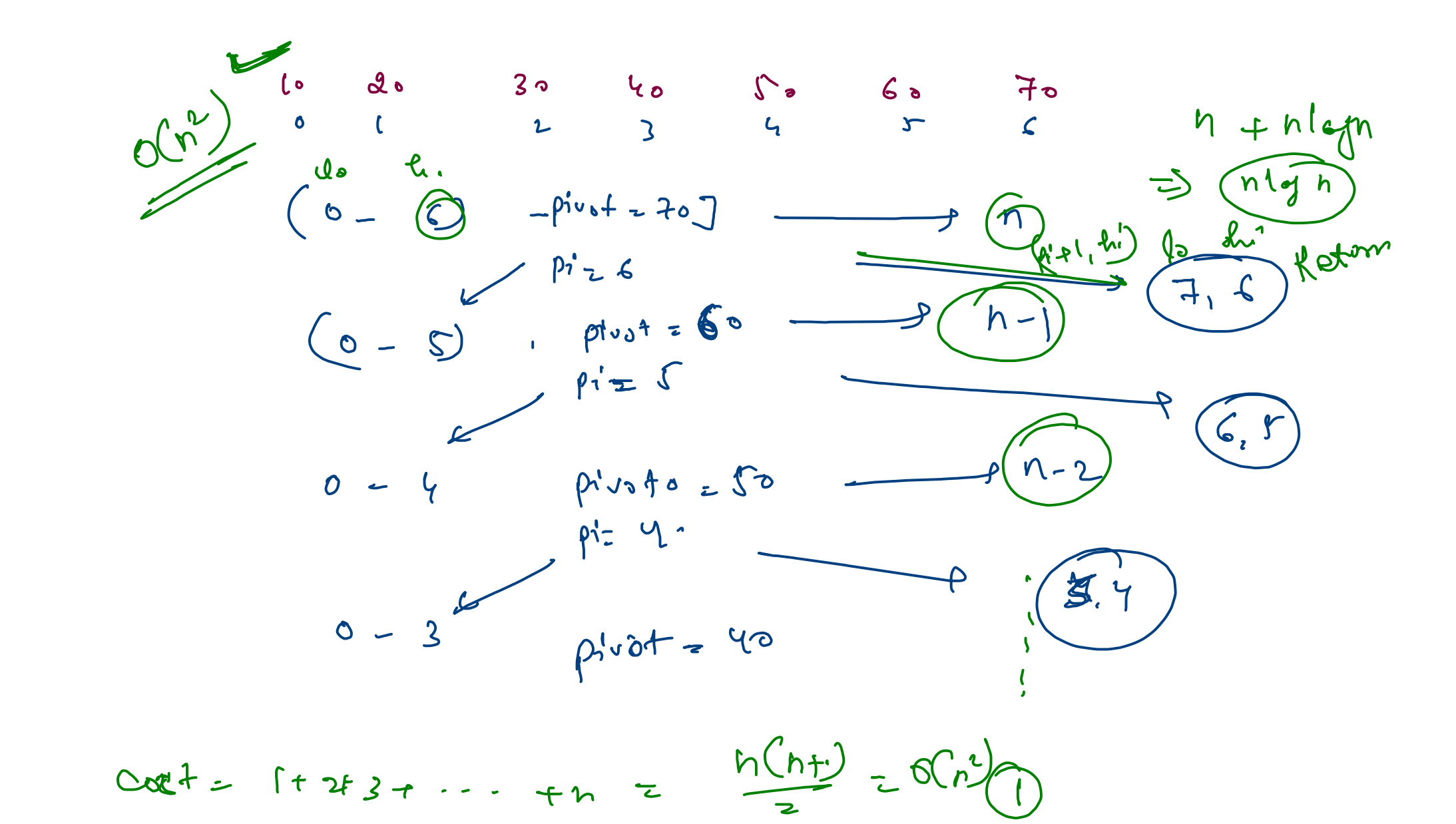
High level

```java
public static void quickSort(int[] arr, int lo, int hi) {
    int pivot = arr[hi];

    int pi = partitionIndx(arr, lo, hi, pivot);

    quickSort(arr, lo, pi - 1);
    quickSort(arr, pi + 1, hi);
}
```

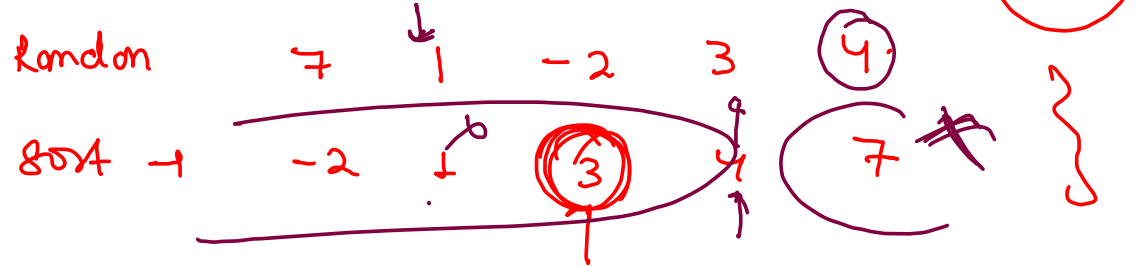
quicksort → Inplace Sorting.

```
Array Before sort : [10, 15, 19, 57, 25, 36, 14, 12, 11, 24, 27]
lo : 0 hi : 10pi : 8 Array : [10, 15, 19, 25, 14, 12, 11, 24, 27, 36, 57]
lo : 0 hi : 7pi : 6 Array : [10, 15, 19, 14, 12, 11, 24, 25, 27, 36, 57]
lo : 0 hi : 5pi : 1 Array : [10, 11, 19, 14, 12, 15, 24, 25, 27, 36, 57]
lo : 0 hi : 0pi : 0 Array : [10, 11, 19, 14, 12, 15, 24, 25, 27, 36, 57]
lo : 2 hi : 5pi : 4 Array : [10, 11, 14, 12, 15, 19, 24, 25, 27, 36, 57]
lo : 2 hi : 3pi : 2 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
lo : 3 hi : 3pi : 3 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
lo : 5 hi : 5pi : 5 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
lo : 7 hi : 7pi : 7 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
lo : 9 hi : 10pi : 10 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
lo : 9 hi : 9pi : 9 Array : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
Array after sort : [10, 11, 12, 14, 15, 19, 24, 25, 27, 36, 57]
```




10 20 30 40 50 60 70
0 1 2 3 4 5 6
O(n²)
n + nlogn
⇒ nlog n
lo hi
(0 - 6) -pivot = 70]
pi = 6
n
(p+1, hi)
lo hi
7, 6
Return
(0 - 5) pivot = 60
pi = 5
n-1
6, 5
0 - 4 pivot = 50
pi = 4
n-2
3, 4
0 - 3 pivot = 40
Cost = 1 + 2 + 3 + ... + n = n(n+1)/2 = O(n²)
1

k^{th} Smallest → 3^{rd} Smallest, (k=3)



find correct position for pIdx = k-1, after sorting. (log n)

```
int quickselect (arr, lo, hi, int k)
    if (lo > hi) return -1;
    int pivot = arr[hi];
    int pi = partitionIdx(arr, pivot, lo, hi);

    if (pi > k) { // left call
        return quickselect (arr, lo, pi-1);
    } else if (pi < k) {
        return quickselect (arr, pi+1, hi);
    } else {
        return pivot;
    }
```