Sorting 2

Monday, 22 November 2021 8:58 PM

Running stream of input. (Sort in ascending order)

2, 10, 8, 7, 9, 5..

1) How many elements to shift?

- Travel from left } O(N)
- Travel from right
- ~~Binary Search~~ – O(log(N)) + # large elements ≤ N

sorted

0	1	2	3	4	5 ---
2	5	7	8	9	10

2) Shift the elements.

$SC = O(1)$

$TC = O(N * N)$ → ∀ elements; shift all greater element to right.

Insertion Sort

Shift → $a[i] = a[i-1]$ ∀i st. $a[i-1] >$ cur. element

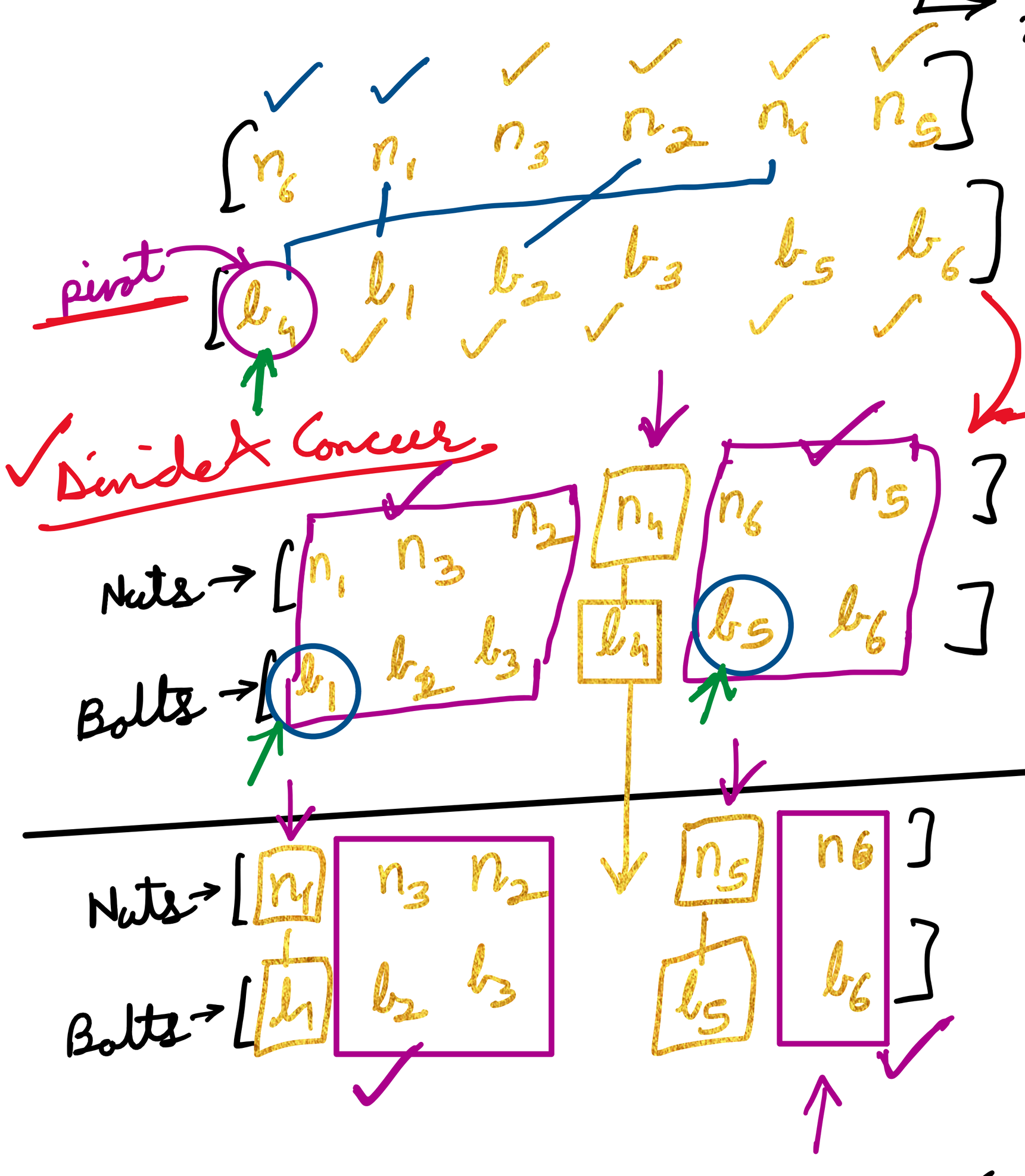
Nuts & Bolts

N nuts & bolts pair are available to us. 1:1 mapping

∀ nuts there is exactly 1 bolt & vice versa

*→ Cannot compare nut with nut & bolt with bolt.

Compare nut with bolt → exactly fits / nut is small / nut is big.



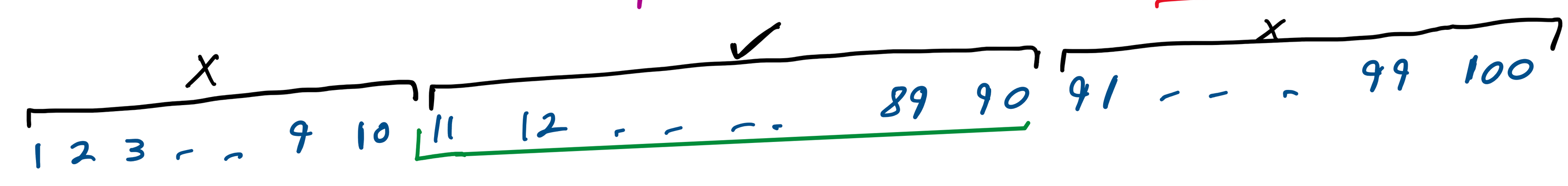
// compare every nut with every bolt $TC = O(N^2)$

O(N)

Divide & Concur

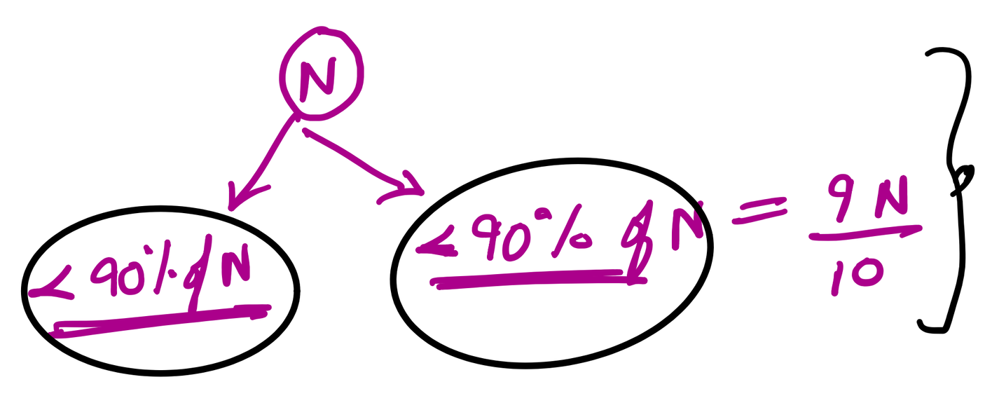
Best case → if sub problems are almost half the size $TC = O(N \log_2(N))$

Worst case → if subproblem is of size almost (N-1). $TC = O(N^2)$

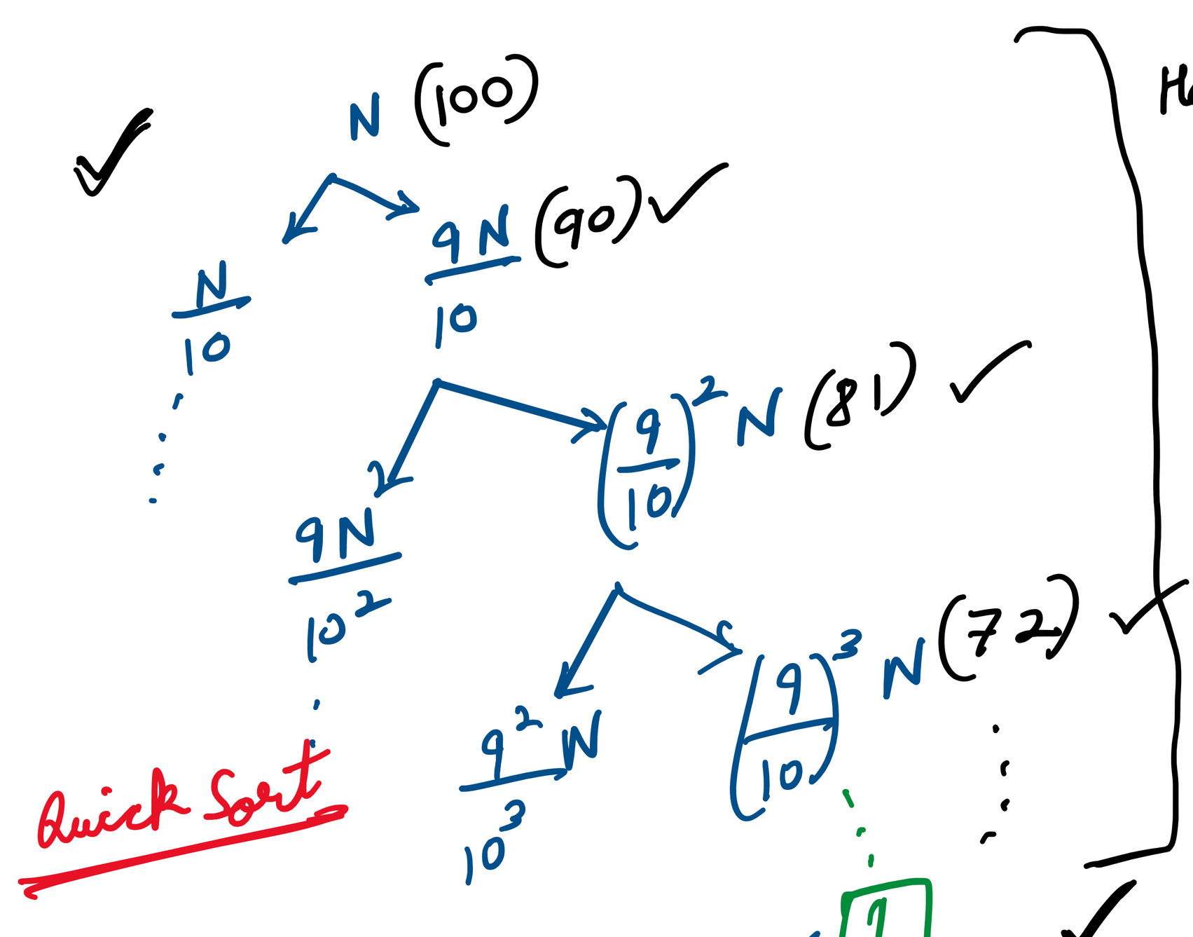


good choice of pivot = 50, 51

bad choice of pivot = 1, 100



probability of selecting pivot $= \frac{80}{100} = 0.8$



Quick Sort

Height?

$N \to \frac{9N}{10} \to \left(\frac{9}{10}\right)^2 N \ldots \left(\frac{9}{10}\right)^k N = 1$

$\Rightarrow N = \left(\frac{10}{9}\right)^N \Rightarrow k = \log_{\frac{10}{9}}(N)$

$N = 10^5$

$\log_2(N) \approx 17$ $\log_{\frac{10}{9}}(N) \approx 109$

$TC = O(N \log_{\frac{10}{9}}(N)) = 10^5 * 10^2 = 10^7$

(acceptable in 1 sec)

Sort array using quick sort.

Try to cod it

A = [4 2 5 1 6 (3)] good

A = [2 (1) 3 4 5 (6)] not good

A = [1 (2) 3 4 (5) 6]

A = [1 2 3 4 5 6]

```
void swap(a[], i, j) {
    t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

TC = O(1)

```
void quickSort(a[], l, r) {      //SC = O(log(N))   worst → O(N)
    if (l < r) {
        pi = partition(a, l, r);   //pi element is at correct location.
        quickSort(a, l, pi-1);
        quickSor(a, pi+1, r);
    }
}
```

l → 0, r → N-1

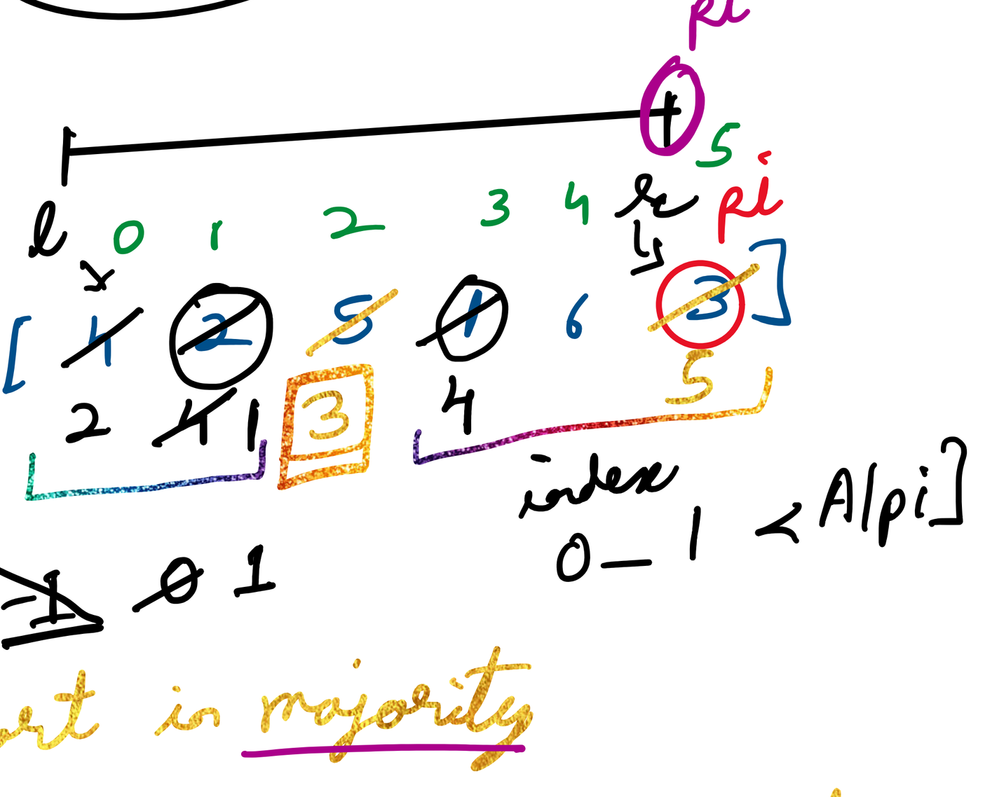
TC = O(N)

```
int partition(a[], l, r) {      //SC = O(1)
    pi = a[r];                  // random pivot
    i = l-1;                    // ∀ elements A[l]... A[i] are < pi
    for (j = l; j <= r-1; j++) {
        if (a[j] < pi) {
            i++;
            swap(a, i, j);
        }
    }
    swap(a, i+1, r);
    return (i+1);
}
```

< pi | pi

(l - - - i) < A[pi]

j = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4



index 0 – 1 < A[pi]

i = ~~-1~~ ~~0~~ 1

SC of Q sort in majority cases is better than merge sort.

Q→ Given an array with N elements.

Goal → Make all elements unique.

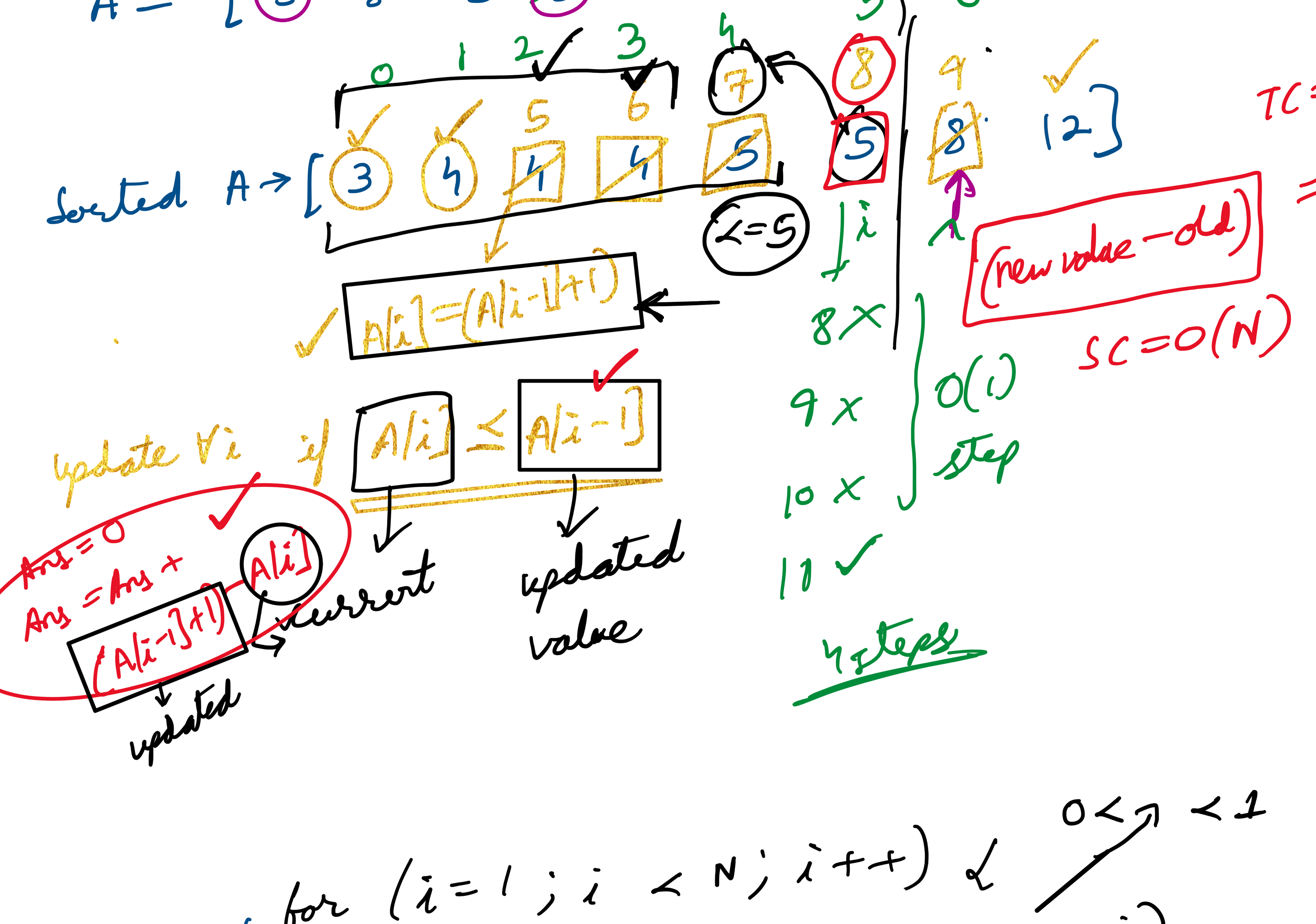
Operation → Increase any element by 1.

Find min no. of operations to achieve the goal.

→ update any no. to as small value as possible which is unique

A = [2 3 ~~2~~ 5] → ~~3~~ 4 Ans = 2

A = [(3) 8 5 ~~5~~ 4 ~~4~~ ~~4~~] 4 + 2 + 3 = 9 Ans = 9



$TC = O(N \log(N) + N) = O(N \log(N))$

$SC = O(N)$ Merge Sort

(new value – old)

$A[i] = (A[i-1] + 1)$

update ∀i if $A[i] \le A[i-1]$ (current, updated value)

Ans = 0

Ans = Ans + (A[i-1]+1) – A[i]

```
for (i = 1; i < N; i++) {
    j = (int) (Math.random() * i);
    swap(a, i, j);
}
```

O(N) 0 < r < 1