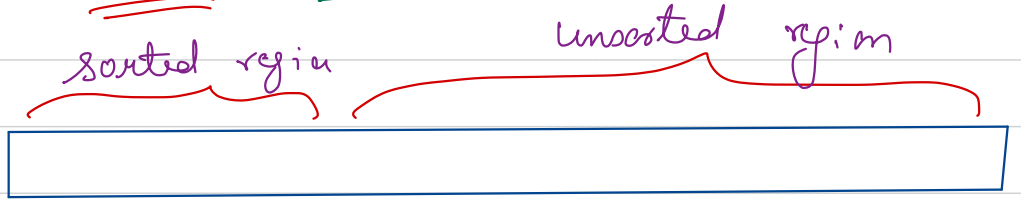
⇒ Selection Sort $O(n^2)$

→ comparison → to find the min element



Heap Sort

issue with selection sort → from the unsorted region, we try to find the min element again & again using something similar to linear search.

↓

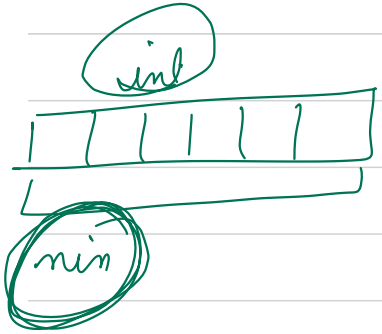
How to optimize ??

using heaps

In a subarray, find the min element.

* Heaps → Complete Binary Tree

→ These help us to find the highest priority element in $O(1)$ time and removals / additions in $O(\log n)$ time. Priority of parent > priority of child



C_1 C_2 C_3 → lower the price, better the choice

Max heap

arr → | 12 | -1 | 0 | 6 | 8 | 3 |

↓ build heap → O(n)

| -1 | 0 | 3 | 6 | 8 | 12 |

isolated

sorted

highest O(1)

pri → 0 -1

1

In place → Yes

Stable → No

Min heap

TC → Ω(n log n) O(n log n) Θ(n log n)

SC → O(1)

s2 = 6

5

4

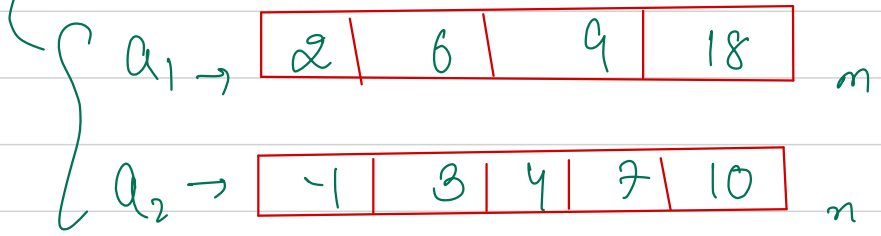
3

2

1

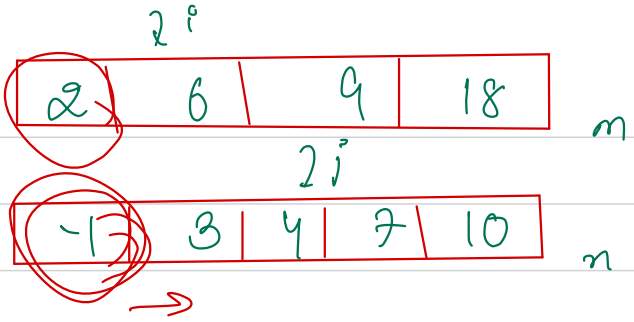
Merge Sort

Sorted arrays



Merge them 2 Sorted array

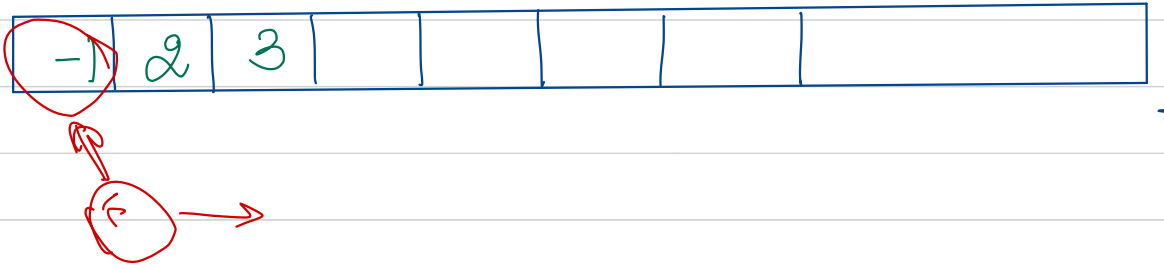
final new list $\longrightarrow$ -1, 2, 3, 4, 6, 7, 9, 10, 18

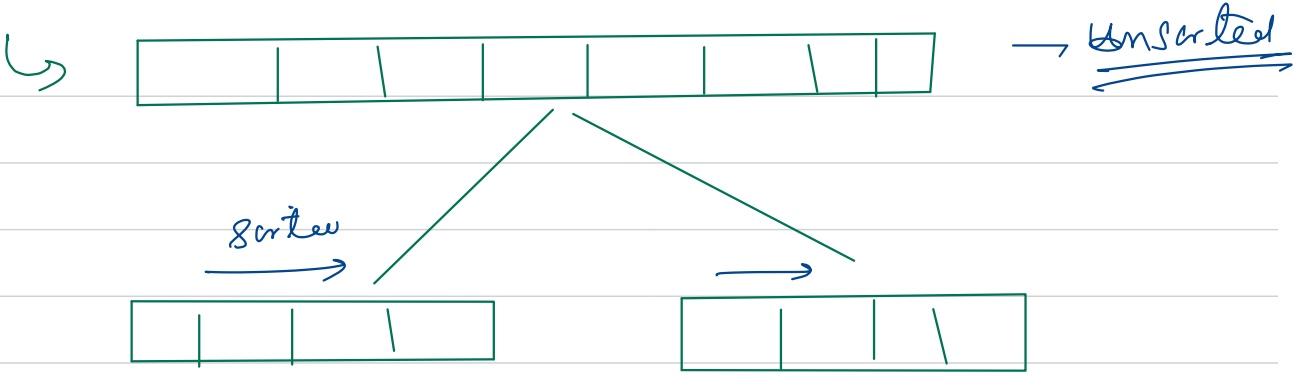


total

$m+n$

$O(m+n)$

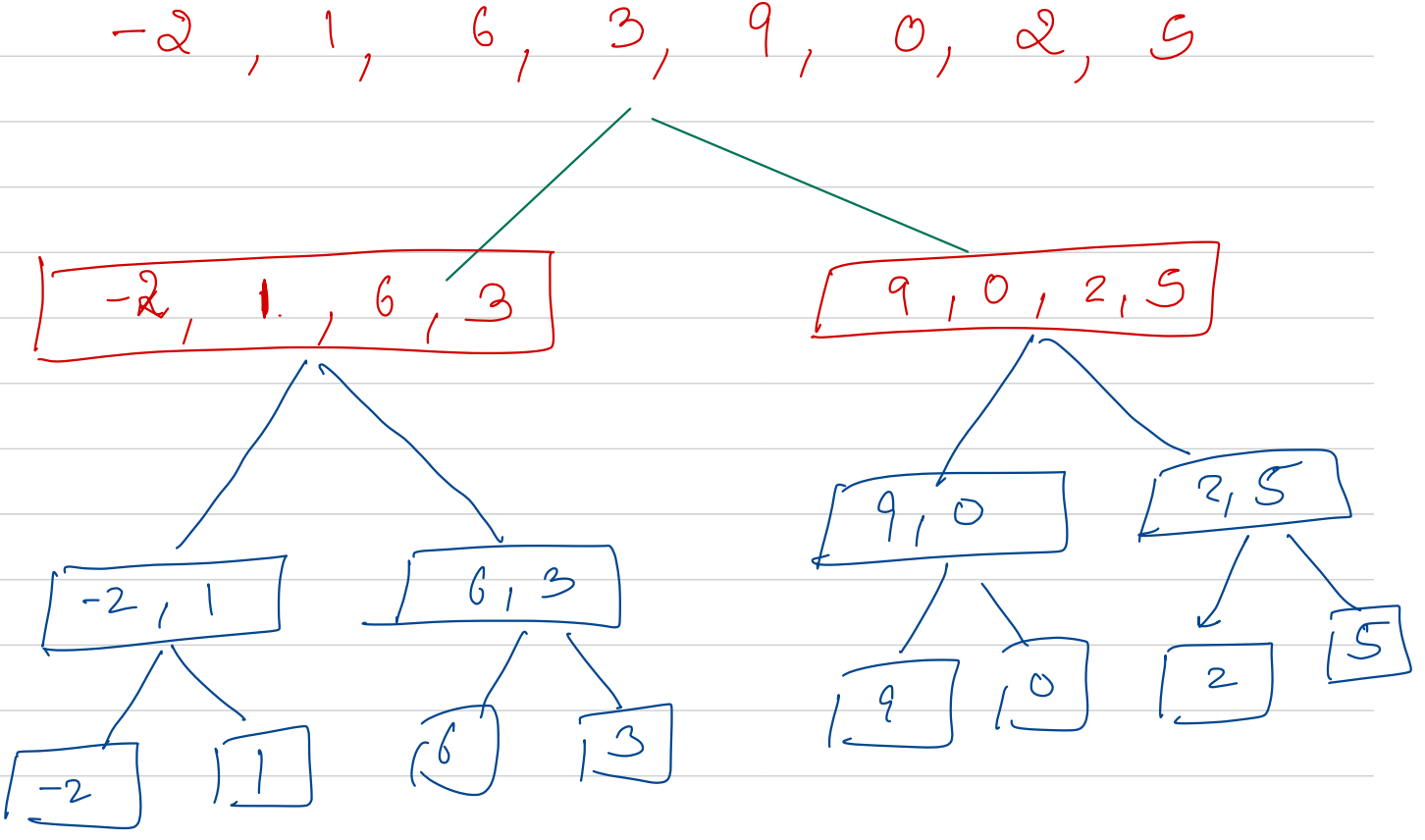


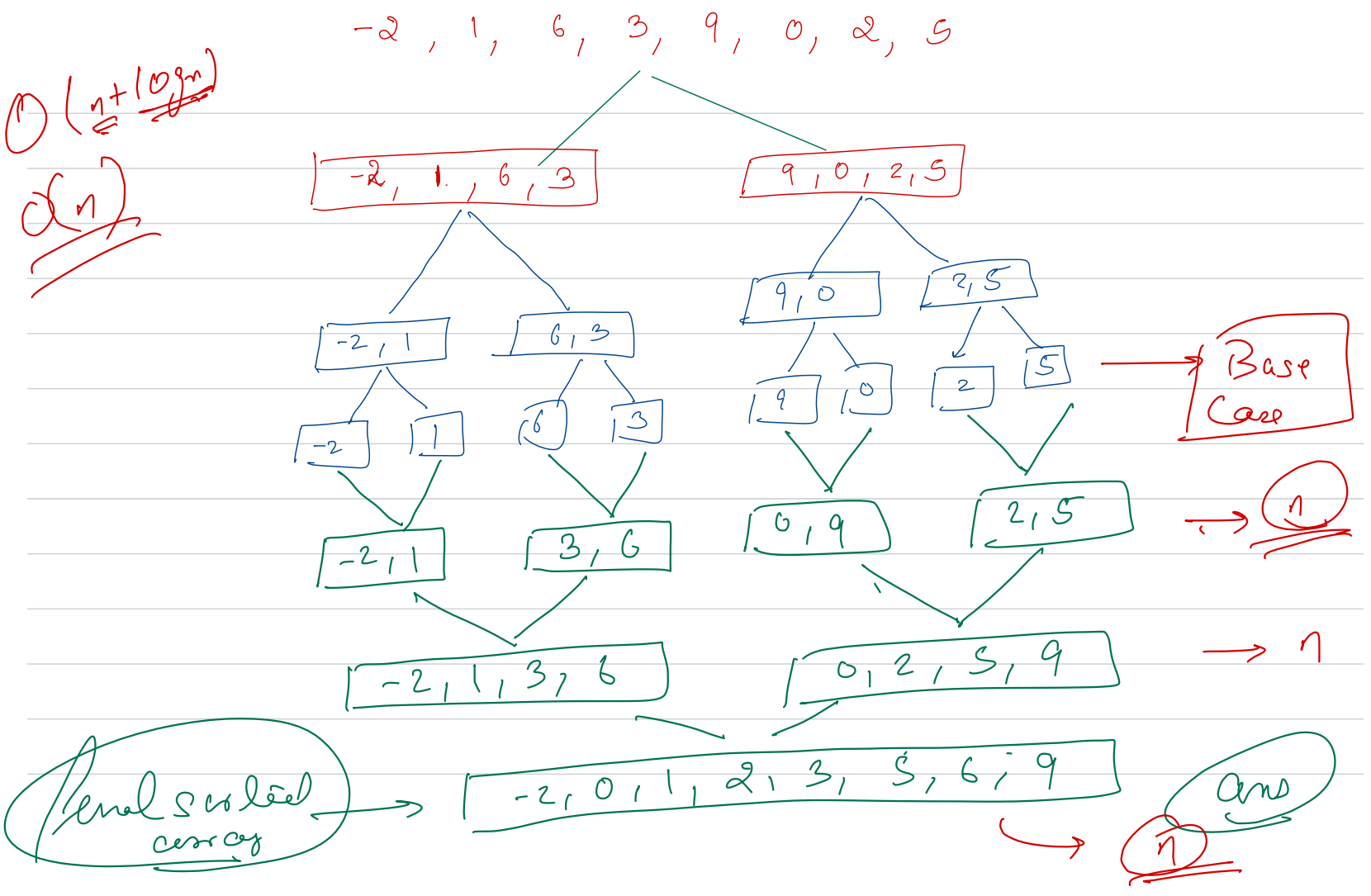


recursive fun

MS(arr[])

it will sort arr


-2, 1, 6, 3, 9, 0, 2, 5
-2, 1, 6, 3
9, 0, 2, 5
-2, 1
6, 3
9, 0
2, 5
-2
1
6
3
9
0
2
5


-2, 1, 6, 3, 9, 0, 2, 5
O(n+log n)
O(n)
-2, 1, 6, 3
9, 0, 2, 5
-2, 1
6, 3
9, 0
2, 5
-2
1
6
3
9
0
2
5
Base Case
-2, 1
3, 6
0, 9
2, 5
n
-2, 1, 3, 6
0, 2, 5, 9
n
Final sorted array
-2, 0, 1, 2, 3, 5, 6, 9
ans
n

$f(arr[0, n-1], n) =$ $f(arr[0, mid], \frac{n}{2}) + f(arr[mid+1, n-1], \frac{n}{2}) +$

↓

it merge sorts an array of size n

merge sorted arrays (lr, rr)

↓ ↓

left result right result

$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$

$T(n)$ → gives you no. of operation to merge sort an array of size n

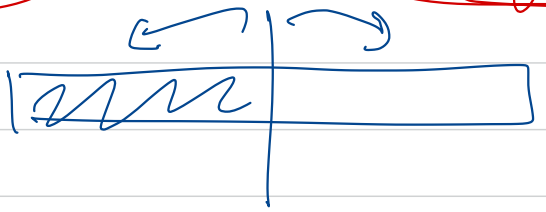
$T\left(\frac{n}{2}\right)$ → apply merge sort on left half

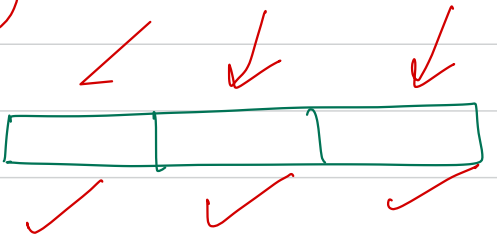
$T\left(\frac{n}{2}\right)$ → apply merge sort on right half

$O(n)$ → merge the 2 halves

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Binary Search ⟷ ternary Search





$n \longrightarrow \frac{n}{2} \longrightarrow \frac{n}{4} \cdots\cdots$

$n \longrightarrow \frac{n}{3} \longrightarrow \frac{n}{9} \cdots\cdots$

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

$T(n) = T\left(\frac{n}{3}\right) + 2$

$O(\log n)$

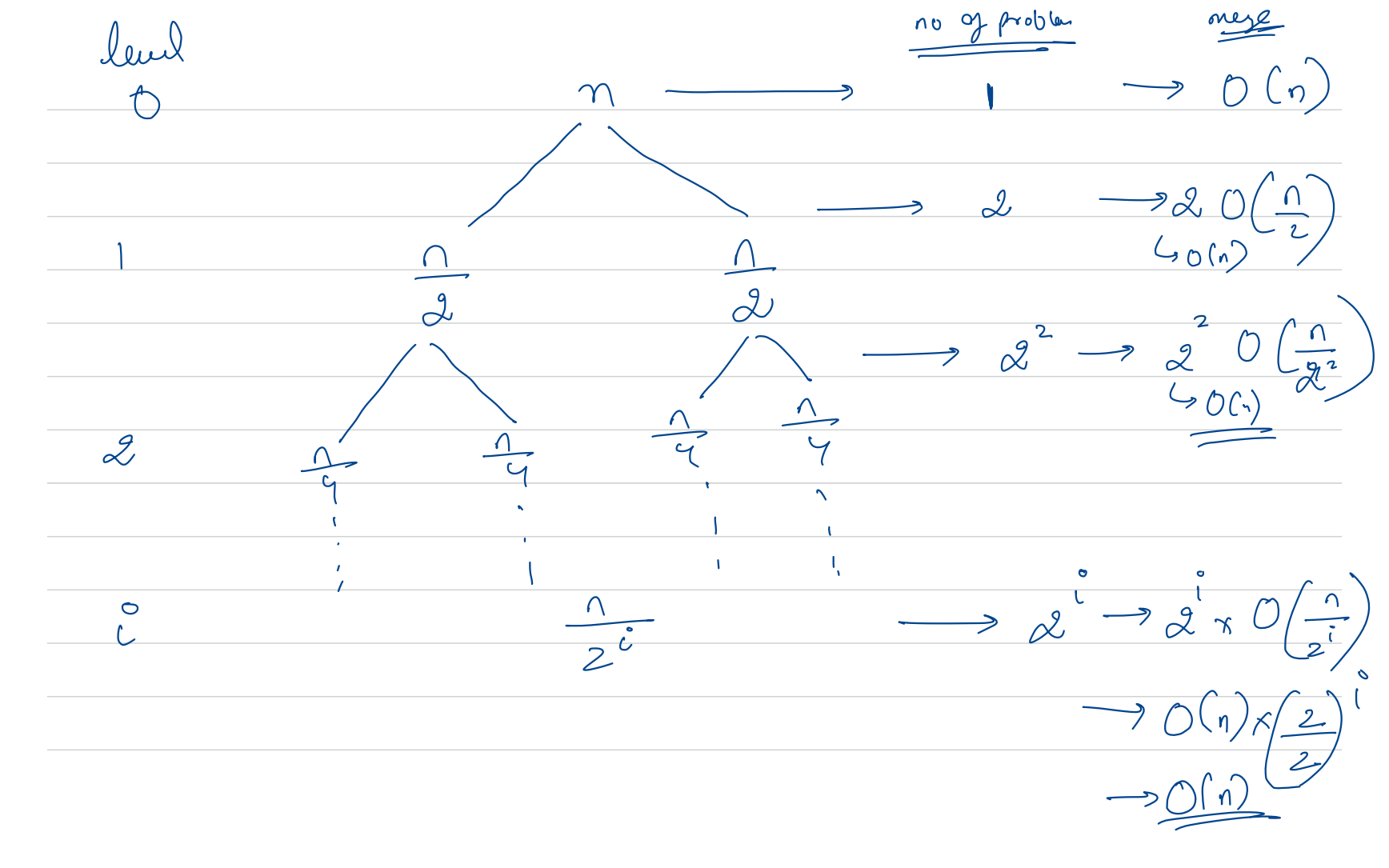
$O(\log n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Bloomberg

master theorem

Recursion Tree


level
no of problem
merge
0
n
1
O(n)
1
2
2 O(n/2)
O(n)
n/2
n/2
2
2^2
2^2 O(n/2^2)
O(n)
2
n/4
n/4
n/4
n/4
i
n/2^i
2^i
2^i x O(n/2^i)
O(n) x (2/2)^i
O(n)

On each level we do $O(n)$ task

Time Complexity = $\sum_{i=0}^{\log n} O(n)$ $\longrightarrow$ $O(n \log n)$

TC $O(n \log n)$

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \cdots\cdots \frac{n}{2^k}$

$\frac{n}{2^k} = 1$

$k = \log_2 n$

SC $\rightarrow O(n)$

Inplace → No

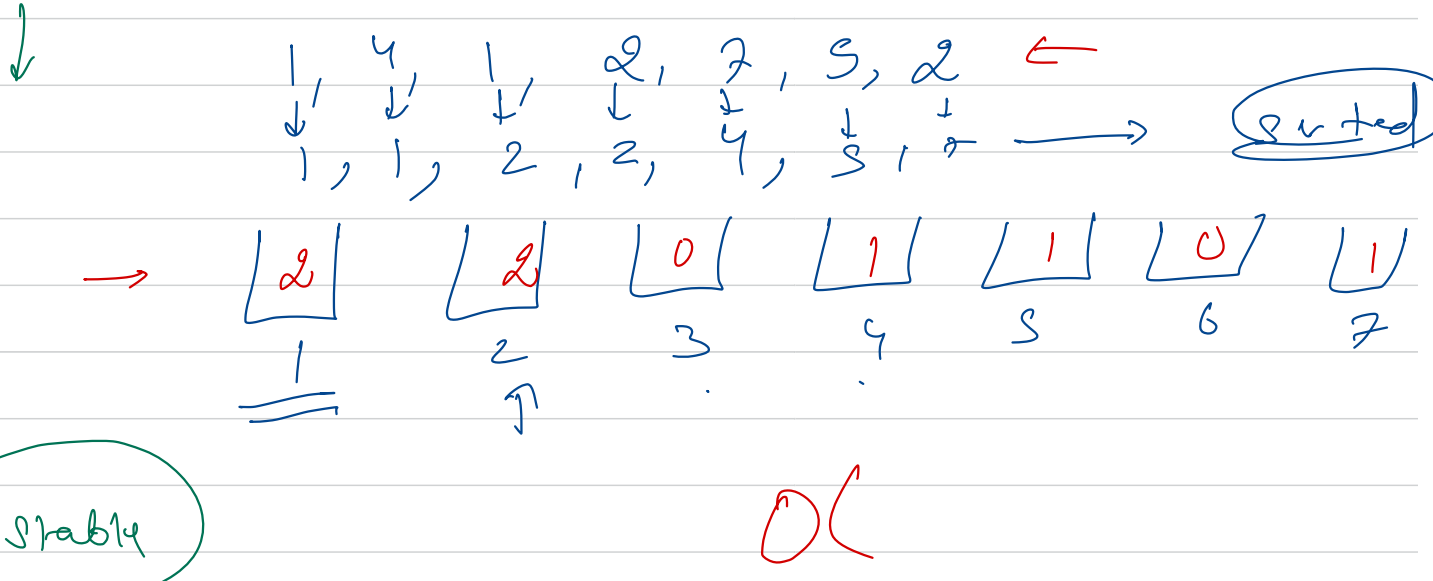
Stable → Yes

No of compars → O(nlogn)

$n! \rightarrow n^2 \longrightarrow n \log n \longrightarrow$ linear time

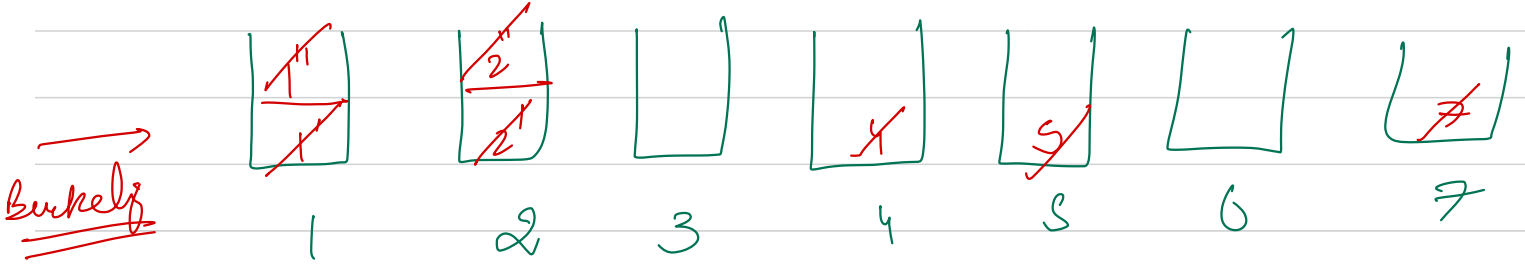
Counting Sort

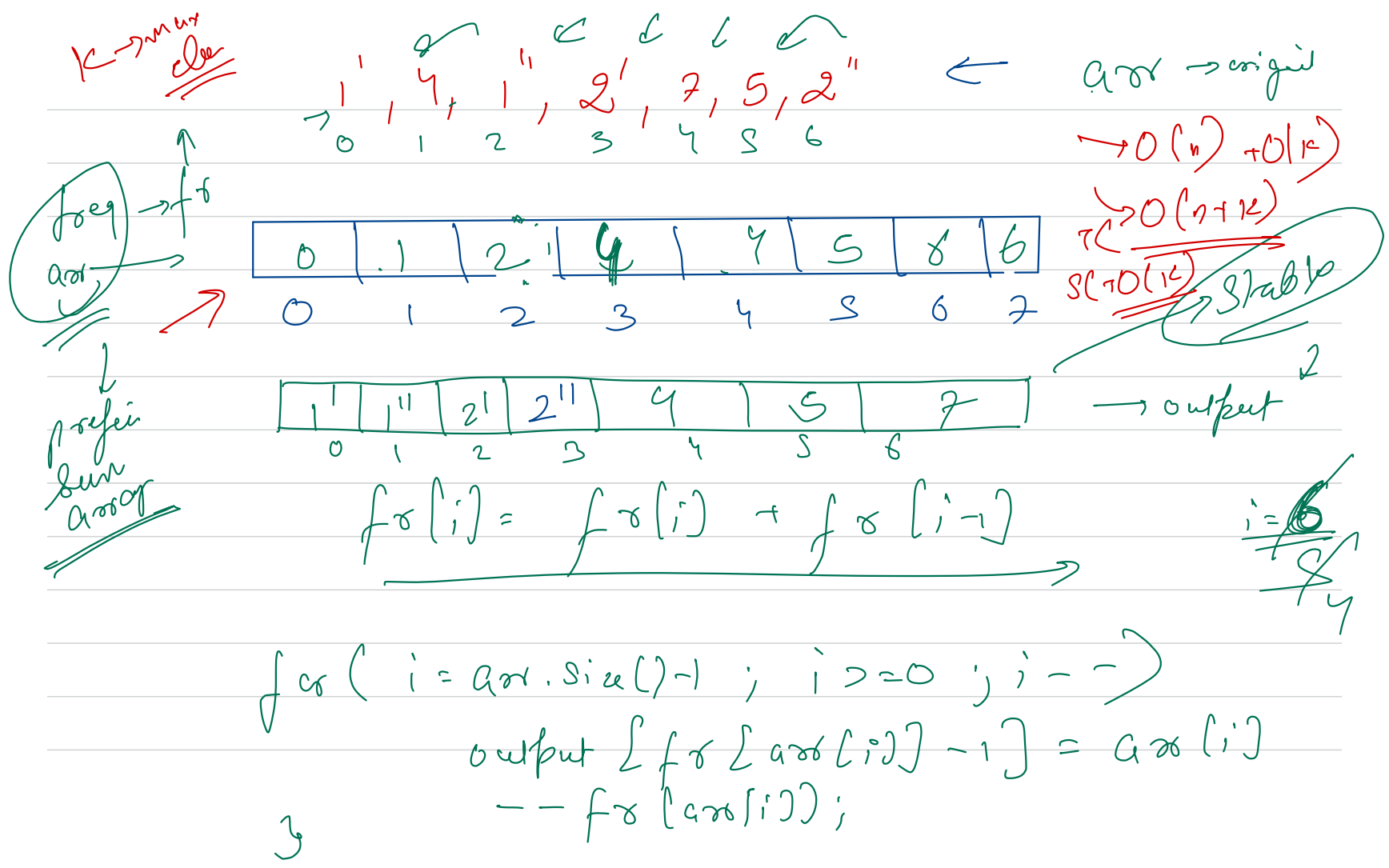
I not use cuber counting sort



→ $1', 4, 1'', 2', 7, 5, 2''$

$1'', 4, 1', 2', 7, 5, 2'$




k → number of digits
1', 4, 1'', 2', 7, 5, 2''
0 1 2 3 4 5 6
arr → original
→ O(n) + O(k)
→ O(n+k)
S(+O(k)
Stable
freq
arr
prefix sum array
0 1 2 1 4 4 5 6 7
0 1 2 3 4 5 6 7
1' 1'' 2' 2'' 4 5 7
0 1 2 3 4 5 6
→ output
2
freq[i] = freq[i] + freq[i-1]
i = 6
for (i = arr.size()-1 ; i >= 0 ; i--)
output [freq [arr [i]] - 1] = arr [i]
-- freq [arr [i]) ;
}

What is pr denoting after prefix sum.

At any i^{th} index, pr[i]-1 denotes the index of the last element i from arr (original array)

Counting sort fails for inputs with large values

$a[i] \le 10^9$

$K \rightarrow 10^6$

$1, 10^6, 0$

10^6