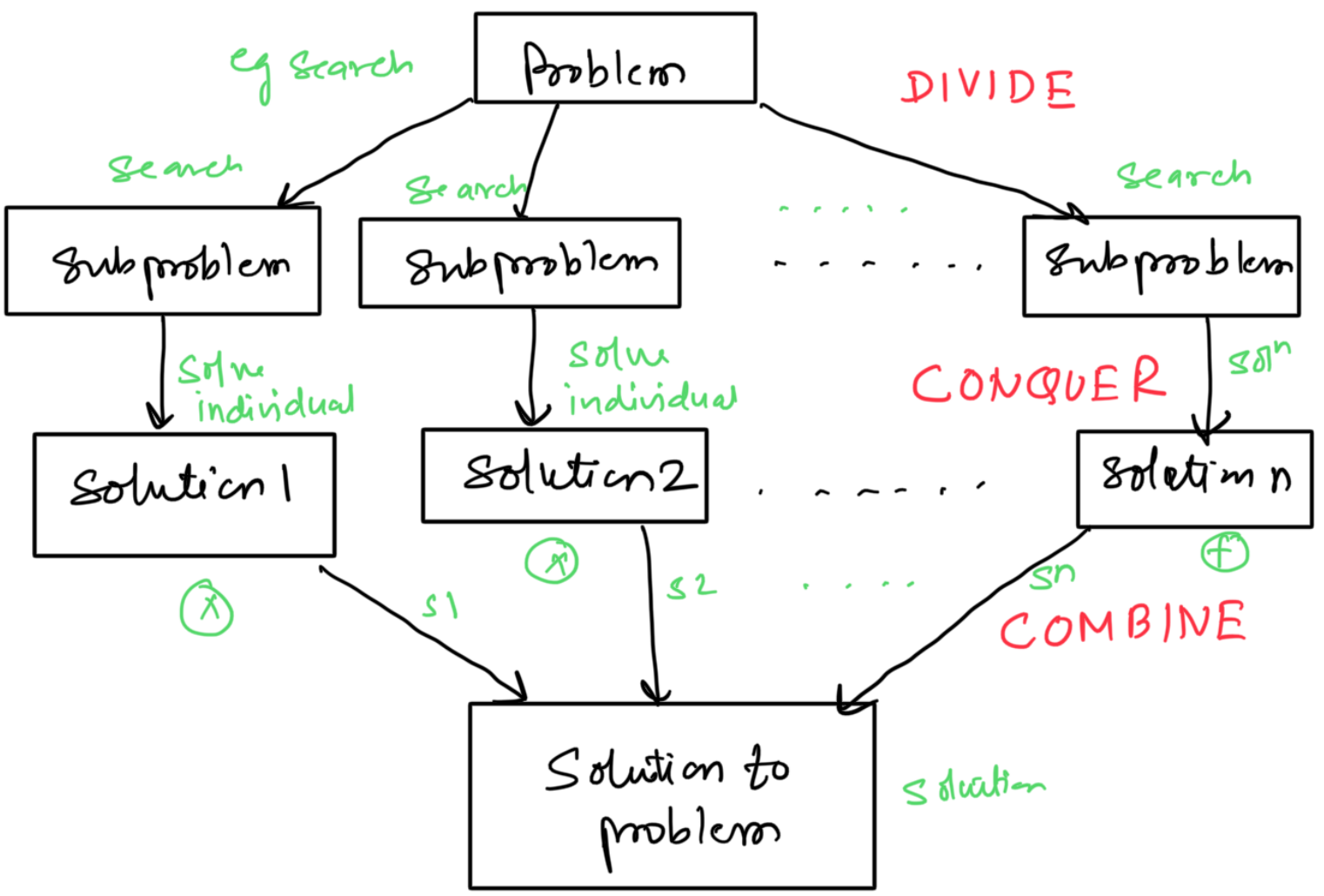
Divide - and - Conquer

— popular strategy for algorithm



Divide & Conquer

— breaks a problem into subproblems that are similar to the original problem. Recursively solves the subproblem & finally combines the solutions to the sub problem to solve the original problem

— Consists of 3 parts —

① Divide the problem → into a number of subproblems that are smaller instances of the same problem.

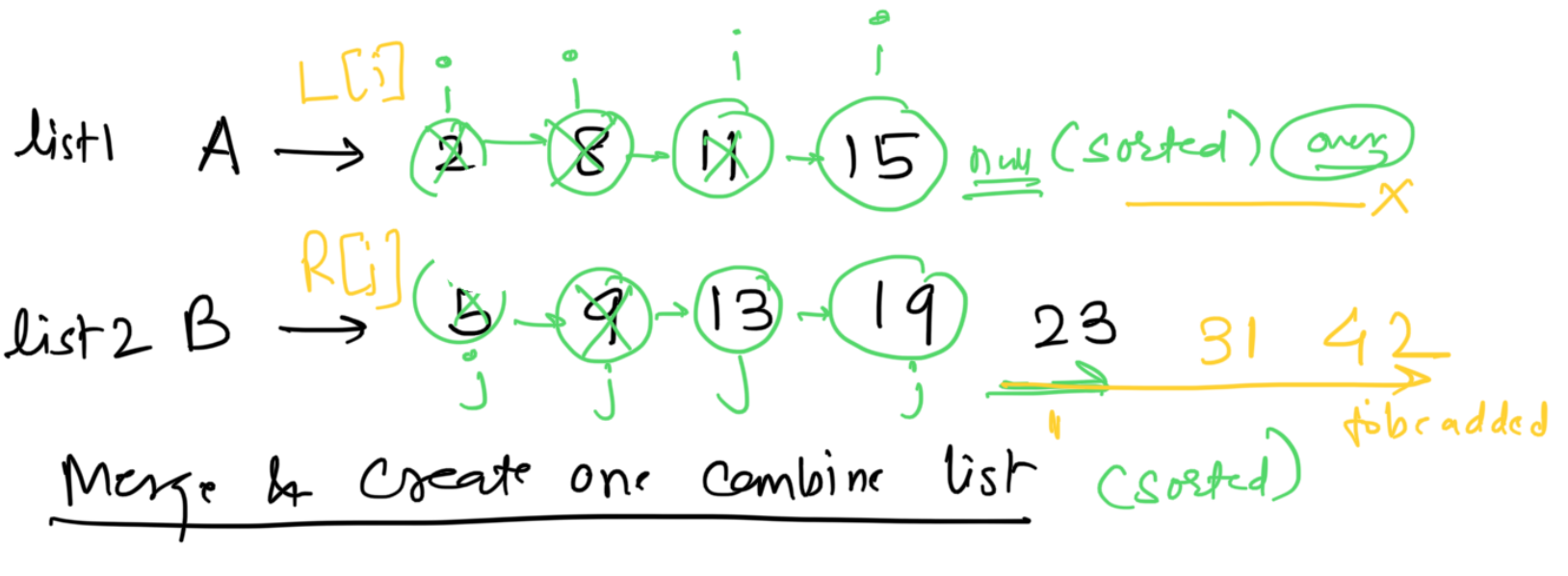
②

Conquer the problem → solving using Recursive Strategy.

(3) Combine the problem → solution to the subproblems into the solution for the original.

Merge Sorting →

Merge →



Merge & Create one combine list

C → 2 5 8 9 11 13 15 19 23 31 42

Compare

Added all remaining element

```
while (1)                                   (2)
{
    if (A[i] < B[j])
        C[K++] = A[i++];
    else
        C[K++] = B[j++];
}
```

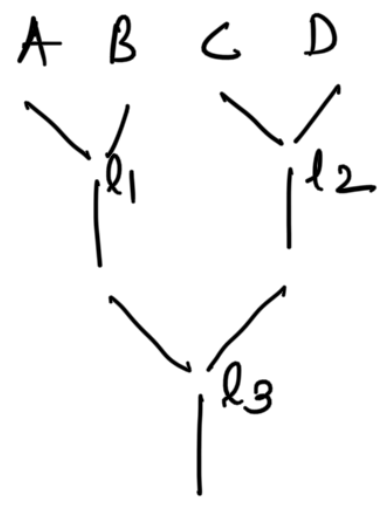
→ Compare

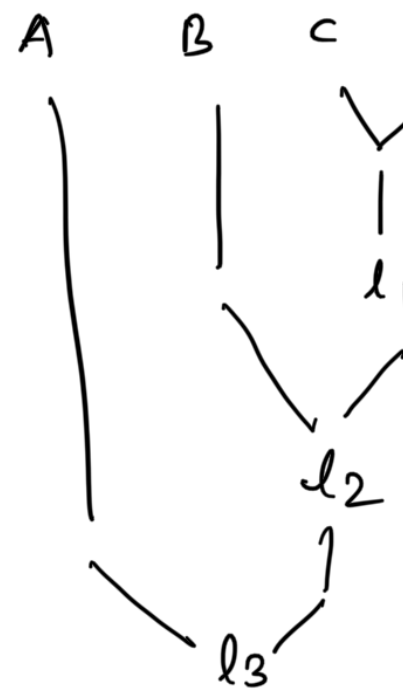
```
list1 ⇒ { for (i=m; i<=m; i++)
              C[K++] = A[i];
```

Add all Remaining

```
List 2 => { for (j=m; i<=n; j++)
              C[k++] = B[j];
```
elements from list1 & list2

Example





Iterative Merging $\Leftarrow$ { m-way merging ; 4-way w merging

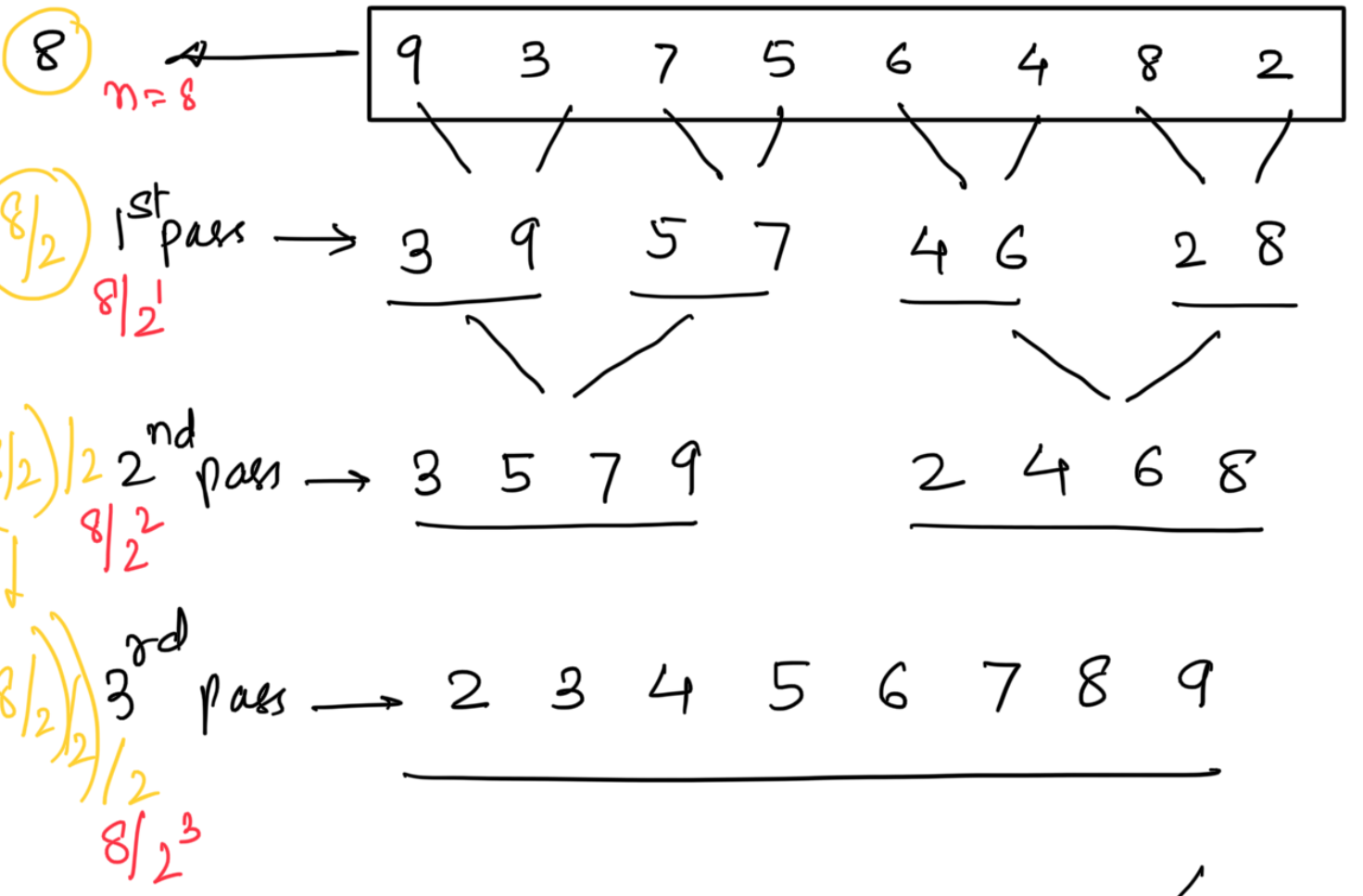
Merge Sort → Merge → Recursive procedure

Array ⇒ 9 3 7 5 6 4 8 2

Base cond^n } ⇒ Single element → Smallest problem (1)

2 or more than 2 → Bigger problem (≥ 2)

combine



$$\frac{8}{2^3} = 1$$

log n

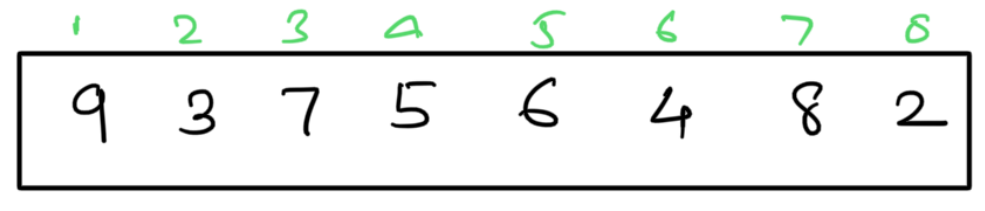
$$\frac{n}{2^3} = 1$$

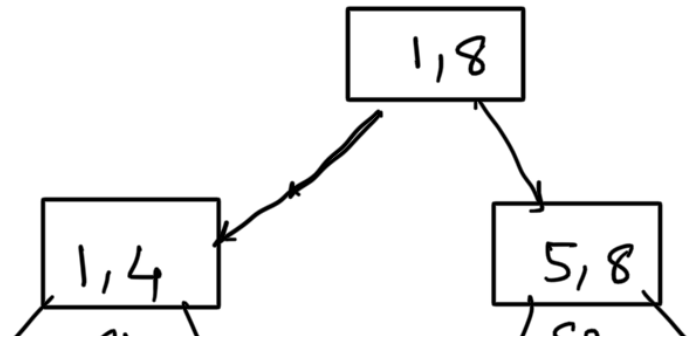
$$n = 2^3$$

$\log n = 3 \Rightarrow$ No of passes

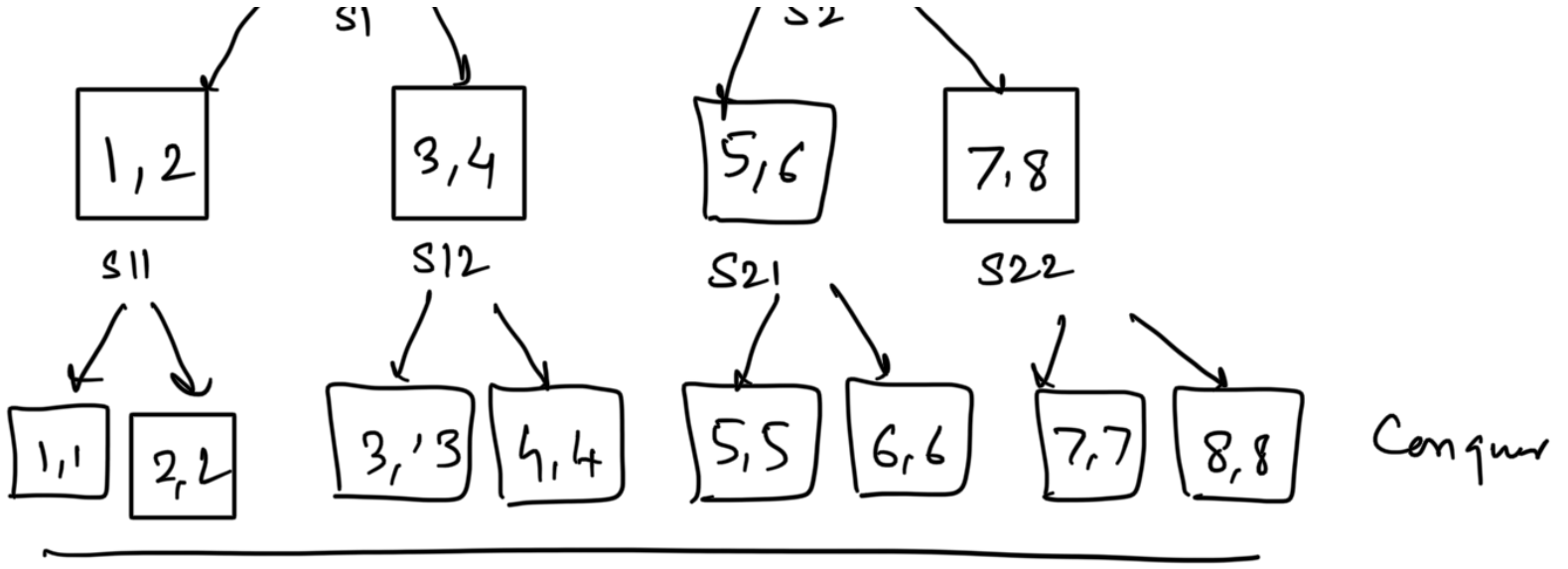
Time Complexity $\Rightarrow (\log n) \times n$

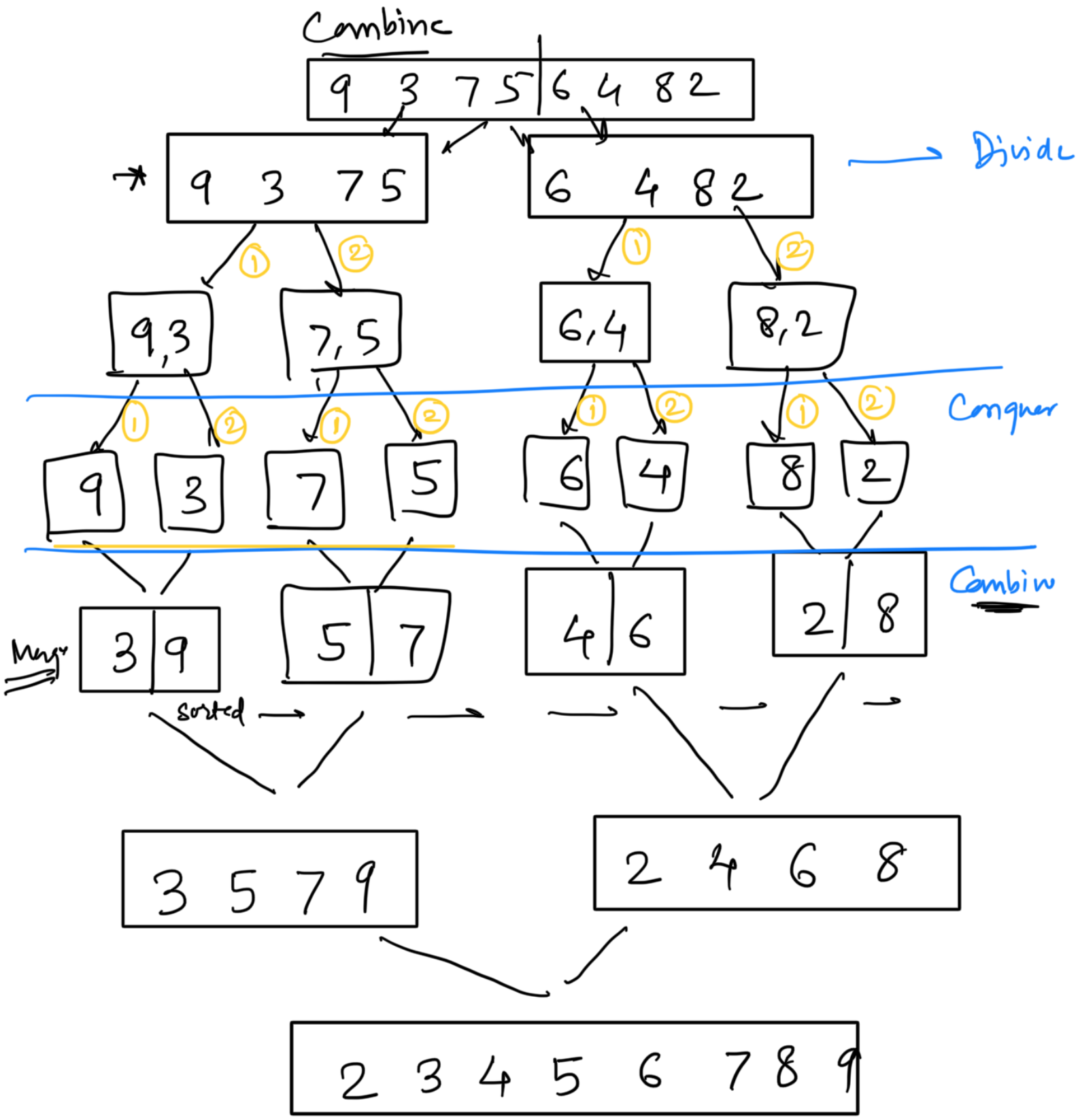
$\Rightarrow n \log n$





Divide






```
Mergesort( l, h)
{   if (l < h)
    {   mid = (l+h)/2;

    Mergesort (l, mid) : }
```

```
        MergeSort(mid+1, h);
        Merge (l, mid, h);
    }
}
```

Adv → 1) Large size list / data

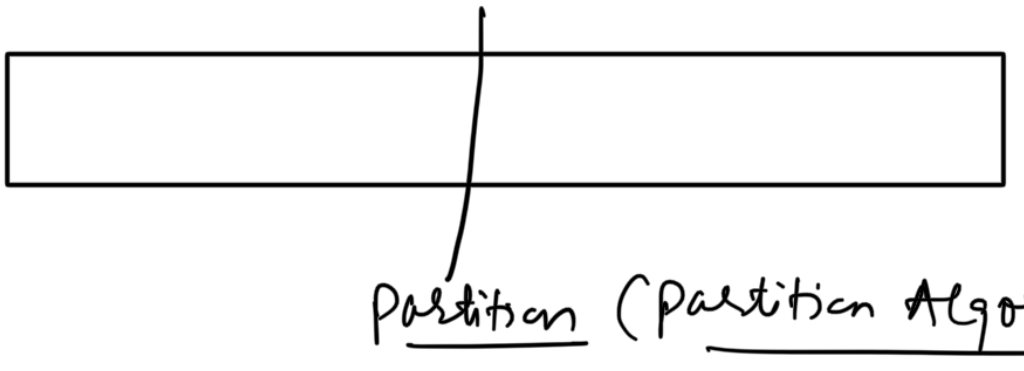
2) linked list merge sort *

3) External sorting

4) Stable

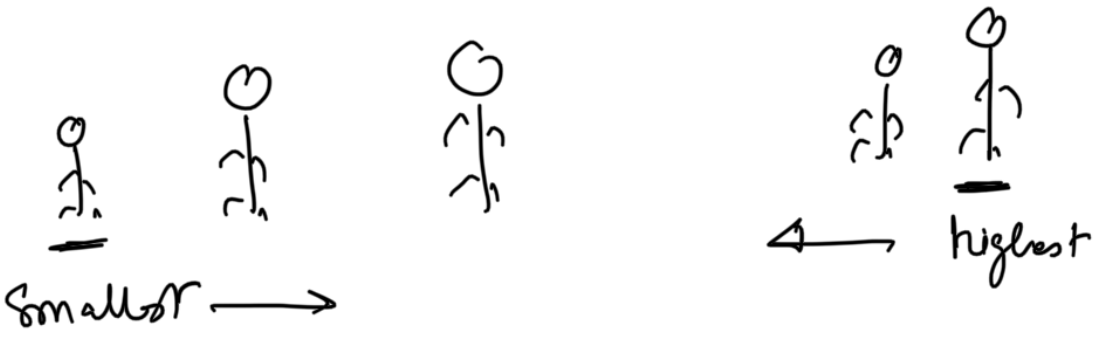
Quick Sort → eg. Height wise sitting arrange

- most popular sorting techniques
- name suggest quick sort
- works with partitioning the array to be sorted & each partition in turn sorted recursively. Hence called as partition exchange sort.

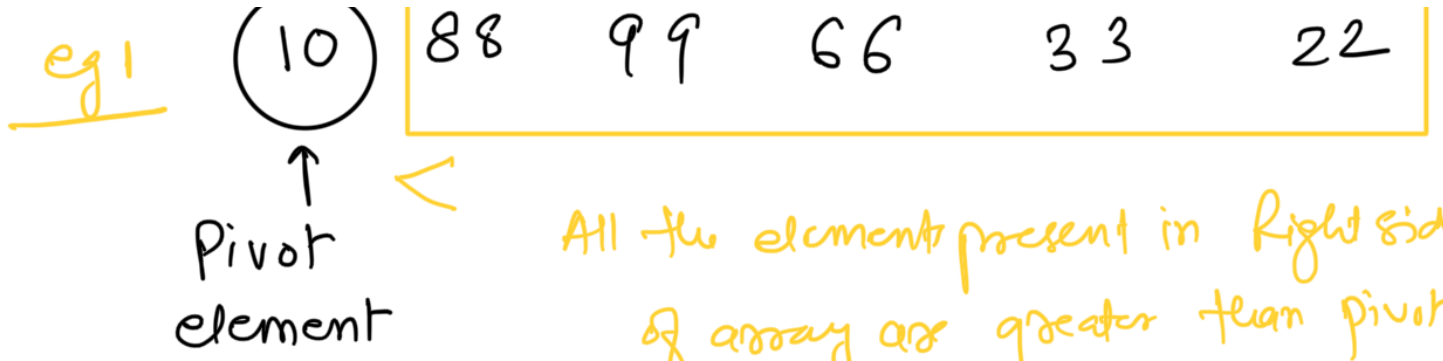


- Recursion & Divide & Conquer technique for sorting is implemented

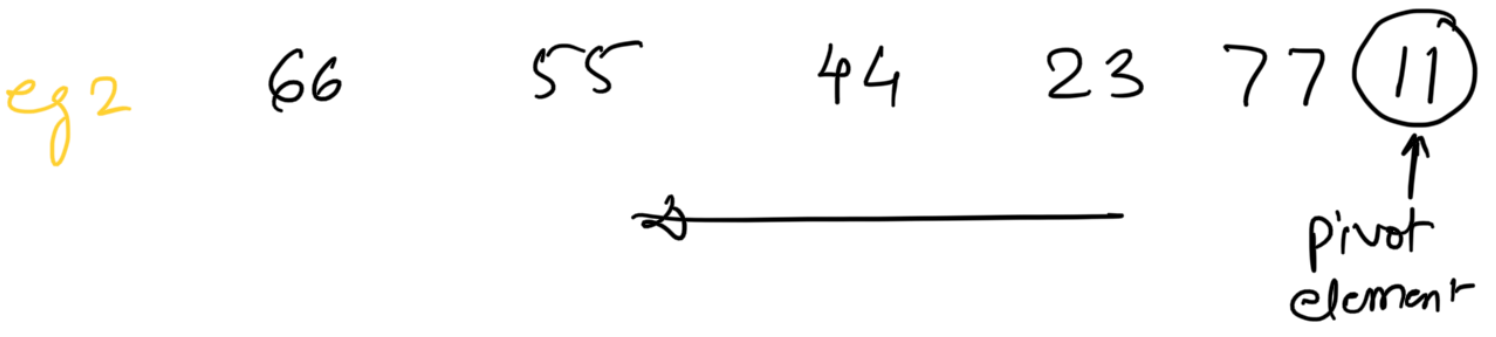
Ex1



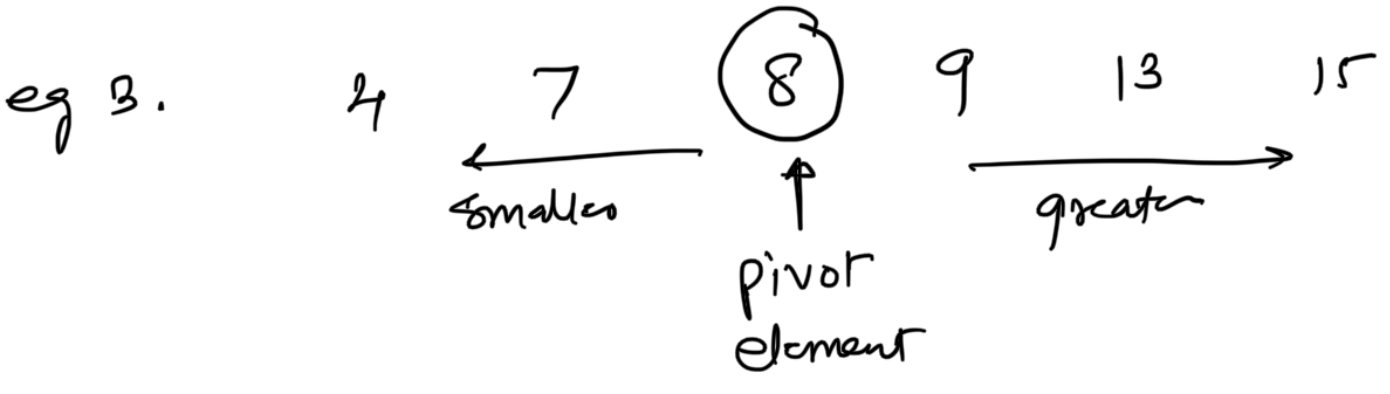
eg 1



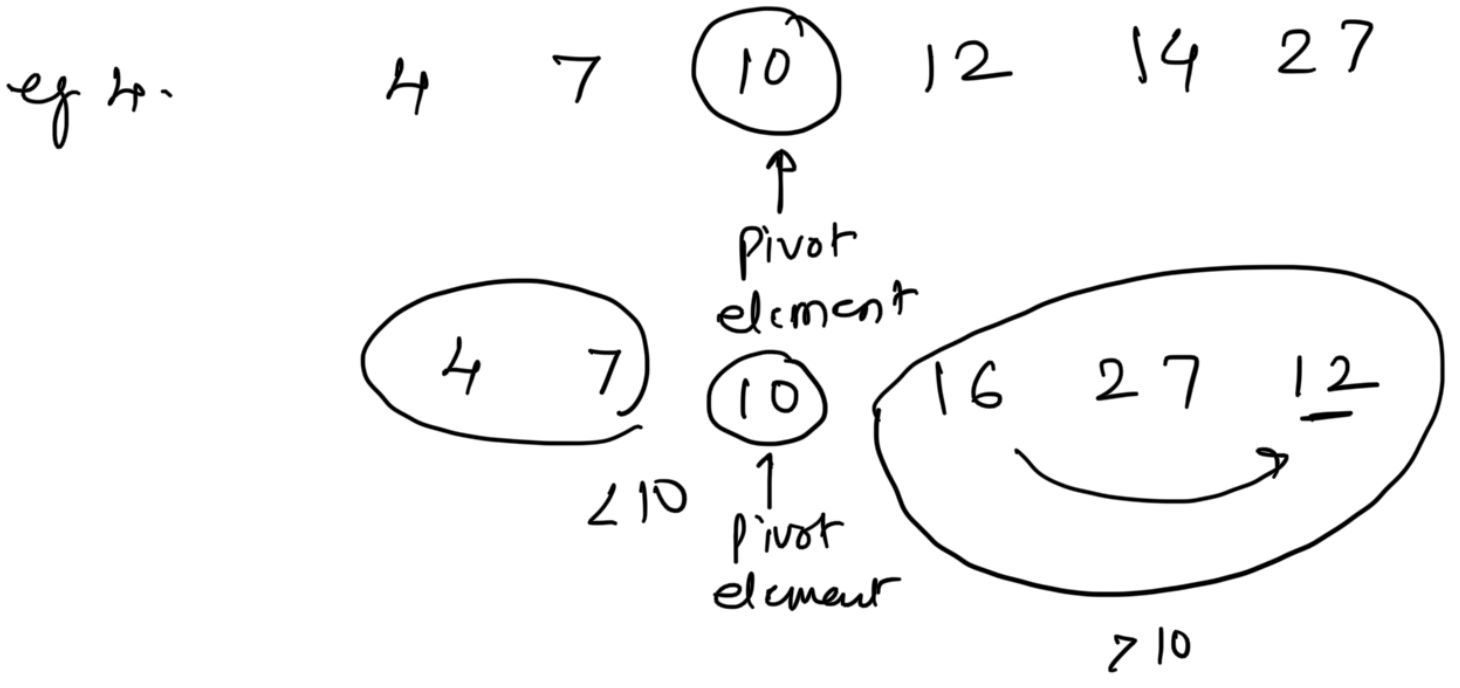
eg 2



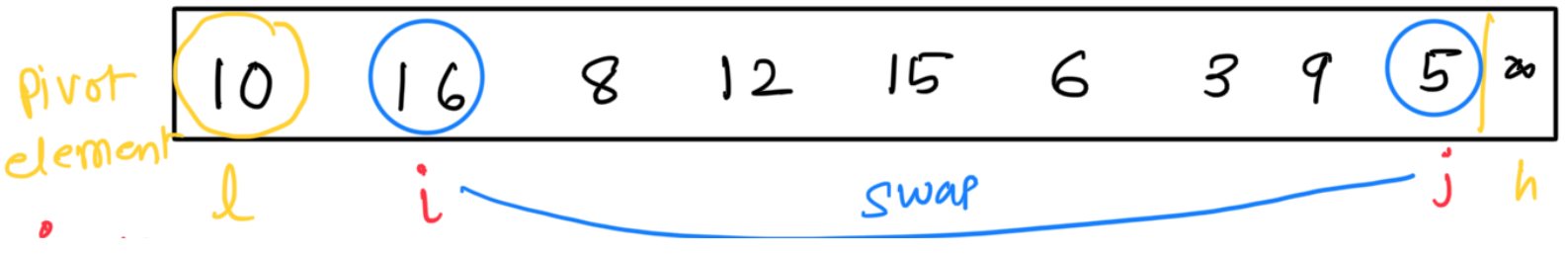
eg 3.



eg 4.



Quick Sort Example -



i > 10
j < 10

10 5 8 12 15 6 3 9 16

i i j

swap

10 5 8 9 15 6 3 12 16

i swap j

10 5 8 9 3 6 15 12 16

j i

interchange

stop the iteration

j & pivot ⇒ interchange

---

6 5 8 9 3 10 15 12 16

i j pivot

Recursion Repeat

j i Recursion Repeat

6 5 3 9 8 10 15 12 16

j x i

3 5 6 9 8 10 12 15 16

3 5 6 8 9 10 12 15 16

pivot fix

Left i > pivot → stop

Right j < pivot → stop

i & j swap

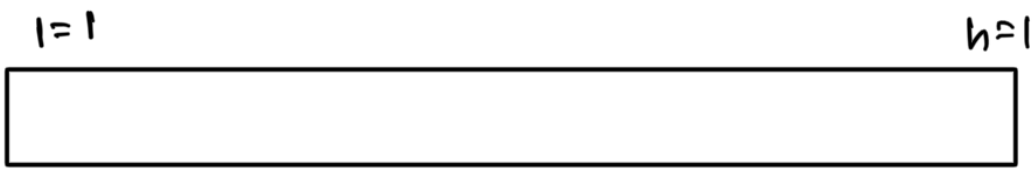
j x i → cross (stop point)

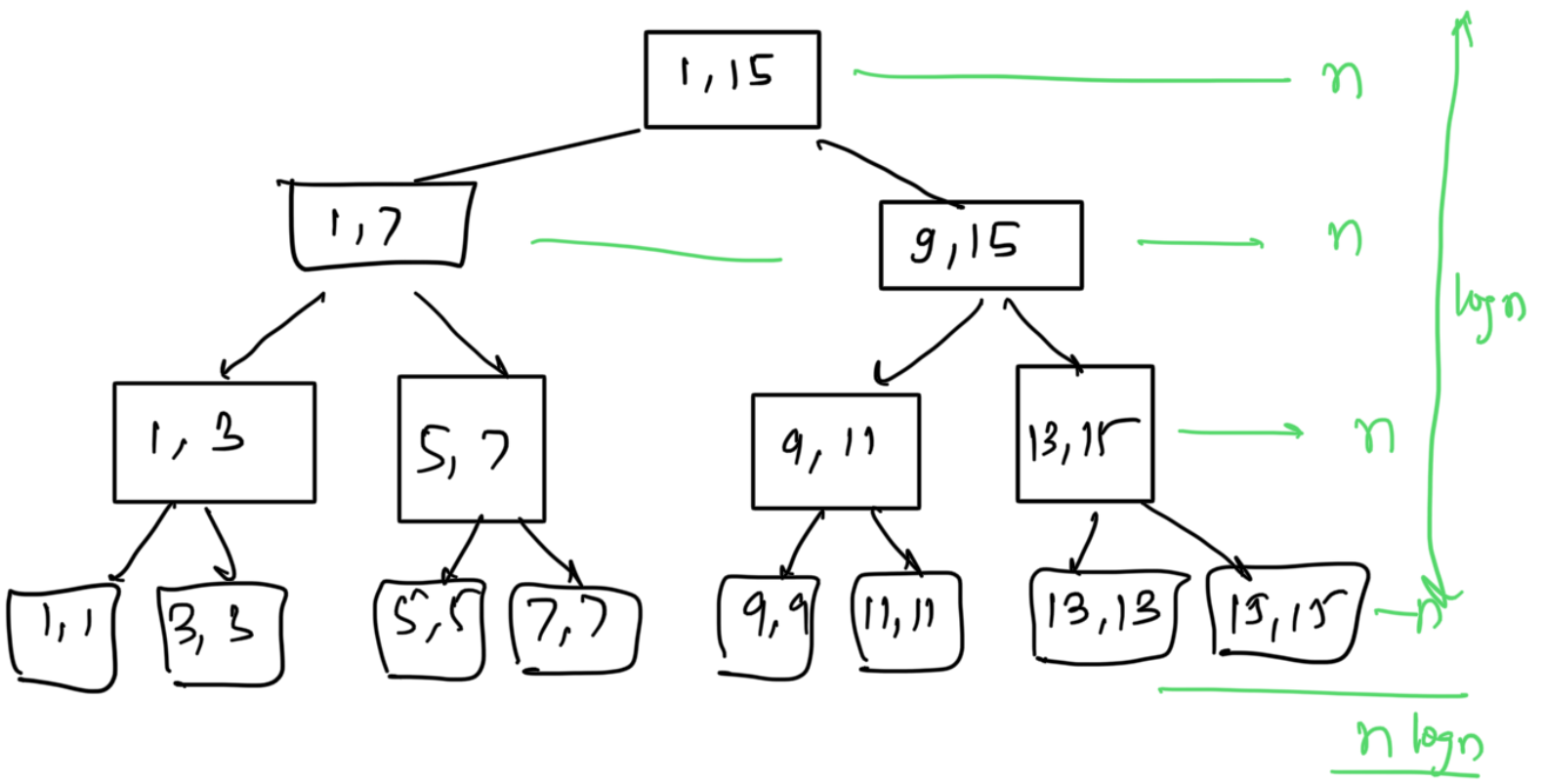
(small) (large)

pivot & j exchange

} partitioning process

pivot ⇒ j





Time Complexity ⇒ $O(n \log n)$

Assume ⇒ partition is in mid value

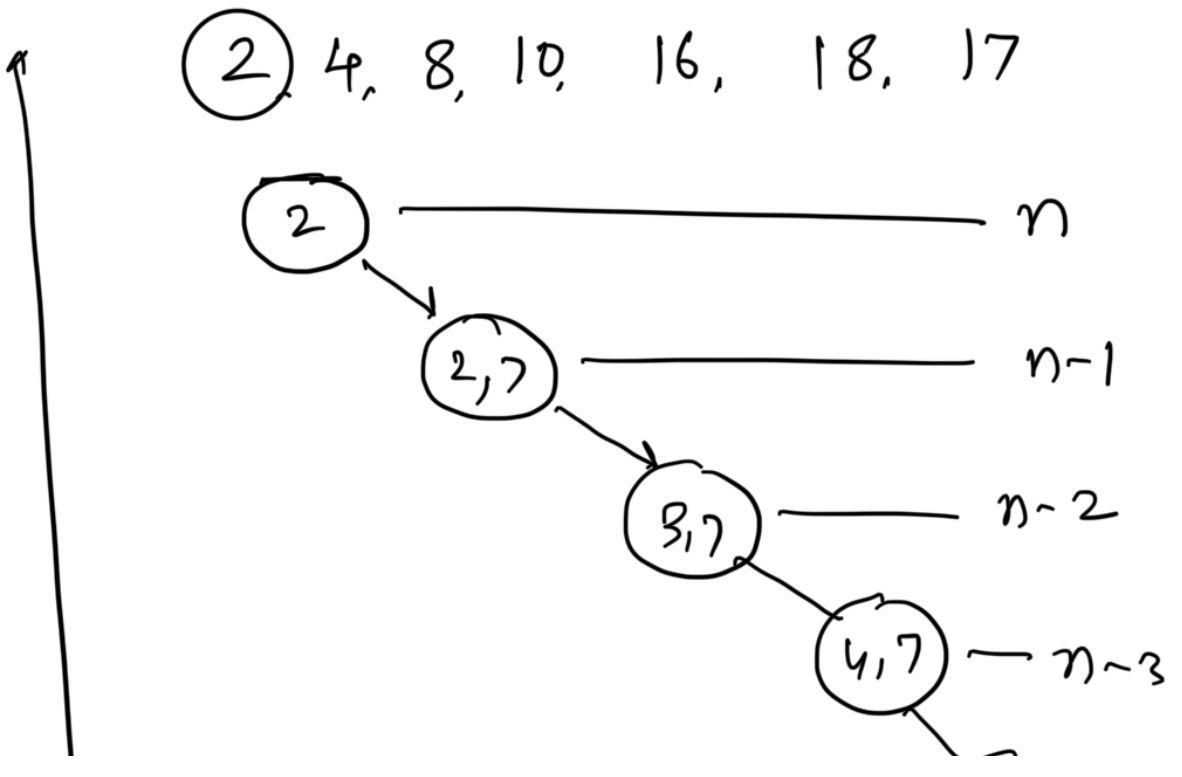
Pivot ⇒ Middle element or Random element

Best Case ⇒ Middle element (pivot)

1, 2, 3, 4, 5, 6, 7

Median ⟶ Best case

Worst Case ⇒ Sorted Array. (A↑ or ↓D)



log n

$$n+(n-1)+(n-2)+\cdots 1$$

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

$$= O(n^2)$$ ← Worst case Time Complexity