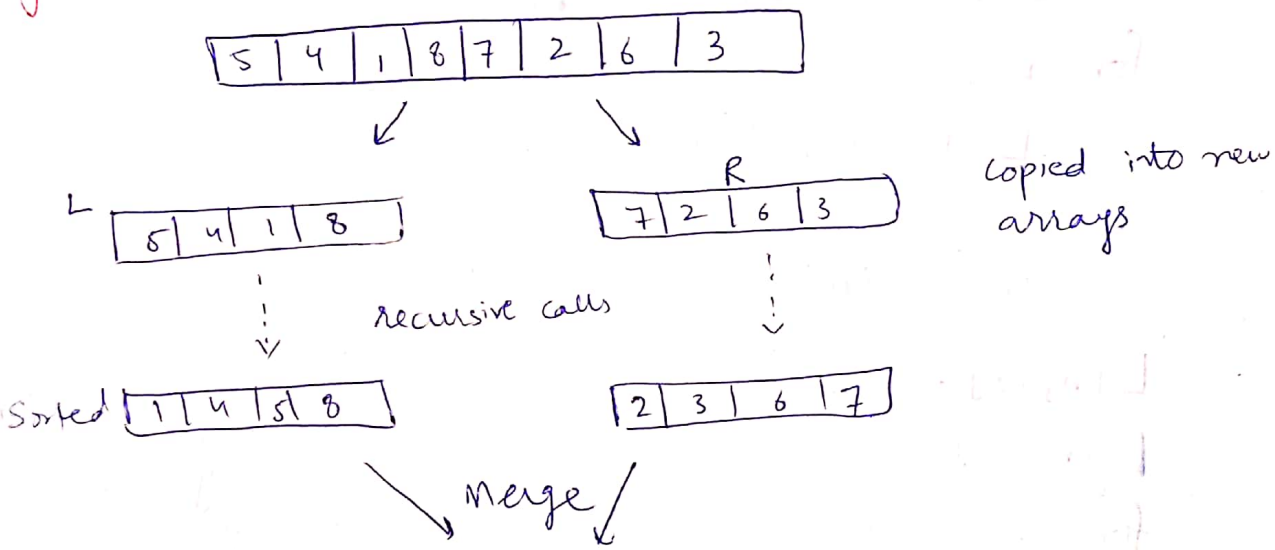
previously we did sorting in $O(n^2)$ Insertion sort.
Can we do any better?

Divide and Conquer Rule

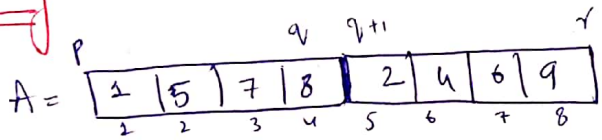
An major algorithm design technique
it leads to recurrences

- Divide
- Conquer → solve recursively
- → Combine

Merge Sort (recursive algo)

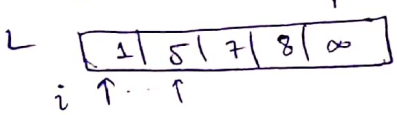


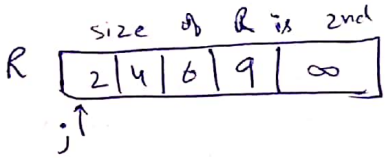
Merging



2 lists sorted from p to q & q+1 to r

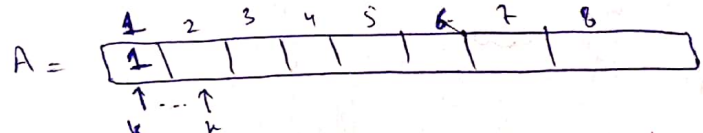
create new list L p to q+1





Time taken to copy is $O(n)$

Now A is gonna be over-written



compare i & j, fill in k with the smaller number & increament k, and the index of smaller no.

n comparisons + n copies
so $O(n+n)$

space complexity is also ordered by n.

Why add ∞ at end?

```
MergeSort(A, p, r)
{ if p < r
    q = ⌊(p+r)/2⌋        ⟶ Θ(1)
    → MergeSort(A, p, q)
    → MergeSort(A, q+1, r)   } ⟶ 2T(n/2)
    merge(A, p, q, r)        ⟶ Θ(n)
```

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

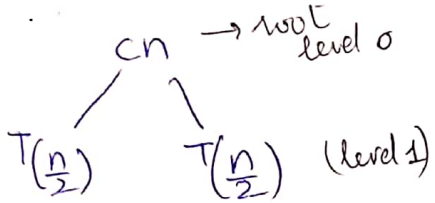
```
merge(A, p, q, r)
    n1 = q - p + 1
    n2 = r - q
    let L[1 .... n1 + 1] and R[1 .... n2 + 1]
    for i = 1 to n1
        L[i] = A[p + i - 1]
    for j = 1 to n2
        R[j] = A[q + j]
    L[n1 + 1] = ∞
    R[n2 + 1] = ∞
    for i = 1, j = 1
    for k = p to r
        if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
        else A[k] = R[j]
            j = j + 1
```

Merge Sort Analysis

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \longrightarrow (A)$$

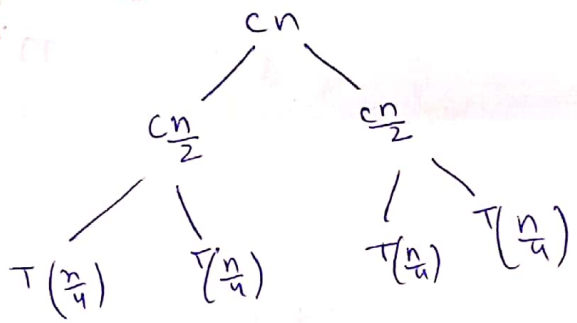
Solving by Recursion Tree Method.

If I do cn amount of work, I can reduce the problem into two smaller size problems.



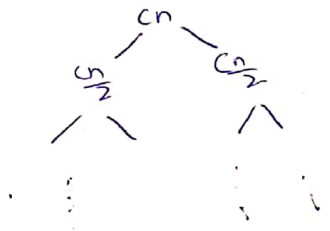
Put $n = \frac{n}{2}$ in (A)

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{cn}{2}$$



Put $n = \frac{n}{4}$ in (A)

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + c\frac{n}{4}$$



$T(1) \cdots T(1) \cdots T(1)\ T(1)$

for levels $j = 0, 1, 2, \cdots$

No. of sub-problems (1, 2, 4, 8, ...) at level $j = 2^j$

Size of " " " $= \frac{n}{2^j}$

$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \cdots$

ASSUME n is even so $n = 2^k$.

To reach i/p size of 1

$\frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \cdots \frac{n}{2^k}$

we need $\boxed{k+1}$ steps.

$n = 2^k \Rightarrow \log_2 n = \log_2 2^k \Rightarrow k = \log_2 n$

level	Work done
0	cn
1	cn
2	⋮
3	
4	
$\log_2 n$	cn

Total work

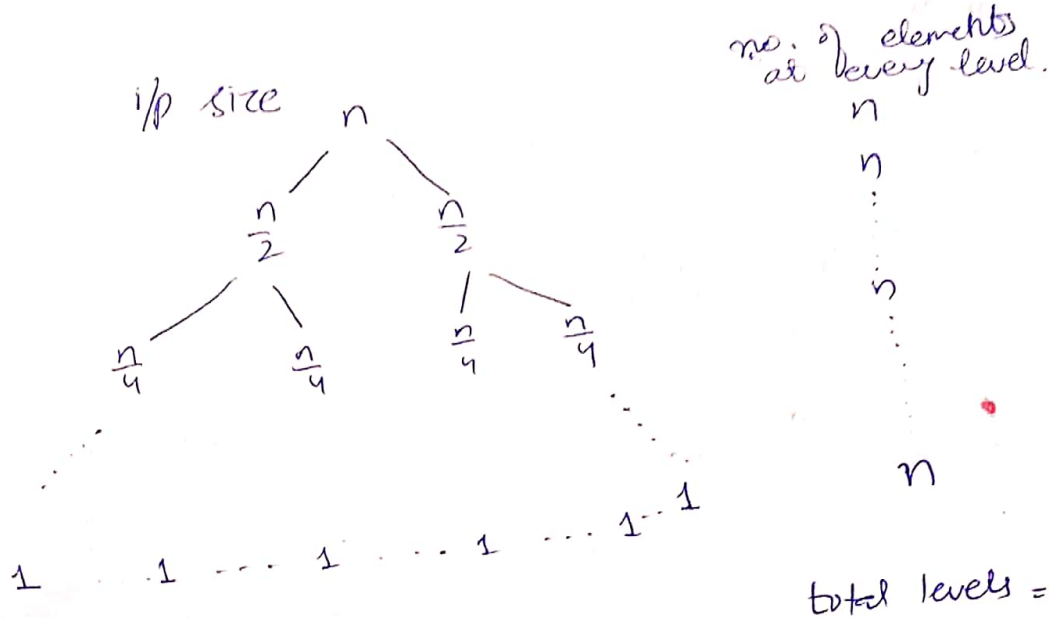
$cn + cn + \cdots$ upto $\log_2 n$ times

$\boxed{cn \log_2 n} = O(n \log n)$

Space Complexity of Merge Sort.

When merge is called it declares new (2) arrays.

2 arrays of sizes 1.



So there's $n\log_2 n$ total space that is occupied by all the arrays that are ever created. But as these are recursive calls, the older array no longer exists when the recursive call ends. So all these arrays do not exist simultaneously.

So, these tempory arrays are storing n elements at max so we are using only O(n) extra space.

Space Complexity of Insertion Sort:

| 5 | 2 | 6 | 0 |

key = index i, j

No new Array declared.

"inplace Sorting Algorithm"

We are not using any extra space.