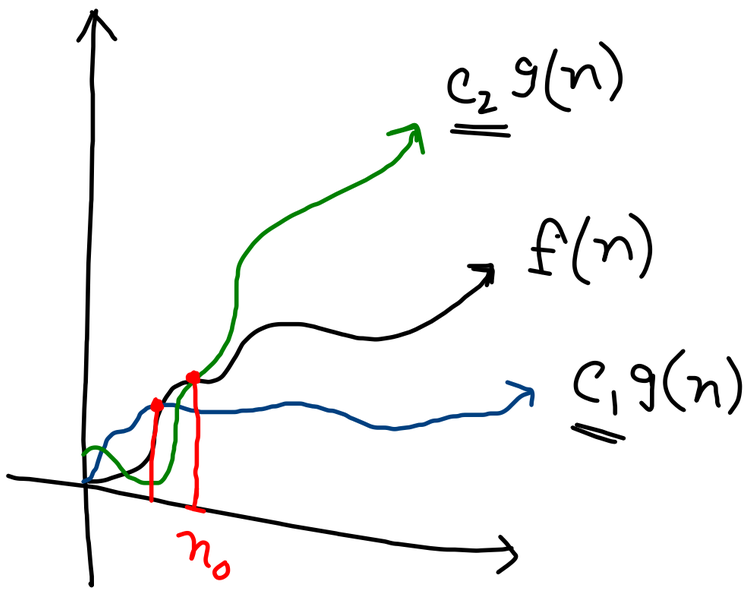
Asymptotic Notation

(H) → Tight Bound



$$\Theta(g(n)) = \{ f(n) : \exists\ c_1, c_2 > 0,\ n_0 \text{ s.t. } 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)\ \forall\ n \ge n_0 \}$$

$6n^2+8n+7 \overset{?}{\in} \Theta(n^2)$

- $6n^2+8n+7 > 6n^2$, $n \ge 0$
- $6n^2+8n+7 < 7n^2$, $n \ge 9$

$n^2 > 8n+7$

$\downarrow$

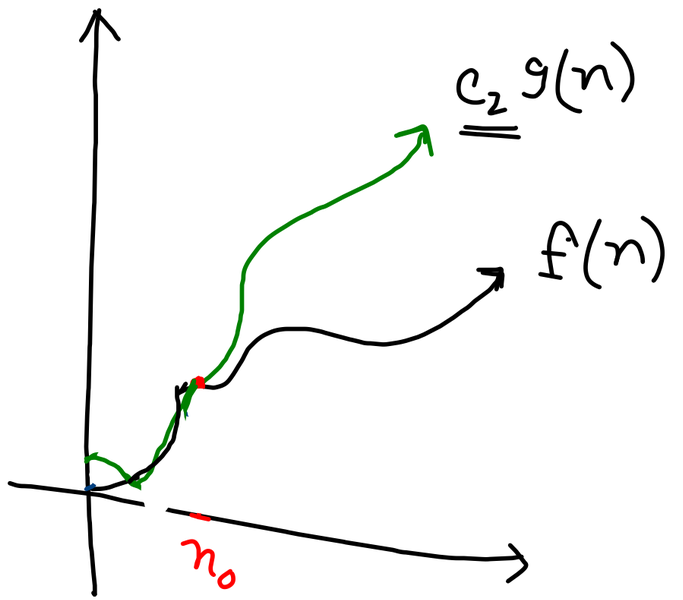
$n = 9$

$n_0 = 9,\ c_1 = 6,\ c_2 = 7$

Big 'Oh'

Asymptotic Notation

$O \rightarrow$ Upper Bound



$$O(g(n)) = \{ f(n) : \exists_{n_0} \; c_2 > 0, \; s.t. \; 0 \leq f(n) \leq c_2 \cdot g(n) \} \; \forall \; n \geq n_0$$

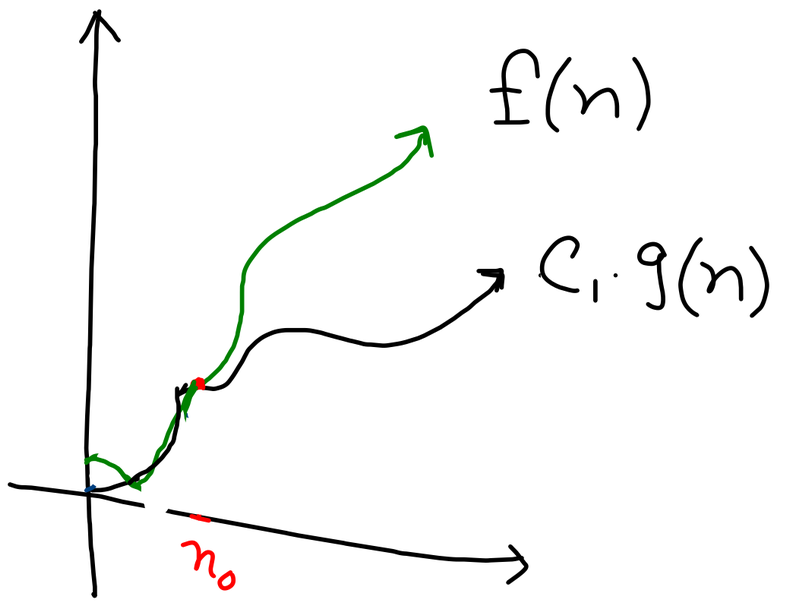
$$n^2 \stackrel{?}{=} O(n^3) \quad \checkmark$$

$$\stackrel{?}{=} \Theta(n^3) \quad \times$$

Asymptotic Notation

Big 'Omega'

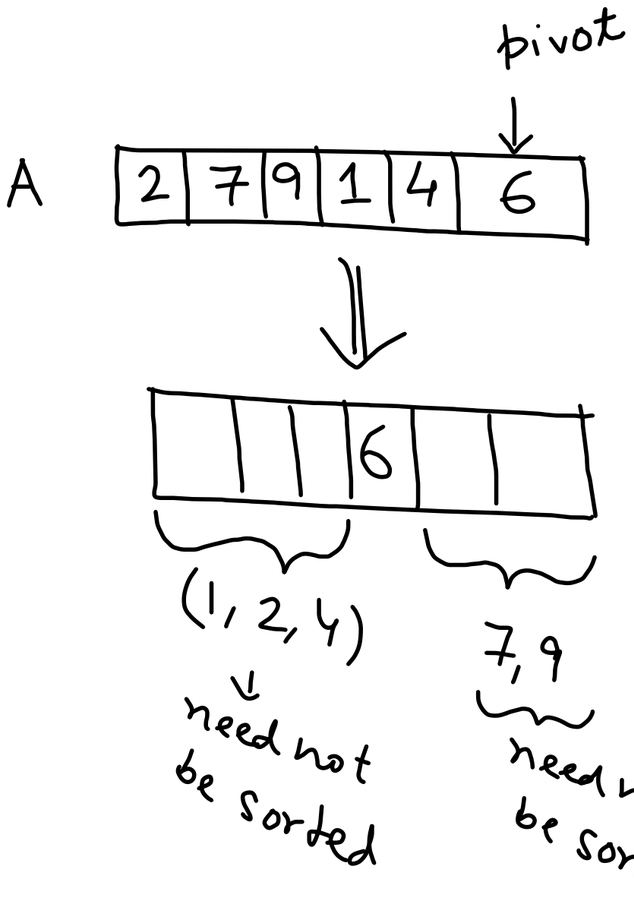
Ω → Lower Bound



$$\Omega(g(n)) = \{ f(n) : \exists_{n_0} \; c_1 > 0, \; s.t. \; 0 \le c_1 \cdot g(n) \le f(n) \; \forall \, n \ge n_0 \}$$

$n^2 = \Omega(n)$? ✓

Algorithm 1



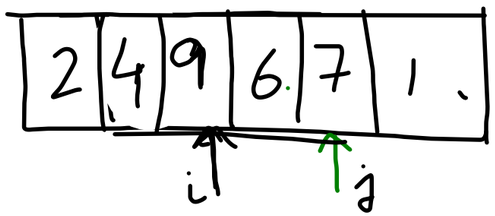
Partitioning around pivot element

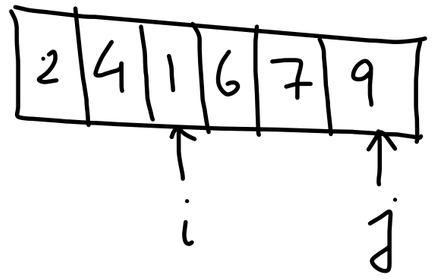
Sorting

$\Theta(n \log n)$

$\Theta(n)$

Counter → 3





Ok

$A[i] \leq pivot \leq A[j]$

$i = i-1$

$j = j+1$

Ok

$A[j] \leq pivot \leq A[i]$

$Swap(A[j], A[i])$

$i = i-1,$

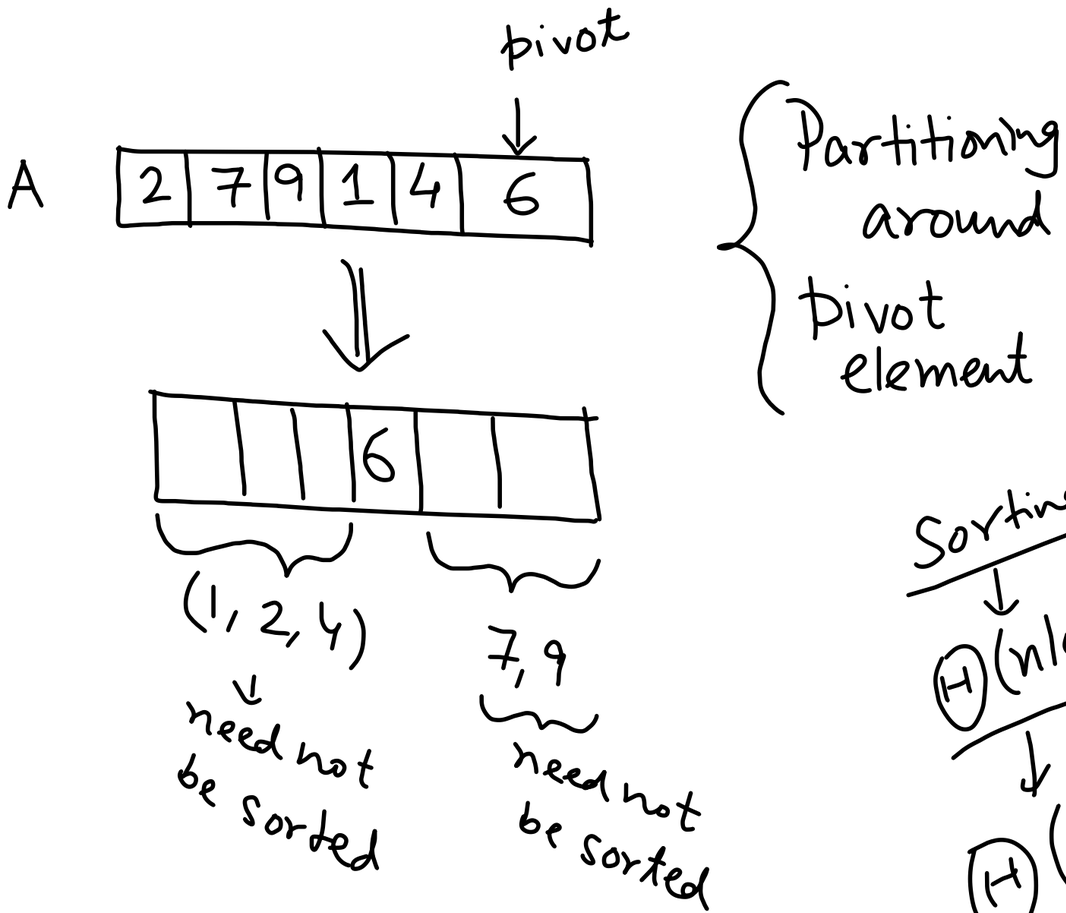
$j = j+1$

$O(n)$

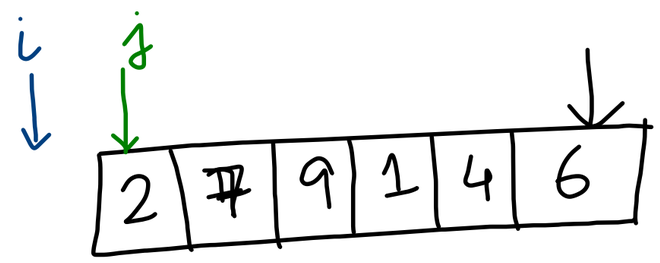
$A[i] \leq pivot,$
$A[j] \leq pivot,$
$i = i-1$

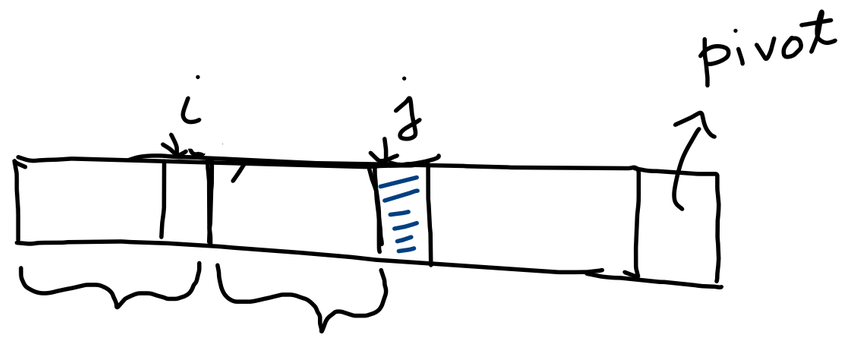
$A[i] > pivot$
$A[j] > pivot$
$j = j+1$

Algorithm 1



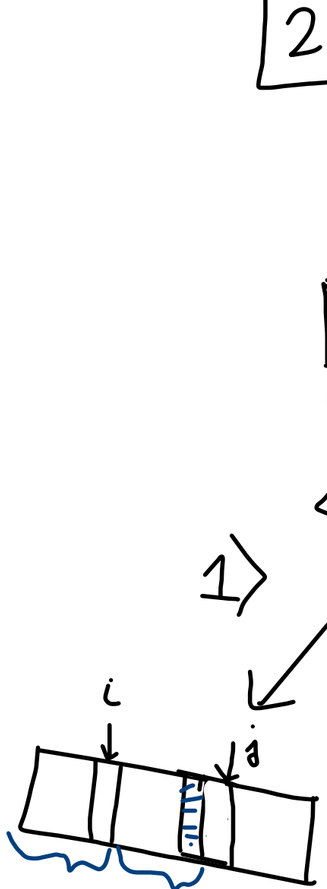
Sorting → Θ(nlogn) → Θ(n)

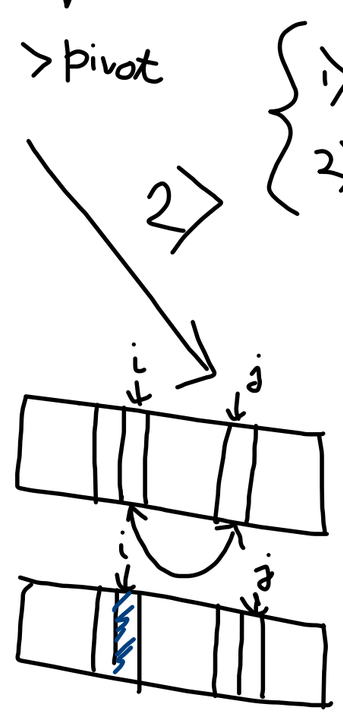




1> $A[j] \geq$ pivot

2> $A[j] <$ pivot





Partition (A, n)

$x = A[n]$

$i = 0$

for $j = 1$ to $n-1$

  if $A[j] \leq x$

    $i = i+1$

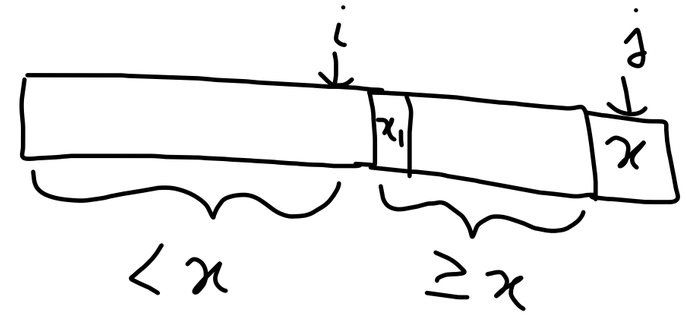
    swap $(A[i], A[j])$

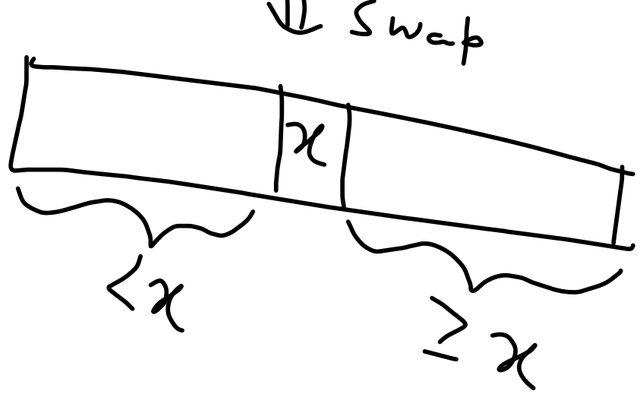
swap $(A[i+1], A[n])$

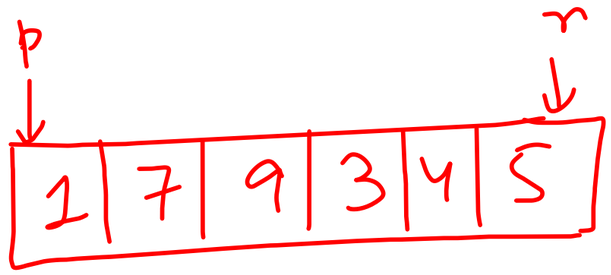
return $(i+1)$.

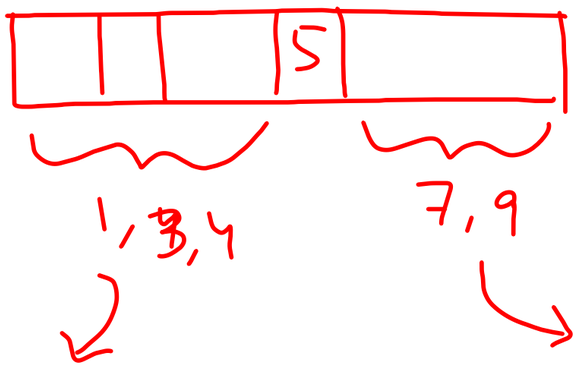
$O(n)$ time

Correctness









Divide-and-Conquer

Quick-Sort

QSort (A, p, r)

q ← Partition (A, p, r)

- QSort (A, p, q-1)
- QSort (A, q+1, r)

> QSort(A, 1, n)

Time Complexity

$$T(n) = T(i-1) + T(n-i) + O(n)$$

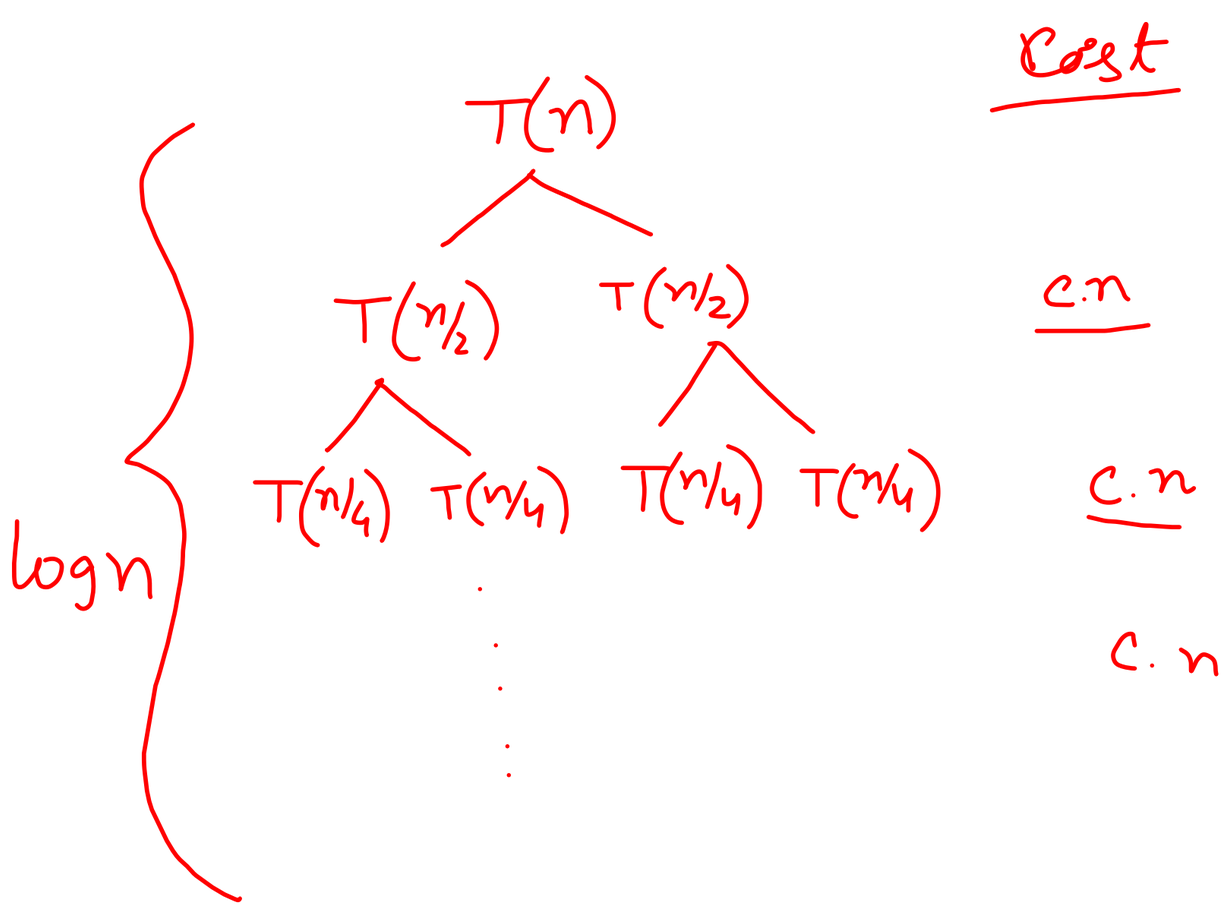
Avg Case

↳ $\Theta(n \log n)$

Array is Sorted $\Rightarrow$

$$T(n) = T(n-1) + O(n)$$
$$= T(n-1) + cn$$
$$= T(n-2) + c(n-1) + cn$$
$$\vdots$$
$$= c' n^2 = O(n^2)$$

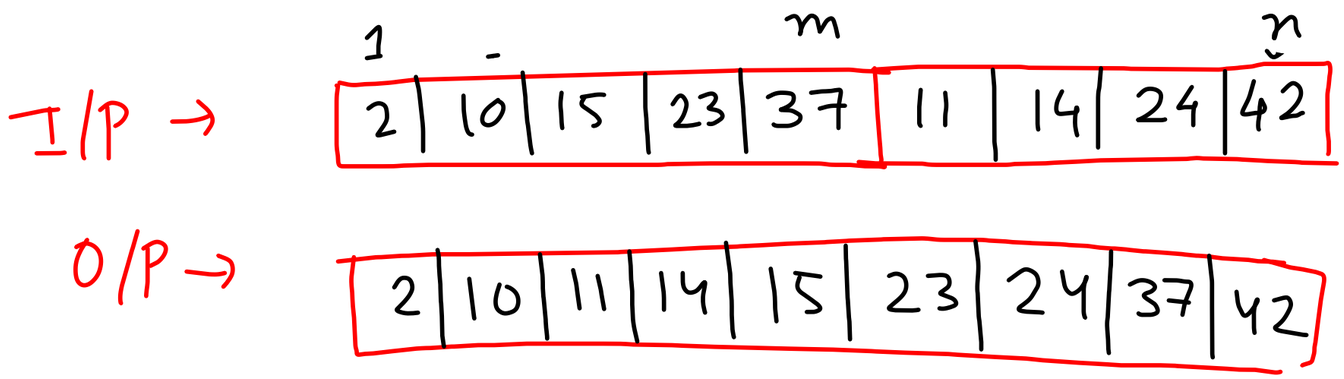
$$T(n) = T(n/2) + T(n/2) + cn$$
$$= 2T(n/2) + cn$$
$$\approx O(n \log n)$$



Overall cost

$\leq c \cdot n \cdot \log n$

Ex: You have two sorted subarrays $A[1..m]$, $A[m+1, \ldots, n]$. Sort the whole array.

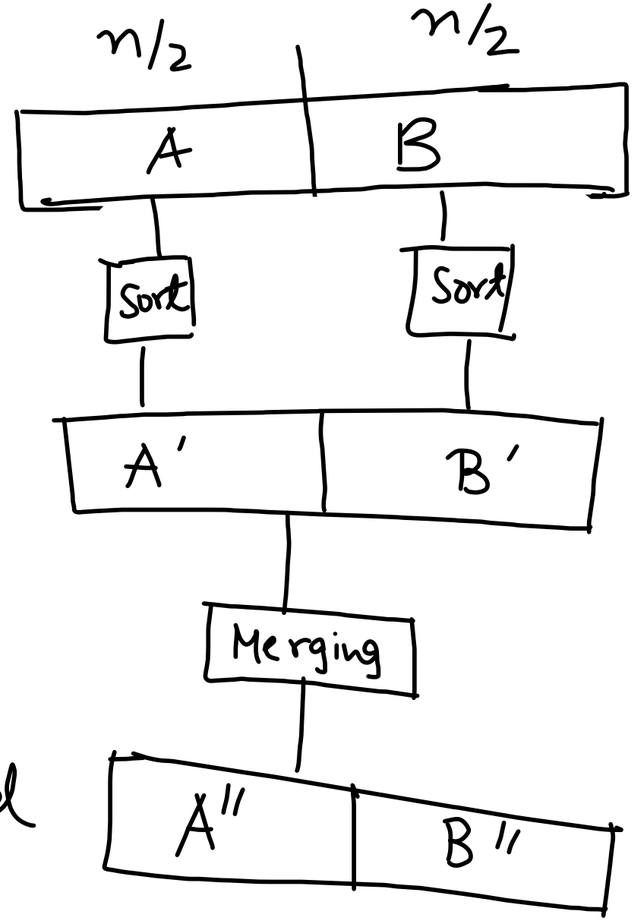


Merging two sorted Arrays

O(n) - time

Inplace Algo → additional Storage should be O(1)

- O(n) time, O(n) space
- O(n²) time, O(1) space




```
Merge Sort (A, 1, n)
    Merge Sort (A, 1, n/2)
    Merge Sort (A, n/2+1, n)
    Merge (A)
```

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

$$\Rightarrow T(n) = O(n \log n)$$