Recap: Sorting → Mergesort, Quicksort } $O(n \log n)$ → best general sorting TC.

```
Mergesort (arr):
    mid = len(arr)/2
    left_half = arr[:mid]
    right_half = arr[mid:]
    mergesort (lh)
    mergesort (rh)
    merge (arr, lh, rh)
```

assume here that mergesort here knows how to sort these half arrays.

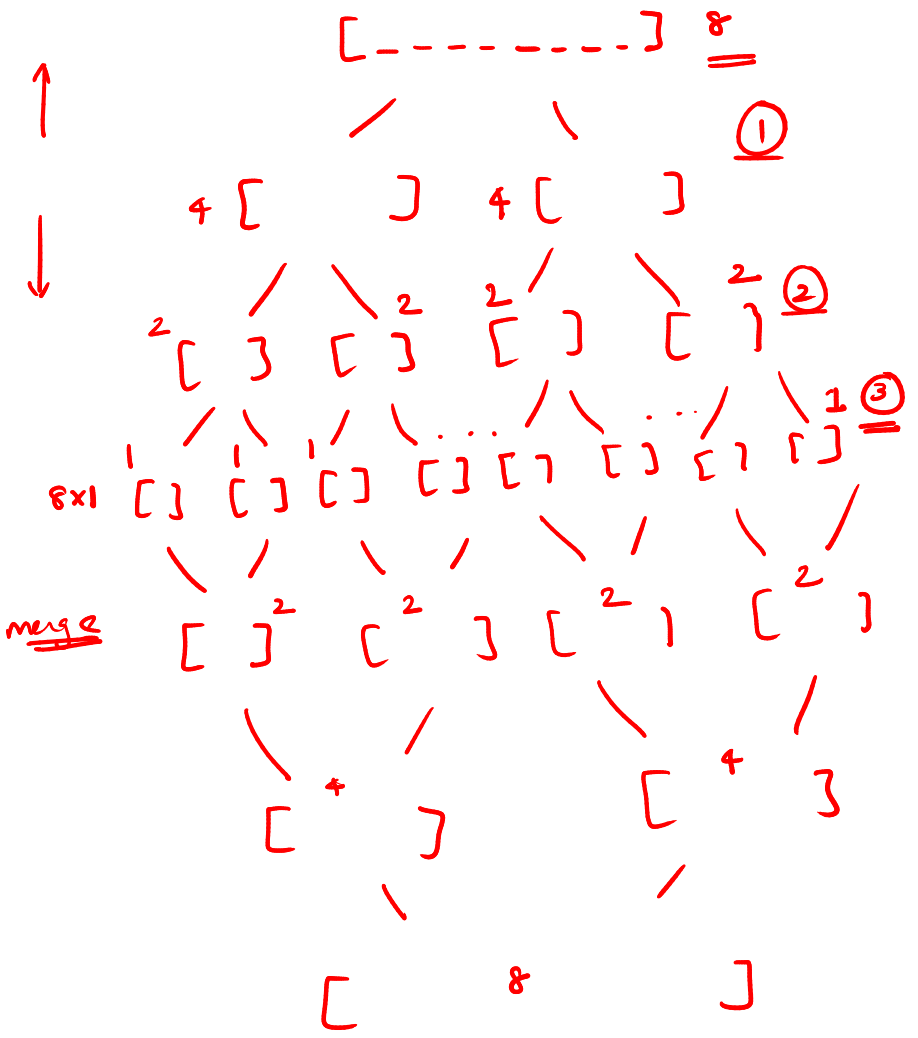
⇒ you want to take 2 sorted arrays and merge them in original array arr.

```
merge (arr, lh, rh):
    i = j = k = 0
    n = len(lh)
    m = len(rh)

    while i < n && j < m:
        if lh[i] < rh[j]:
            arr[k] = lh[i]
            i++  k++
        else
            arr[k] = rh[j]
            j++  k++

    while i < n:
        arr[k] = lh[i]
        i++  k++

    while j < m:
        arr[k] = rh[j]
        j++  k++
```



③ = $\log_2 8$

levels = $\log_2 n$

At each level, you perform $O(n)$ work

TC ~ $O(n \log_2 n)$.

$10^5 < n_{max} < 10^6$

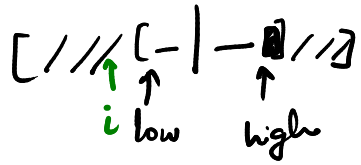
QUICKSORT. ⇒ Template

⊛ I want to get the process of partition. [3 1 6 5 8 2 7 (4)] ← pivot.

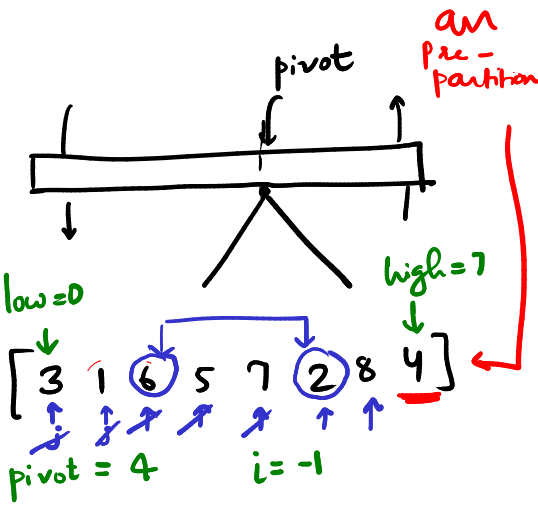
Agenda : Pick the last element & put it at its right position.

Expected o/p: [3 1 2 (4) 5 7 6 8] all elements smaller than this element should be on LHS and greater should be on RHS.

Return: The correct position/index of pivot after partition.




```
int partition (arr, low, high):
    pivot = arr[high]          ← pick last element.
    i = low - 1                ← (i+1) elements are smaller than pivot. Eventually,
    for (j = low, j < high, j++)   ← This loop runs through out array except last element.
        if arr[j] < pivot :    ← partition cond?
            i++                ← # elements on LHS of pivot
            swap (arr[i], arr[j])
    swap (arr[i+1], arr[high])   ← brings pivot at its right position
    return i+1                 ← Index of pivot element
```



j = 0 to 6

j=0	i=0	swap (arr[0], arr[0])
j=1	i=1	swap (arr[1], arr[1])
j=2	—	j=3 —
j=4	—	
j=5	i=2	swap (arr[5], arr[2])
j=6	—	swap (arr[1], arr[3])

last value of i

i+1 → return 3

O(n) TC.

[3 1 2 5 7 6 8 4]

Array post partition ⇒ [3 1 2 (4) 7 6 8 5]

Eg highlighting imp of i = low - 1 → [9 5 4 1] → return 0 → [1 5 4 9]

Eg 2

[9 5 4 1] → pivot = 1

j = 0 to 2

i = -1, j = 0 —, j = 1 —, j = 2 —

swap (arr[0], arr[3])

Doubt: [1 5 4 9] pivot = 9 No need actually to perform partition.

3

Q. How will 5 and 4 swap? → IDC.

Intuition: whenever a smaller element is found, you bring it to $i+1^{th}$ position. Start i = -1. Increment i.

Condense idea: i acts like as a placeholder for smaller elements compared to pivot.

Why i=low-1 ? Imagine ⇒ i=low & not i=low-1.

Perform partition on this arr. Eg. → [1 2 4 3 5]

0 1 4

low=0 high=4

i = 0 i = -1

for j = low, high-1

if arr[j] < pivot:

i++

swap(i^{th}, j^{th})

j=0 i=1

?? swap(arr[0], arr[1])

j=0 i=0

swap( ) ← no impact!

Wrap with quicksort:

```
void qs(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        qs(arr, low, pi-1)
        qs(arr, pi+1, high)
```

f^n call: qs(arr, 0, n-1)

n: no of elements in arr

pi → pivot_index

[3 1 5 6 2 4]

3 1 2 4 5 6

1 2 3 5 6

[1 2 3 4 5 6]

Just like mergesort, splitting the array in left & right parts.

Proof is out of scope. Trust demo. → TC ~ $O(n \log_2 n)$.

qs is preferred over ms ∵ It doesn't create extra arrays.
∴ It uses less space.

Eureka moments. Variations.

Q.1 2N elements in an array N → +ve N → -ve.

Rearrange the array: [_ _ _ _ _ ...]
+ve -ve +ve -ve +ve

Eg. [3 4 -1 2 -5 6 -7 -8] ← i/p array.

Idea : pivot = 0 | partition. [-1 -5 -7 -8 3 4 2 7]
x=0, y = i+1

swap(x^{th}, y^{th}) x+=2 y+=2 y < 2n x < n

Partition cond^n : All elements that satisfy this if cond^n will end up at the start of the array, then the elements which falsify the cond^n will start. The partition switch happens at i+1.

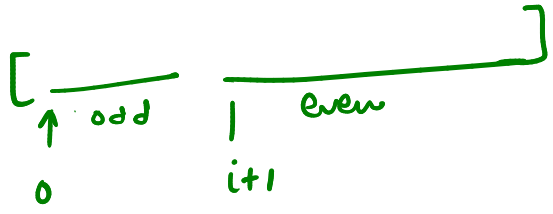
Q.2 2N elements N → odd N → even

an = [_ _ _ _ ...]
even odd even odd

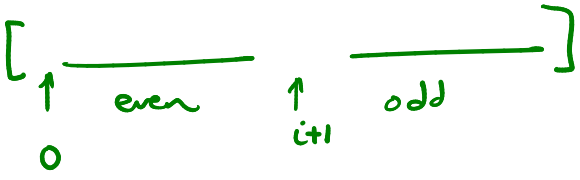
[_ _ _ ...]
odd even odd
2nd

Idea : partition

if an[j] % 2 != 0 :



2nd if an[j] % 2 == 0 :



Usecase : If I have an array of elements, some elements belong to a group & others don't. You can club them at the start of an using partition(). There are i+1 such elements.

Q.3 Push zeros at the end. [1 0 2 0 0 3 0 0 4 5]

↓

[1 2 3 4 5 0 0 0 0 0]

non zero | zeros

Idea : if arr[j] != 0 :

⋮

Q.4 Dutch National Flag Problem [0 1 2 0 1 1 2 1 0 1 1 2]

↓

[0 0 0 1 1 1 1 1 2 2 2]

if arr[j] == 0 :

⋮

[zeros | mix of 1s and 2s]

i+1

O(2n)

~ O(n).

from i+1 to end :

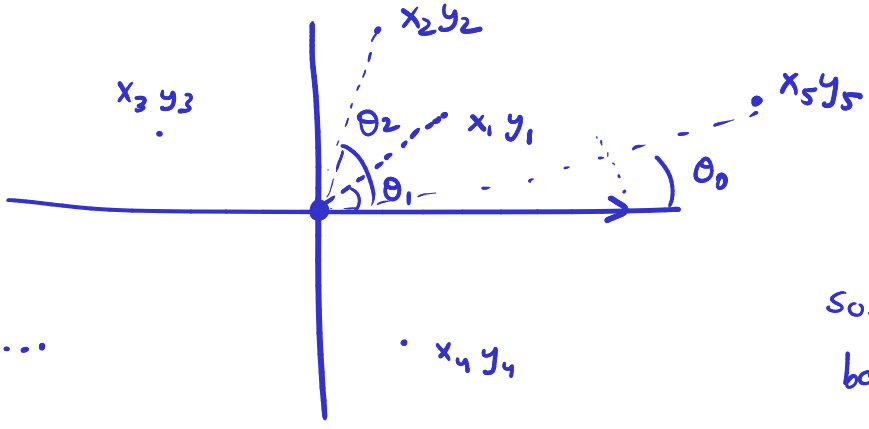
if arr[j] == 1 : [ones | non-ones]

MANUAL SORTING

Up next : sort( ) . Custom sorting .

arr = [(x_1, y_1) (x_2, y_2) (x_3, y_3) ⋮ (x_n, y_n)]

Eg.



$\theta_1 < \theta_2$

(x_1, y_1) , (x_2, y_2) . . .

Sort this array based on polar angle.