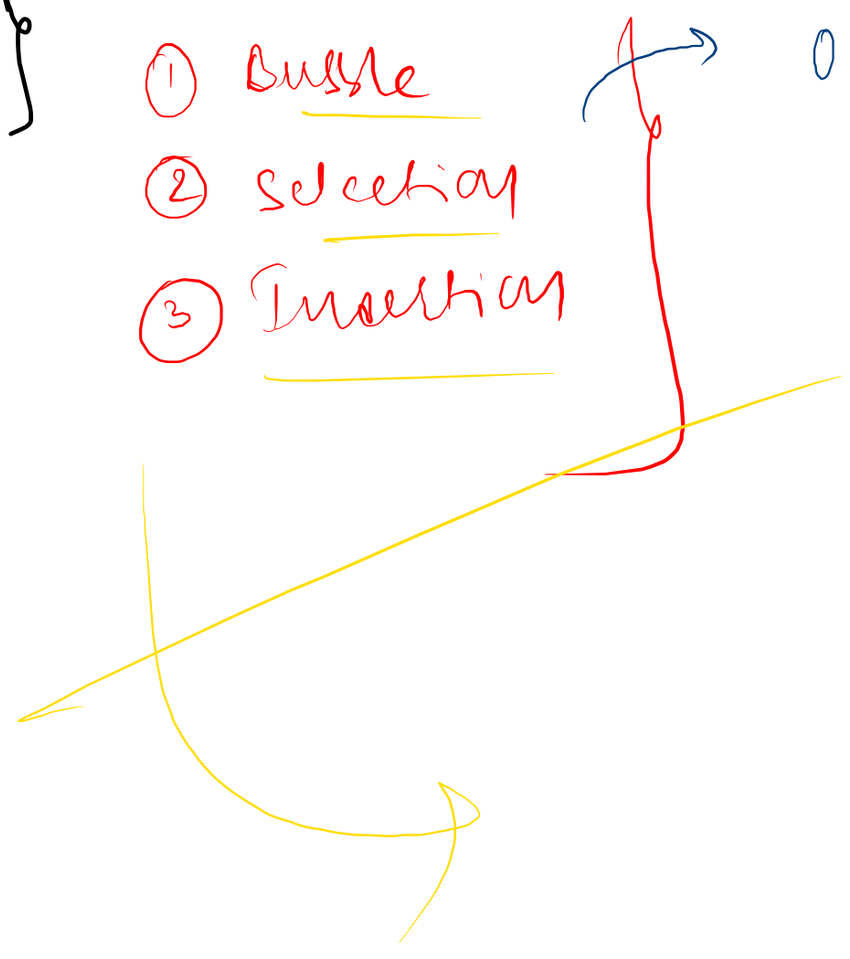
Sorting

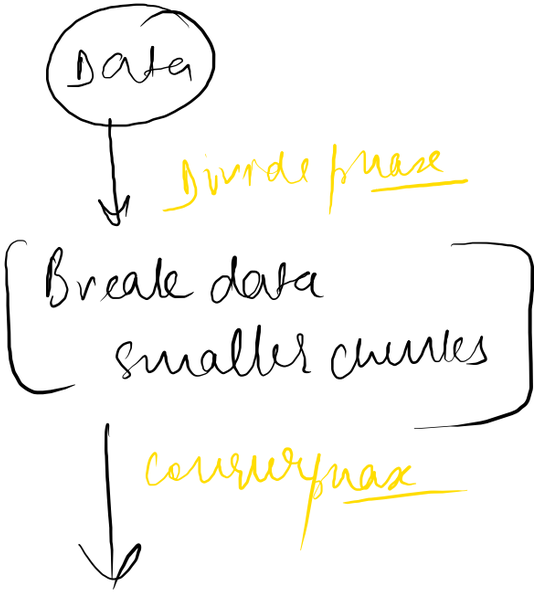
1. Bubble
2. selection
3. Insertion



$O(n^2)$ → expensive

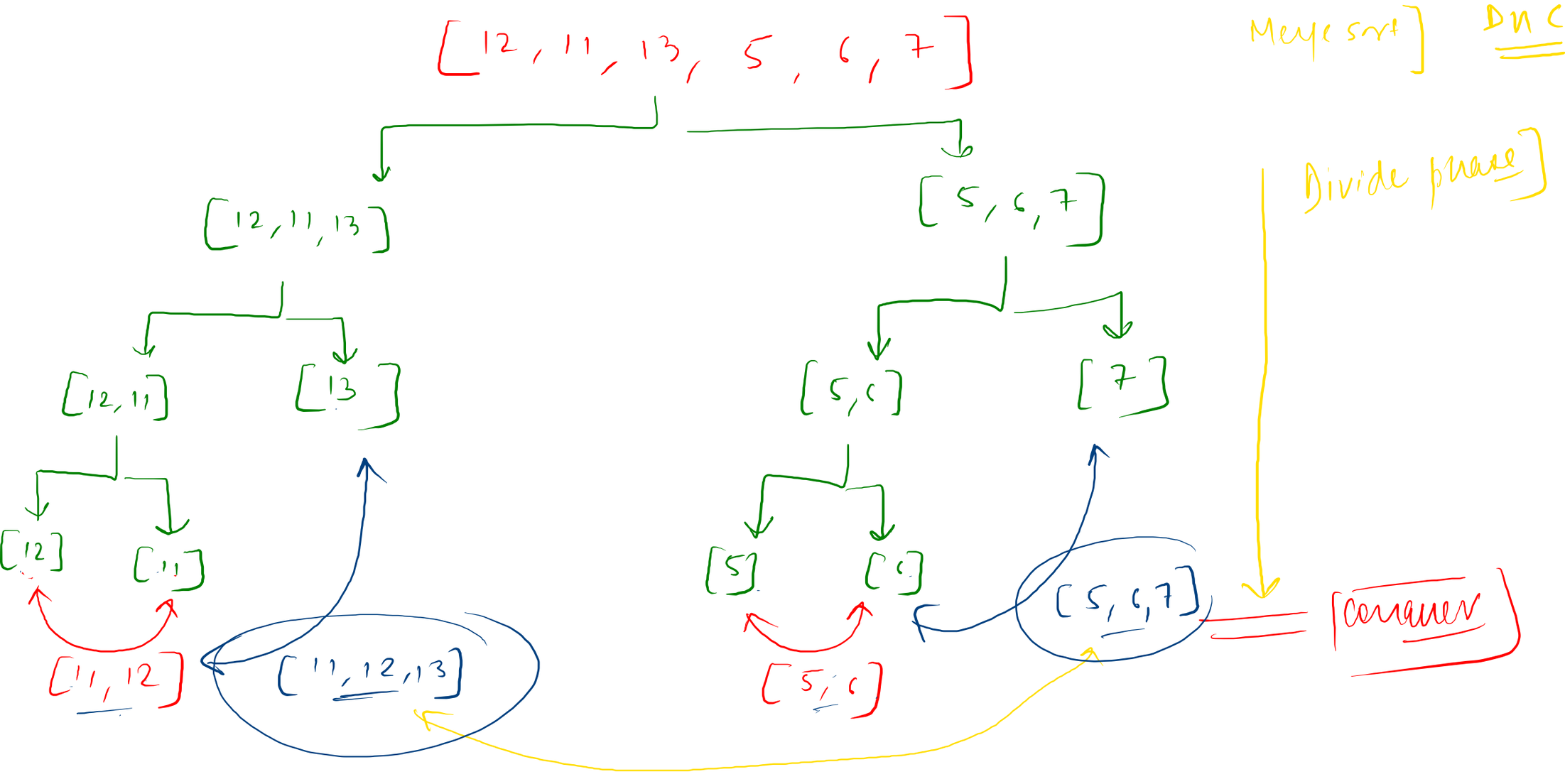
Merge Sort → DnC | Divide and Conquer Algorithms

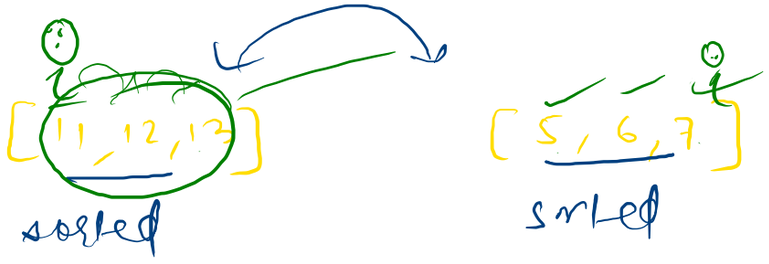
list.sort() ← (internally does merge sort)



(DS) ⟶ Data ⟶ Sorted ⟶ search very fast [BS]

// Median / Mode

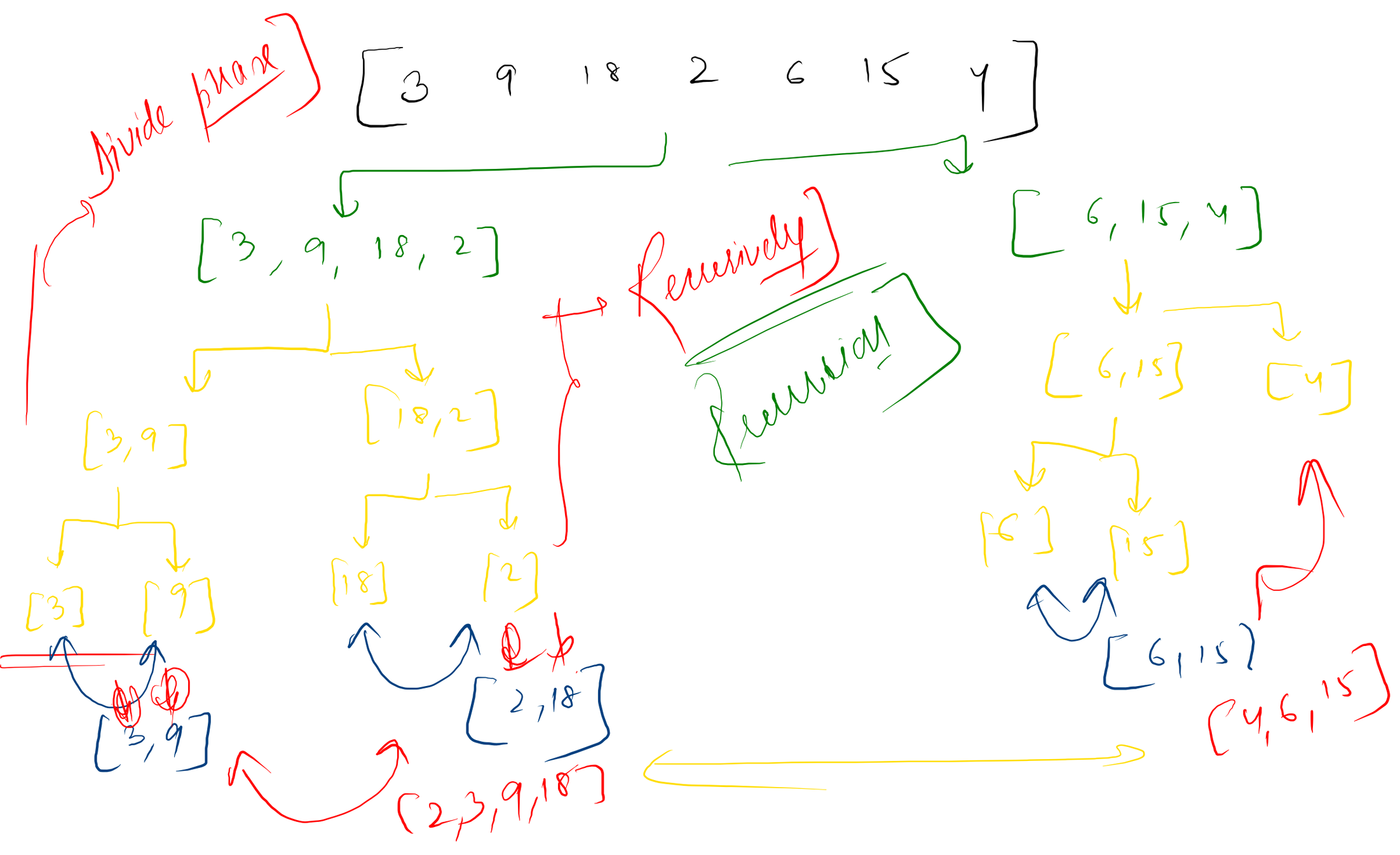

[12, 11, 13, 5, 6, 7]
Merge sort
DAC
Divide phase
[12, 11, 13]
[5, 6, 7]
[12, 11]
[13]
[5, 6]
[7]
[12]
[11]
[5]
[6]
[11, 12]
[11, 12, 13]
[5, 6]
[5, 6, 7]
Conquer

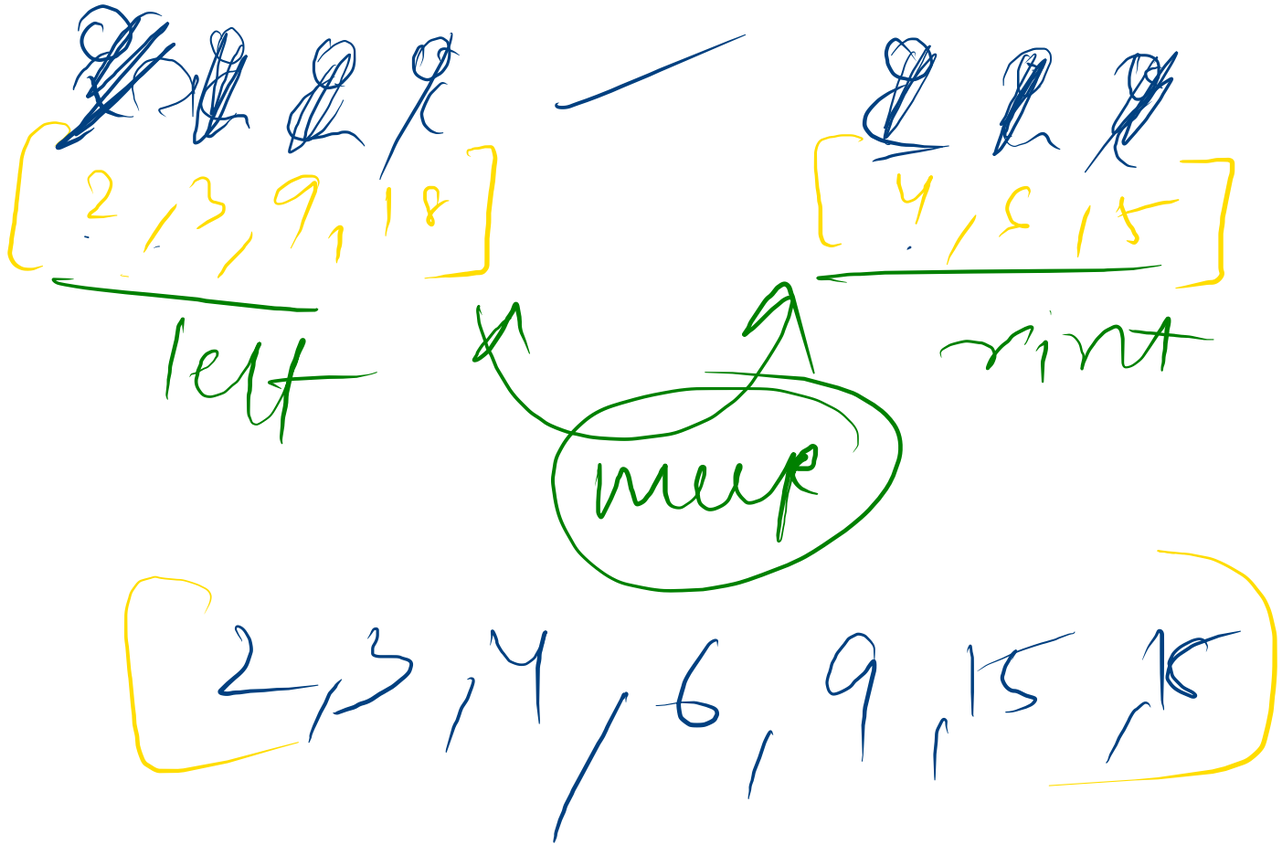


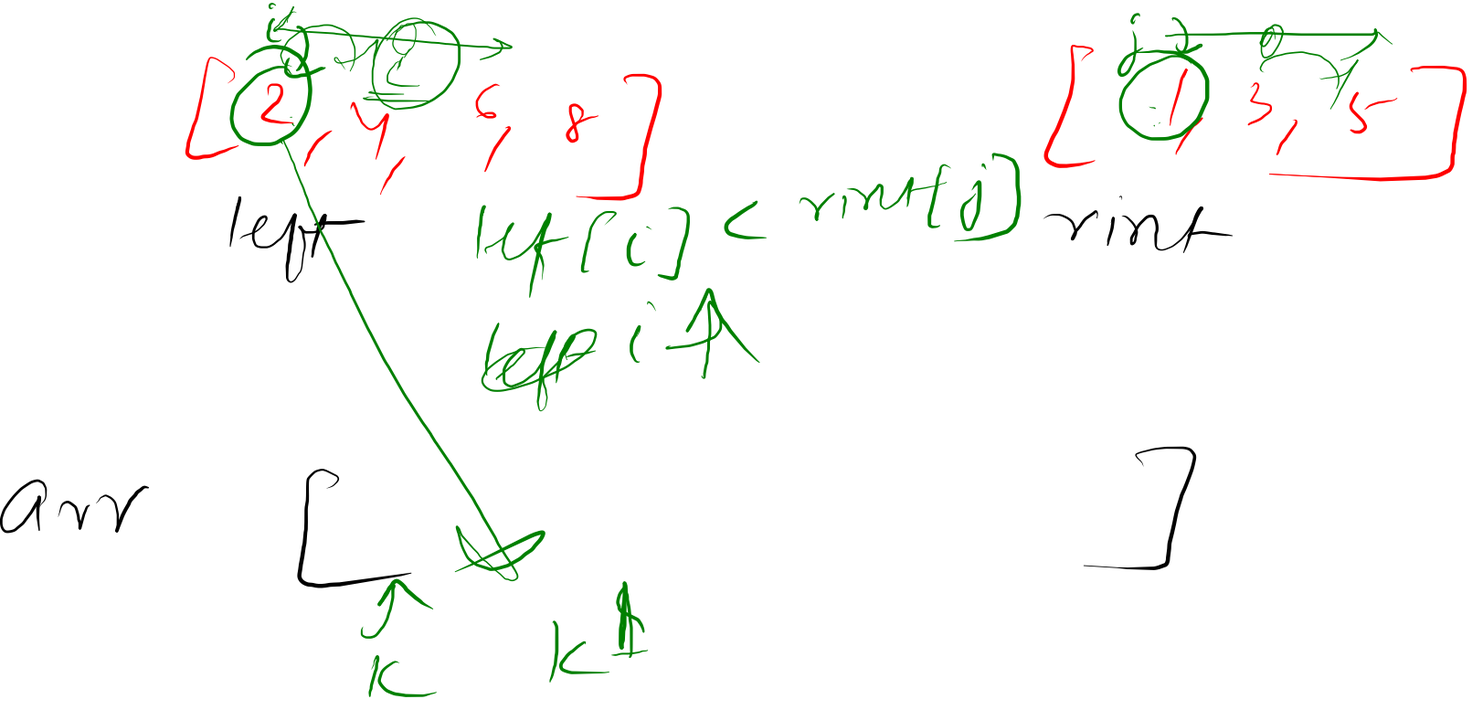
combine so that after combination we get sorted array

[5, 6, 7, 11, 12, 13] sorted array

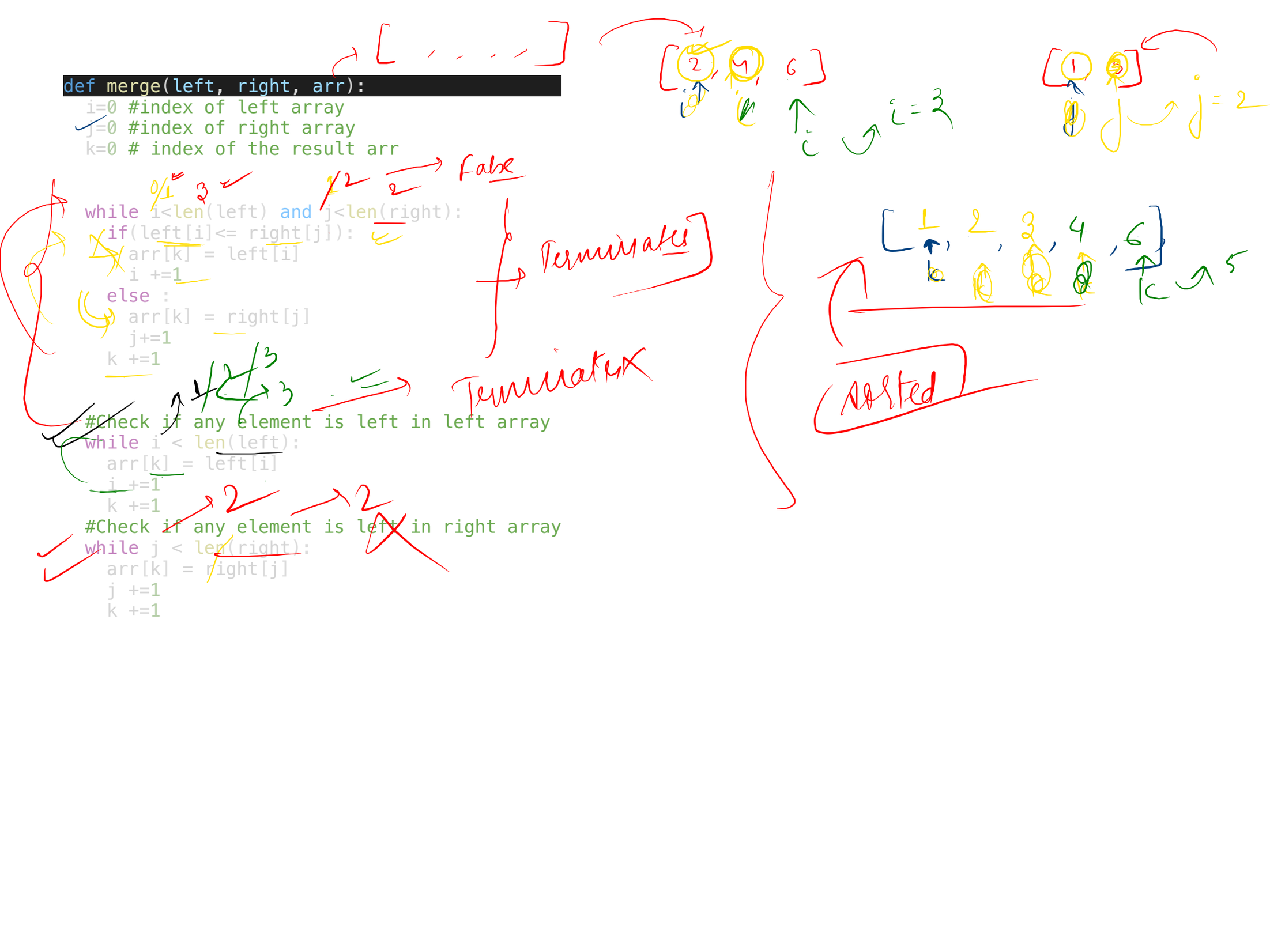
Merge SM


Divide phase
[3 9 18 2 6 15 4]
[3, 9, 18, 2]
[6, 15, 4]
Recursively
Recursively
[3,9]
[18,2]
[6,15]
[4]
[3]
[9]
[18]
[2]
[6]
[15]
[3,9]
[2,18]
[6,15]
[2,3,9,18]
[4,6,15]

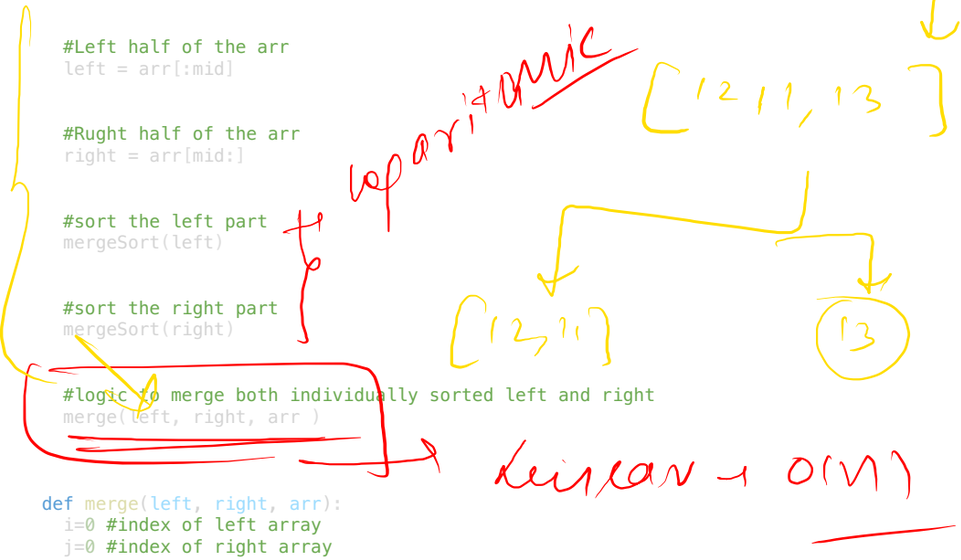

[2, 3, 9, 18]
[4, 6, 15]
left
right
merge
[2, 3, 4, 6, 9, 15, 18]


left
left[i] < right[j]
right
left i
arr
k
k

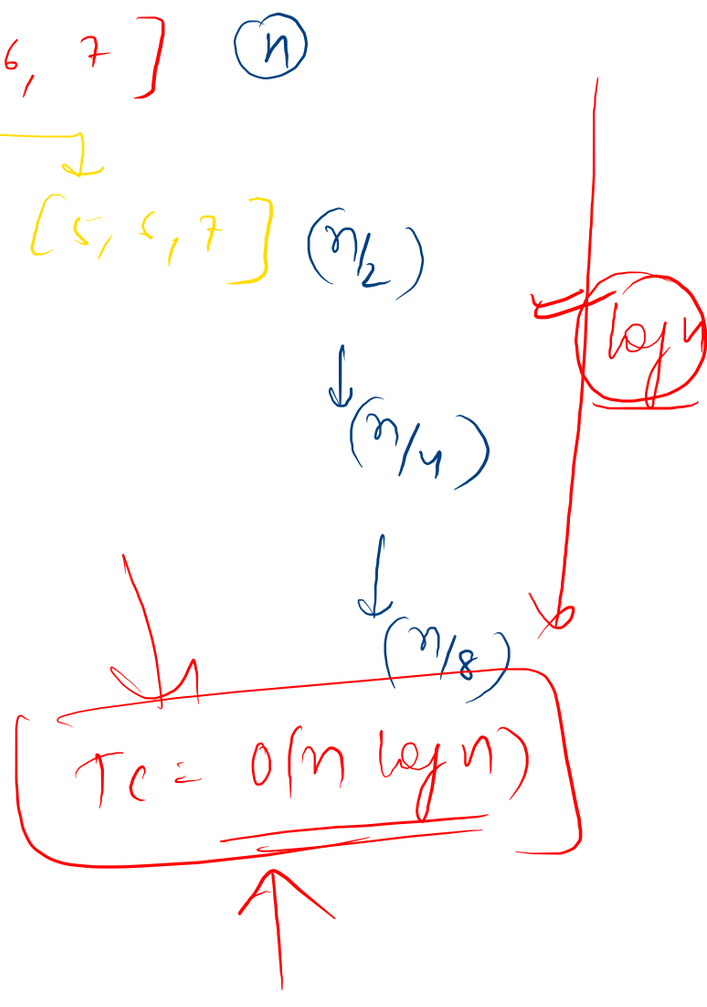
```
def merge(left, right, arr):
    i=0 #index of left array
    j=0 #index of right array
    k=0 # index of the result arr

    while i<len(left) and j<len(right):
        if(left[i]<= right[j]):
            arr[k] = left[i]
            i +=1
        else :
            arr[k] = right[j]
            j+=1
        k +=1

    #Check if any element is left in left array
    while i < len(left):
        arr[k] = left[i]
        i +=1
        k +=1
    #Check if any element is left in right array
    while j < len(right):
        arr[k] = right[j]
        j +=1
        k +=1
```



```python
def mergeSort(arr):  # this fn sorts the arr passed as argument

  if (len(arr)<=1):
    return
  # Divide into two parts
  mid = len(arr)//2


  #Left half of the arr
  left = arr[:mid]


  #Rught half of the arr
  right = arr[mid:]


  #sort the left part
  mergeSort(left)


  #sort the right part
  mergeSort(right)


  #logic to merge both individually sorted left and right
  merge(left, right, arr )
```



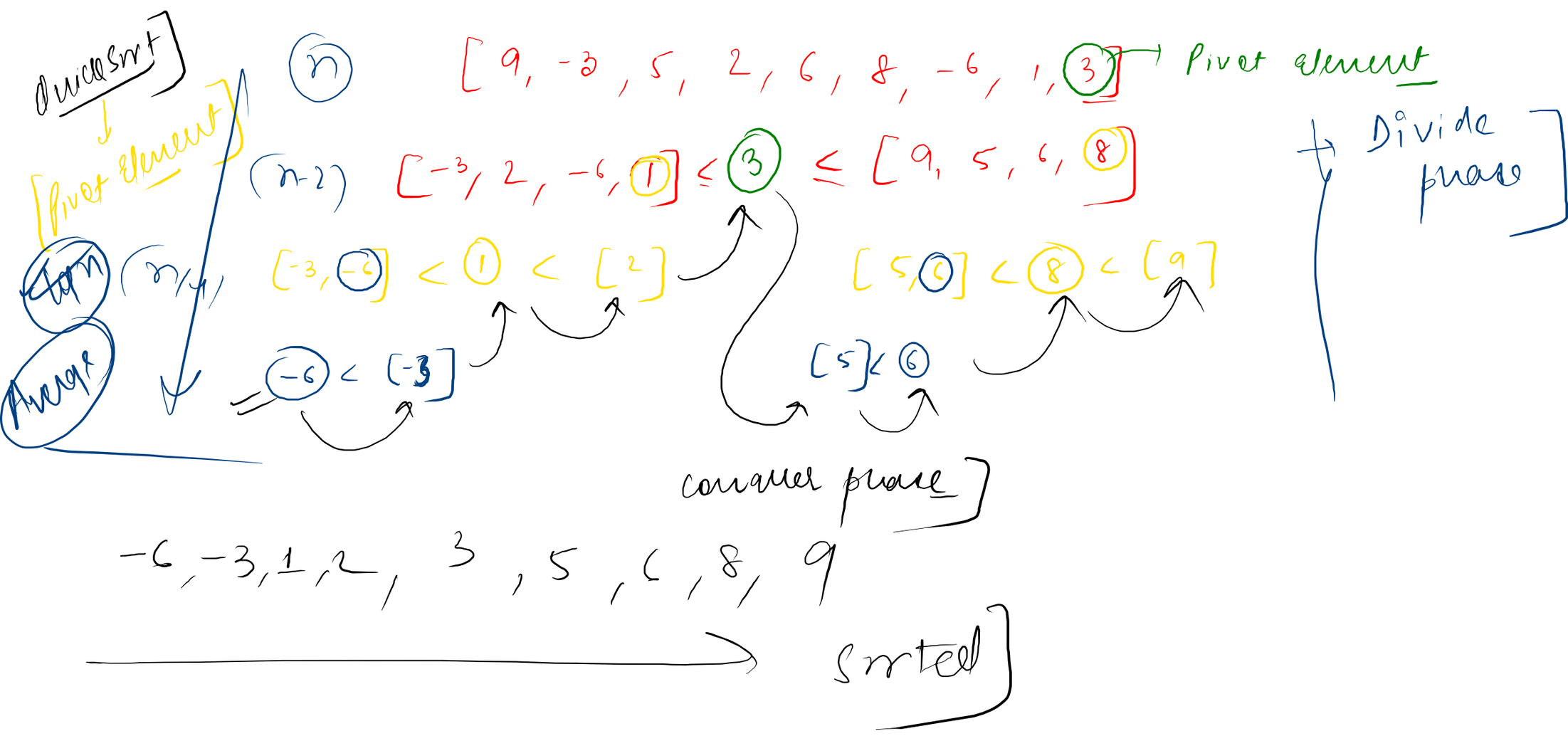

```python
def merge(left, right, arr):
  i=0 #index of left array
  j=0 #index of right array
  k=0 # index of the result arr


  while i<len(left) and j<len(right):
    if(left[i]<= right[j]):
      arr[k] = left[i]
      i +=1
    else :
      arr[k] = right[j]
      j+=1
    k +=1
```

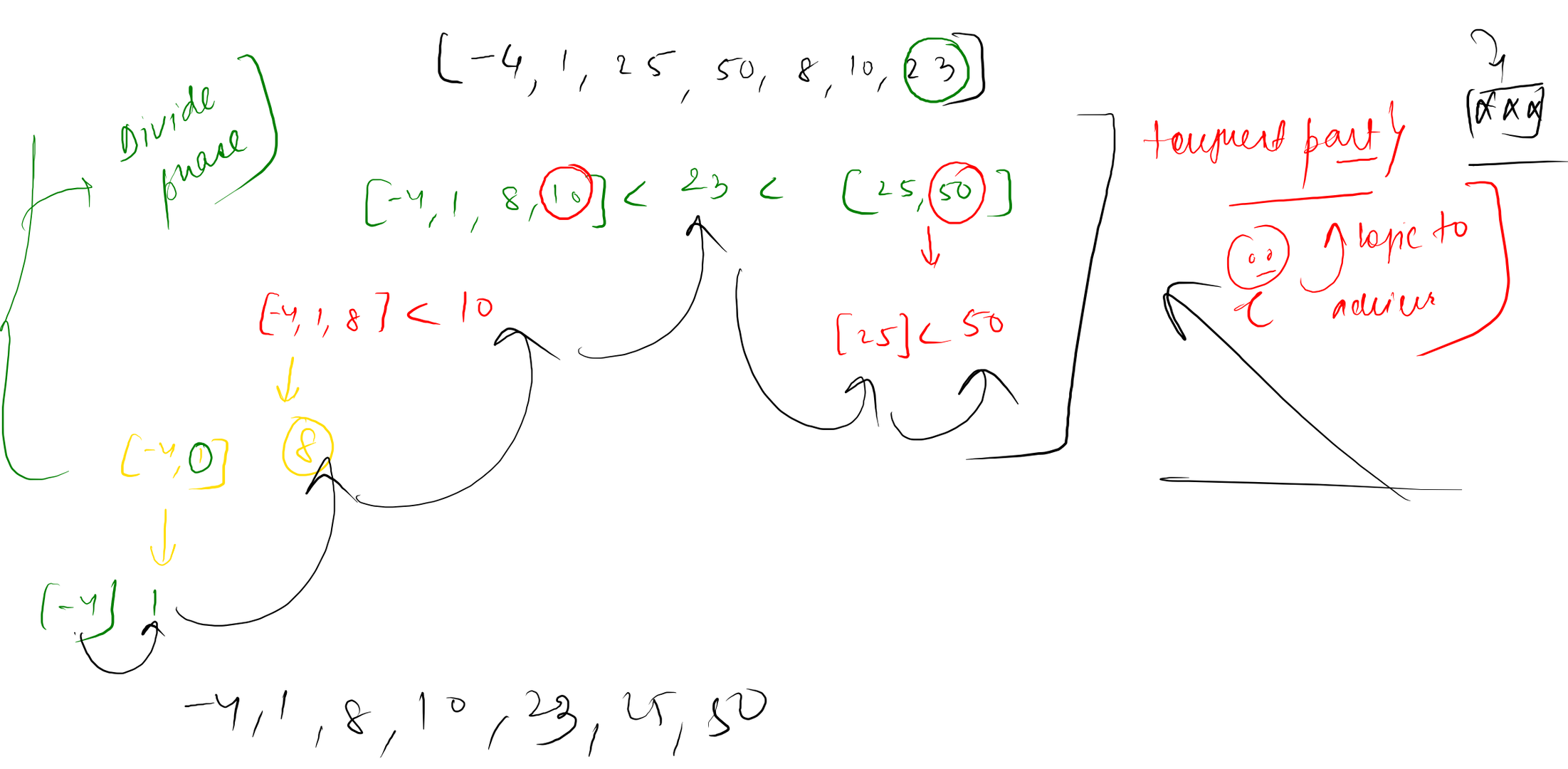


for (log n) → ~~X~~ n steps to move

O(n log n)

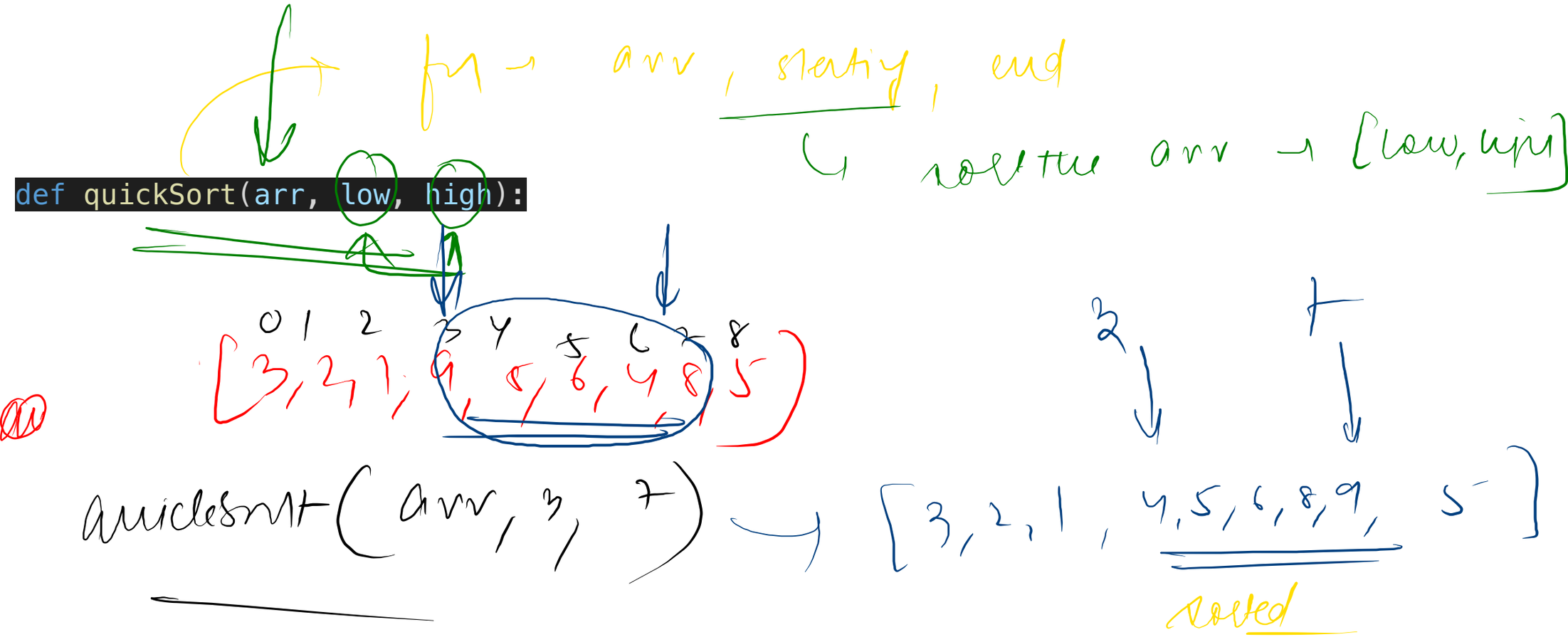
Quick Sort } Divide and Conquer Algorithm


Quicksort
Pivot element
Partition
Merge
n
[9, -3, 5, 2, 6, 8, -6, 1, 3]
Pivot element
(n-2)
[-3, 2, -6, 1] ≤ 3 ≤ [9, 5, 6, 8]
Divide phase
[-3, -6] < 1 < [2]
[5, 6] < 8 < [9]
-6 < [-3]
[5] < 6
conquer phase
-6, -3, 1, 2, 3, 5, 6, 8, 9
sorted


[-4, 1, 25, 50, 8, 10, 23]
Divide phase
[-4, 1, 8, 10] < 23 < [25, 50]
[-4, 1, 8] < 10
[25] < 50
[-4, 0]
8
[-4]
1
tougher part
topic to advise
-4, 1, 8, 10, 23, 25, 50

$O(\log n) + O(\log n) \longrightarrow O(n)$

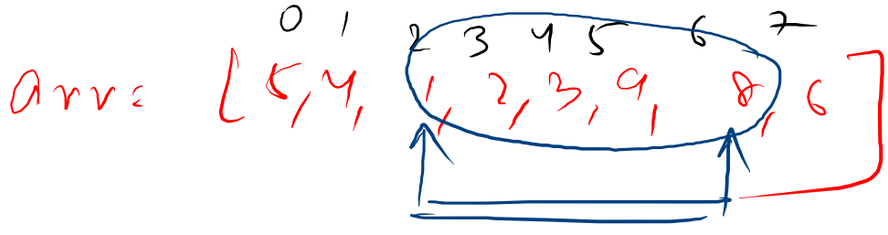
Map


fn → arr, starting, end
↳ sort the arr → [low, high]
def quickSort(arr, low, high):
0 1 2 3 4 5 6 7 8
[3, 2, 1, 9, 5, 6, 4, 8, 5]
quicksort(arr, 3, 7) → [3, 2, 1, 4, 5, 6, 8, 9, 5]
sorted

fn ← arr, start I, end I

↳ sort the array [low, high]

```
def quickSort(arr, low, high):
```

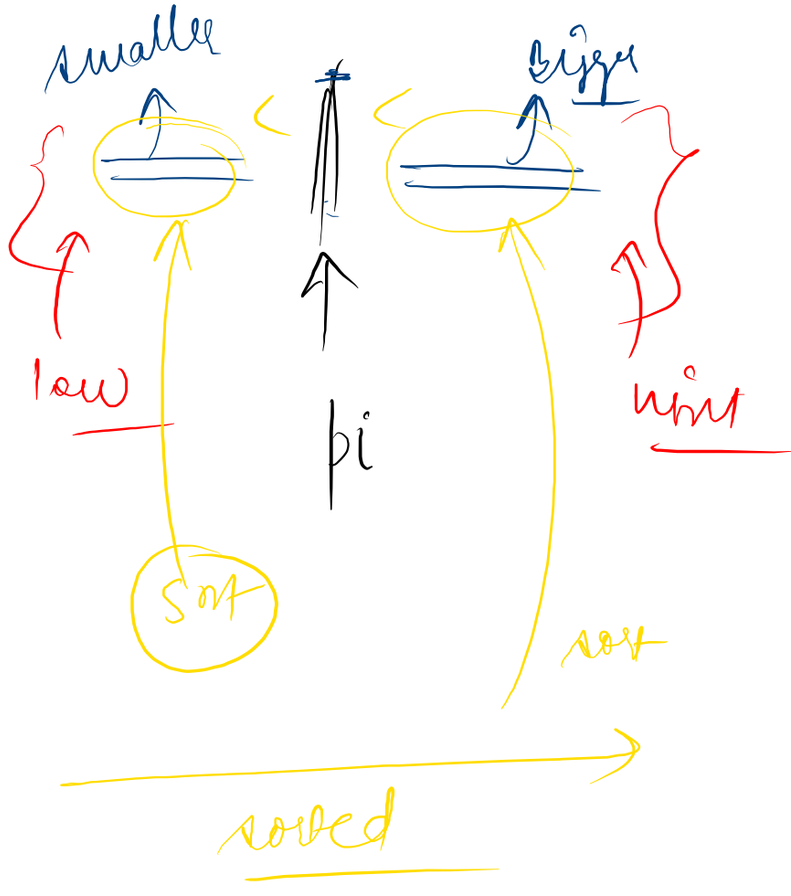


quickSort(arr, 2, 6) ↳ [5, 4, 1, 2, 3, 8, 9, 6]

sorted

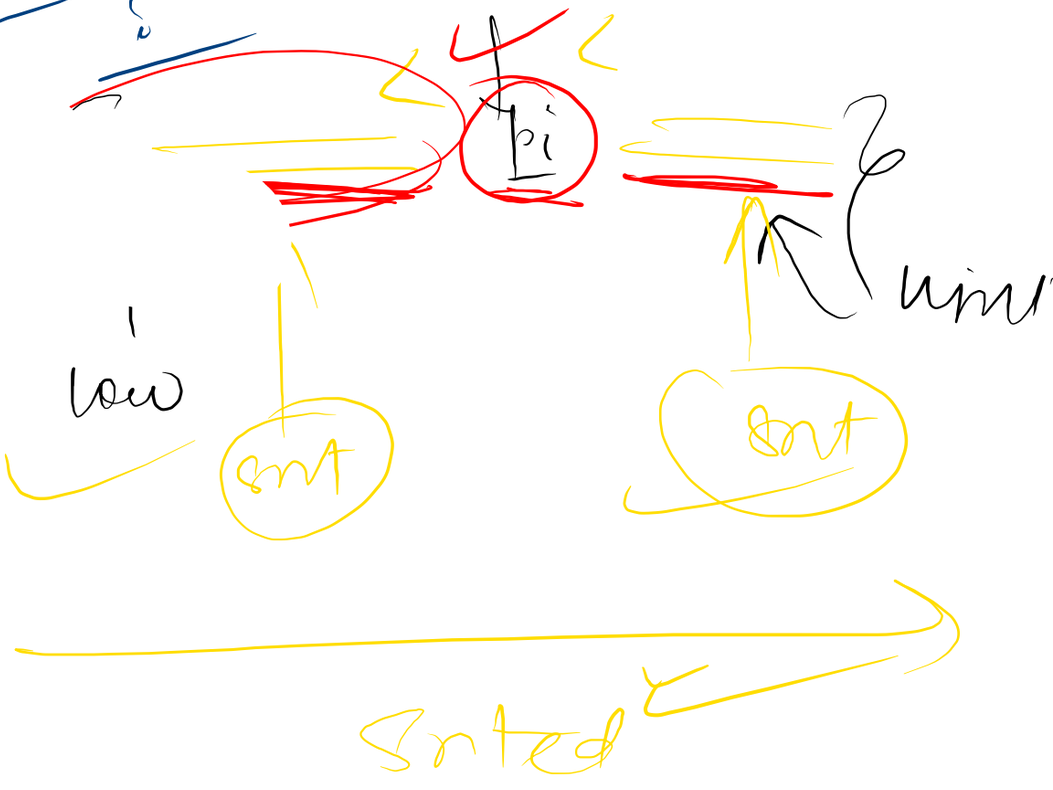
Any ideas!

```python
def quickSort(arr, low, high):
  # Base condition
  if low >=high :
    return

  #find the pivot point and re-arrange the array around it
  pi = partition(arr, low, high)


  #left of pi will be all small
  # right of pi will be all big
```

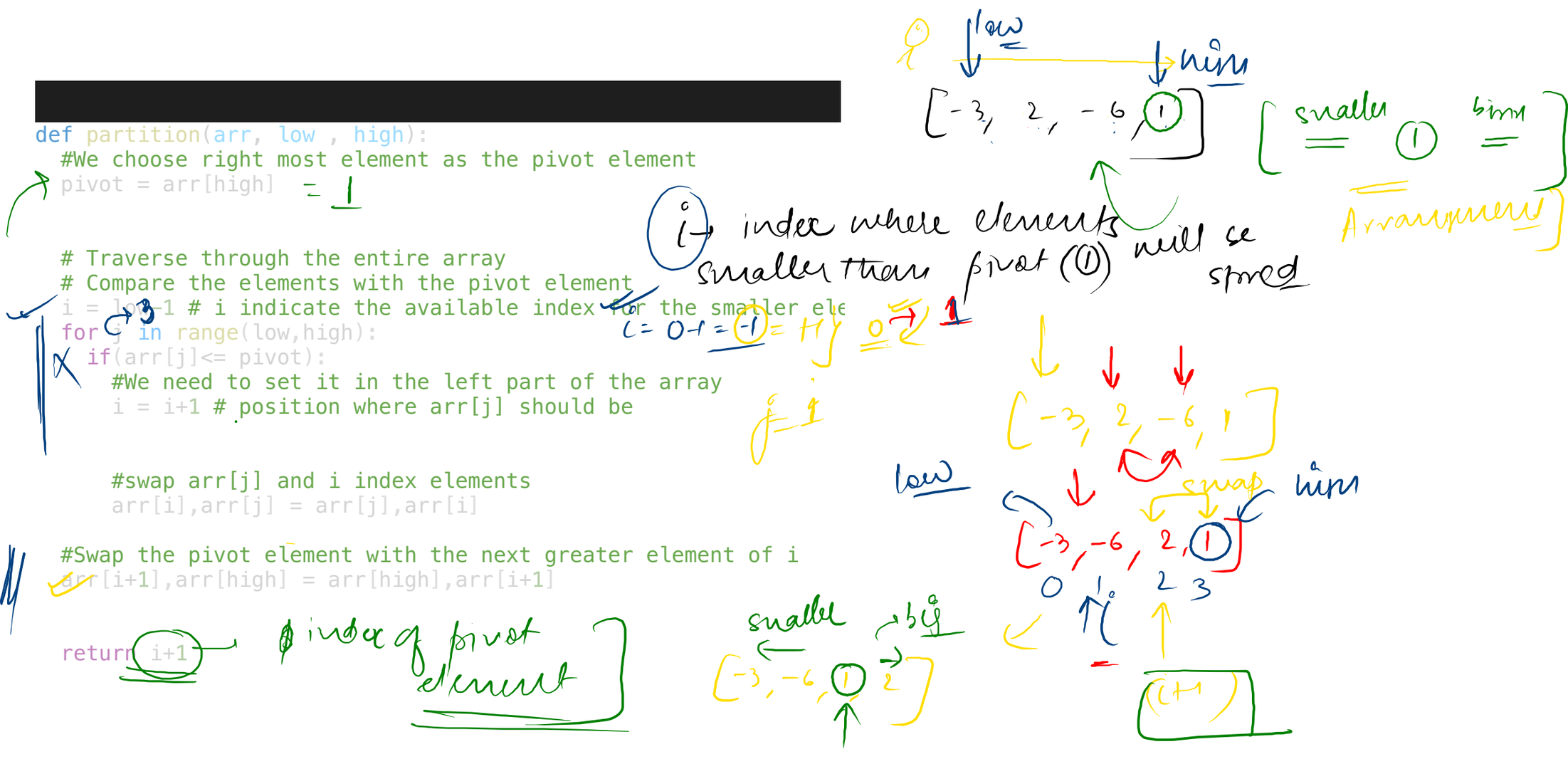


```python
def quickSort(arr, low, high):
  # Base condition
  if low >=high :
    return

  #find the pivot point and re-arrange the array around it
  pi = partition(arr, low, high)


  #left of pi will be all small
  # right of pi will be all big


  #sort the left part
  quickSort(arr, low, pi-1)


  #sort the right part
  quickSort(arr,pi+1, high)
```

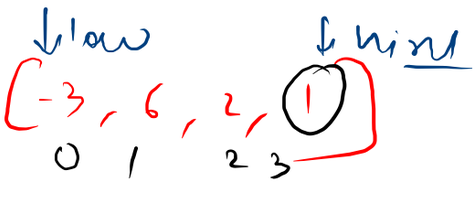
how we partition the array ?

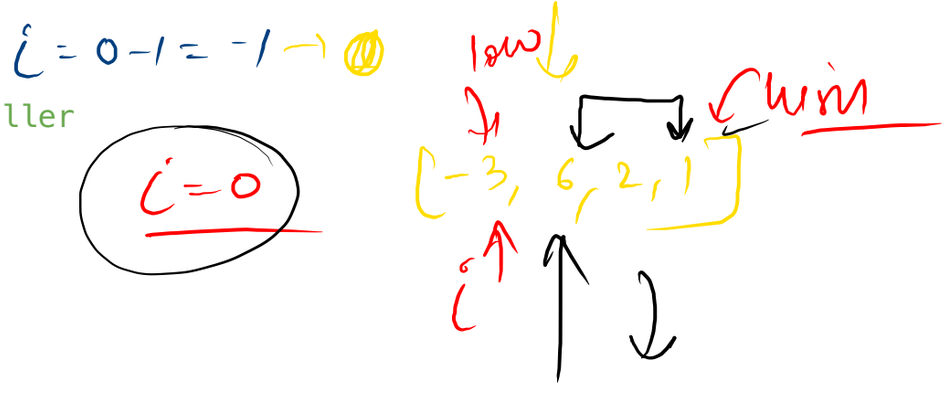


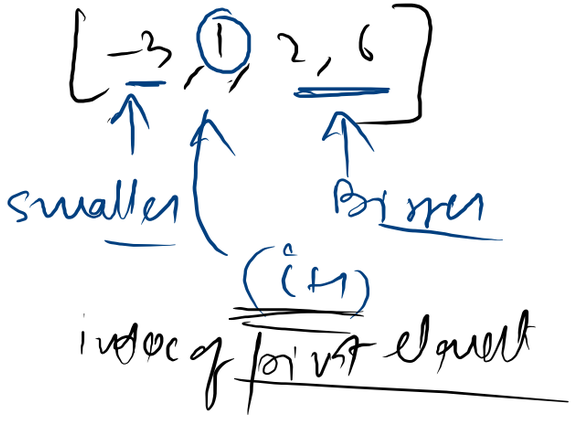
```
def partition(arr, low , high):
  #We choose right most element as the pivot element
  pivot = arr[high]

  # Traverse through the entire array
  # Compare the elements with the pivot element
  i = low-1 # i indicate the available index for the smaller ele
  for j in range(low,high):
    if(arr[j]<= pivot):
      #We need to set it in the left part of the array
      i = i+1 # position where arr[j] should be

      #swap arr[j] and i index elements
      arr[i],arr[j] = arr[j],arr[i]

  #Swap the pivot element with the next greater element of i
  arr[i+1],arr[high] = arr[high],arr[i+1]

  return i+1
```



```
def partition(arr, low , high):
  #We choose right most element as the pivot element
  pivot = arr[high]

  # Traverse through the entire array
  # Compare the elements with the pivot element
  i = low-1 # i indicate the available index for the smaller
elements than pivot
  for j in range(low,high):
    if(arr[j]<= pivot):
      #We need to set it in the left part of the array
      i = i+1 # position where arr[j] should be

      #swap arr[j] and i index elements
      arr[i],arr[j] = arr[j],arr[i]

  #Swap the pivot element with the next greater element of i
  arr[i+1],arr[high] = arr[high],arr[i+1]

  return i+1
```

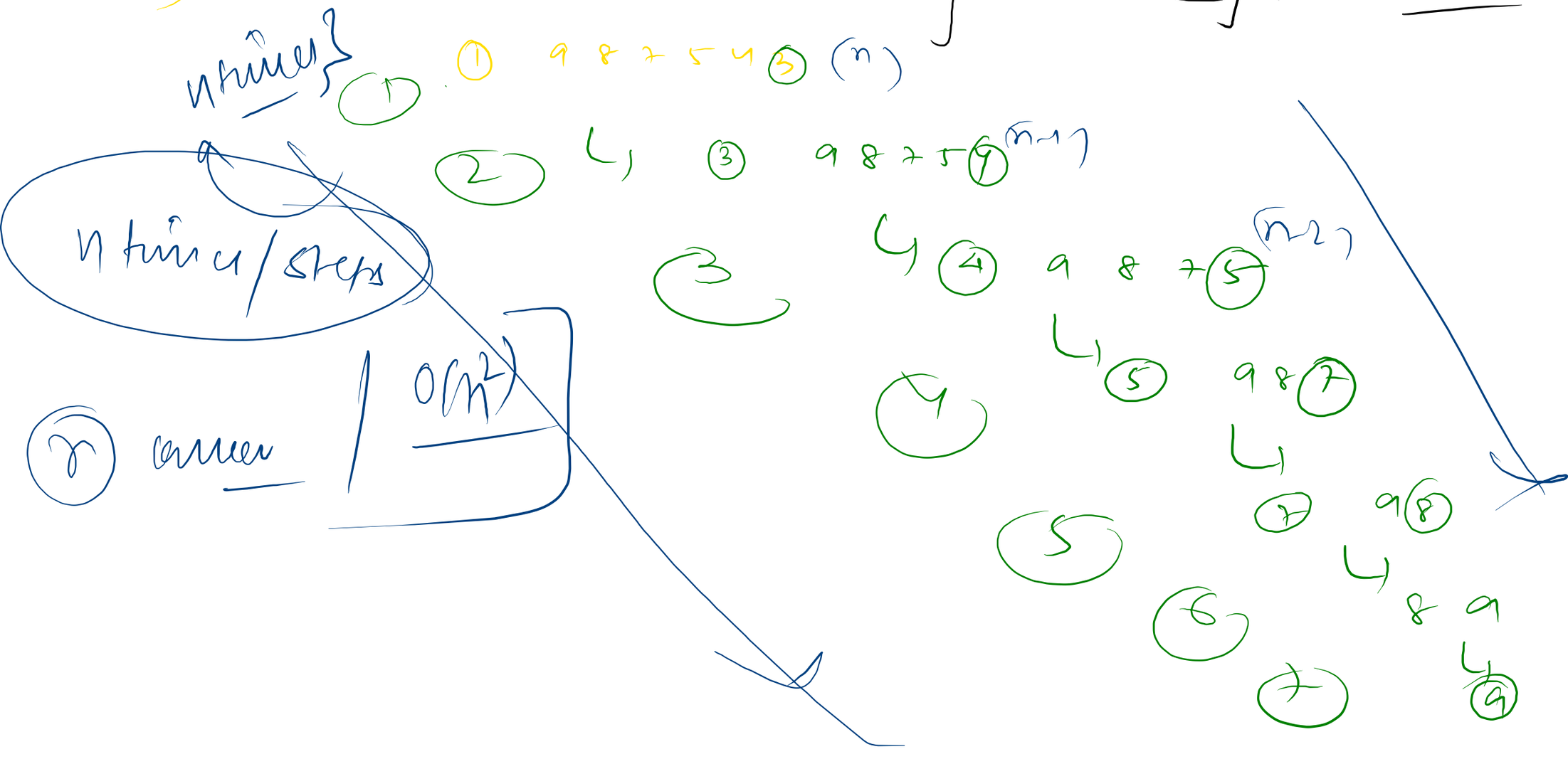




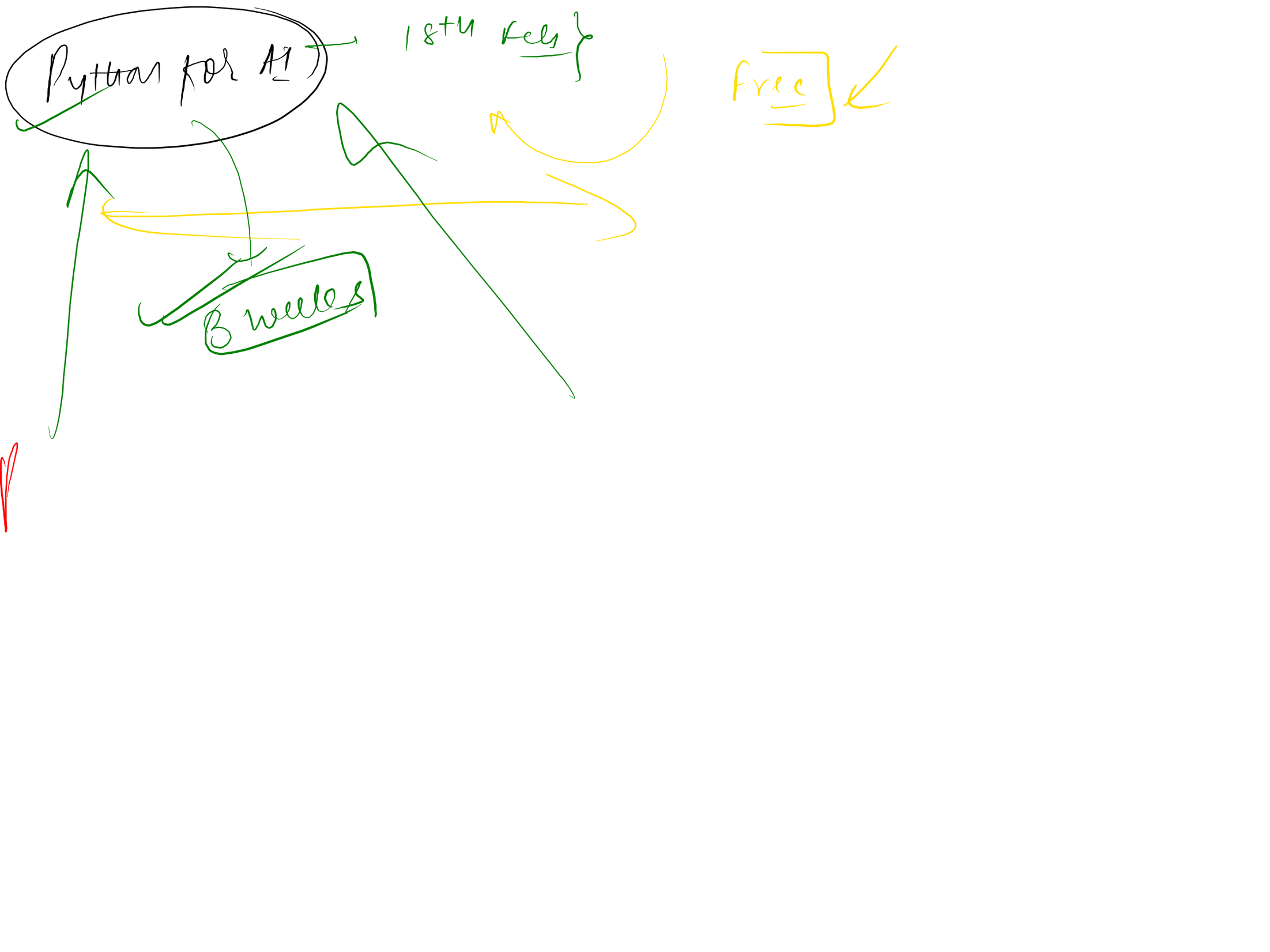


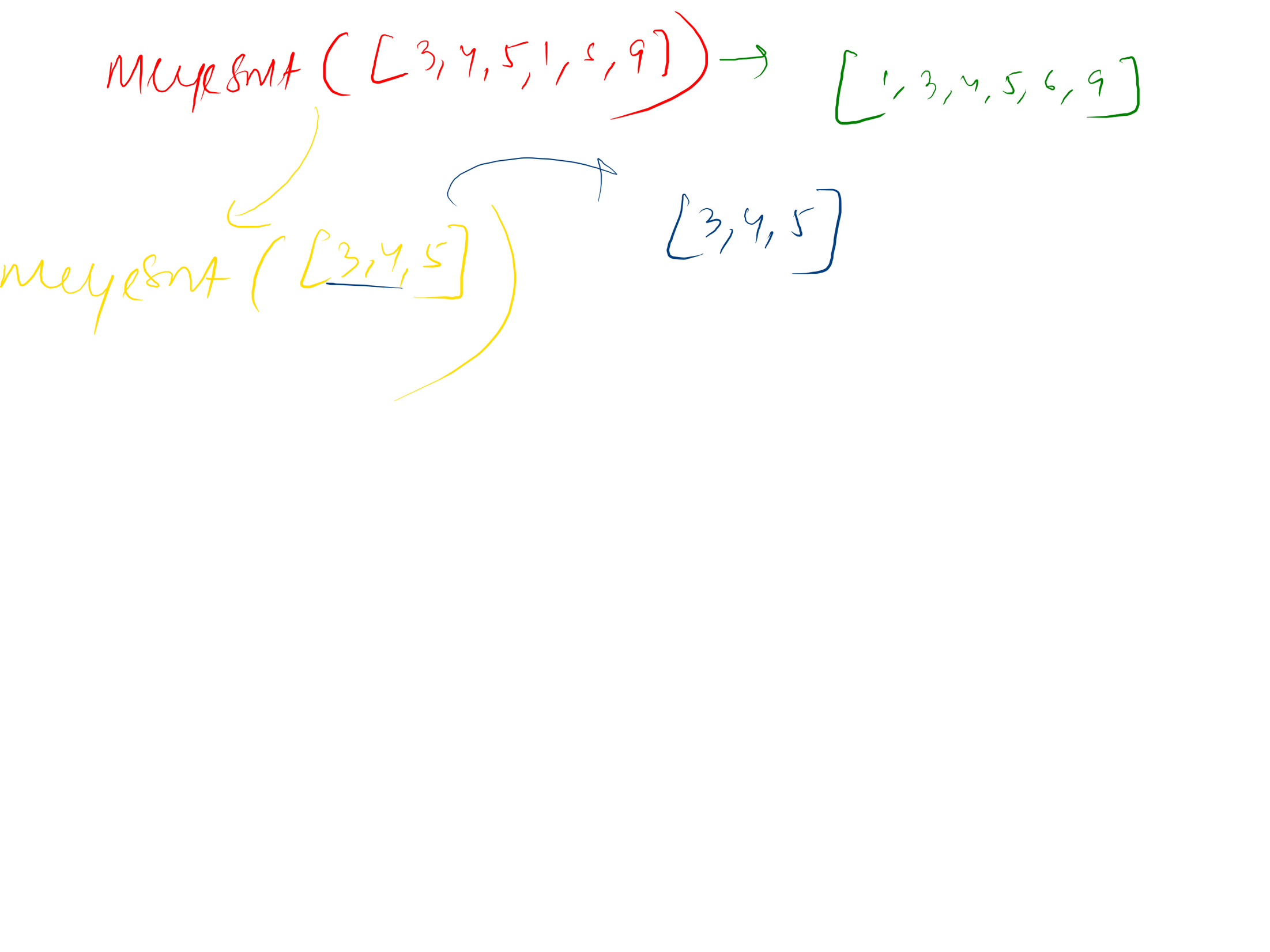
Reversed: 9 8 7 5 4 3 1

worst case $T = O(n^2)$



Interview → Accidental → $O(n^2)$ → Reverse sorted array


Python for AI
1 & 4 Feb
Free
8 weeks


Merge([3,4,5,1,6,9])
[1,3,4,5,6,9]
Merge([3,4,5])
[3,4,5]