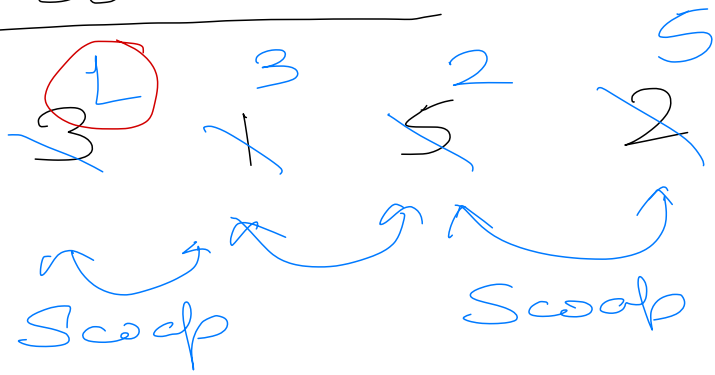
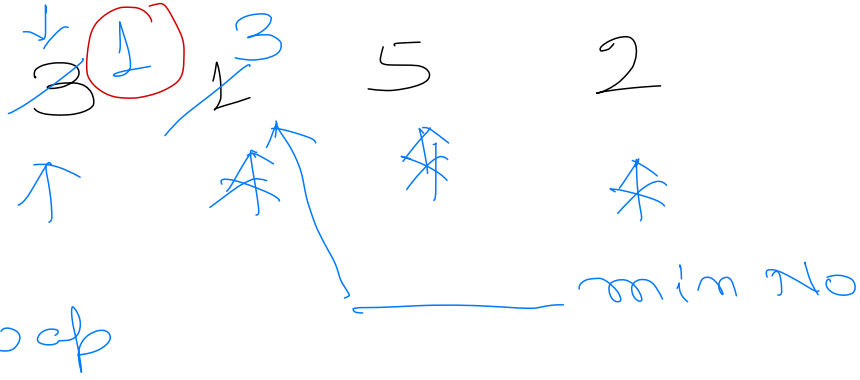
Bubble Sort



Selection Sort



Algo

⇒ for putElementAt → 0 to (n-2)

→ Assume last elem is smallest
smallestElemPos = n-1

⇒ for i → (n-2) down to putElementAt

→ if elem[i] <

elem[smallestElemPos]

→ smallestElemPos = i

→ Swap smallestElemPos & putElementAt elements.

Time Complexity ⇒ $O(n^2)$

Random access

→ Bubble Sort

for putElemAt → 0 to (n-2)

→ swapCount = 0

for right → (n-1) to (putElementAt + 1)

→ left = right - 1

→ if !(elem[left] < elem[right])

if (elem[right] < elem[left]) → Swap left & right elements.

→ Incr swap count

Sequencial access

→ if swapCount = 0 then swp.

Stable Sort ? → Bubble Sort ✓

Selection Sort ✗

Polygon filling

(Key, value)

(1, 5) (2, 3) (1, 2)

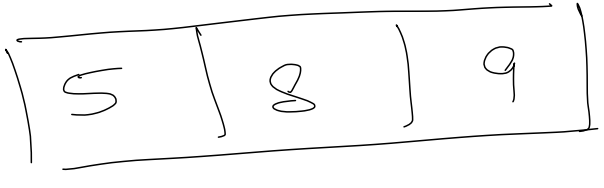
1) Stable Sort

(1, 5) (1, 2) (2, 3)

Insertion Sort

→ We have some element in sorted order

→ Insert the new element so that all elements remains sorted.



← Sorted elements

newElem → 3

5 8 9

↑ Shift 9 to right

5 8 → = 9

↑ shift 8 to right

5 → 8 9

↑ shift 5 to right

3 = 5 8 9

---

[3] 1 5 2

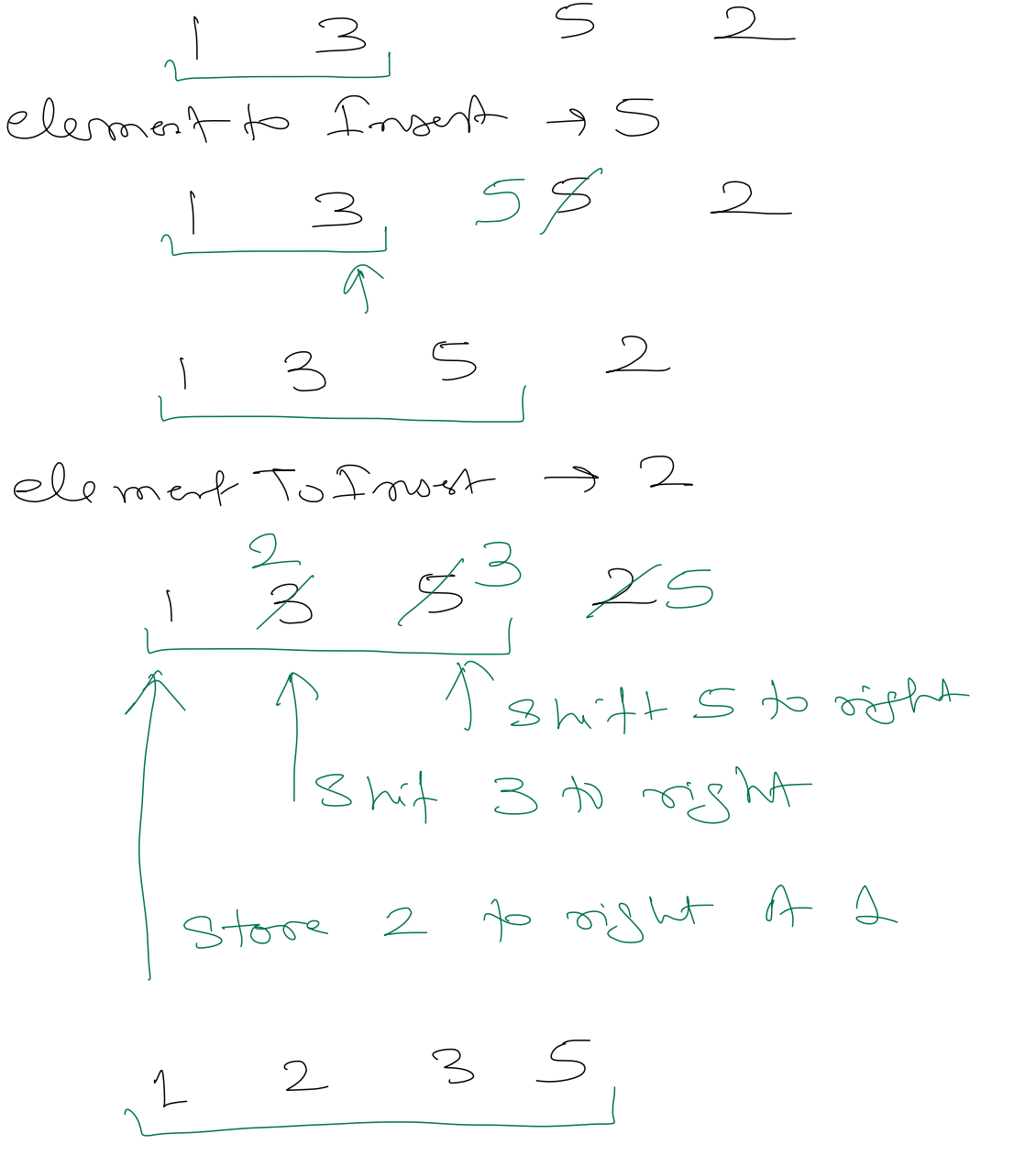
↑ initial sorted array

insert 1 in [3] so that elements remain sorted.

element to Insert → 1

1 [3] x 3 5 2

↑ Shift 3 to right


1 3 5 2
element to Insert → 5
1 3 5 5 2
1 3 5 2
elementToInsert → 2
2 3
1 3 5 5 2
shift 5 to right
shift 3 to right
store 2 to right of 1
1 2 3 5

InsertionSort(elements)
- for sortedArraySize -> 1 to (n - 1)
  - elementToInsert = elements[sortedArraySize + 1]
  - positionToInsert = sortedArraySize
  - while (positionToInsert >= 1)
    - if element[positionToInsert] > elementToInsert
      - Shift element at positionToInsert one place to right

      Else
      - End the loop
    - Decrement positionToInsert by 1
  - Set elementToInsert at (positionToInsert + 1)

$(n-1)-1+1$

$(n-1)$

sortedArraySize $-1+1$

⇒ Array indexing starts at 1, in the above algorithm.

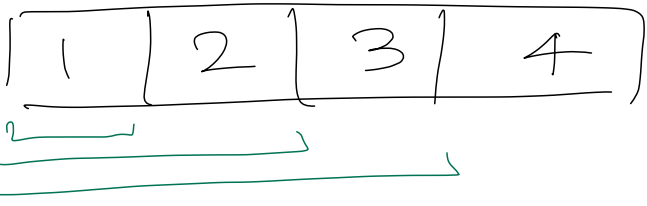
Dry Run

1	2	3	4
~~5~~	~~9~~ 5	~~9~~ x	2

$n \to 4$

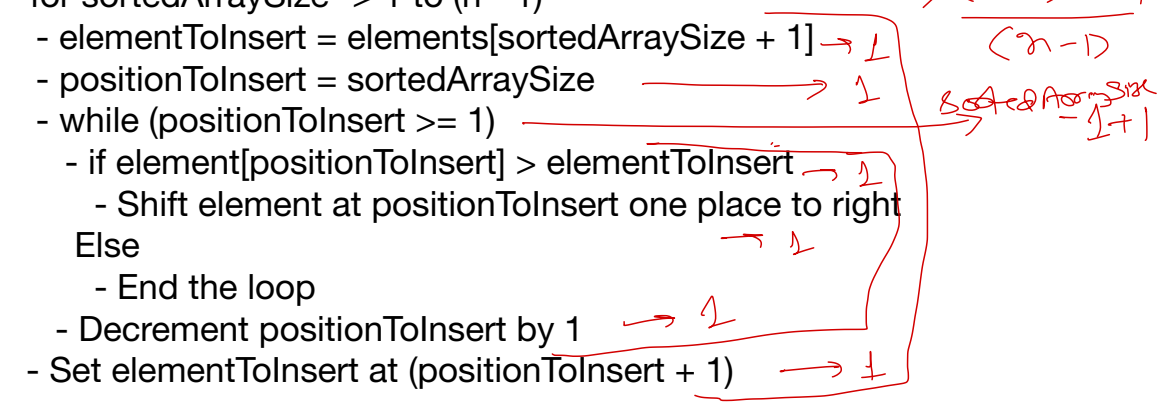
sortedArraySize → 1
elementToInsert → 9
positionToInsert → 1

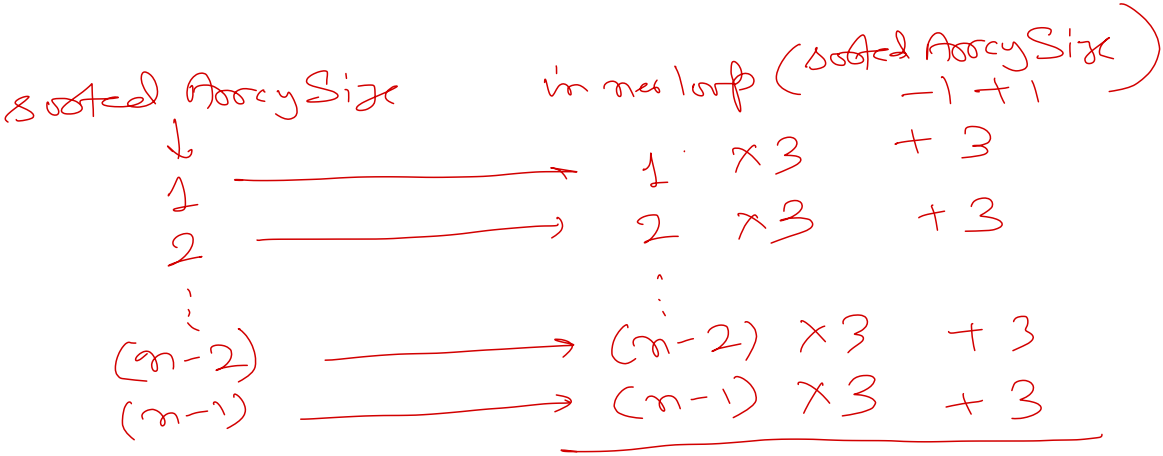
→ 2
→ 1
→ ~~2~~ x ○



InsertionSort(elements)
- for sortedArraySize -> 1 to (n - 1)
  - elementToInsert = elements[sortedArraySize + 1]
  - positionToInsert = sortedArraySize
  - while (positionToInsert >= 1)
    - if element[positionToInsert] > elementToInsert
      - Shift element at positionToInsert one place to right

      Else
      - End the loop
    - Decrement positionToInsert by 1
  - Set elementToInsert at (positionToInsert + 1)





$$[1\times3 + 2\times3 + \dots + (n-2)\times3 + (n-1)\times3] + 3\times(n-1)$$

$$3\times[1 + 2 + \dots + (n-2) + (n-1)] + 3\times(n-1)$$

$$3\times\frac{(n)(n-1)}{2} + 3\times n - 1$$

$$\frac{3}{2}(n^2 - n) + 3n - 3$$

⇒ Remove constants

$$n^2 - n + n$$

⇒ Take highest power of n

$$n^2 \Rightarrow O(n^2)$$

Best Case

↳ Element are already sorted.

$O(n)$ → Bubble Sort

$O(n^2)$ → Selection Sort

$O(n)$ → Insertion Sort

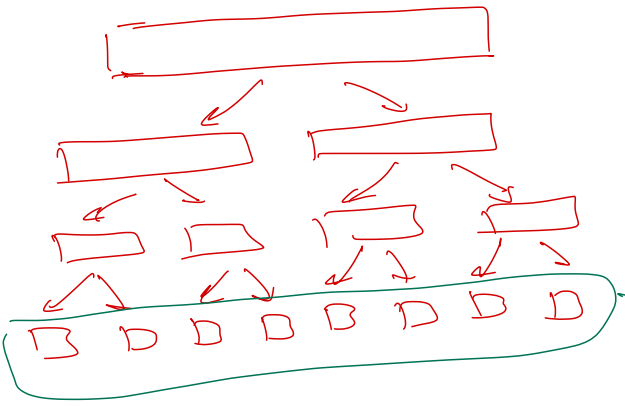
↓

Sorting Arrays, less efficient

Better than insertion sort

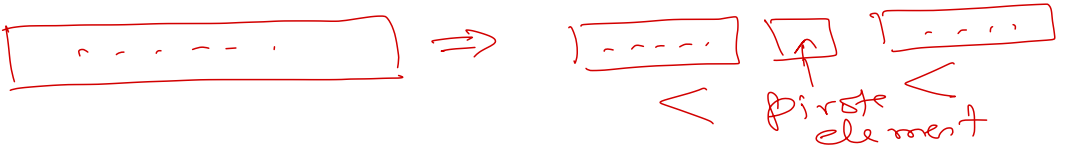
Quick Sort

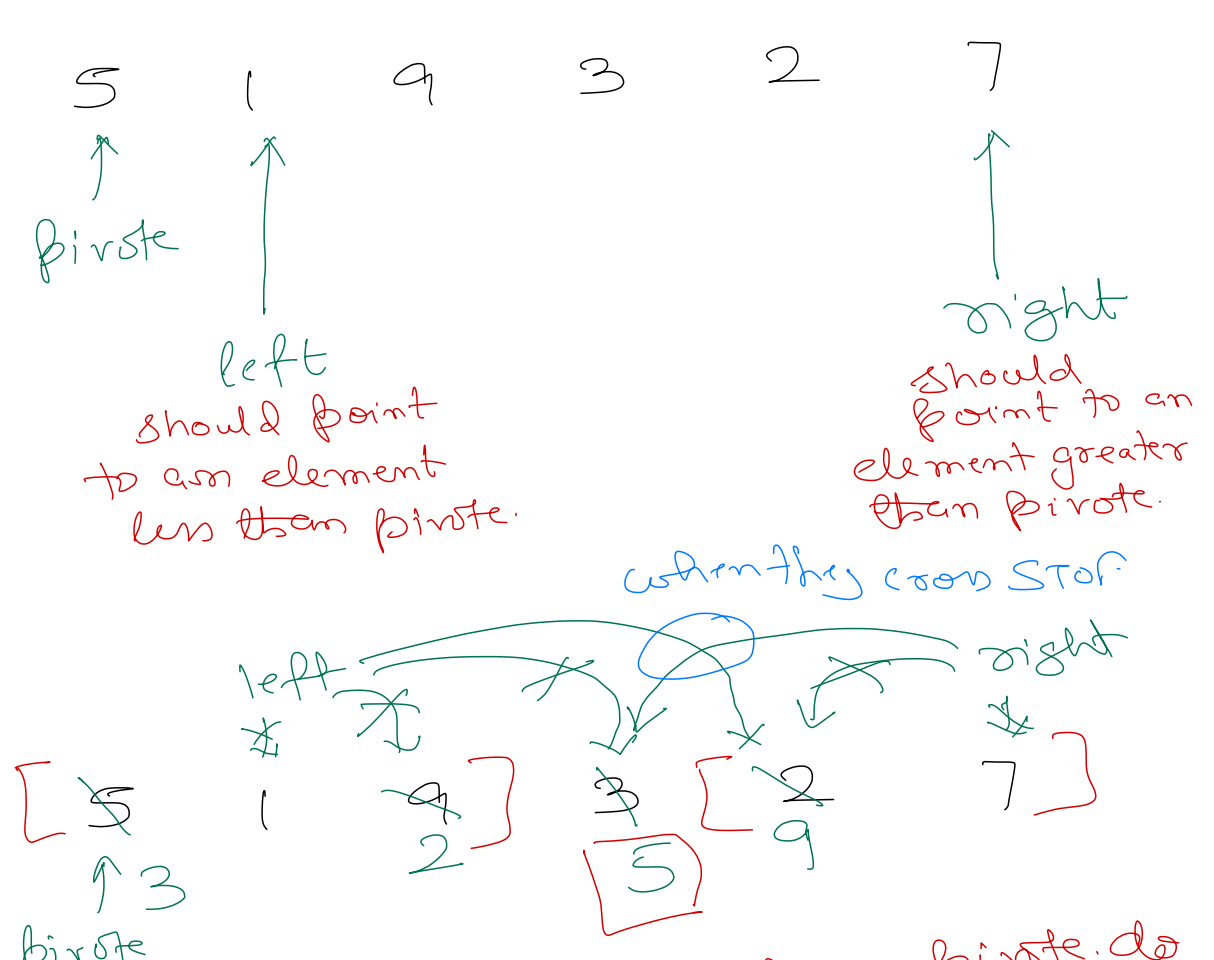
→ Divide elements into two groups and sort each of them individually.



each group is having one element in it, we say group is sorted.

→ QuickSort (elements)
  → if element count is 1 then
    → STOP.
  → Divide element into two groups
  → Sort first group via QuickSort
  → Sort second group via QuickSort.
    → Use Pivote element
    Elements less than pivote falls in first group
    Elements greater than pivote falls in second group.





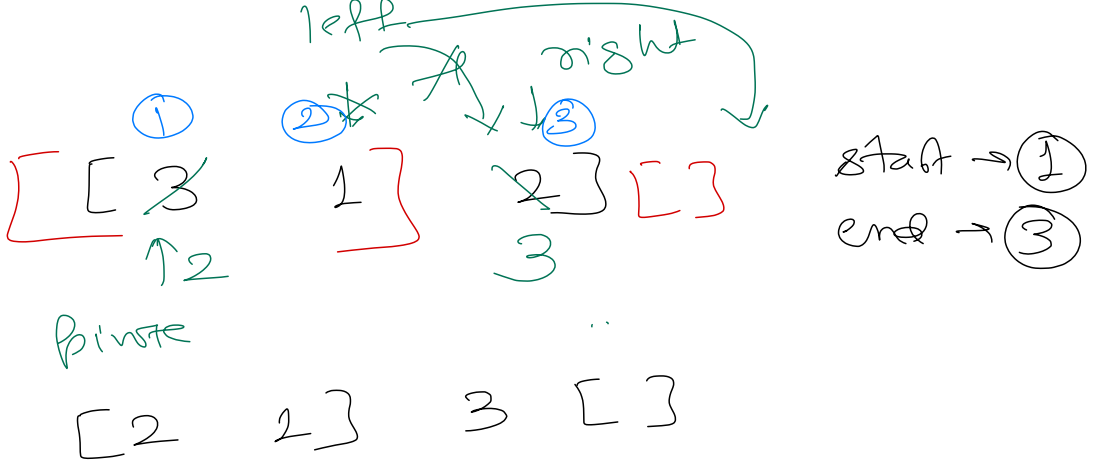
while left element is less than pivote. do
more left to next element.

while right element is greater than pivote do
more right to previous element.

→ Swap element pointed by left & right.

→ Swap pivote & right element.

→ Create two groups around right.



1 .. N
start end

QuickSort(elements, start, end)
- If elements count = 1 then
  - Stop

left → 2
right → N

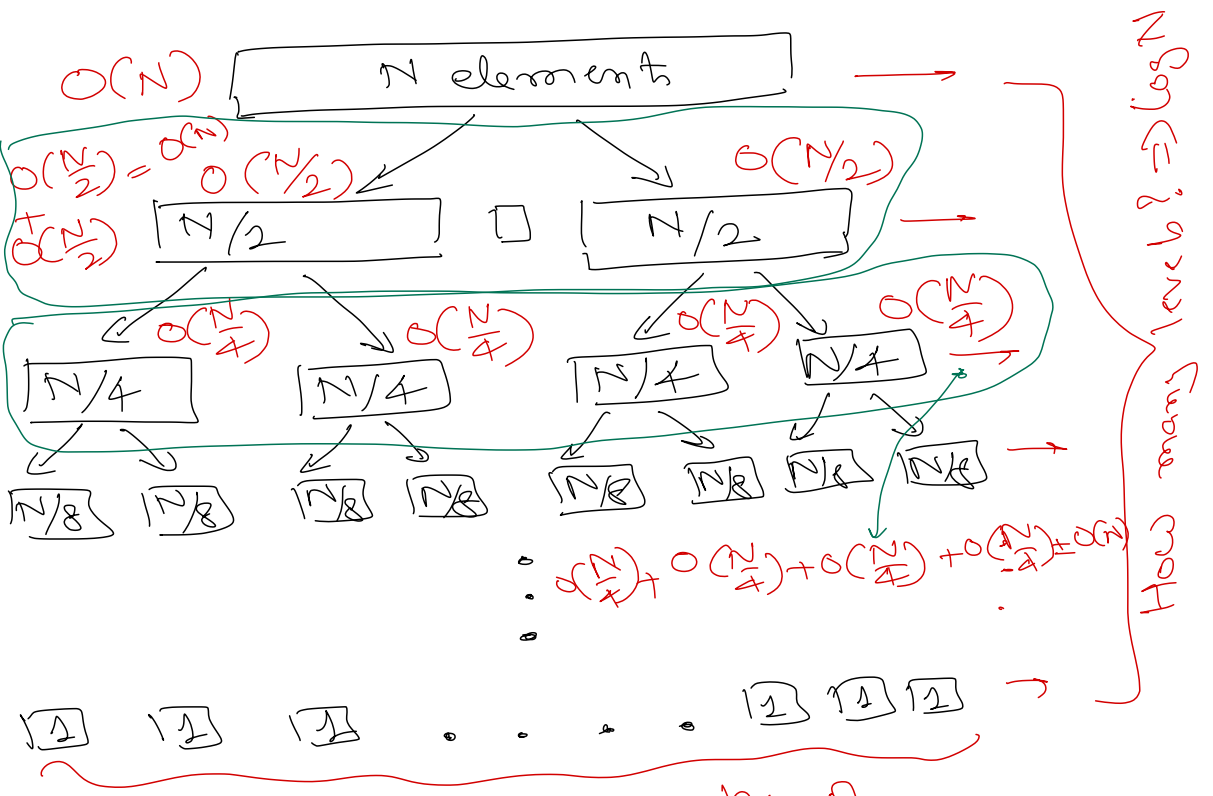
// Divide elements in two groups
- Set pivote to first/start element
- Set left to element after pivote
- Set right to last/end element
- while (left <= right) do

(N - 2 + 1) times
⇒ O(N)

    // Find two elements pointed by left and right, which are incorrect.
    - while left element is less than pivote do
      - Move left to next element.
    - while right element is greater than pivote do
      - Move right to previous element.
    - Swap left and right elements.
- Swap pivote and right elements

what if left cross over at this point!

- if start < (right - 1) then
  - QuickSort(elements, start, right - 1)
- if end > (right + 1)
  - QuickSort(elements, right + 1, end)



How many groups of 1 element each? ⇒ N groups

$O(N) + O(N) + O(N) + \dots + O(N)$

terms count? = number of levels

$$\frac{O(N) + O(N) + \dots + O(N)}{\log n \text{ times}}$$

$$= \log N \times N$$

$$\Rightarrow O(N \log N)$$

Coorse case for Quick Sort

↳ Sorted elements

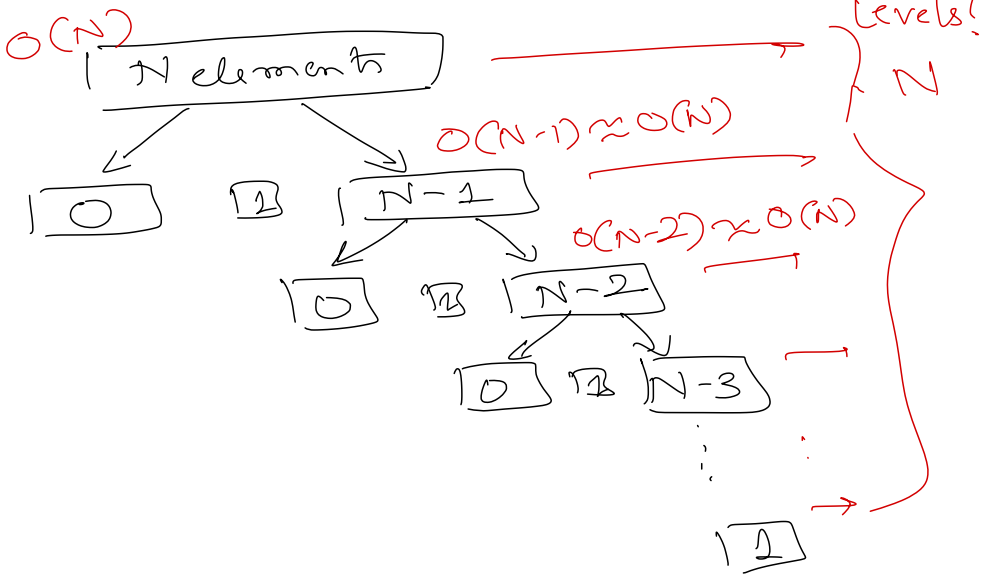
[1 2 3 4 5]

Pivot

[ ] [1] [2 3 4 5]

Pivot

[ ] [2] [3 4 5]



$O(N) + O(N) + \dots O(N)$

N times.

$= N \times N = N^2 = O(n^2)$

⇒ Time complexity of quick sort is $O(n^2)$ when array is sorted.

⇒ Selection of pivot matters as it decides forming of groups.