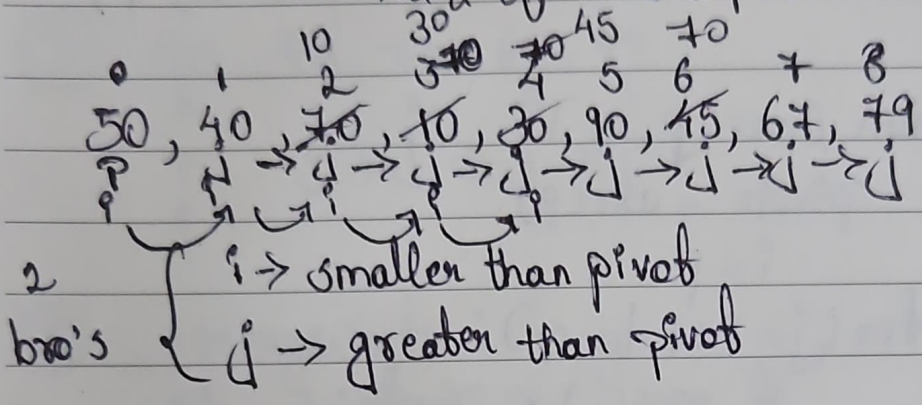
Date:

# QuickSort

→ Inplace sorting Algorithm
→ Not a stable sorting Algorithm.
→ Partition Algorithm → hero
↳ Divides the array → smarter way

↓

no need of combine part.

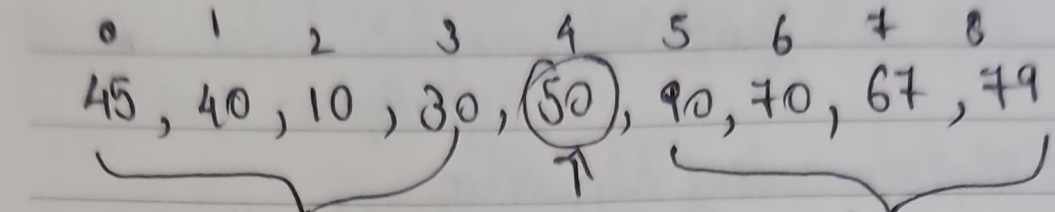



Pivot → start element.

- j keeps comparing with pivot element & informs i if value is smaller than j

If value is greater j increments.
Value of i increments only when the swap happens.
i → will update position & then swap.
j will connect with i only when j value is smaller than pivot
j → i+1 initally.
At the end of each iteration i & pivot elements swap with each other.

Date:




Smaller <50 → p

greater >50 → p

Partition → (4) → pivot element

QS(arr, 0, 3)

QS(arr, 5, 7)

```
partition (arr, p, q):
    pivot = arr(p)
    i = p
    for (j = i+1 to q):
        if arr(j) ≤ pivot:
            i += 1
            swap(arr(i), arr(j))
    swap(arr(i), arr(pivot))
    return i

QS (arr, p, q):
    if i < j:
    m = partition (arr, p, q)
T(m-p) ← QS(arr, p, m-1)
T(q-m) ← QS(arr, m+1, q)
```

↳ q - (m+1) + 1 ⇒ q - m

Date:

② 30 45, 40, 10, 30 45
P j → j → j
i 1 2 3

i = 3

(30, 40, 10,) 45
0 1 2 3
QS(arr, 0, 2)

Small problem
↳ p == q
↳ return arr(p)

③ 50, 90, 70, 67, 79
0 1 2 3 4
P J → j → j → j
i

④ MergeSort

mid = p + (q-p)/2

T(n/2) = mergeSort(arr, p, mid)

T(n/2) = mergeSort(arr, mid+1, q)

pivot element can be → 1st element
last element
mid element
random element.

Ending Index - Start Index + 1

$$T(n) = \begin{cases} C & n = 1 \\ T(m-p) + T(q-m) + n \; ; & n > 1 \end{cases}$$

Date:

## Best case / Average case

$$T(n) = T(n/2) + T(n/2) + n$$
$$= 2T(n/2) + n$$
$$= \Theta(n \log n)$$

n is almost equal no. of elements in both left & right side.

Worst case:

$$T(n) = T(n-1) + 1 + n$$
$$= T(n-1) + n$$
$$= \Theta(n^2)$$

↳

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 50 | 70 | 60 | 69 | 72 | 87 | 68 |
| P | j → | j → | j → | j → | j → | j |
| i | | | | | | |

50 (70, 60, 69, 72, 87, 68)

↳ QS(arr, 1, 6)

20 40 13 | 50 | 60 57 79

n/2 | 1 | n/2

Best case.

Date:

Worst case:-

$n-1$

50 {60 57 79 52}

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

Worst case scenario -> when?

↳ Almost or completely sorted Arrays

Hence we use QuickSort when/in practical real life use-cases where actual sorting is required.

-> Insertion Sort -> $O(n^2)$

Randomized Quicksort

↳ Pivot element randomly

random index -> pivot element

6 <=> 0 -> same code

Probability -> worst case scenario

↳ Quite Less < normal Quicksort

↳ Research

Date:

## ⇒ Questions :-

1) $k^{th}$ smallest element :-

↳ Quicksort or Mergesort → SortedArray ↳ $O(n \log n)$

$k = 2$

[2nd smallest element]

→ return arr(k-1)

→ Brute force Approach

So use, selection procedure

↳ $O(n)$

2

20, 40, 5, 7, 9, 12, 46

(Quicksort)

Partition → return position of pivot

↳ m = 4

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 5, | 7, | 9, | 12, | 20, | 40, | 46 |

$k = 2$

Output = 7

Left → SelectionProcedure(arr, k, i, m-1)

Right → selectionProcedure(arr, k, m+1, j)

Date:

Best case

$T(n) = T(n/2) + n$

$= O(n)$

Worst case

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

$= O(n^2)$

2) kth largest element

{5, 7, 9, 12, 20, 40, 46}

k = 2

Output => 40

3) sort colors :-

nums = [2, 0, 2, 1, 1, 0]

result = [0, 0, 1, 1, 2, 2]

↳ Quicksort → O(nlogn)

Can we optimize this without sorting algorithm?

Two pointer approach → O(n)

$P_0$ → zeroes

$P_2$ → twos

$P_0 = 0$

$P_1 = len(nums) - 1$

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

0 curr $n-1$

nums[curr] == 0

↳ swap(nums(curr), nums[p0])

nums[curr] == 1

curr += 1

$P_0$ += 1

curr += 1

Date:

```
nums[curr] == 2:
swap(nums[curr, nums[p2]])
p2 -= 1
```

Time complexity = O(n)
Space complexity = O(1)

More loop === more time to execute.

## 4) Majority Elements:

Don't use recursion → stack space
↳ space complexity O(n)
arr = [2, 2, 1, 1, 1, 2, 2]
Hashing Data Structure.
↓
(key - value) pair.

| key | value |
|---|---|
| 2 | (4) → output. |
| 1 | 3 |

↳ Hash table.

Boyer-Moore Voting Algorithm
↳ O(n) TC, SP → O(1)

Date:

## 6) Peak Element :-

[1, 2, 1, 3, 5, 6, 4]
 0  1  2  3  4  5  6

Output → 6 → Peak Element.

[4, 3] → 4
 0  1

```
nums[0] > nums[1]
    return 0
nums[-1] > nums[-2]
    return len(nums)-1

while start < end:
    mid = start + (end-start)//2
    if (nums(mid) < nums(mid+1))
        start = mid+1
    else
        end = mid
return end
```