agenda!

1. Pivot Partition
2. Quick Sort
3. Comparator Problems

↳ it is about custom method of sorting,

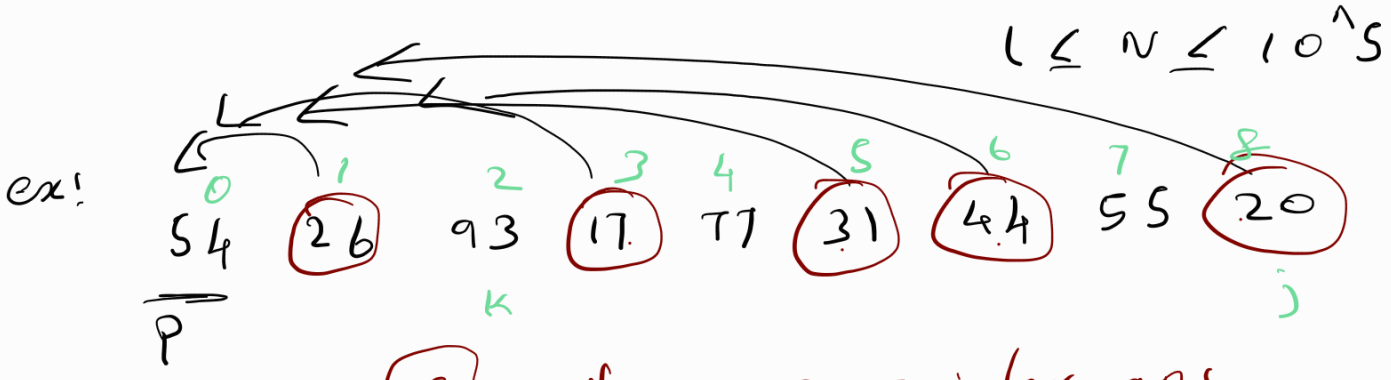
<Question>:- Find Pivit Element

1. Given an int [] A,
2. consider first elements as pivot, re-arrange the element such that for all i:

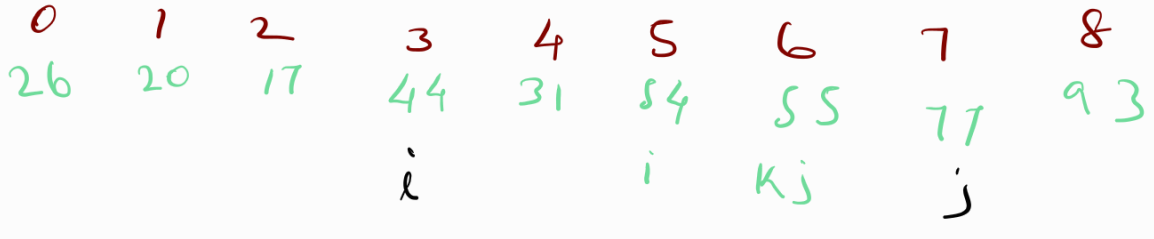
   if A[i] <= P then it should be present on LHS

   if A[i] > P then it should be present on RHS

3. Return index of pivot element.



if LHS = 5, then P = 5 index ans



①

1. iterate over A and count elements <= P
2. return ans

②

lhsArr = [ ]

rhsArr = [ ]

lhsArr + P + rhsArr

require additional array

③ Count Sort

Same (N-1)

④

i = -1 2

k = 3

i = 0

j = 5

P = A[0] = 5

while (k < j) # stops when k and j becomes equal

if A[k] <= P:

swap (k, i), k += 1, i += 1

else:

swap (k, j), j --

return i

1. i = 0, k = 1, j = N-1, P = A[0]

```
i=1, j=n-1, p=A[0]

while (i <= j)
    if A[i] <= p:
        i++
    elif A[j] > p:
        j--
    else:
        swap(i, j)

swap(0, j) # as j is the index of pivit value.

return j
```

TC: O(N), as we are touching every elements only once.

SC: Constant O(1)

Quick Sort

⇒ it is a sorting algo.

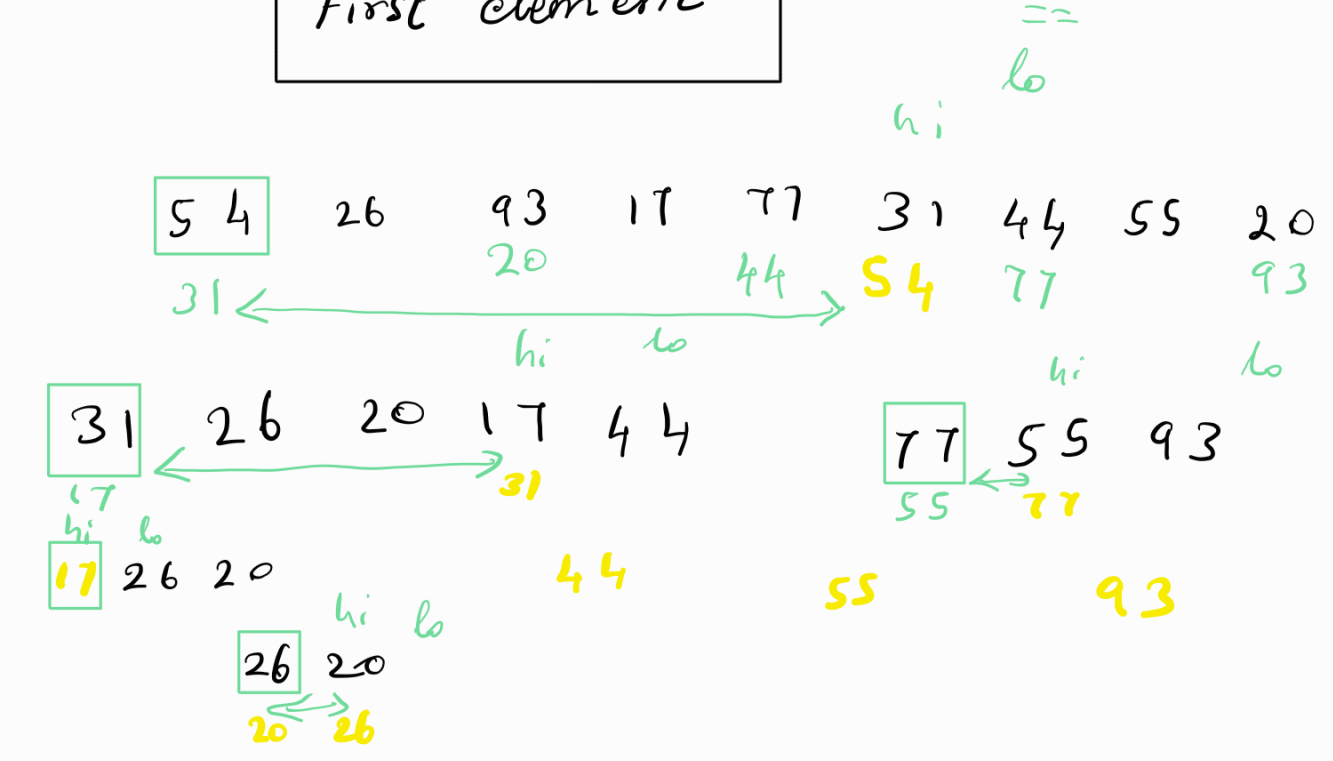
⇒ Taking a pivit point and arrange smaller than pivit value into LHS, grater values into RHS.

⇒ Recursively apply the same process to the two partitioned subarrays

⇒ Recursion stops at basecase, when there is only one element left in sub-array

Question which one is pivote point?

First element



ans = 17 20 26 31 44 55 77 93

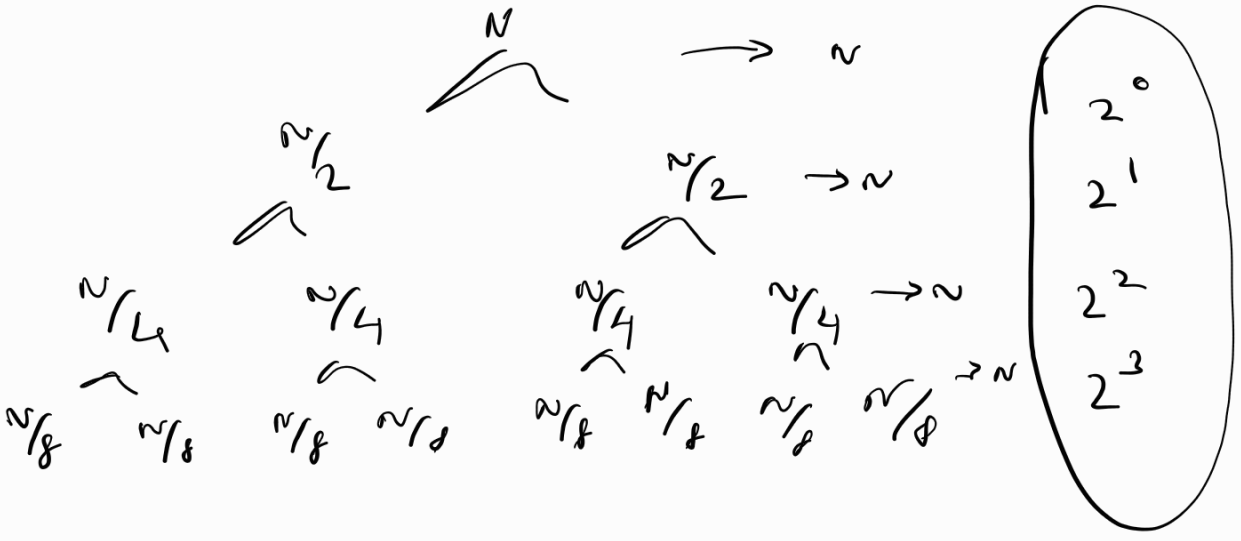
Code

```
def quickSort(A, start, end):
    # base case
    if start >= end: return
    # Recursive case
    pivit = partition(A, start, end)
    # Handle left and right sub-array
    quickSort(A, pivit+1, end) # right
    quickSort(A, start, pivit-1) # left
    return A

def partition(A, start, end):
    i = start+1
    P = A[start]
    while (i <= end):
        if A[i] <= P:
            i+=1
        elif A[end] > P:
            end-=1
        else:
            swap(A, i, j)
    swap(A, start, end)
    return end
```

```
def swap(A, i, j):
    temp = A[i]
    A[i] = A[j]
    A[j] = temp
```

Best case / Average case: {we are able to divide array by equal half (or) almost halfs}

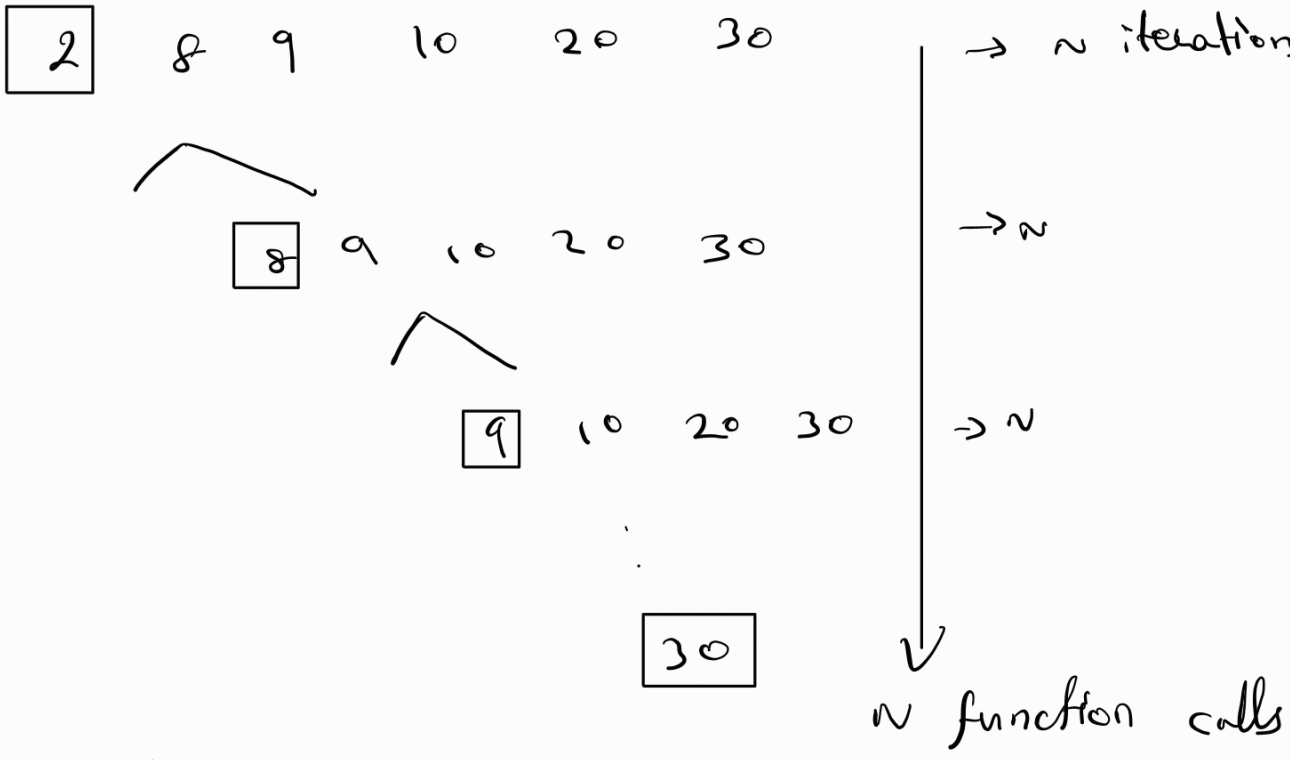


num of function calls = $\log_2 N$

Time per function = N (N iteration in every level)

worst case of Quick Sort

1. when we can not devide array by equal half.
2. (or) when array is alread sorted.



$\Rightarrow N \times N$

Worst Case $\Rightarrow N^2$

Note: There is a way of getting rid of worst case situation.

$\Rightarrow$ by using Randomized pivot point

Randomized Quick Sort

1. Select pivote point using random function.

   math.random(0, N-1)

2. Swap it with First element (index 0)
3. the same script will work

Note: if we pick the minimum element always it will lead to worst case senario

Lets see:

The probablity of picking smallest element in arra of (N) is : $1/N$

The probablity of picking smallest element in array of $(N-1)$ is = $1/N-1$

The probablity of picking smallest element in array $(N-2)$ is = $1/N-2$

⇒ This is continous sequence of events, what is over all probablity

$$\Rightarrow \frac{1}{N} \times \frac{1}{N-1} \times \frac{1}{N-2} \cdots \frac{1}{1}$$

$$\Rightarrow \frac{1}{N \times N-1 \times N-2 \times N-3 \cdots 1} = N!$$

$$\Rightarrow \frac{1}{N!}$$

Factorial of 3 : $3 \times 2 \times 1$

⇒ as the N! is very very large.
the over all probablity is very very less

⇒ so, the chance of choocing smallest elemen every time is very less, we can ignore it.

```
def partition(A, start, end):
    i = start + 1
    P = A[start]
    while (i <= en):
        if A[i] <= P:
            i += 1
        elif A[end] > P:
            end -= 1
        else:
            swap(A, i, j)
    swap(A, start, end)
    return end
```

Math.random(start, end)
swap(A[start], value)

Comparator (which one to keep first, and second by using below)

1. if first arg to come before second **return** +ve
2. if second arg to come before first **return** -ve
3. if both are same **return** 0 # if we dont care wich comes first and second.

=> Given int[] A,

=> sort the data in **ascending order of count of factors.**

=> if count of factors are equal, then sort the array basis of values.

A = 9 3 10 6 4

factors 2 2 4 4 2

ans = 3 4 9 6 10

9 → 1 3 9

3 → 1 3

10 → 1 2 5 10

6 → 1 2 3 6

4 → 1 2 4

ans = 2

1 = 0
2 = 1
3 = 1
4 = 1

25 → 1 5 25 (excluding)

23 → 1 23

23 ** 0.5

22/1 = 22

11 2 1/2 = 11

```
count = 0, i = 1
while ((i x i) <= N):
    if N % i == 0:
        if N % i == i:
            count += 1   # if N is perfect square
        else:
            count += 2
    i += 1
```

count factors

$\sqrt{N} = N^{**}.5$

1, iterate till square root of N and get all the factors

```
if factor(val1) < factor(val2):
    # return val1
    return -1
elif factor(val1) > factor(val2):
    # return val2
    return 1
else: # when both factors are same.
    # sort basis of value
    if val1 < val2:
        return -1
    elif val1 > val2:
        return 1
    else return 0
```

return val1 - val2

Python:

```
import functiontools

def factors(N):
    .
    .
    .

def compare(V1, V2):
    .
    .
    .

A= sorted( A, key = funtools.comp_to_key(compare)

    return A
```

Largest Number

⇒ given int [ ]

⇒ arrange and return such that they form the largest number

ex: A = [10, 2] ⇒ 102

ans = [2, 10] ⇒ 210 can be formed.

2. [3, 30, 34, 5, 9]

ans [9, 5, 34, 3, 30] ✓

9 5 3 34 30 ×

~~9 5 3~~ 4 3 0

~~9 5 3~~ 3 4 30

3. [10, 5, 2, 8, 200]

8, 5, 2, 200, 10 = 8 5 2 200 10

int a int b

9 50

if a+b = b+a

return 0

elif a+b > b+a

return -1

else return 1

how to append digits?

int(str(9) + str(50)) = 950

Code:

```
from functools import cmp_to_key

class Solution:
    def largestNumber(self, A):
        def comp_func(x, y):
            if x+y == y+x:
                return 0
            elif x+y < y+x:
                return -1
            else:
                return 1
        nums = [str(num) for num in A]
        nums.sort(key = cmp_to_key(
                                comp_func))
        if nums[0] == 0:
            return 0
        return "".join(nums)
```