agenda!

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

↳ it is about custom method of sorting,

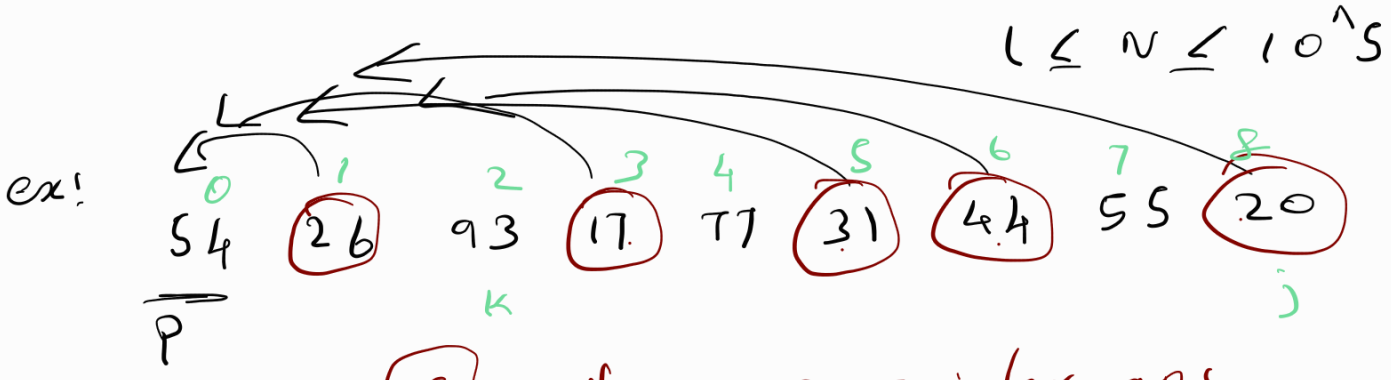
<Question>:- Find Pivit Element

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

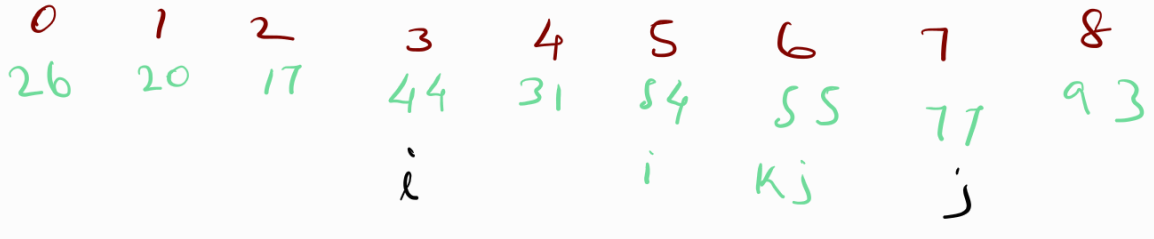
   if A[i] <= P then it should be present on LHS

   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.

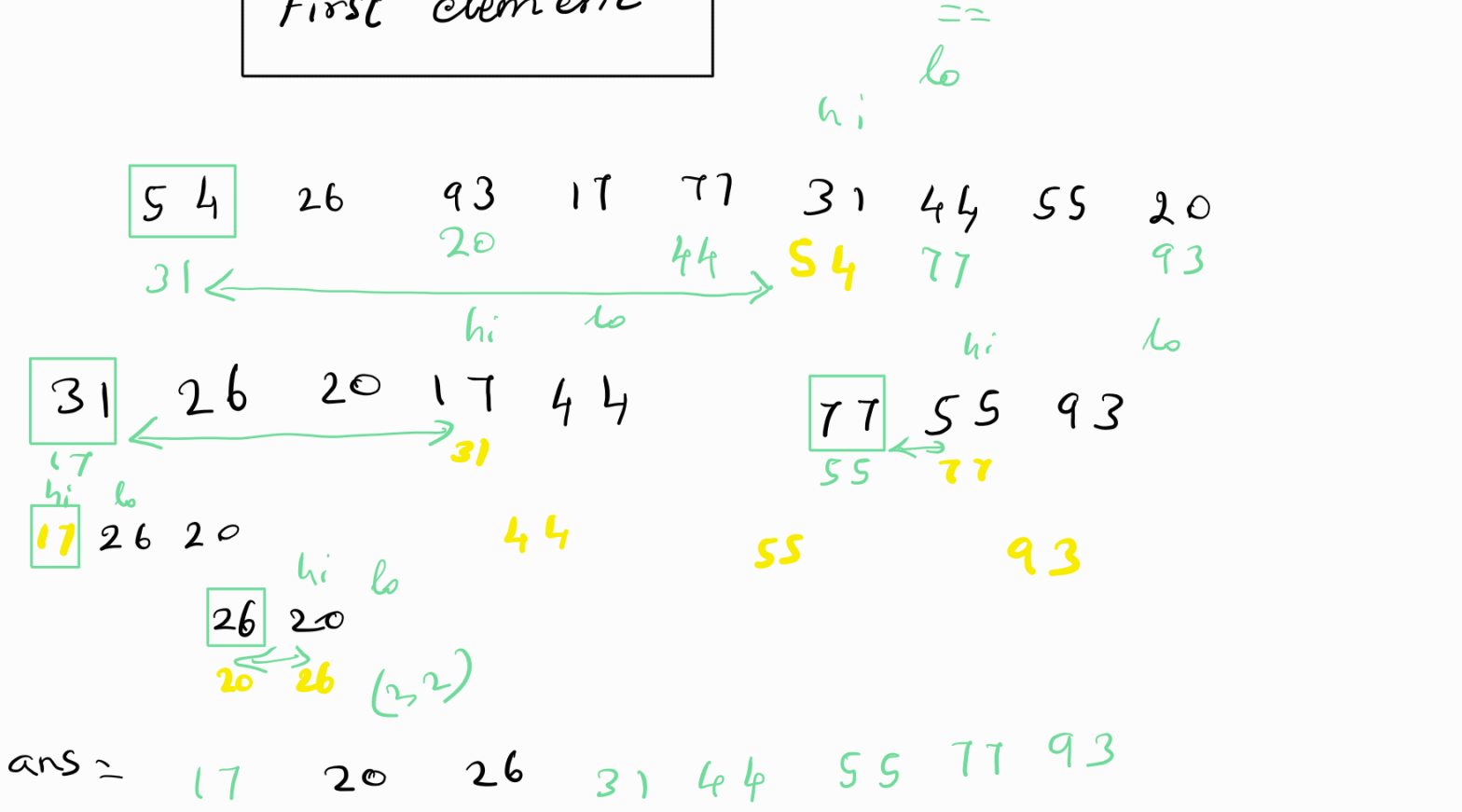
⇒ Taking a pivit point and arrange smaller than pivit value into LHS, grater values into RHS.

⇒ 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

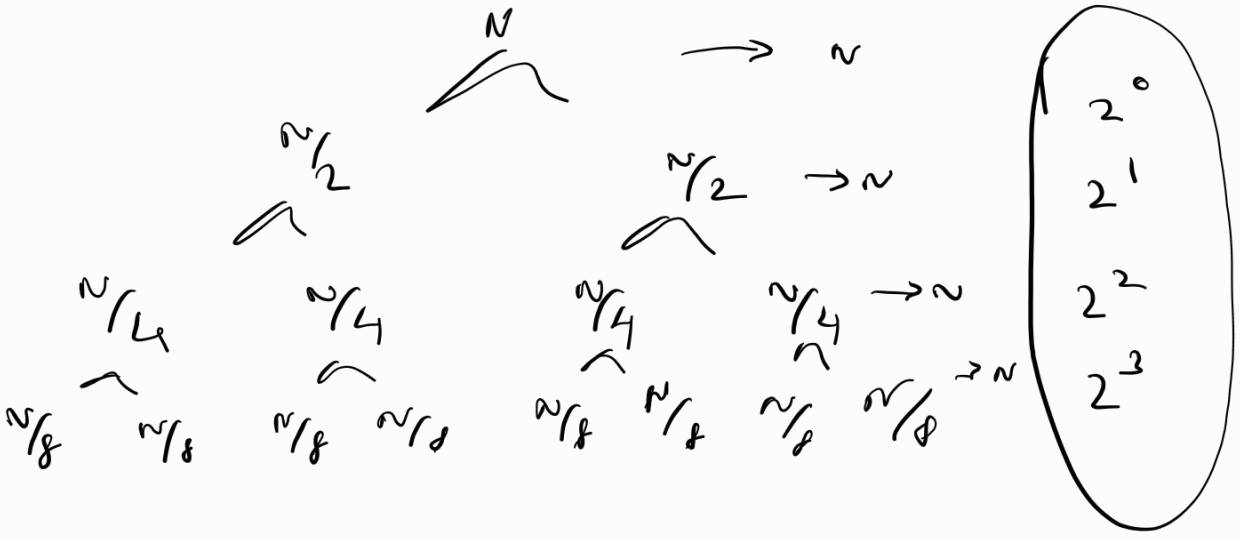
Code

```
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}

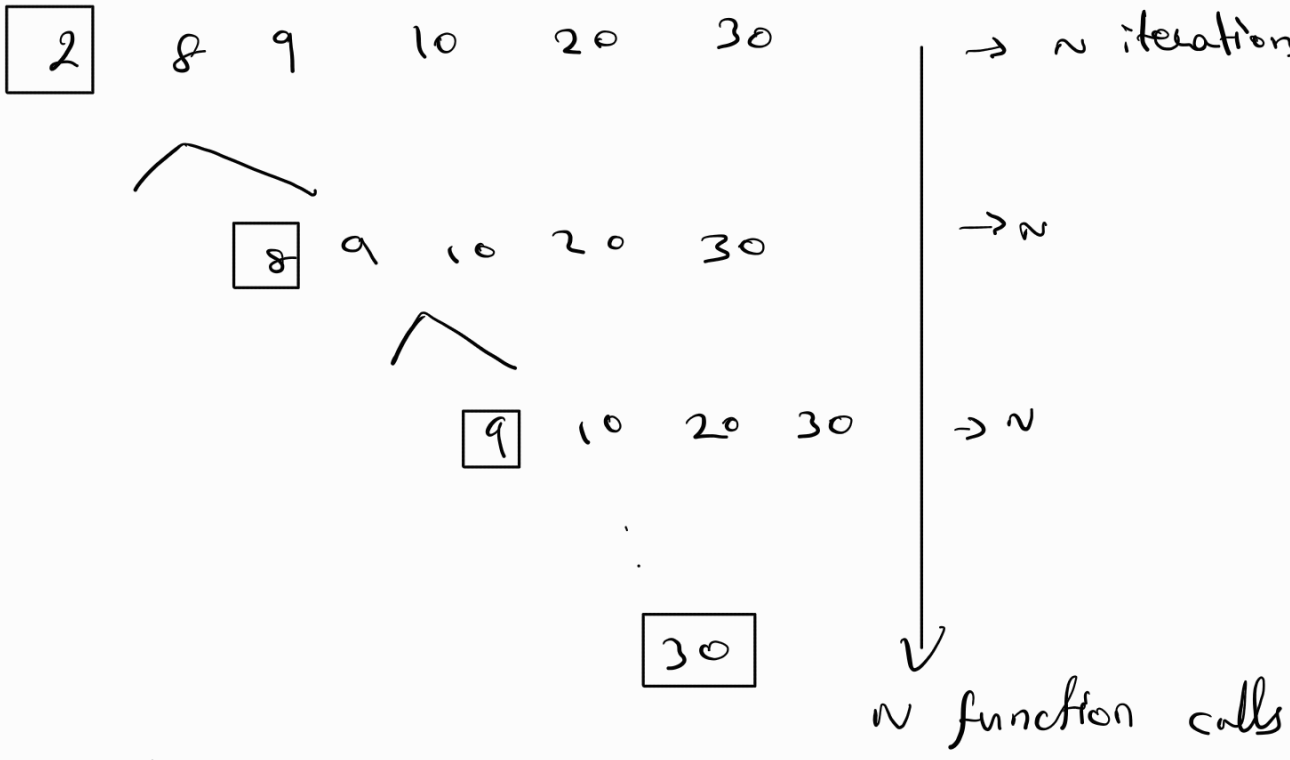


num of function calls = $\log_2 N$

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 = sorted(A, key = cmp_to_key(
                                comp_func))
        if nums[0] == '0':
            return 0
        return "".join(nums)
```

1. Largest Number

=> Given int[] A

=> arrange them such that they form largest num

$1 \leq |A| \leq 10^5$

$0 \leq A[i] \leq \frac{2^x 10^9}{int}$

2 3 9 0

9 3 2 0

from functools import cmp_to_key

---

2. Sort by color

=> Given int[ ] A,

=> sort the array.

$1 \leq N \leq 10^5$

$0 \leq A[i] \leq 2$

O(1)|2

approch 1

Count sort since the value are 3 digit only.

1. create array of 3 size.
2. update element fregency into it.
3. iterat 0 - 2 and update original array A to sort the array.

TC: O(N)

approach 2: Quick Sort

Taking pivit point and re-arranging element such that smallest value in LHS Large vale in RHS.

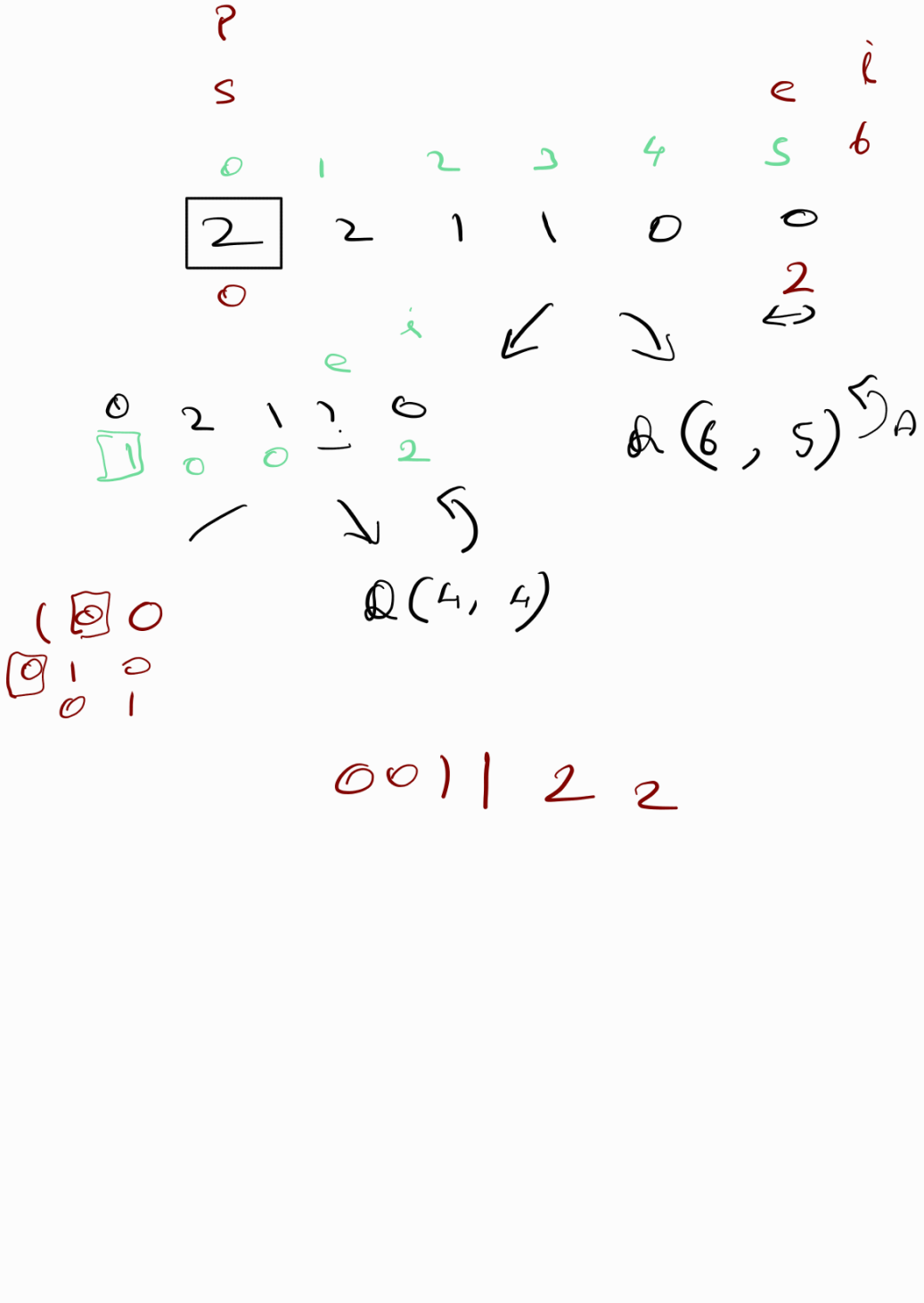
partition (A, s, e)

```
def quickSort(A, s, e)
    if s >= e:
        return A
    pivit = partitioing(A, s, e)
    mergeSort(A, s, pivit-1)
    mergeSort(A, pivit+1, e)
    return A

def partitioing(A, s, e)
    import random
    pindex = random.randint(s, e)
    p = A[pindex]
    swap(A, pIndex, s)
    i = s+1
    while (i <= e):
        if A[i] <= p:
            i = i+1
        elif A[e] > p:
```

```
        e -= 1
    else:
        swap(A, i, e)
swap(A, s, e)
return e
```

i = 1 2 3 p = 2
e = 2



0 0 1 | 2 2

③ ⇒ sort int [] A

⇒ based on its factors

⇒ if factors are same, sort it based on its value.

Ex.

6	8	9
4	4	3

ans

9	6	8

6	8	9
1	1	1
2	2	3
3	4	9
6	8	

Custom sorting: $O(\text{sqrt}(N))$

```
def factor(N):
    n = N**0.5, count = 0
    for i in range(1, n+1)
        if N%i == 0:
            count += 2
    if n*n == N:    # handling perfect square
        count -= 1
    return count

def customSort(x, y)
    a = factor(x)
    b = factor(y)
    if a < b:
        return -1
    elif a > b:
        return 1
    else: return -1
```

(4) Quick Sort:

(5) Partition given array

(6) Wave Array

=> Given int [] A

=> sort the array into wave-like array and return it

arrange the elements in such that

$a1 \geq a2 \leq a3 \geq a4 \leq a5 \ldots$

$8 \quad 7 \quad 7 \geq 9 \ldots$

Ex: $A = [1 \quad 2 \quad 3 \quad 4]$

$4 > 2 < 3 > 1$ and $2 > 1 < 4 > 3$

Note: return lexicographically smallest if more

ans.

1 2 3 4

2 1 4 3

1. Sort the array
2. swap adjecent elements.

```
A.sort()
for i in range(0, N, 2)        → 2 steps
    if (i+1) < N:
        A[i], A[i+1] = A[i+1], A[i]
return A
```

7. Closet Point to origin

int[][] A = [(x, y)]

int B = 4

user current location = (0, 0)

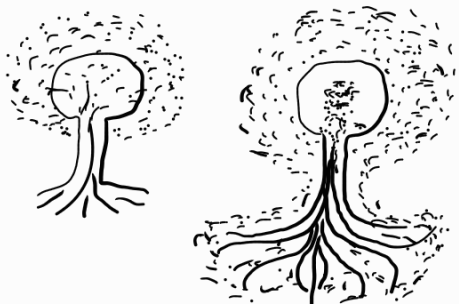
$P_1 (x_1, y_1)$
$P_2 (x_2, y_2)$

$$sqrt[(x_2 - x_1)^2 + (y_2 - y_1)^2]$$

A = [[1 3]
[-2 2]] B = 1

$$\Rightarrow \sqrt{(0-1)^2 + (0-3)^2} \Rightarrow \sqrt{(0+2)^2 + (0-2)^2}$$

$$\Rightarrow \sqrt{-1 - 9} \Rightarrow \sqrt{-10}$$



$2^2 = 2 \times 2 = 4$

$3^2 = 3 \times 3 = 9$

$-3^2 = -(3 \times 3) = -9$

$(-3)^2 = -3 \times -3 = 9$

```
def cust fun(x,y)
    a = ( x[0])^2 + ( x[1])^2
    b = ( y[0])^2 + ( y[1])^2
    if a<  :
        return -1
    elif a>b:
        return 1
    else return -1

c = sorted ( A, key = cmp_to_key(cust func))
return c[0:3]
```

TC: NlogN

SC: O(1)

Tens Digit Sorting

→ Given array of size N

→ Sort the array in increasing order

$A = [15 \quad 11 \quad 7 \quad 19]$

$15//10 = 1$

$10 \quad 10 \quad 0 \quad 10$

$[7 \quad 19 \quad 15 \quad 11]$

$15-1$
$11-1$
$7-0$
$19-1$

```
custfunc(x,y):
    a = (x//10)%10
    b = (y//10)%10
    if a > b:
        ↓ return 1
    elif a < b:
        ↓ return -1
    else: # if a = b
        if x > y:
            ↓ return -1
        elif x < y:
            ↓ return 1
        else:
            return -1
```

$(15//10)\%10$

$1 \%10$

1

$(22//10)\%10$

$2 \%10$

2

$99 \%10$

9

```
from functools import cmp_to_key
return sorted(A, key= cmp_to_key(custfun))
```

Sort 01

⇒ Given array A of 0s and 1s sort this array.

⇒

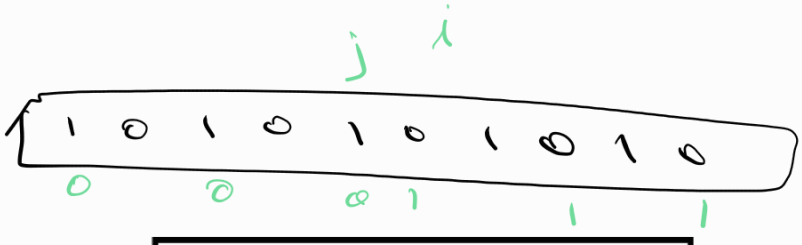
BF: using built in sorting $(n \log n)$

approach 1 Count sort

1. Create freqArr of size 2 as 0s and 1s only
2. update freqArr
3. update A from freqcy count.

O(N), O(N)
TC SC

approach 2 two pointer




```
while (i <= j):
    if A[i] == 0:
        ↓ i++
    elif A[j] == 1:
        ↓ j--
    else:
        ↓ swap(i,j)
```