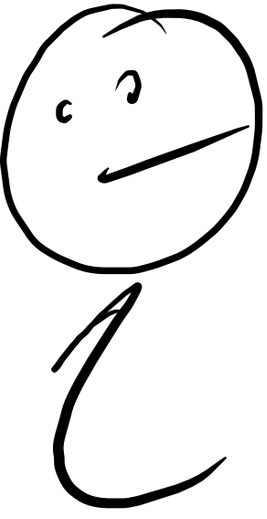
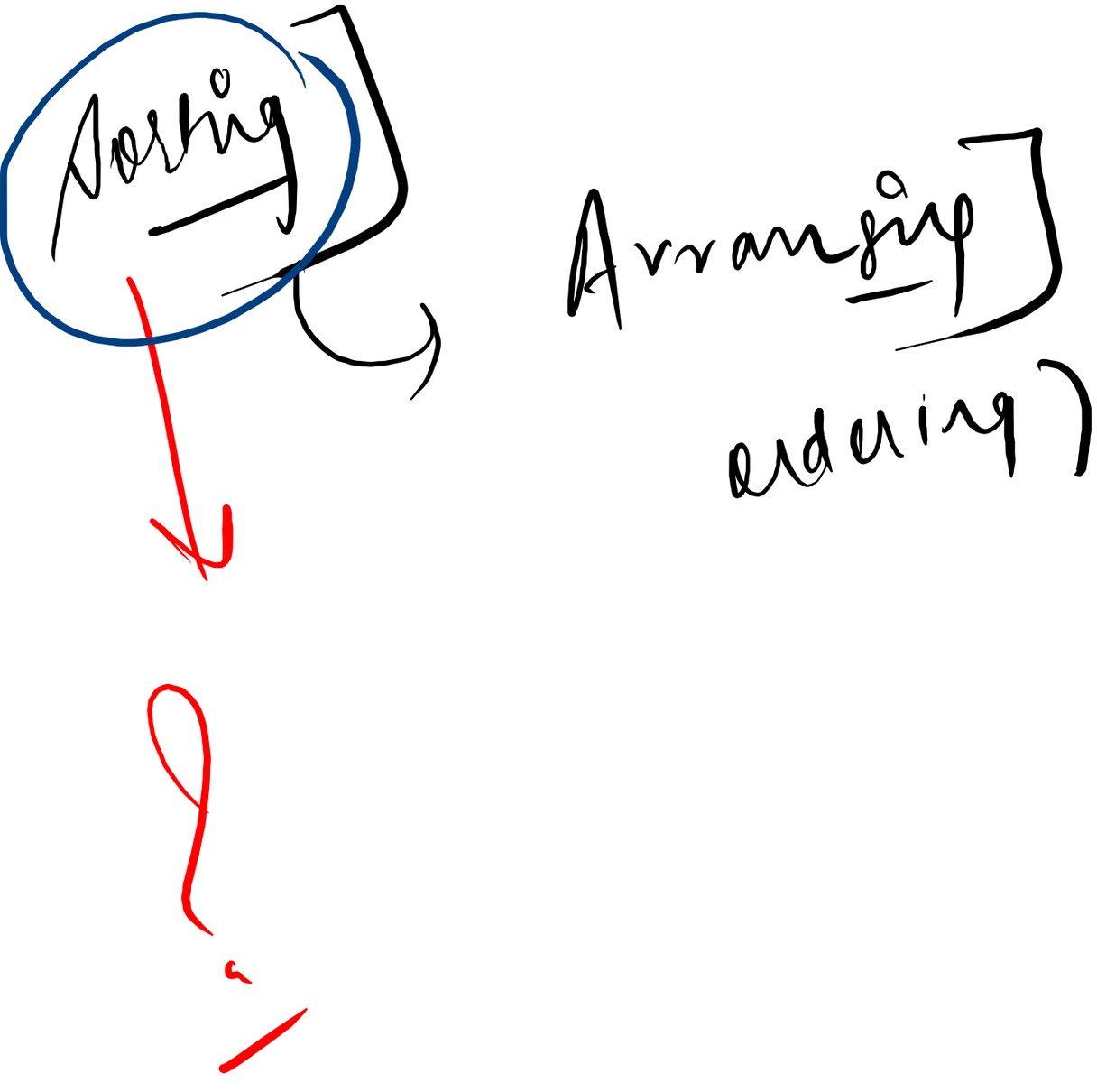
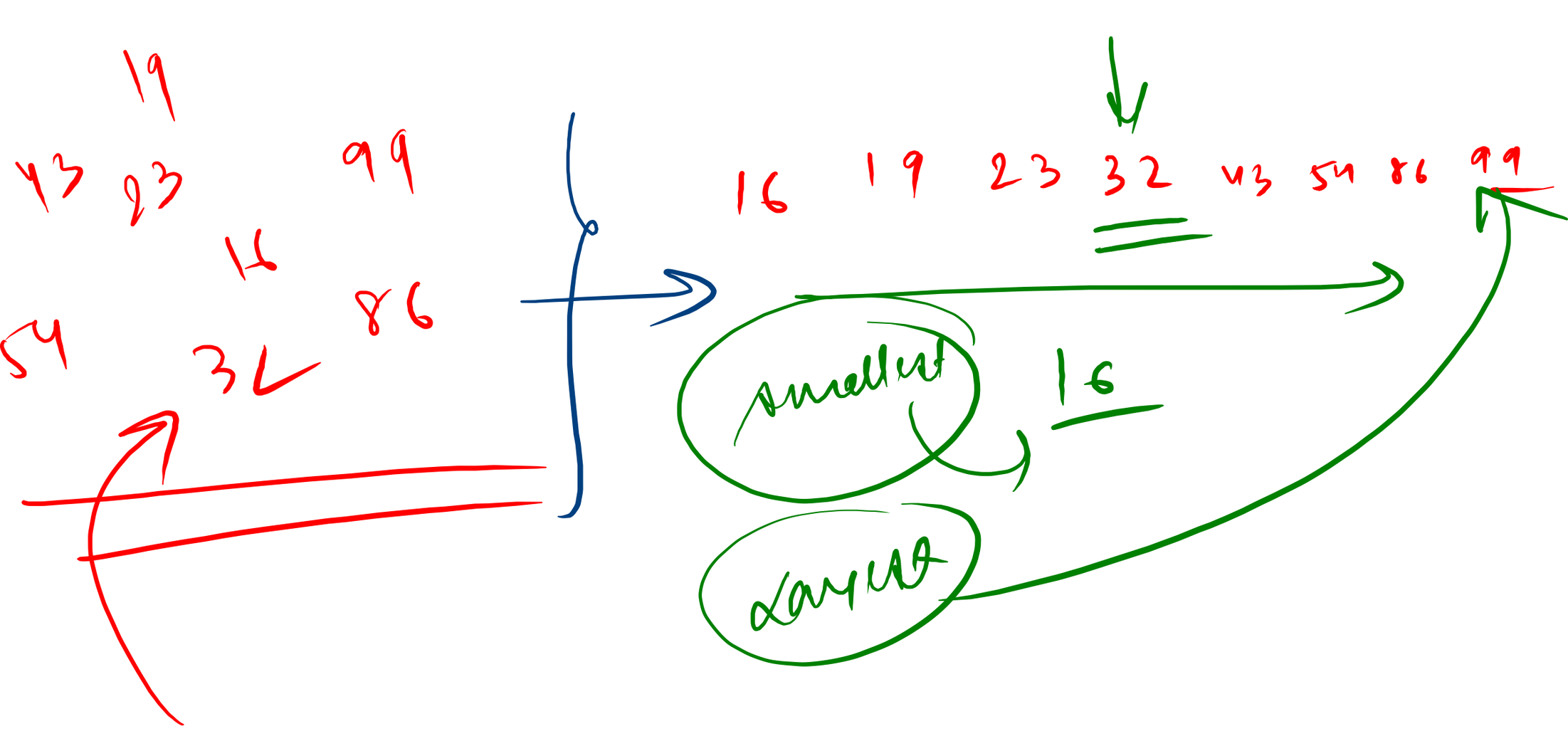
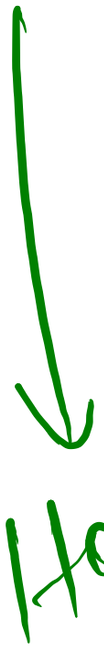



Sorting
Arranging
ordering

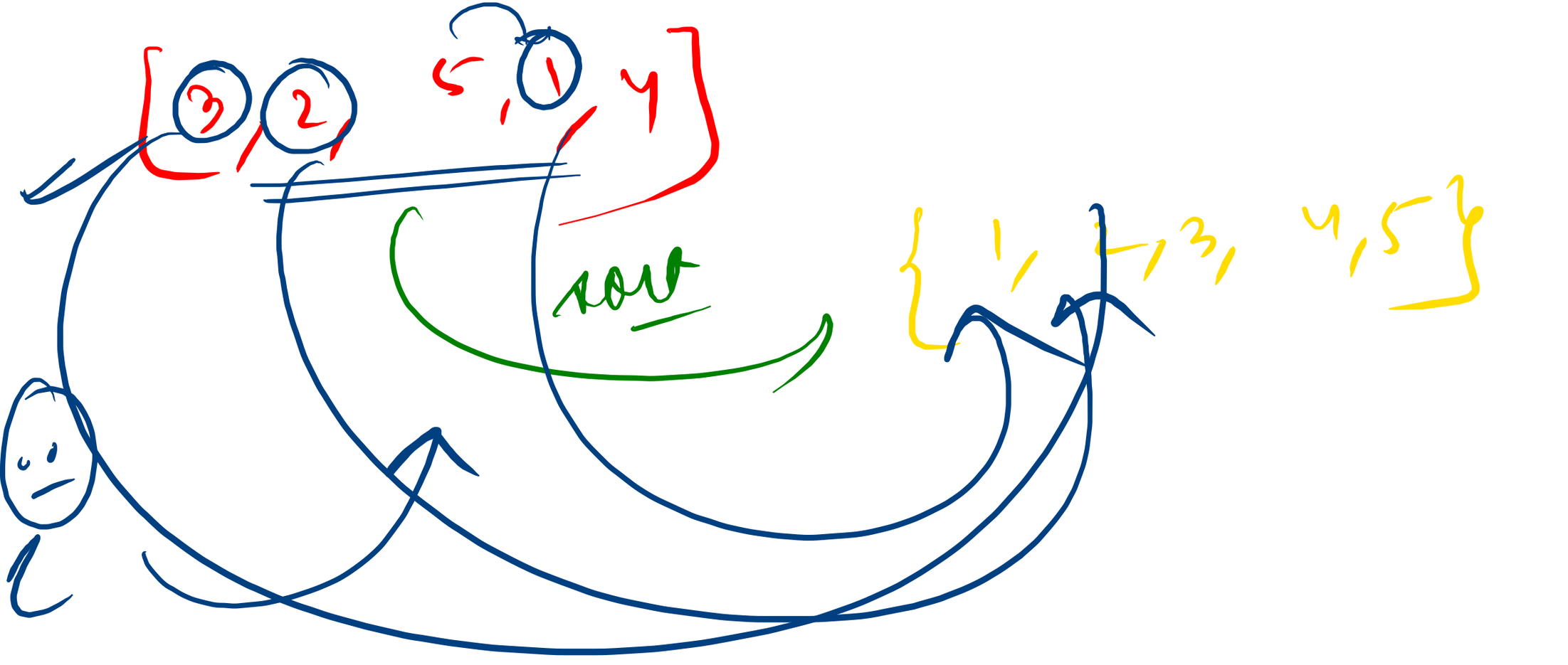

19
43 23
99
14
86
54
32
16 19 23 32 43 54 86 99
smaller
16
larger

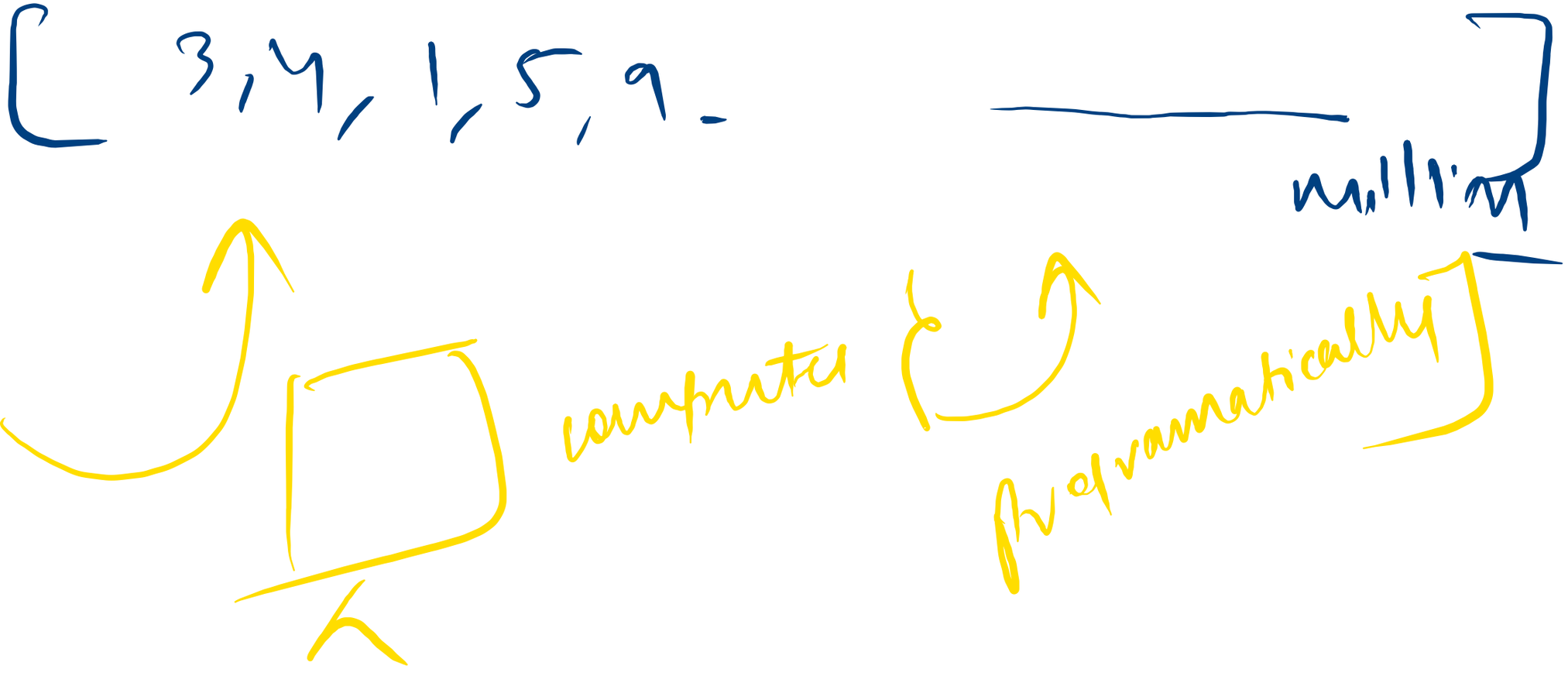
data
information

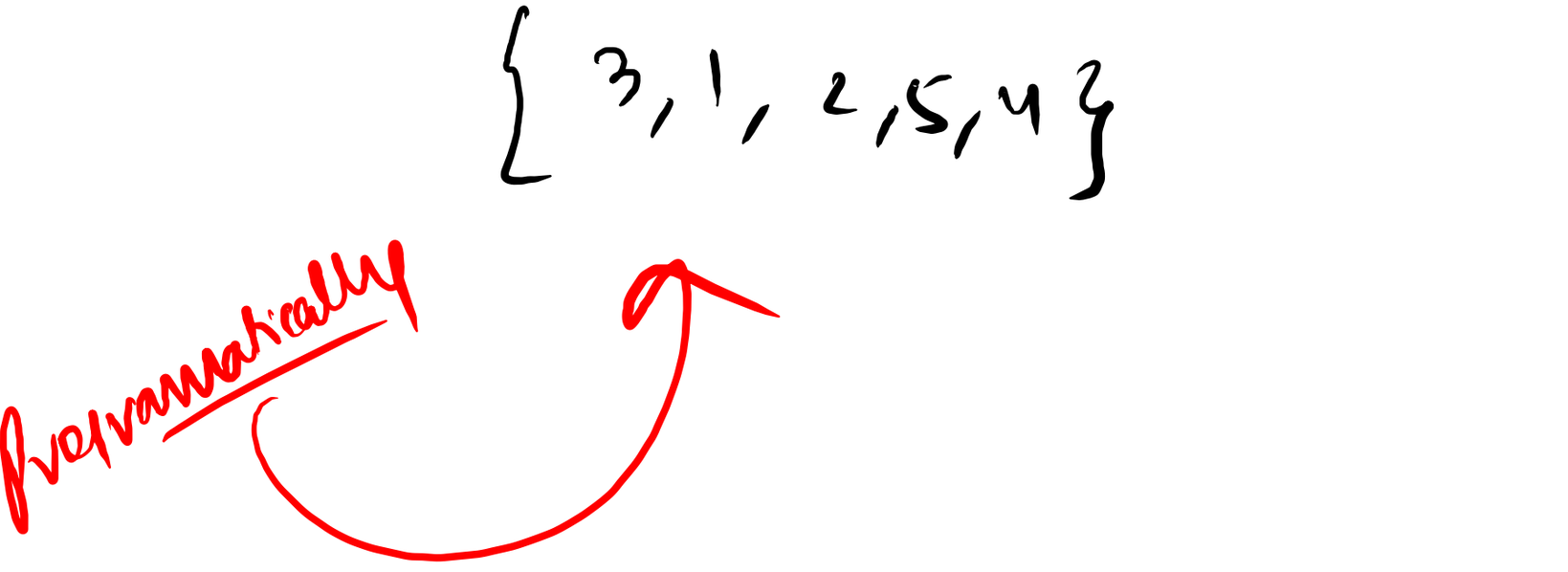
Sorting | what way | ?



How do we do it?


3
2
5
1
4
1010
{ 1, 2, 3, 4, 5 }


3, 4, 1, 5, 9 -
million
computer
programmatically


{ 3, 1, 2, 5, 4 }
Programmatically

Three basic sorting algorithms

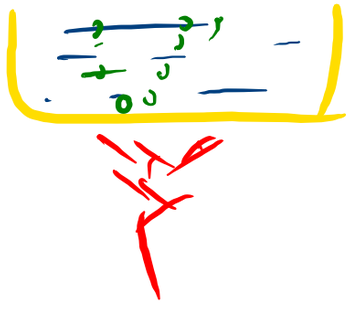
1. Bubble sort
2. selection sort
3. Insertion sort

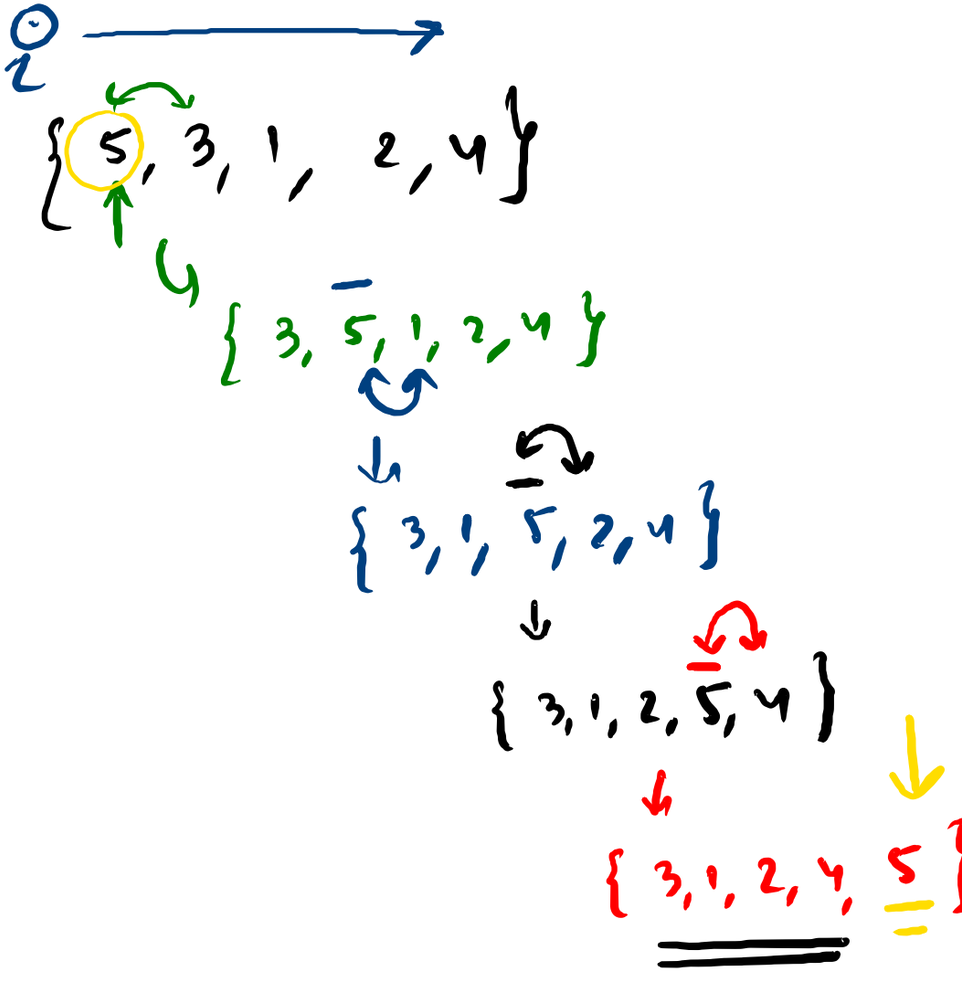
Bubble sort

Tweak → smart

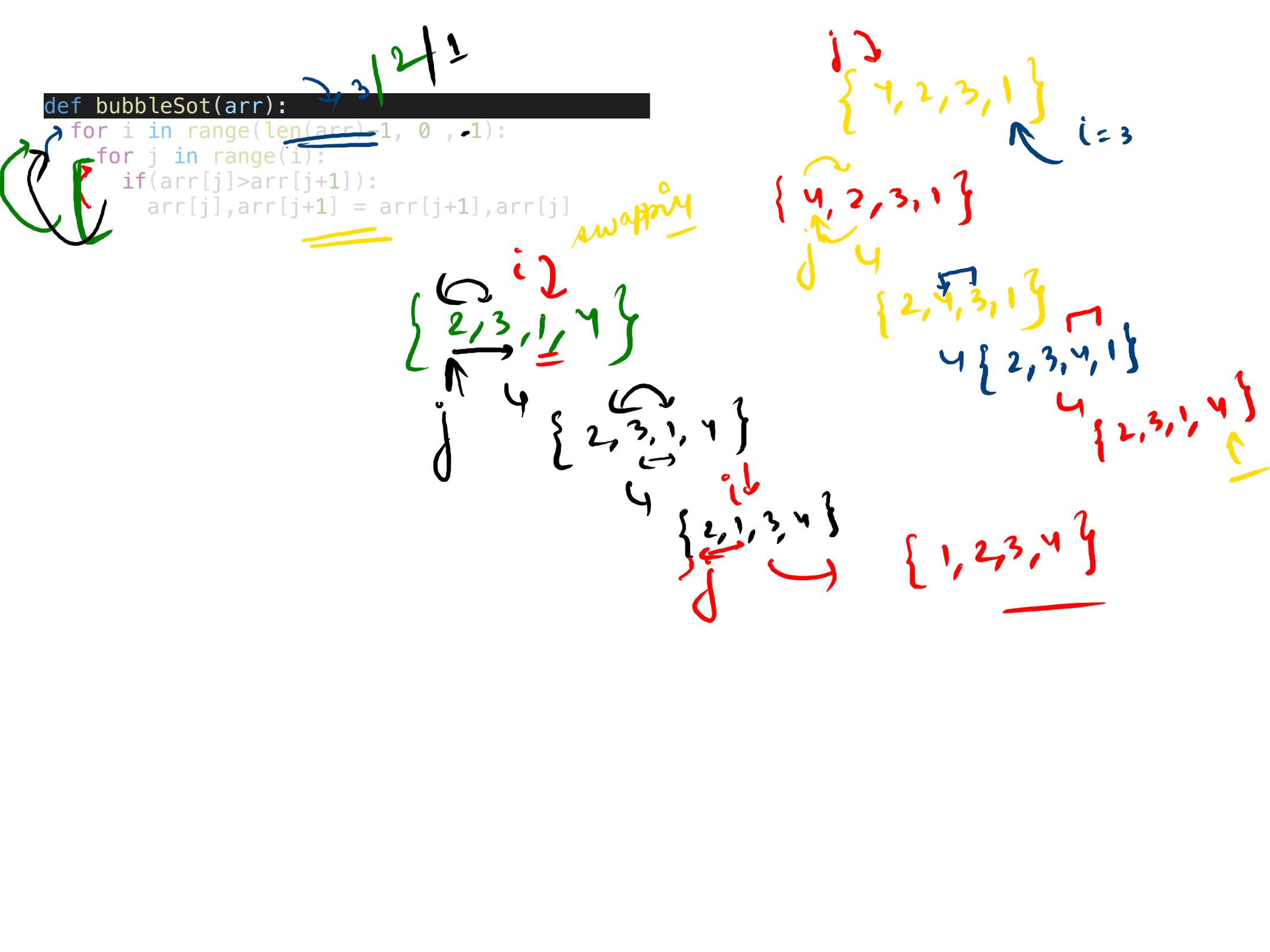
Bubble sort

Inspired

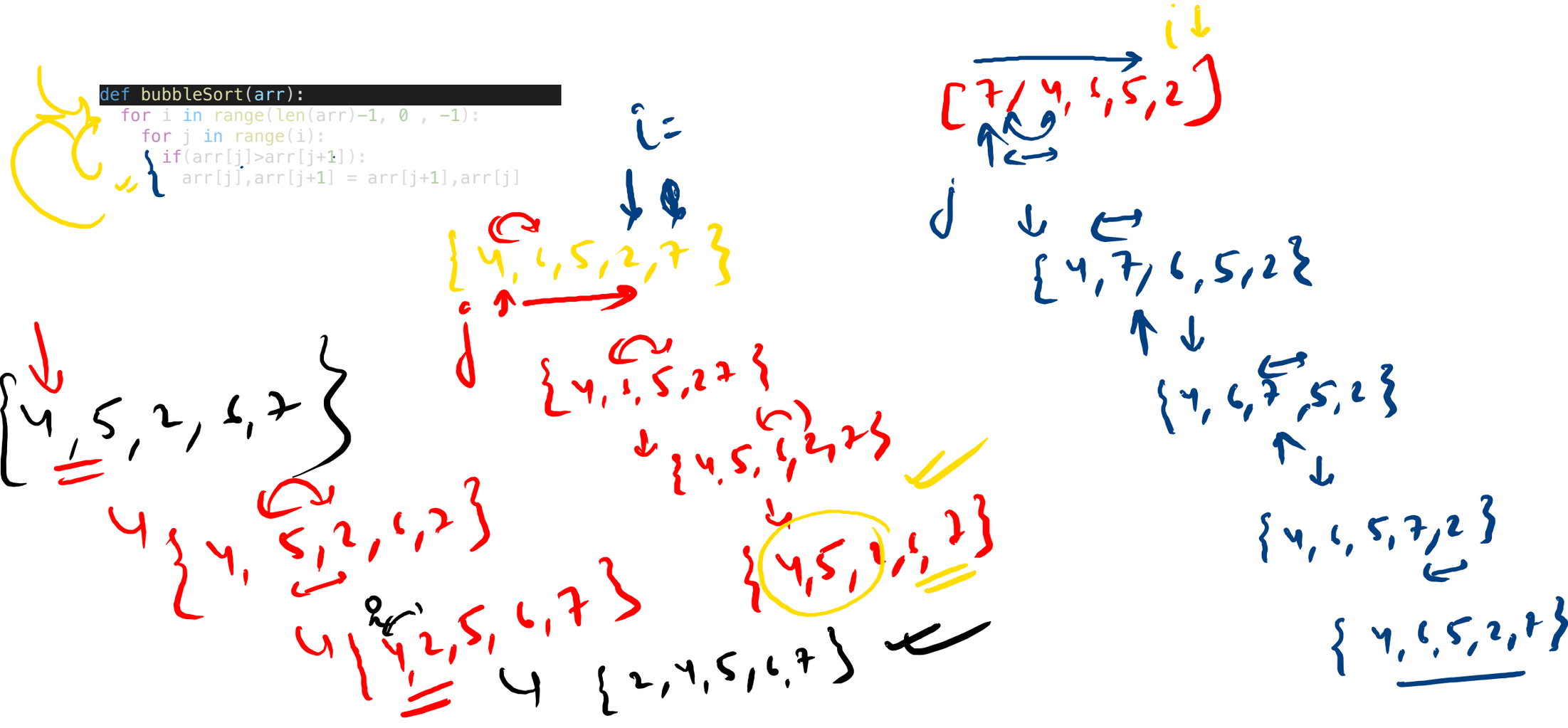




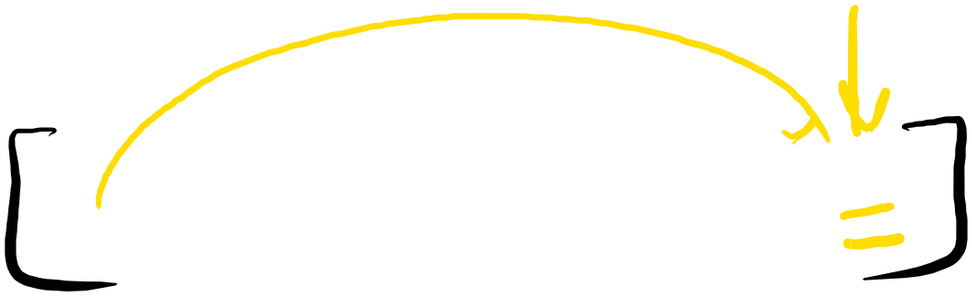
```
def bubbleSot(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```



```
def bubbleSort(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```



BubbleSort



```
def bubbleSort(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```

n

time complexity

$T_c = O(n^2)$

worst case?

{5, 4, 3, 2, 1}

worst case :

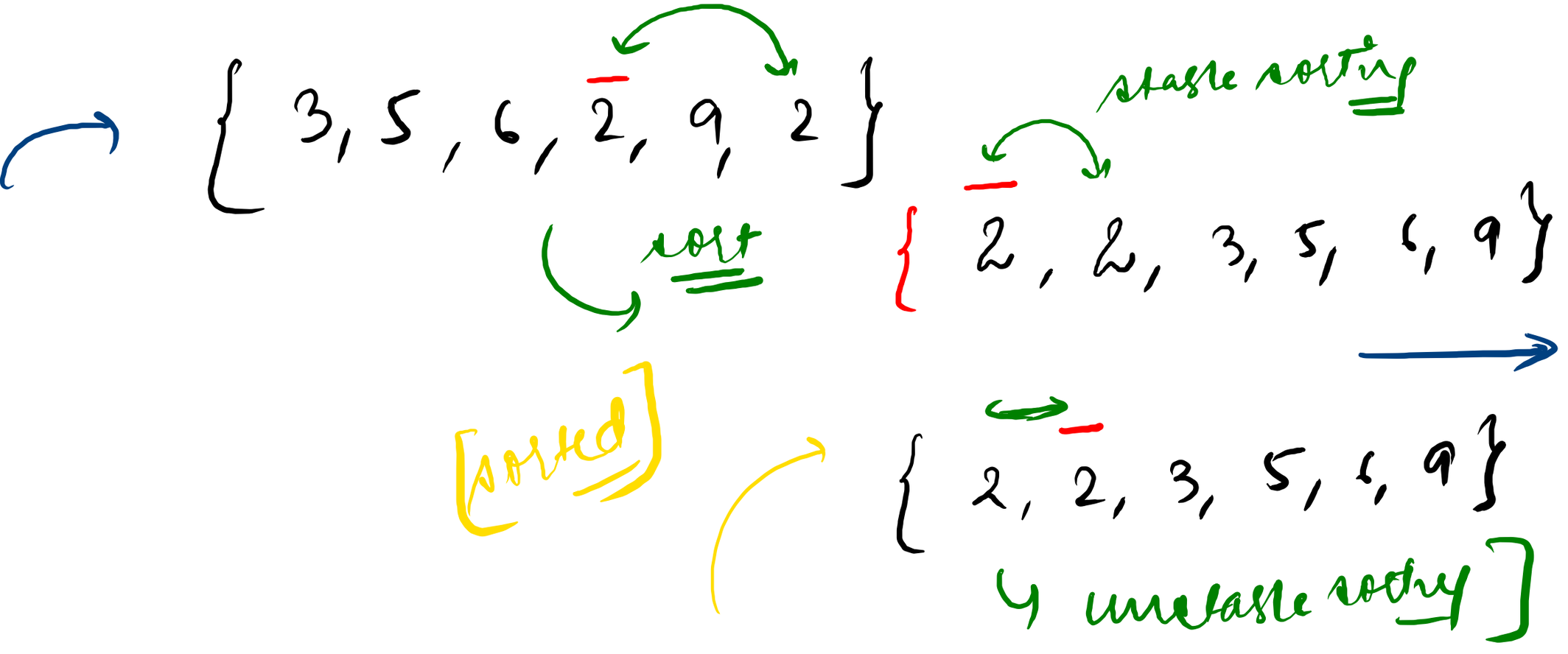
comparisons ? → n^2

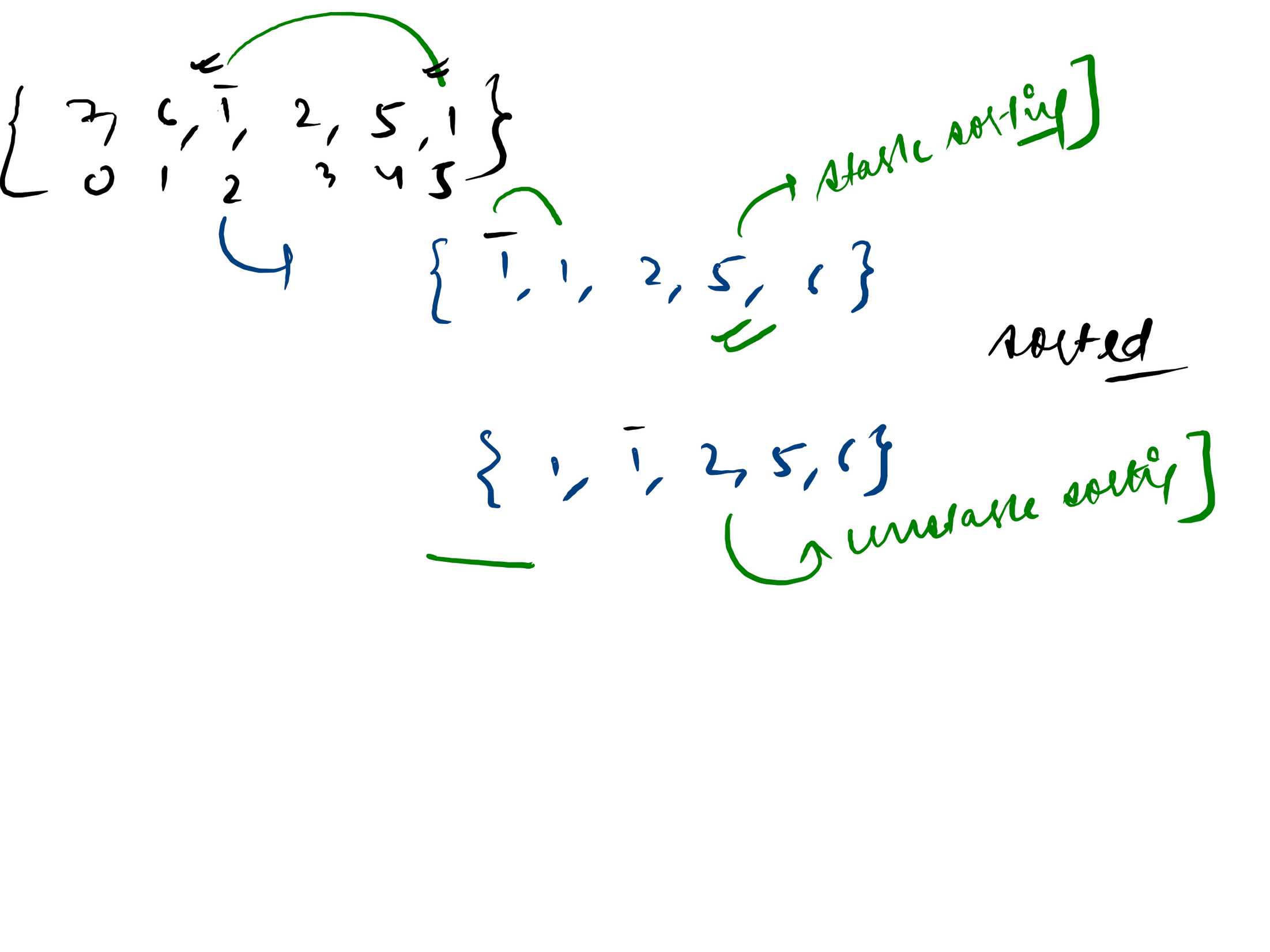
swapping ∝ n^2

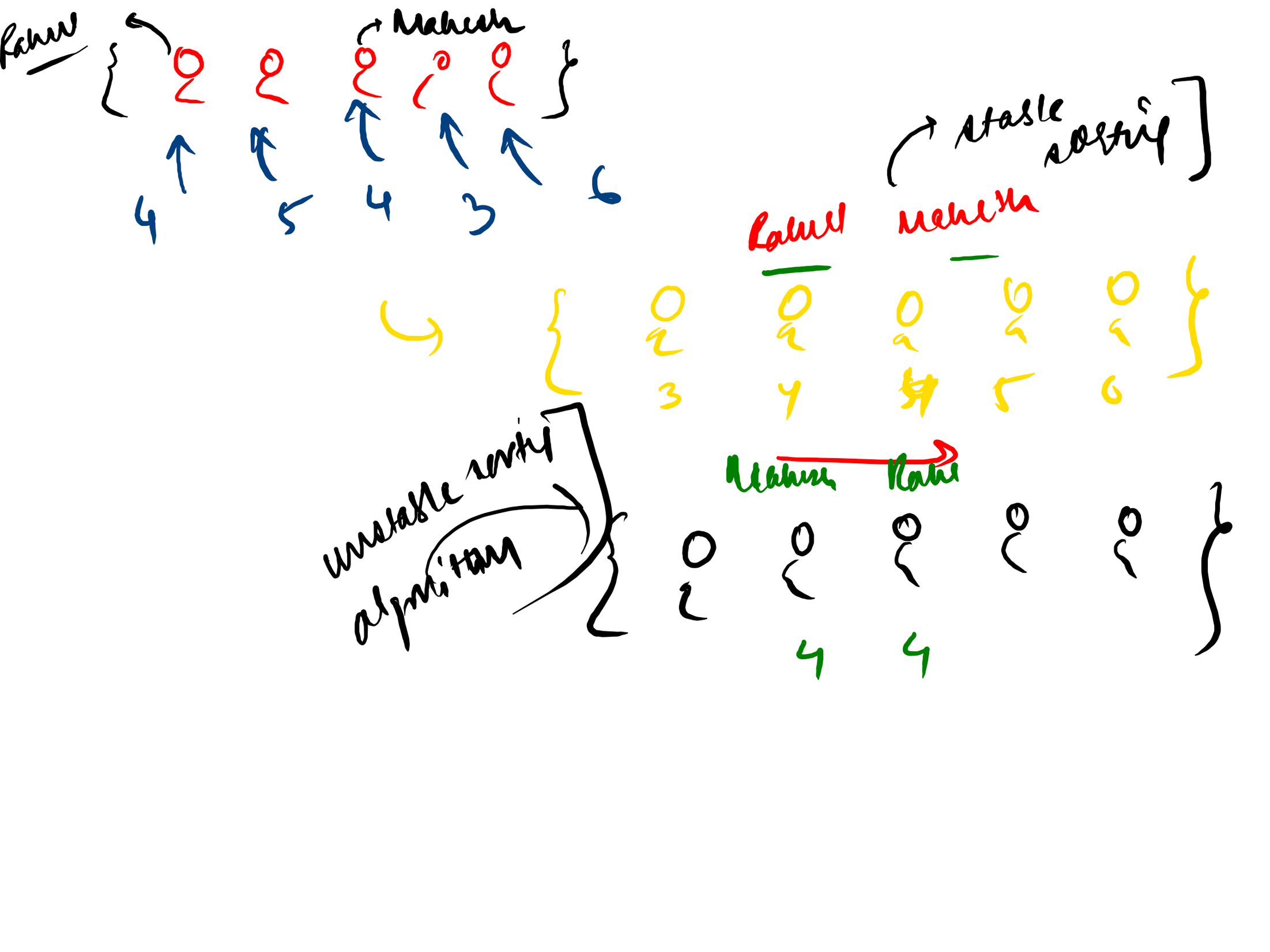
sorting }

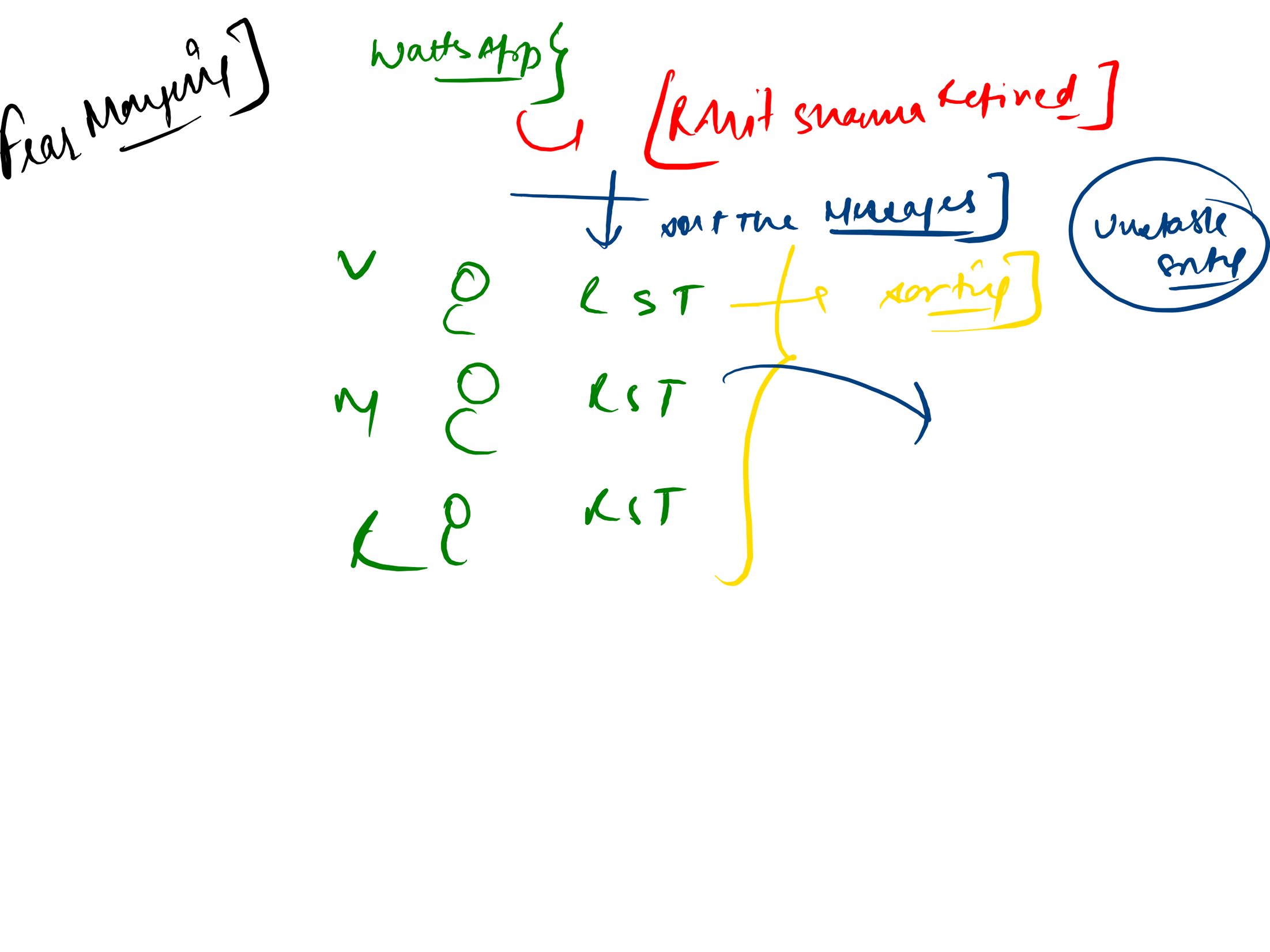
↑ stable sorting

unstable sorting]


3, 5, 6, 2, 9, 2
sort
stable sorting
2, 2, 3, 5, 6, 9
sorted
2, 2, 3, 5, 6, 9
↳ unstable sorting

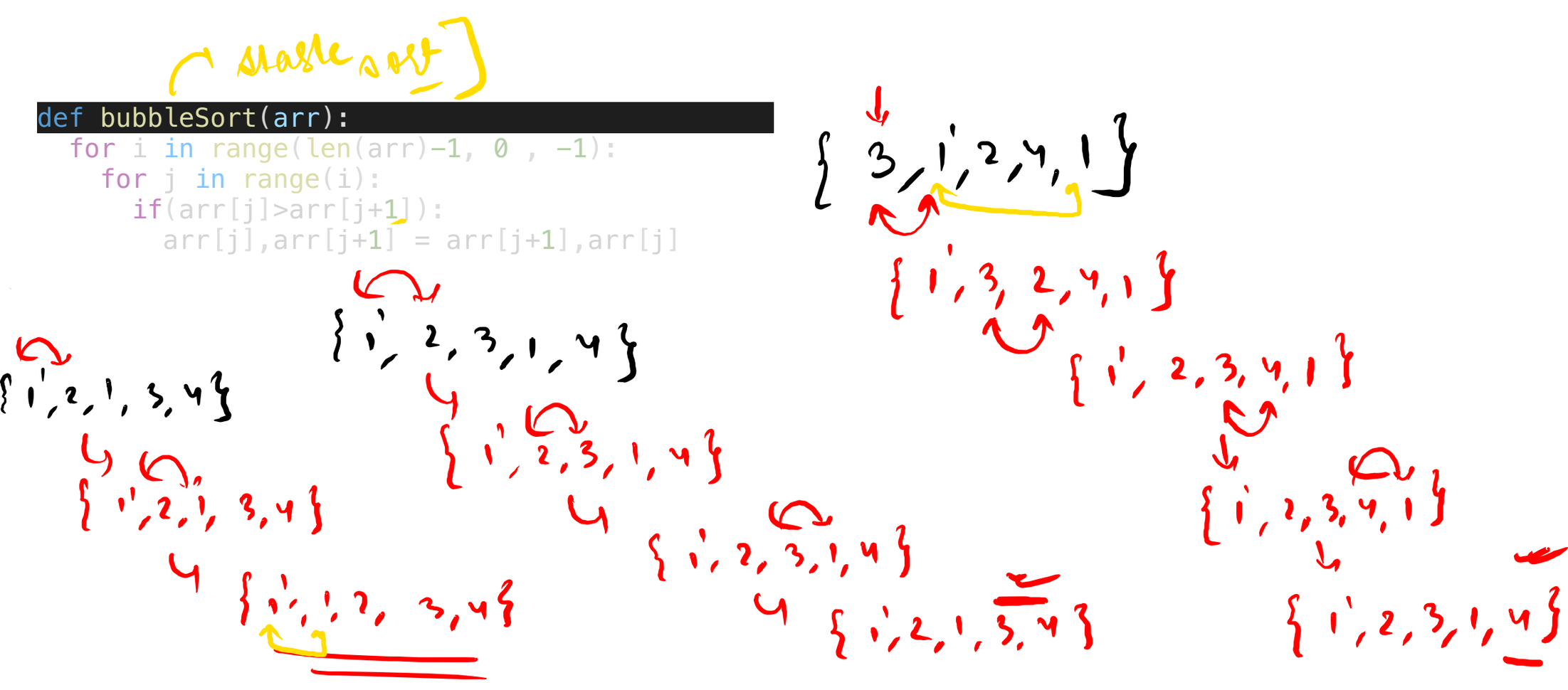

{ 7, 6, 1, 2, 5, 1 }
0 1 2 3 4 5
{ 1, 1, 2, 5, 6 }
static sorting
sorted
{ 1, 1, 2, 5, 6 }
insertion sorting


Pivot
Partition
4
5
4
3
6
stable sorting
Pivot
Partition
3
4
5
6
unstable sorting algorithm
Partition
Pivot
4
4


Fear Mayuri
WattsApp
Kohli sharma retired
sort the Messages
Unstable Entry
RST
RST
RST
sorting

stable sort

```
def bubbleSort(arr):
  for i in range(len(arr)-1, 0 , -1):
    for j in range(i):
      if(arr[j]>arr[j+1]):
        arr[j],arr[j+1] = arr[j+1],arr[j]
```



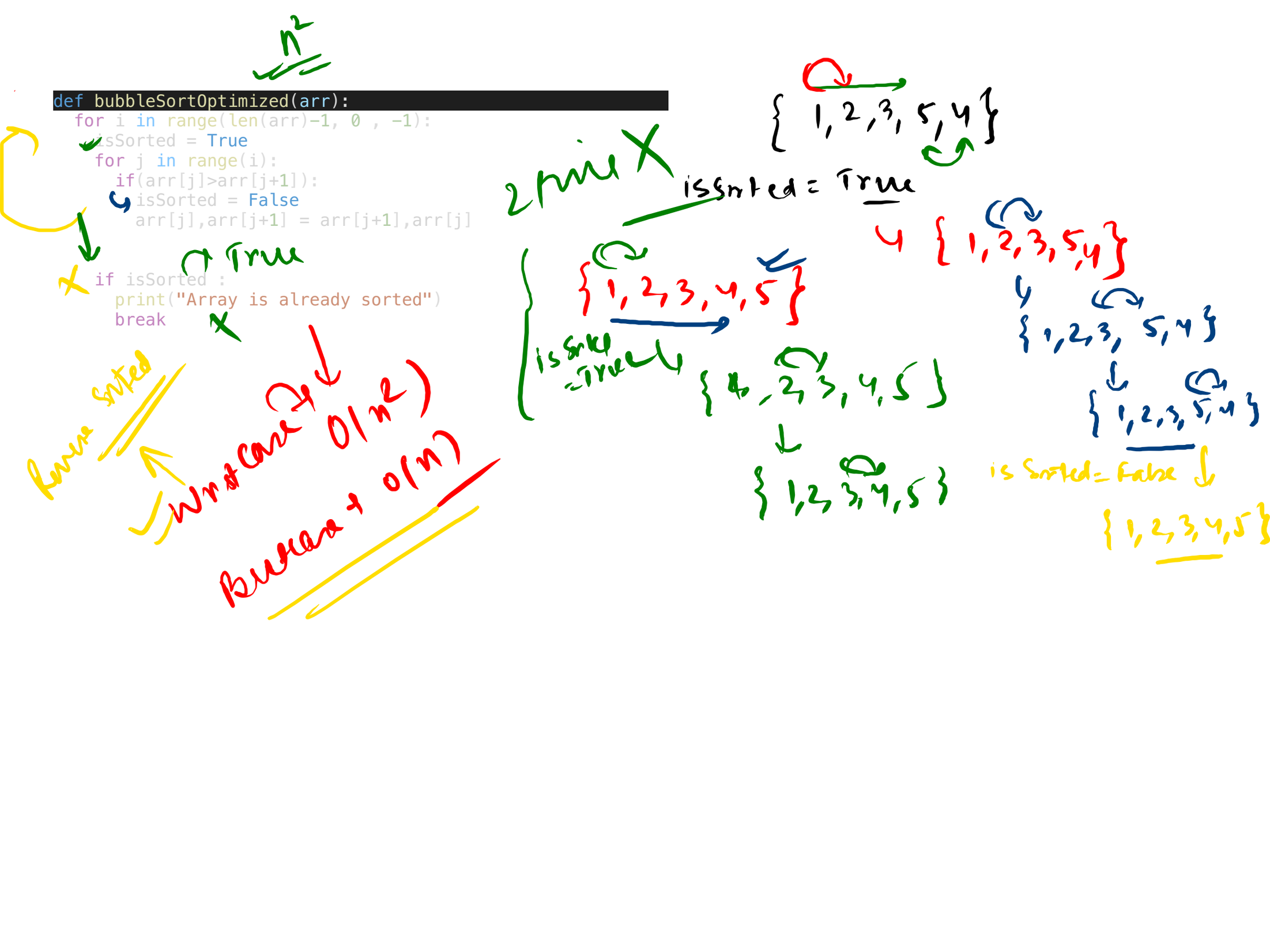
```python
def bubbleSort(arr):
  for i in range(len(arr)-1, 0 , -1):
    for j in range(i):
      if(arr[j]>arr[j+1]):
        arr[j],arr[j+1] = arr[j+1],arr[j]
```

$\rightarrow O(n^2)$

bubble sort → [1, 2, 3, 4, 5] → $O(n^2)$

n^2 ×

```
def bubbleSort(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```


n²
def bubbleSortOptimized(arr):
for i in range(len(arr)-1, 0 , -1):
isSorted = True
for j in range(i):
if(arr[j]>arr[j+1]):
isSorted = False
arr[j],arr[j+1] = arr[j+1],arr[j]
True
if isSorted :
print("Array is already sorted")
break
Worst Case → O(n2)
Best case → O(n)
Array Sorted
2 pass X
isSorted = True
{ 1, 2, 3, 5, 4 }
{ 1, 2, 3, 4, 5 }
isSorted = True
{ 1, 2, 3, 4, 5 }
{ 1, 2, 3, 4, 5 }
4 { 1, 2, 3, 5, 4 }
{ 1, 2, 3, 5, 4 }
{ 1, 2, 3, 5, 4 }
isSorted = False
{ 1, 2, 3, 4, 5 }

```python
def bubbleSort(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```

```python
def bubbleSortOptimized(arr):
    for i in range(len(arr)-1, 0 , -1):
        isSorted = True
        for j in range(i):
            if(arr[j]>arr[j+1]):
                isSorted = False
                arr[j],arr[j+1] = arr[j+1],arr[j]

        if isSorted :
            print("Array is already sorted")
            break
```

sorted → O(n)

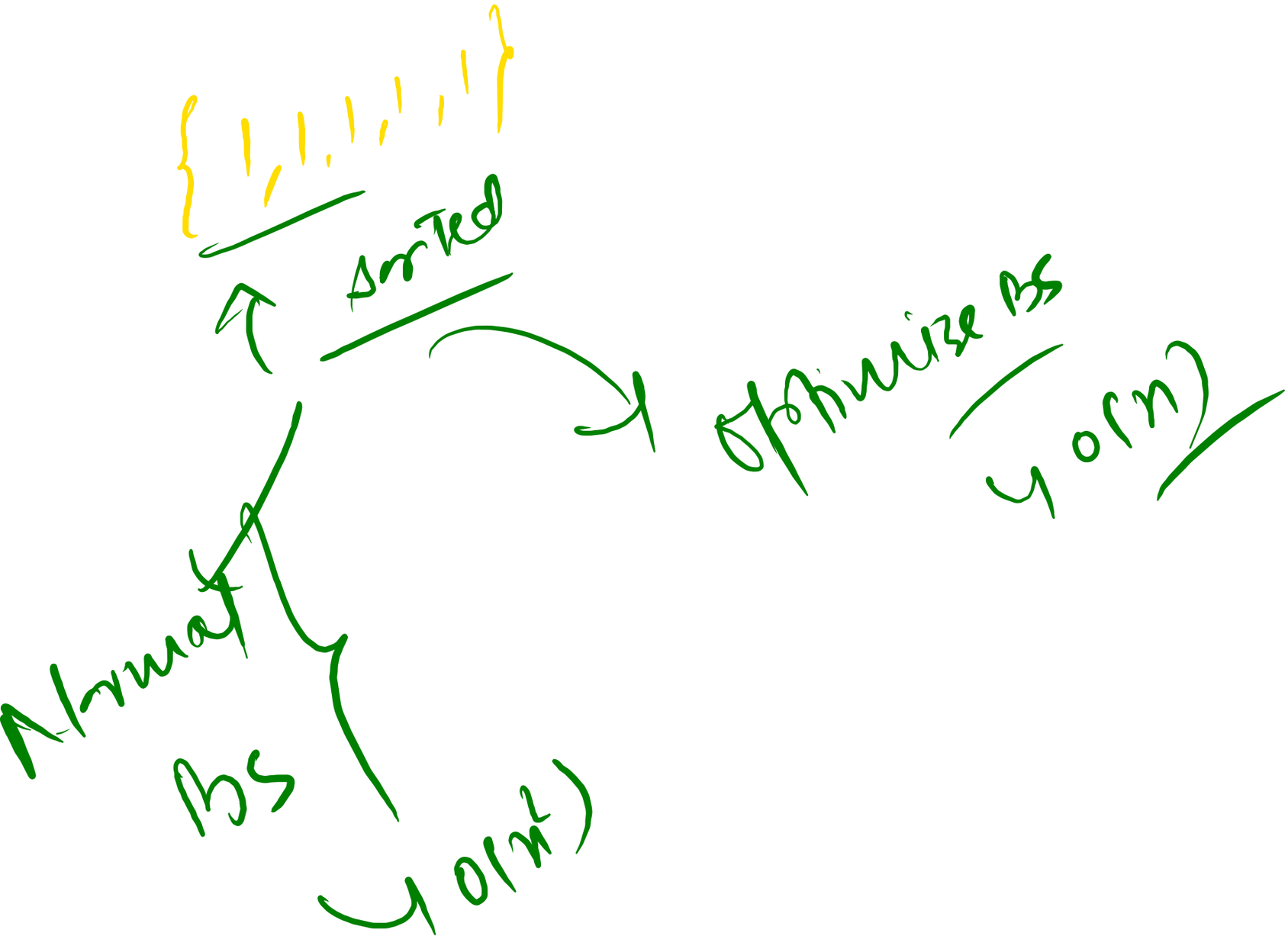
sorted → n

n time (n-1) → n^2

{1, 2, 3, 4, 5}

True

→ TC = O(n)


{ 1, 1, 1, 1, 1 }
sorted
Normal BS
O(n²)
Optimize BS
O(n)

```
for i in range(10)
    print(shit)
```

(n)

for i in range(n)

for j in range(i)

print('hello')

(n-1)

Total Nr of Operations

No of operations = No of Hello printed]

$(n-1) + (n-2) + (n-3) + \cdots \quad (*)$

$= \frac{n(n-1)}{2}$

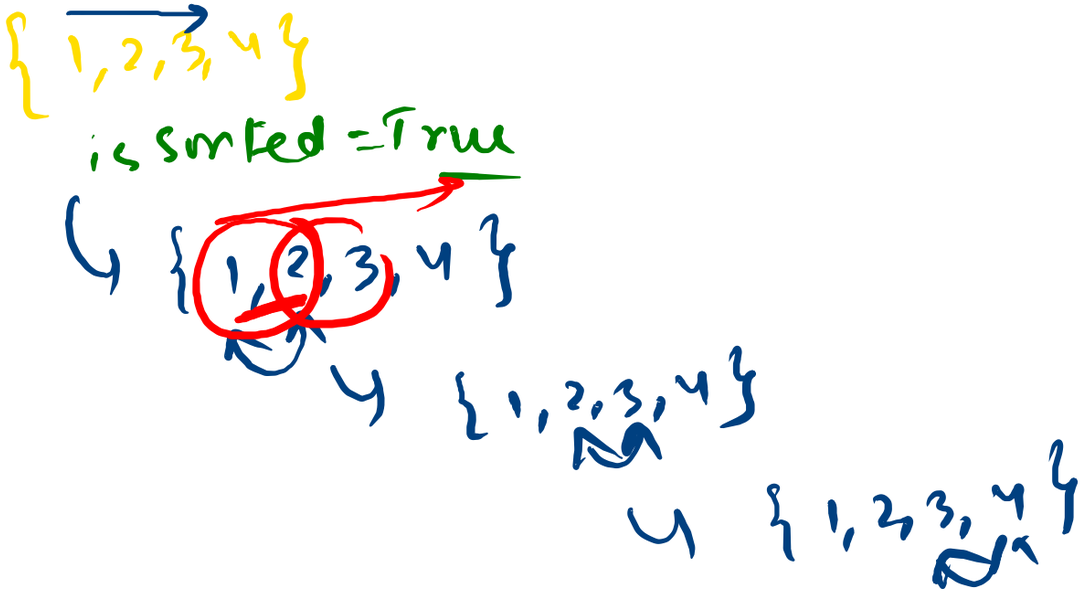
$= \frac{n^2 - n}{2}$

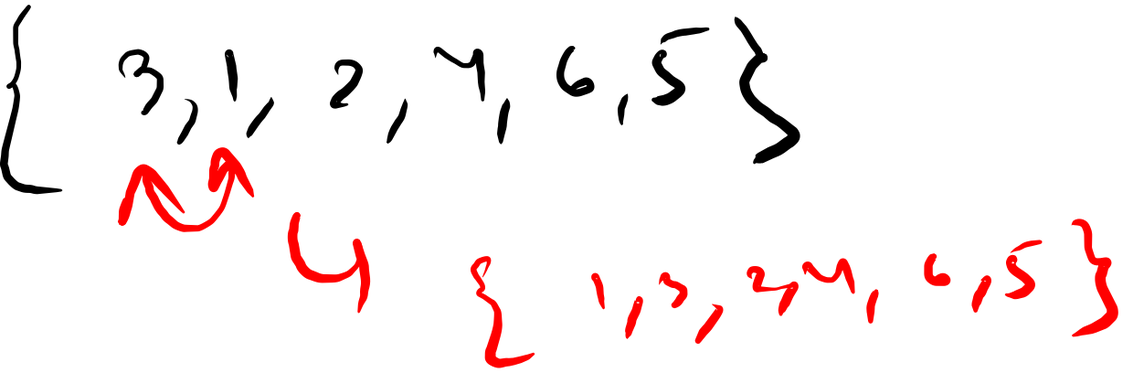
$T_c \in O(n^2)$

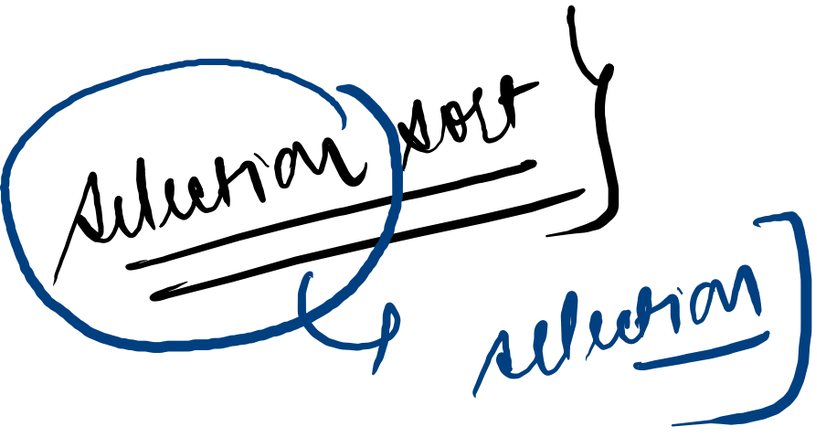
Time complexity

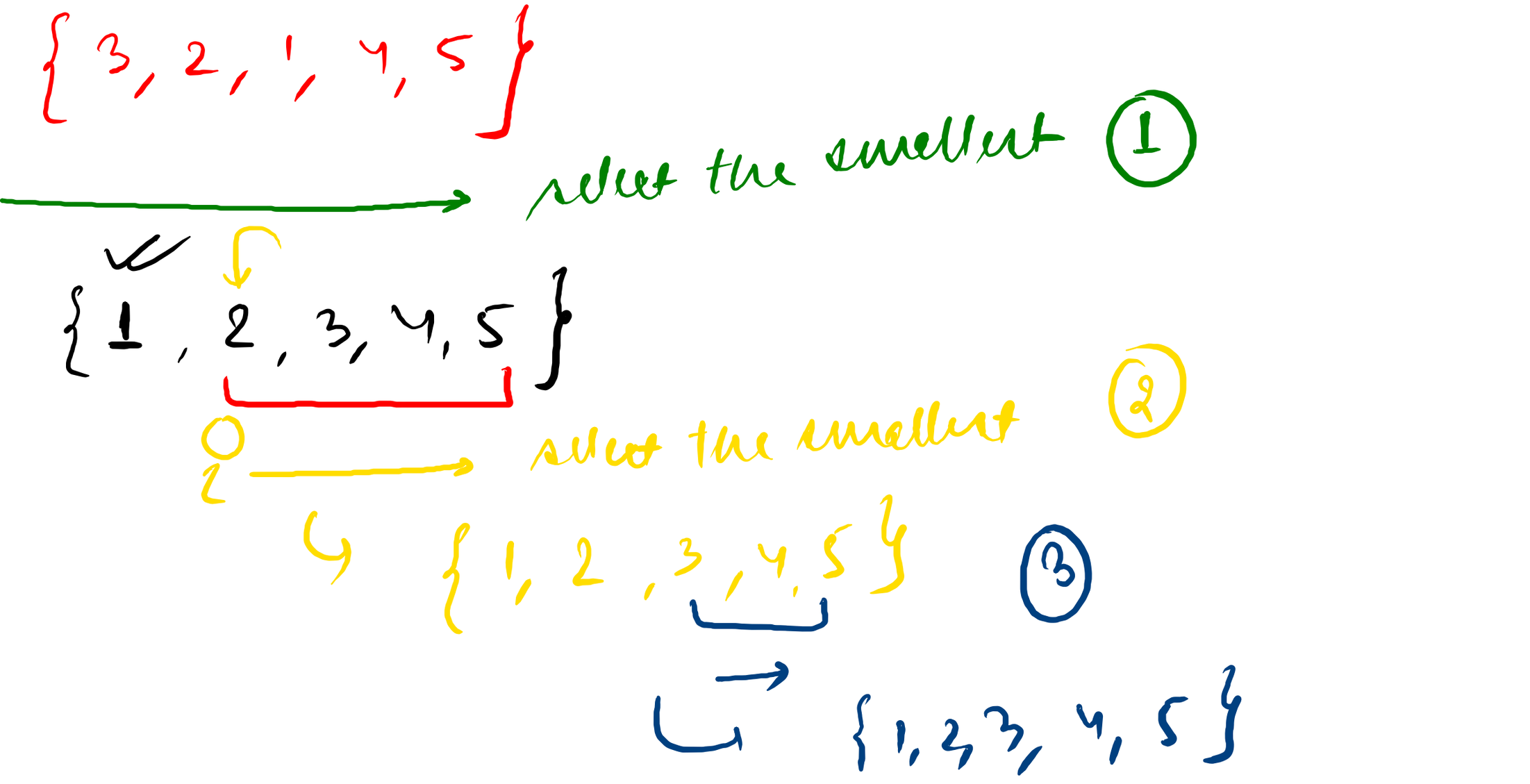
No of execution is proportional to the input size

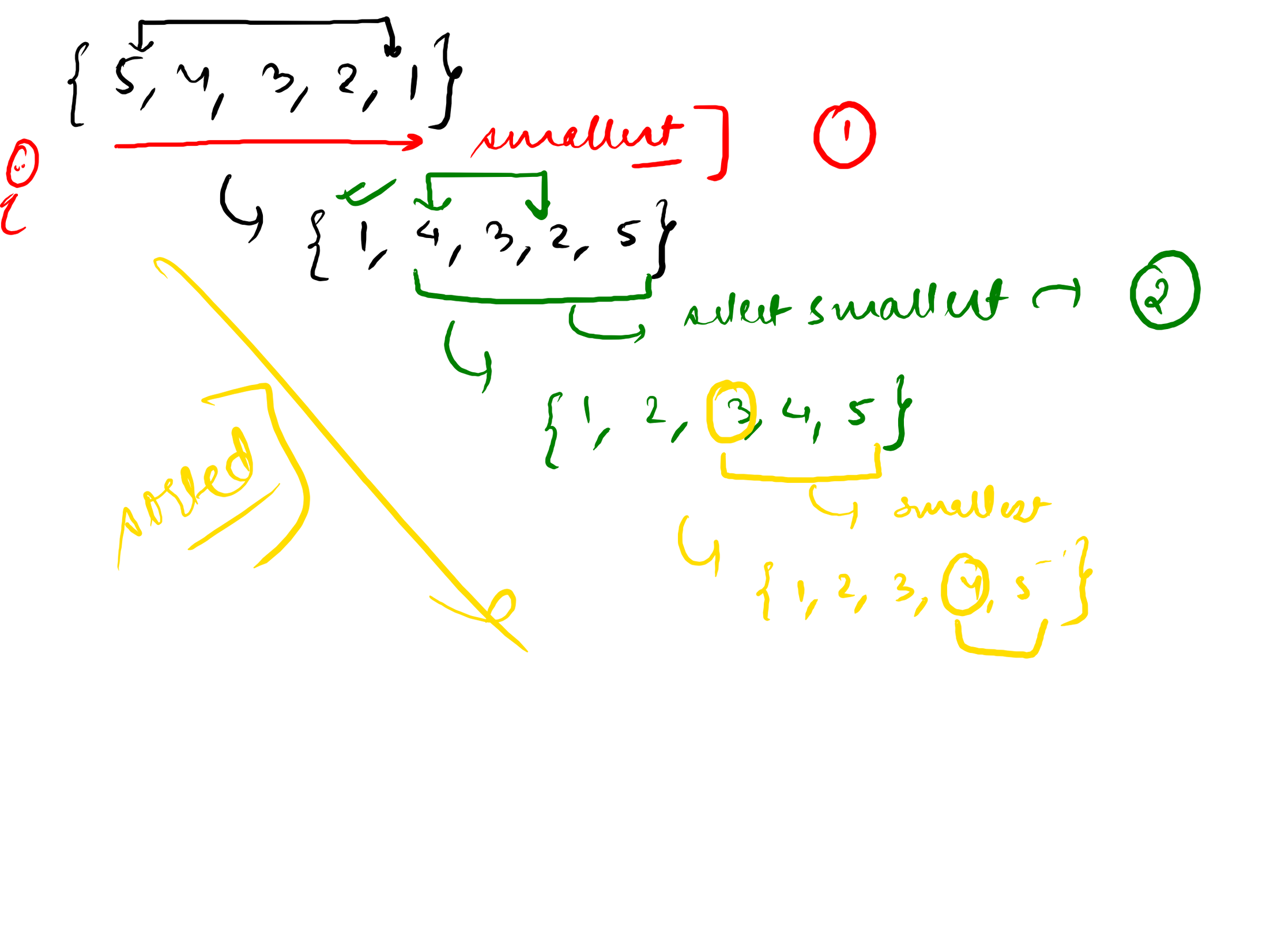
```python
def isSorted(arr):
  isSorted = True
  for i in range(len(arr)-1):
    if arr[i]>arr[i+1]:
      isSorted = False
      break
  if isSorted :
    print("Array is sorted")
  else :
    print("Array is not sorted")
```

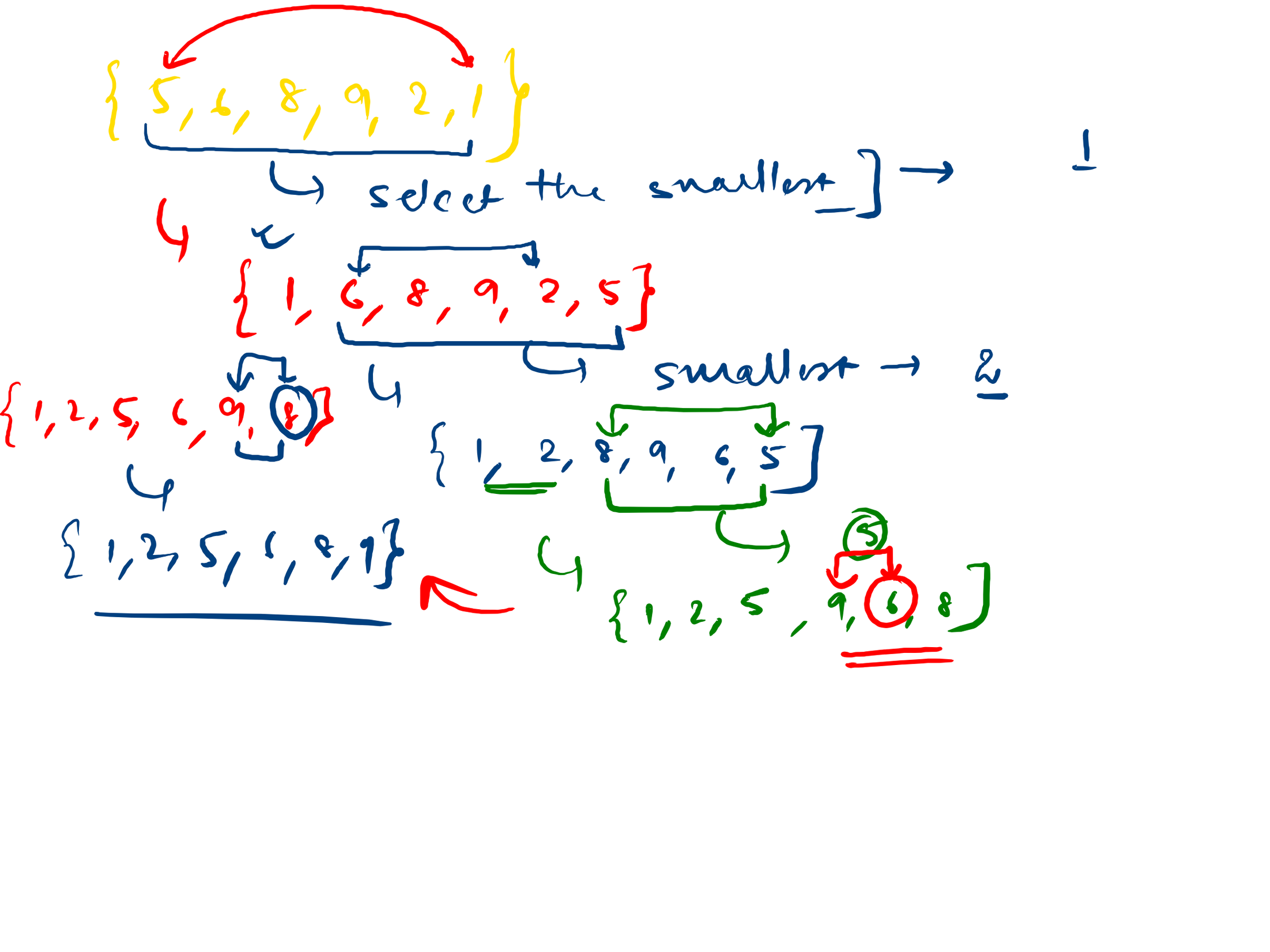


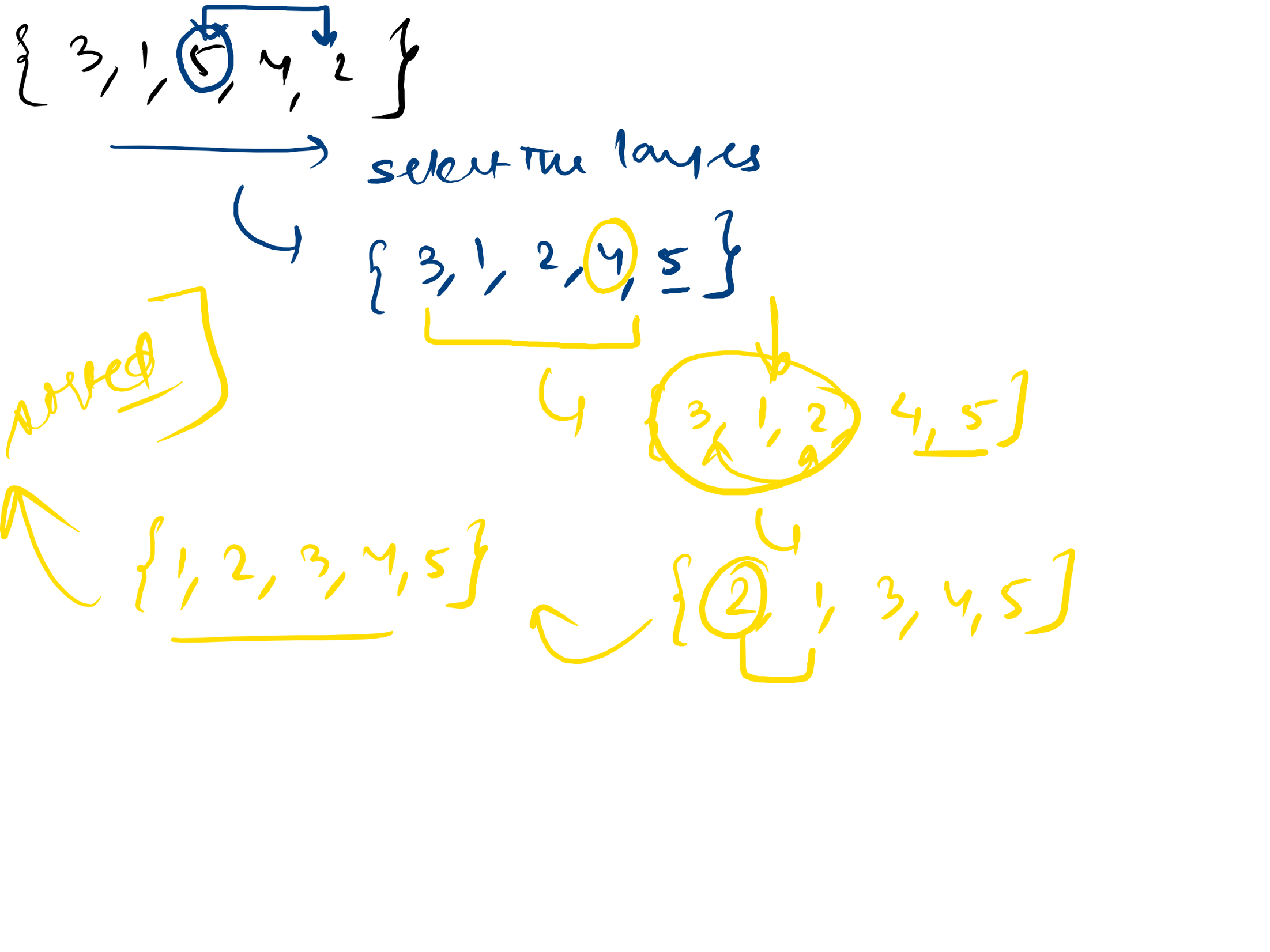

{ 3, 1, 2, 4, 6, 5 }
4 { 1, 3, 4, 6, 5 }

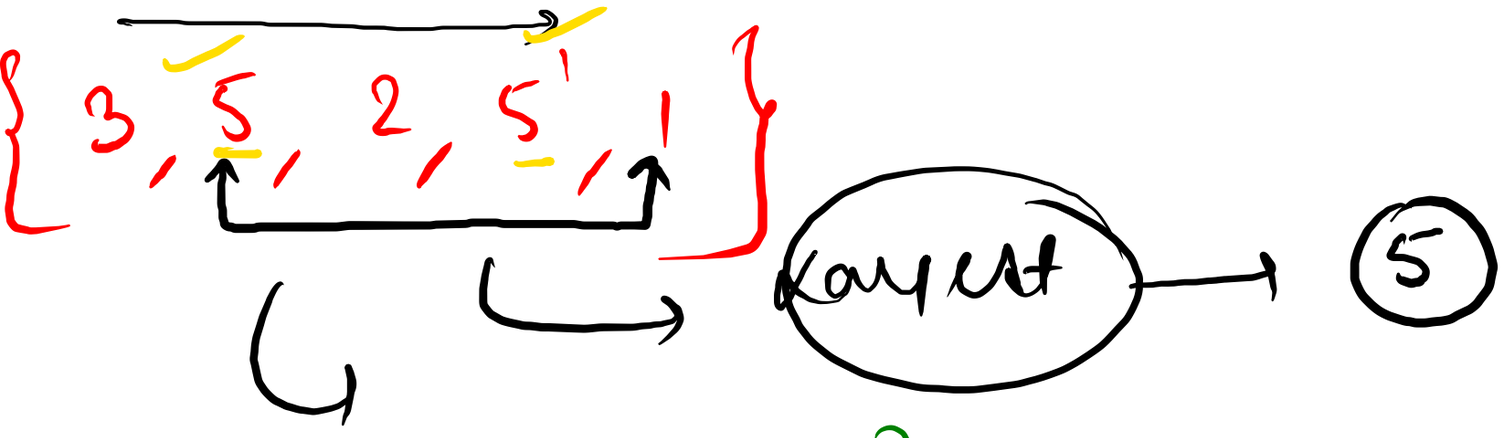

selection sort
selection


{3, 2, 1, 4, 5}
select the smallest
1
{1, 2, 3, 4, 5}
2
select the smallest
2
{1, 2, 3, 4, 5}
3
{1, 2, 3, 4, 5}


{ 5, 4, 3, 2, 1 }
smallest
1
{ 1, 4, 3, 2, 5 }
select smallest
2
{ 1, 2, 3, 4, 5 }
smaller
{ 1, 2, 3, 4, 5 }
sorted


{ 5, 4, 8, 9, 2, 1 }
select the smallest → 1
{ 1, 4, 8, 9, 2, 5 }
smallest → 2
{ 1, 2, 5, 4, 9, 8 }
{ 1, 2, 8, 9, 4, 5 }
{ 1, 2, 5, 4, 8, 9 }
{ 1, 2, 5, 9, 4, 8 }


{ 3, 1, 5, 4, 2 }
select the largest
{ 3, 1, 2, 4, 5 }
3, 1, 2
4, 5
sorted
{1, 2, 3, 4, 5}
{2, 3, 4, 5}

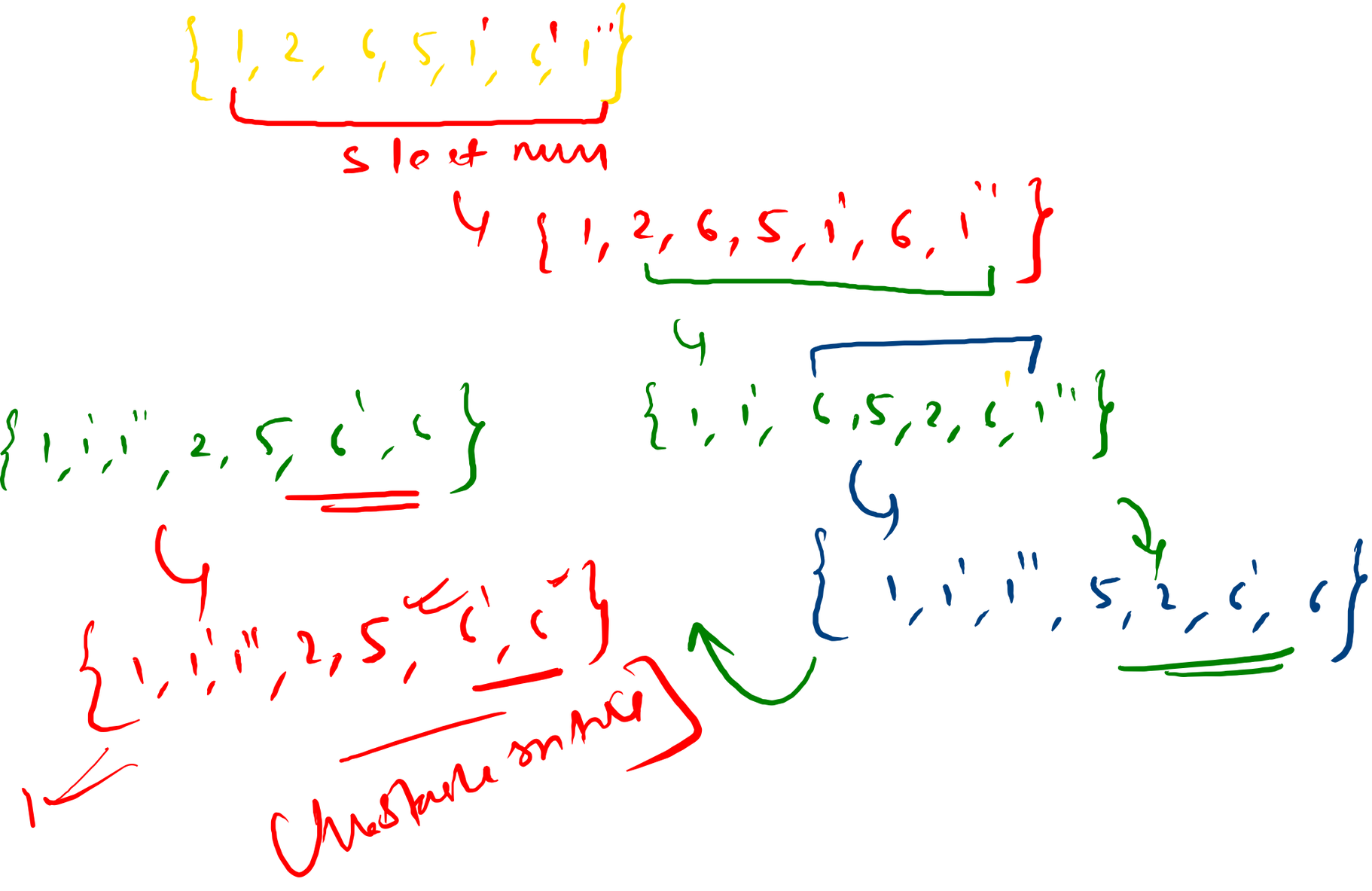


{ 3, 1, 2, 5', 5 }

{ 1, 2, 3, 5', 5 }

{ 3, 1, 2, 5', 5 }

{ 2, 1, 3, 5', 5 }


{1, 2, 6, 5, 1, 6, 1}
select num
{1, 2, 6, 5, 1, 6, 1}
{1, 1, 6, 5, 2, 6, 1}
{1, 1, 1, 2, 5, 6, 6}
{1, 1, 1, 5, 2, 6, 6}
{1, 1, 1, 2, 5, 6, 6}
1
Backtrack on this

```python
#Selection Sorting
def selectionSort(arr):
  for i in range(len(arr)):
    min =i
    for j in range(i+1,len(arr)):
      if arr[j] <arr[min]:
        min =j
    #Find the smallest values
    arr[i],arr[min] = arr[min],arr[i]
```

```python
#Selection Sorting
def selectionSort(arr):
  for i in range(len(arr)):
    min =i
    for j in range(i+1,len(arr)):
      if arr[j] <arr[min]:
        min =j
    #Find the smallest values
    arr[i],arr[min] = arr[min],arr[i]
```

Unstable

```python
def bubbleSort(arr):
  for i in range(len(arr)-1, 0 , -1):
    for j in range(i):
      if(arr[j]>arr[j+1]):
        arr[j],arr[j+1] = arr[j+1],arr[j]
```

Stable

Comparisons → $O(n^2)$ → $\propto n^2$

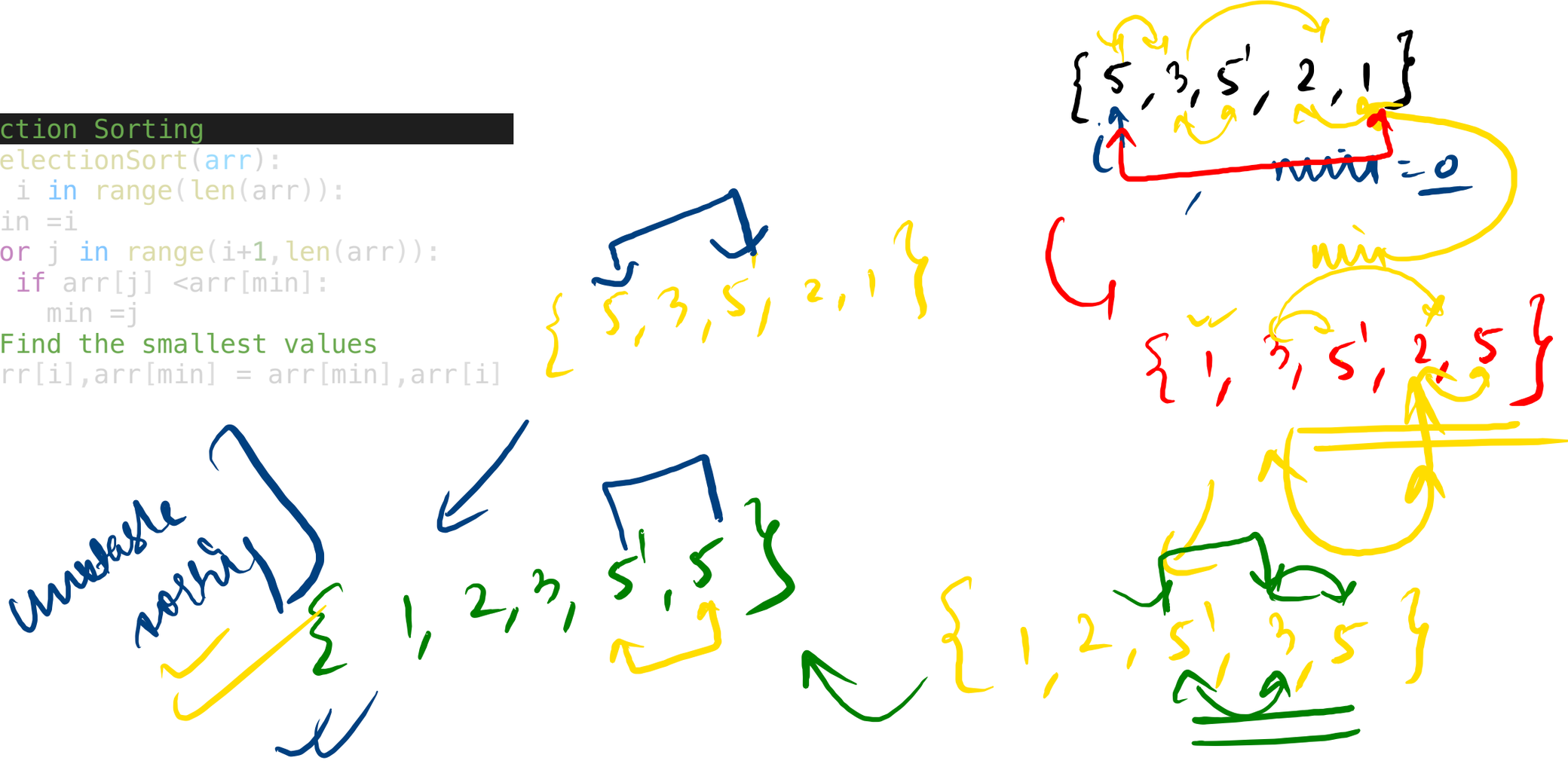
Swaps → n

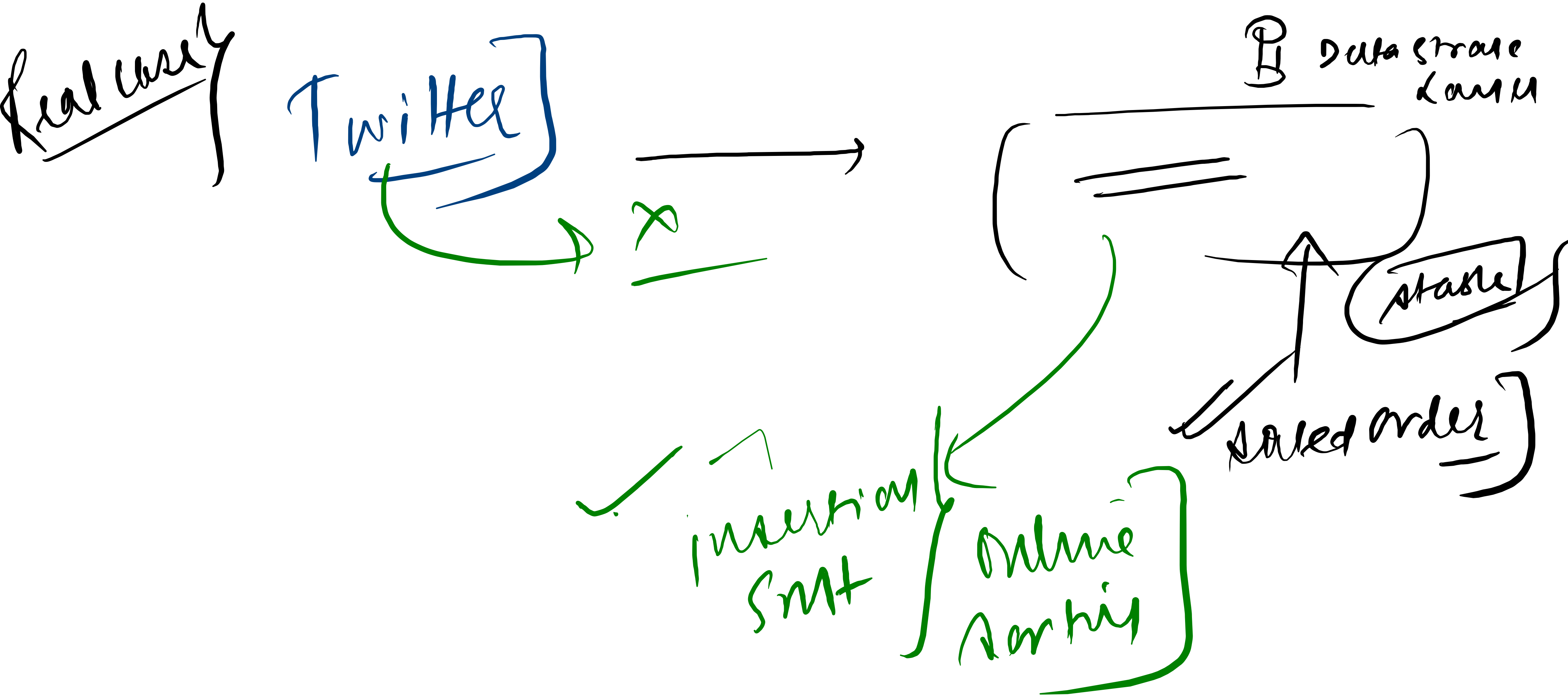
Better performance

$\propto n^2$

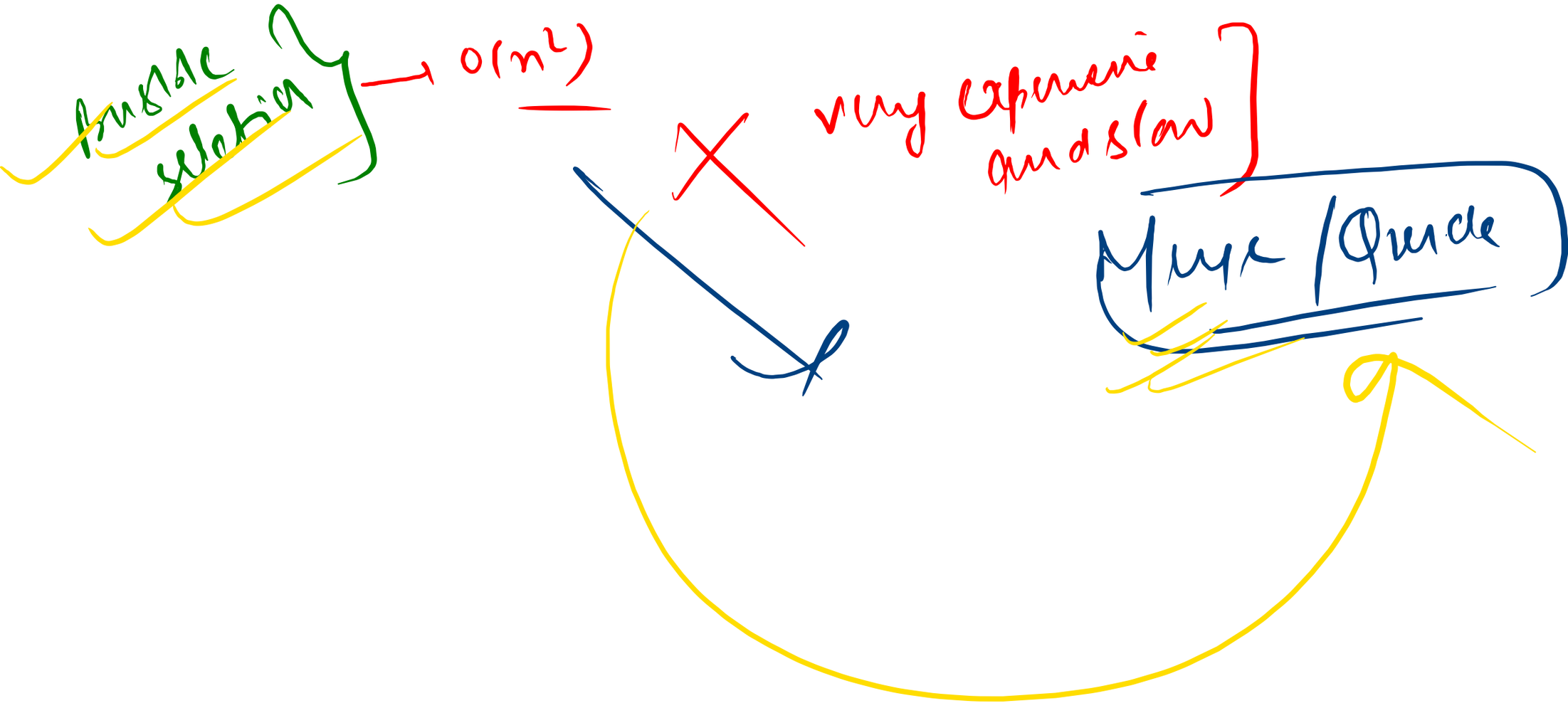
$\propto n^2$

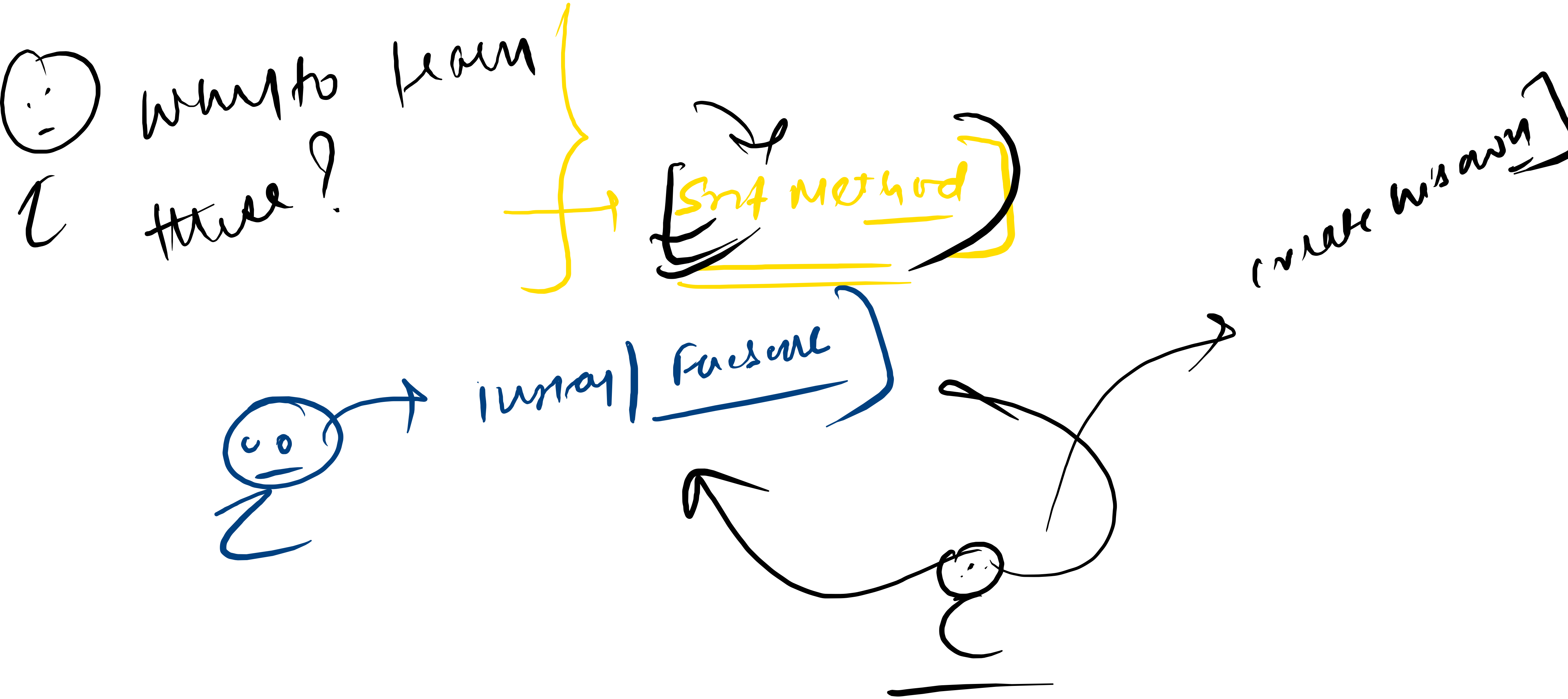
```python
#Selection Sorting
def selectionSort(arr):
    for i in range(len(arr)):
        min =i
        for j in range(i+1,len(arr)):
            if arr[j] <arr[min]:
                min =j
        #Find the smallest values
        arr[i],arr[min] = arr[min],arr[i]
```

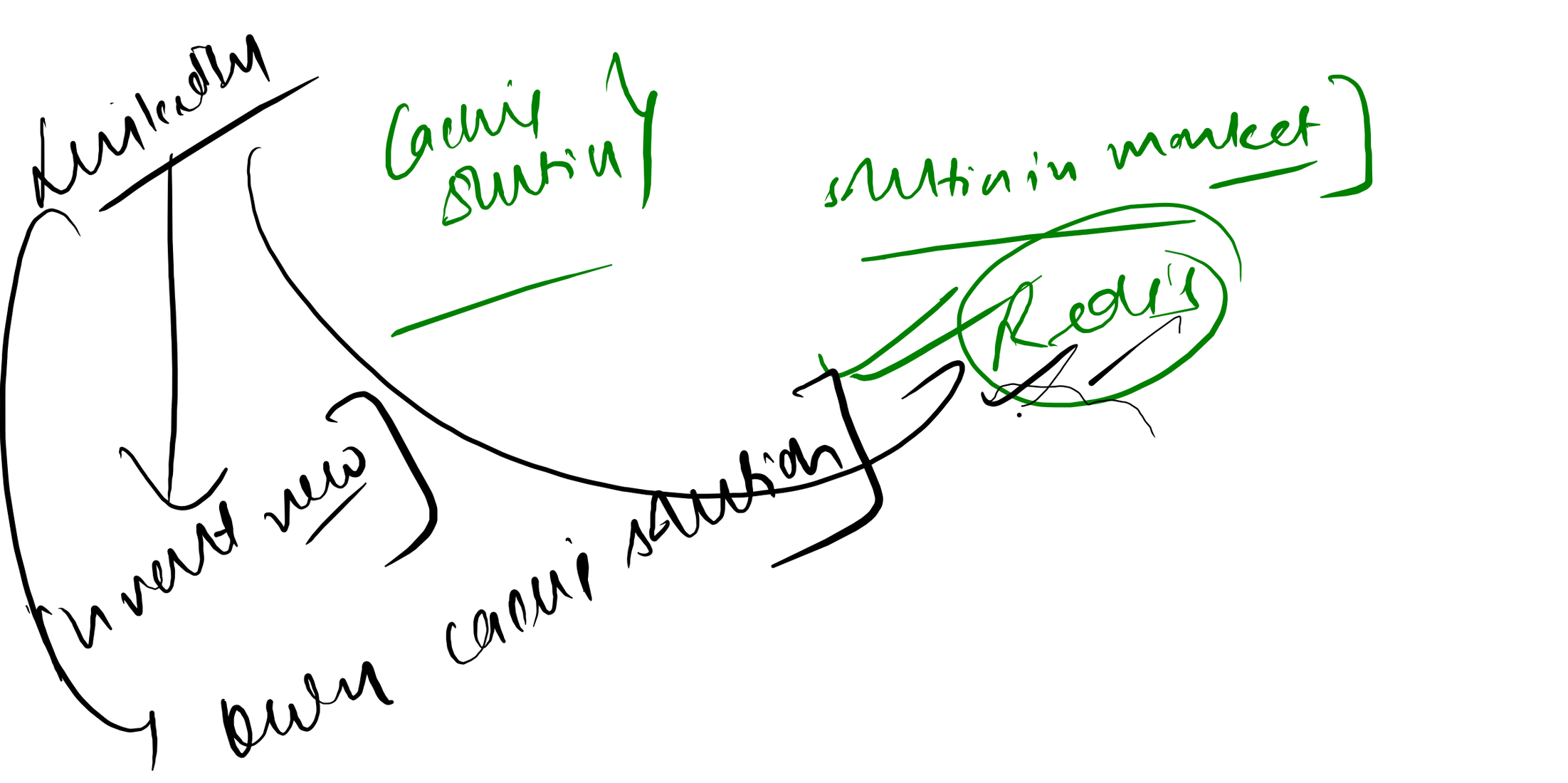


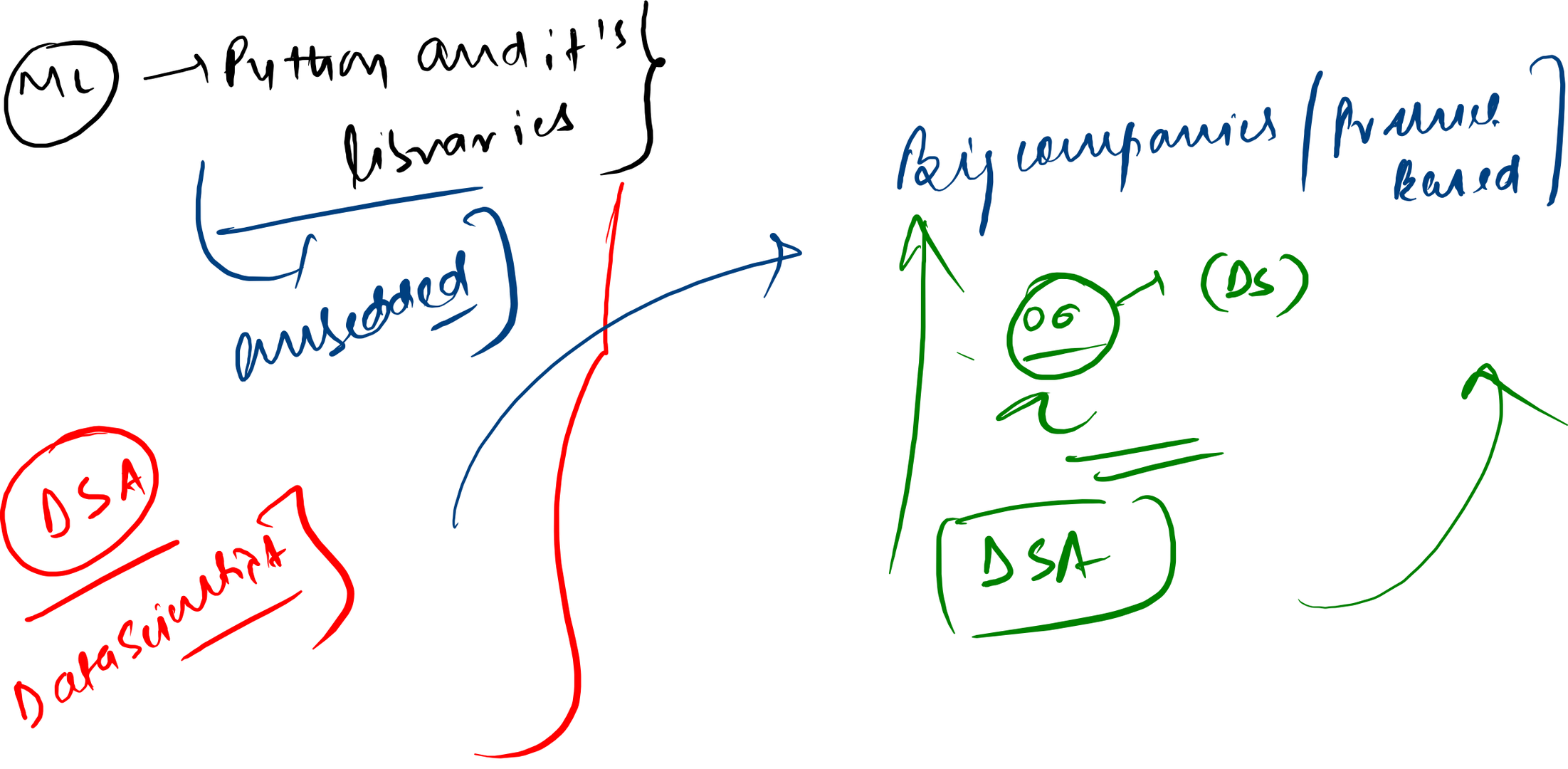


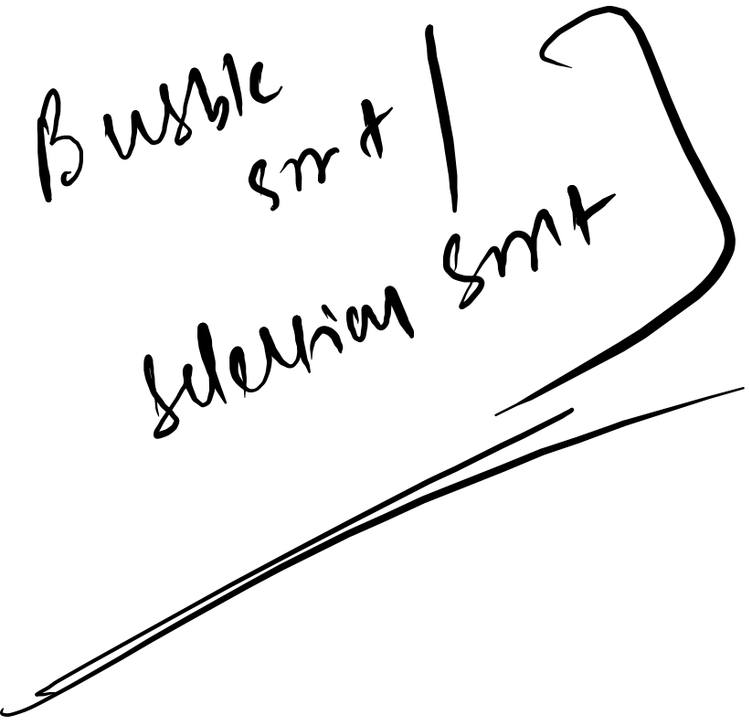
Real case
Twitter
stable
sorted order
insertion sort
Online sorting

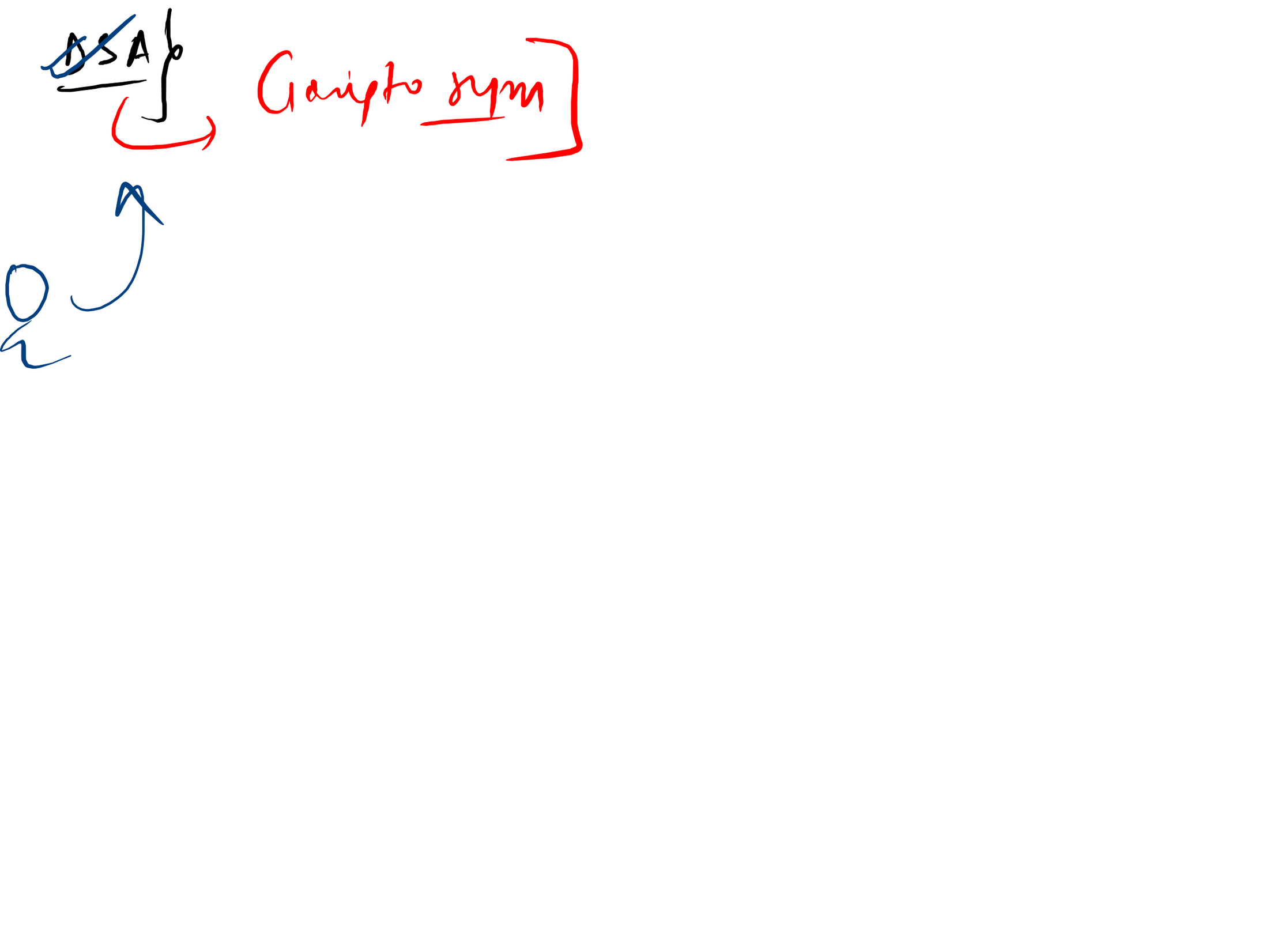

Bubble sort
O(n²)
very expensive and slow
Much faster


Why to learn these?
Sort method
Function
create its own




ML
→ Python and it's libraries
unsupervised
DSA
Data science
Big companies (Product based
OOPS
(DS)
DSA

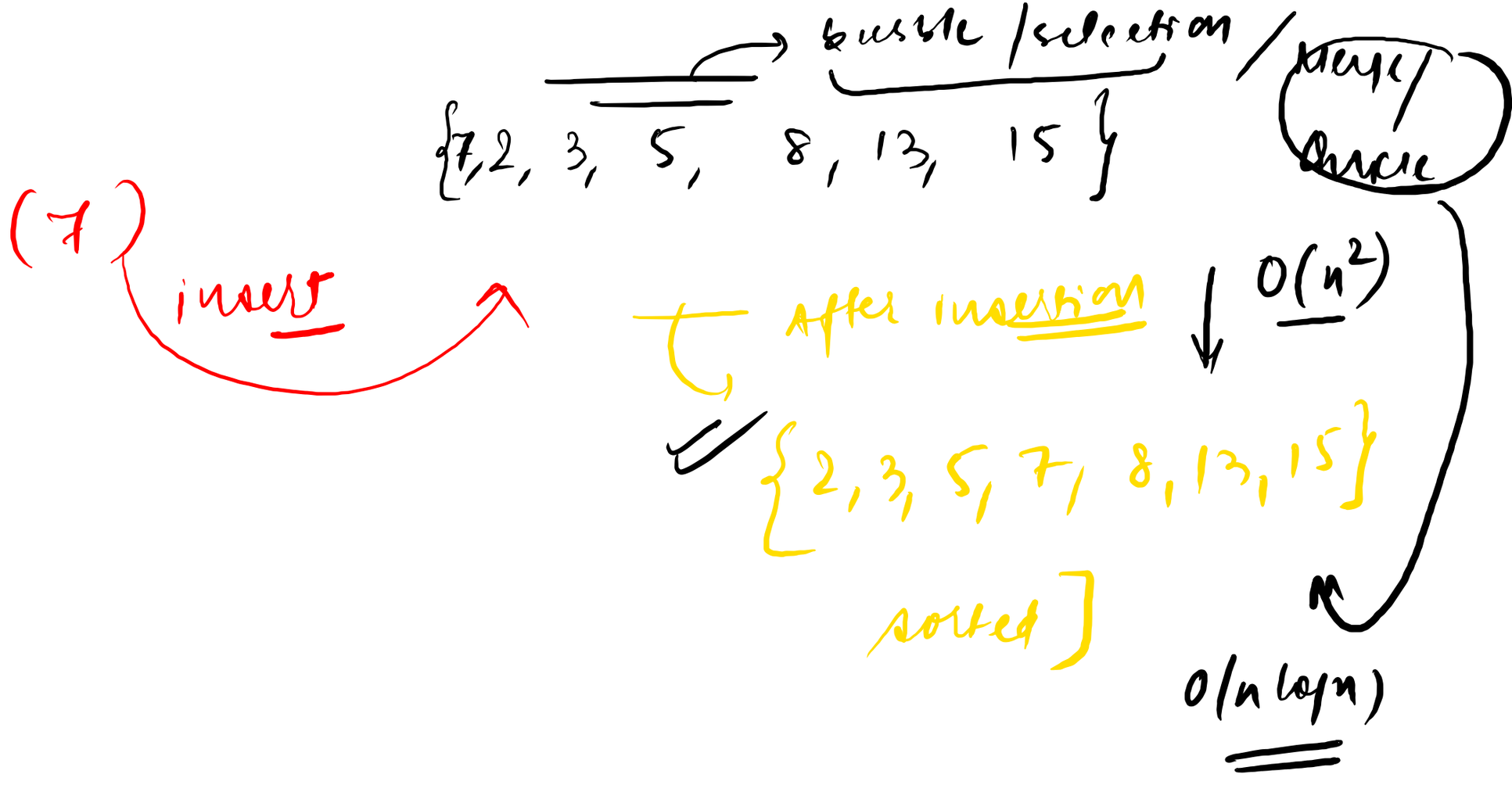

Bubble sort
Selection sort


DSA
Crypto sym

Insertion Sort → online sorting Algorithm

→ complex

Alert


bubble / selection
merge / quick
{7, 2, 3, 5, 8, 13, 15}
(7)
insert
after insertion
O(n²)
{2, 3, 5, 7, 8, 13, 15}
sorted
O(n log n)

[3, 5, 8, 9, 12]

(7) ↑ insert →

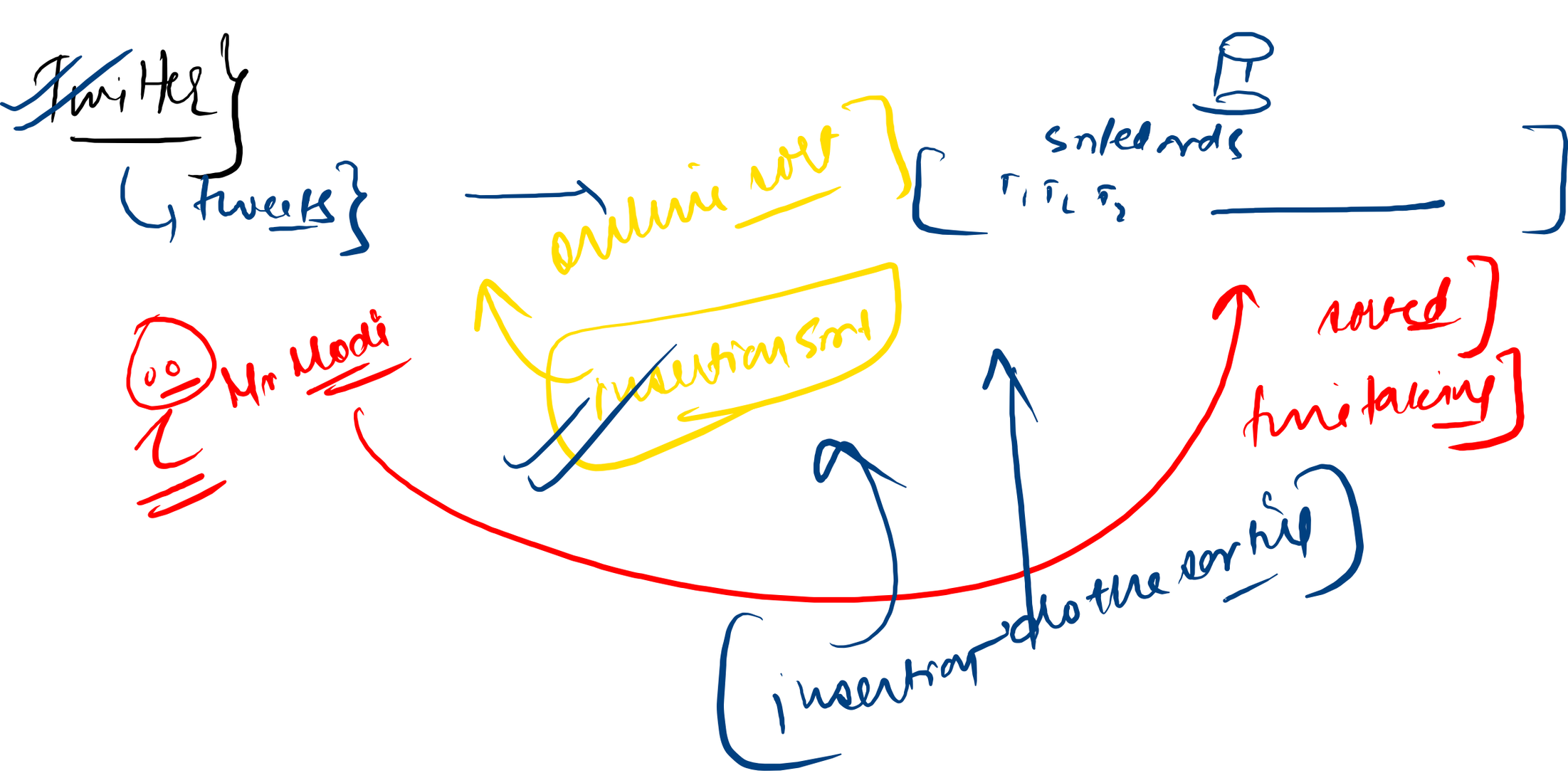
un-sorted

[7, 3, 5, 8, 9, 12]

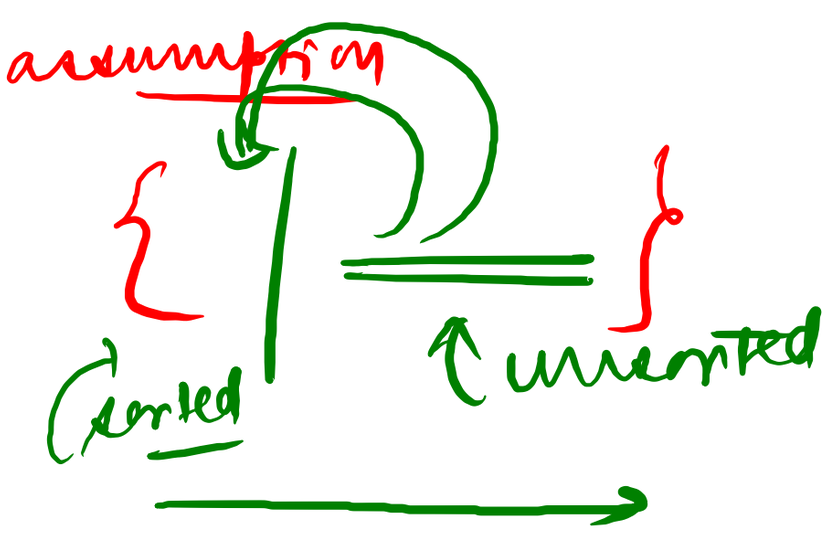
sort (BS → O(n²) SS

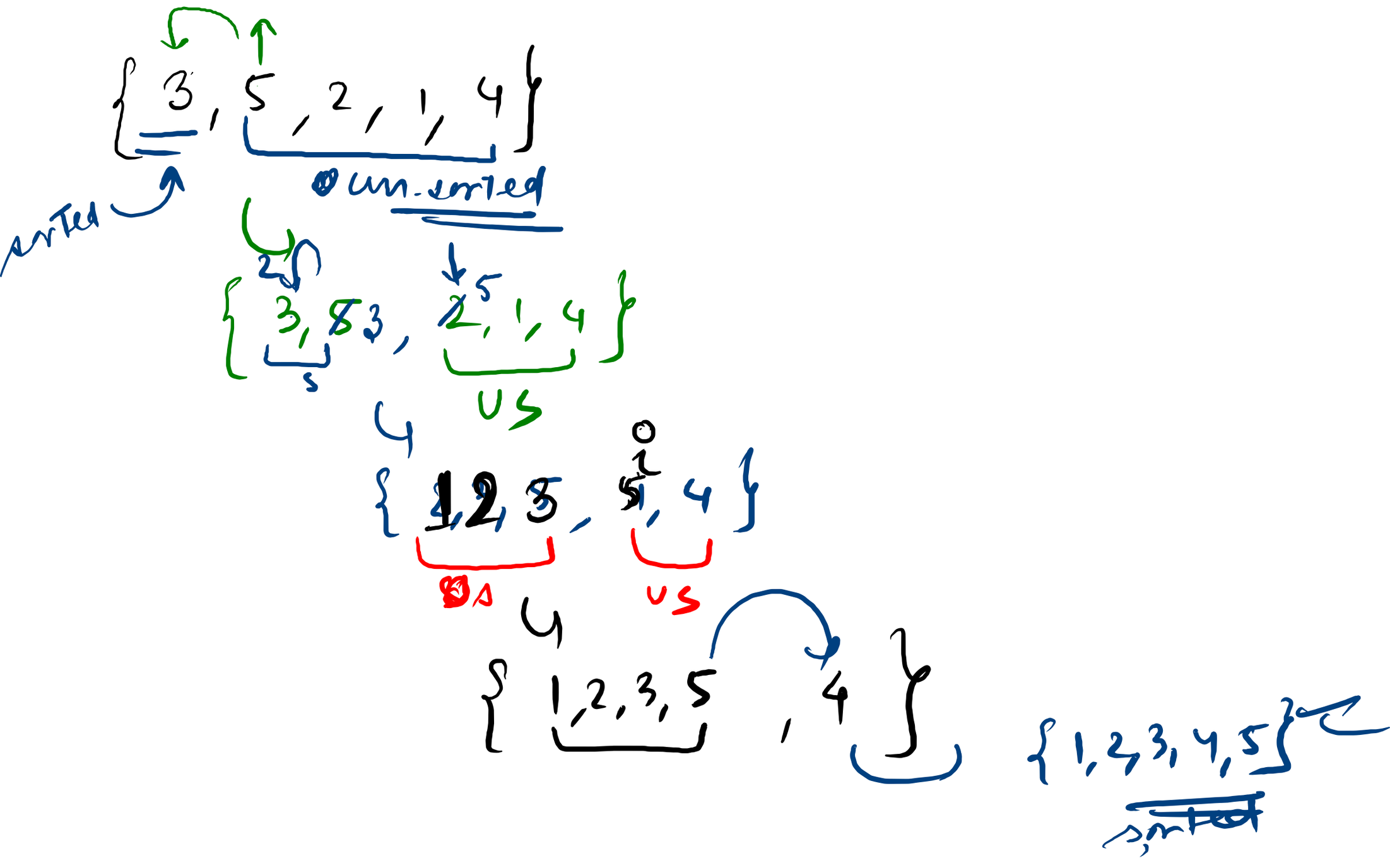
time complexity

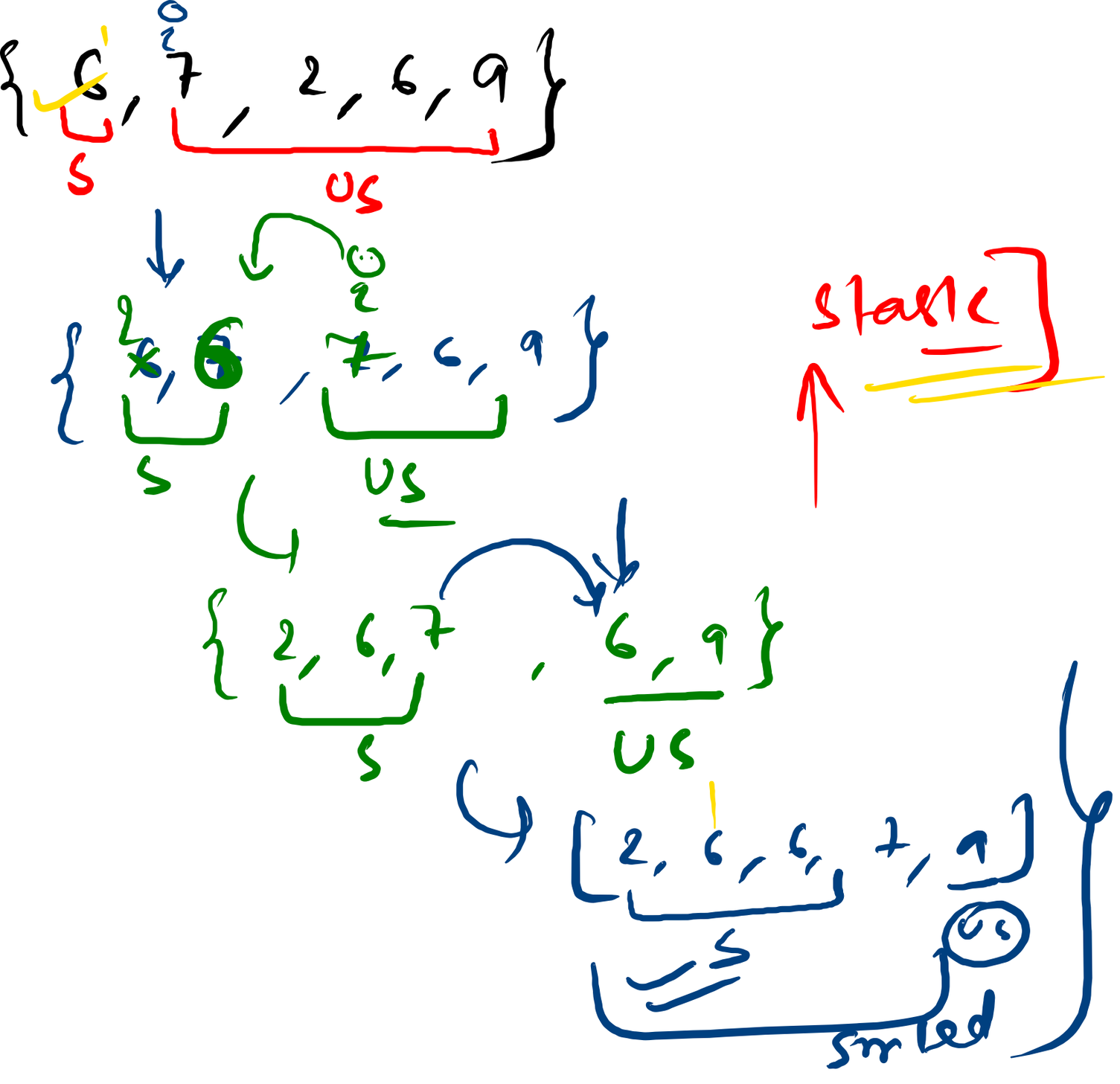
[3, 5, 7, 8, 9, 12]


Twitter
tweets
online user
[r1 r2 r3]
Mr Modi
insertion sort
sorted
tweets
insertion sort the sorted

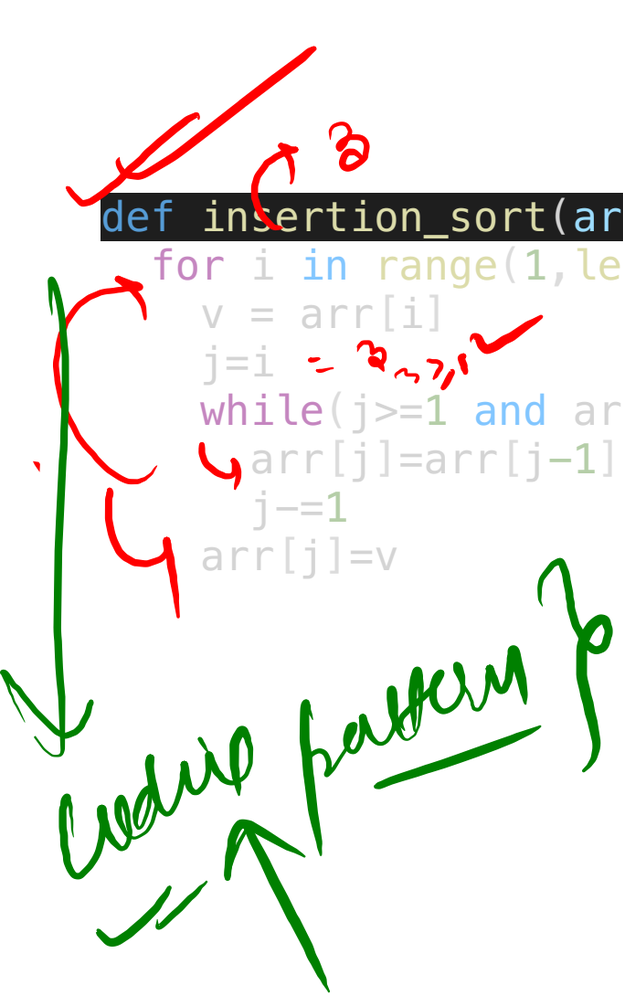
Meeting

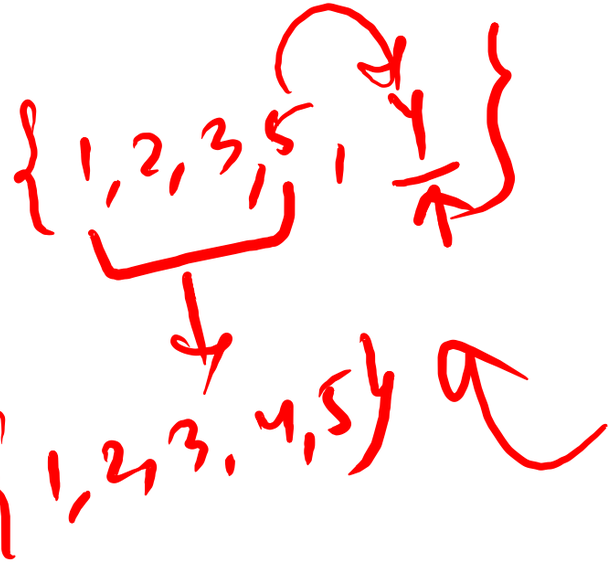



{ 3 , 5 , 2 , 1 , 4 }
sorted
unsorted
{ 3 , 3 , 2 , 1 , 4 }
s
us
{ 1 2 3 , 5 , 4 }
i
s
us
{ 1 , 2 , 3 , 5 , 4 }
{ 1 , 2 , 3 , 4 , 5 }
sorted

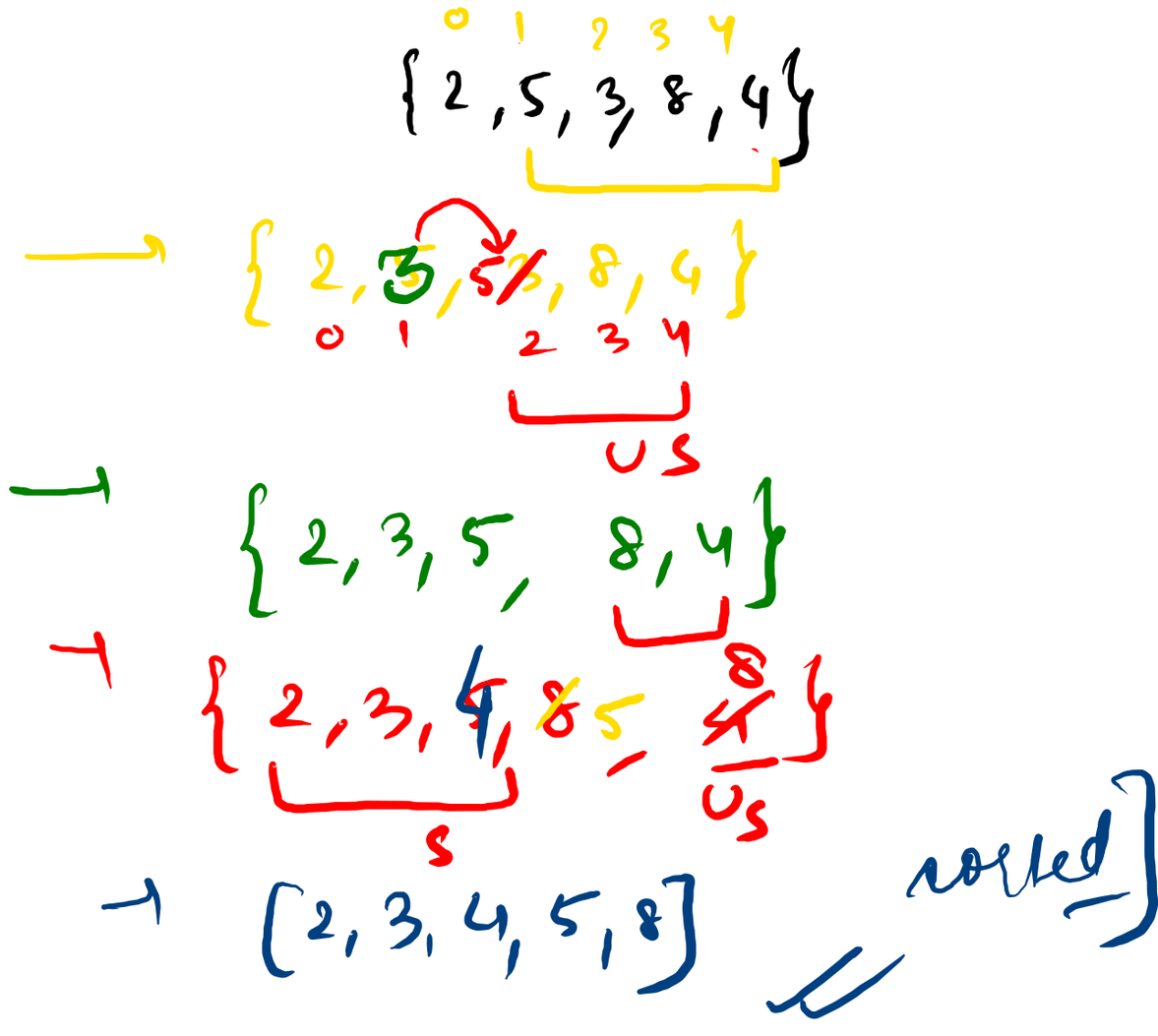

{ 6, 7, 2, 6, 9 }
s
us
{ 2, 6, 7, 6, 9 }
s
us
{ 2, 6, 7, 6, 9 }
s
us
[2, 6, 6, 7, 9]
s
smled
static

```
def insertion_sort(arr):
    for i in range(1,len(arr)):
        v = arr[i]
        j=i
        while(j>=1 and arr[j-1]>v):
            arr[j]=arr[j-1]
            j-=1
        arr[j]=v
```





```
def insertion_sort(arr):
    for i in range(1,len(arr)):
        v = arr[i]
        j=i
        while(j>=1 and arr[j-1]>v):
            arr[j]=arr[j-1]
            j-=1
        arr[j]=v
```



```python
def insertion_sort(arr):
    for i in range(1,len(arr)):
        v = arr[i]
        j=i
        while(j>=1 and arr[j-1]>v):
            arr[j]=arr[j-1]
            j-=1
        arr[j]=v
```

TC $O(n^2)$

comparison $\propto n^2$

swapping $\propto n^2$


```python
#Selection Sorting
def selectionSort(arr):
    for i in range(len(arr)):
        min =i
        for j in range(i+1,len(arr)):
            if arr[j] <arr[min]:
                min =j
        #Find the smallest values
        arr[i],arr[min] = arr[min],arr[i]
```

$O(n^2)$

n^2

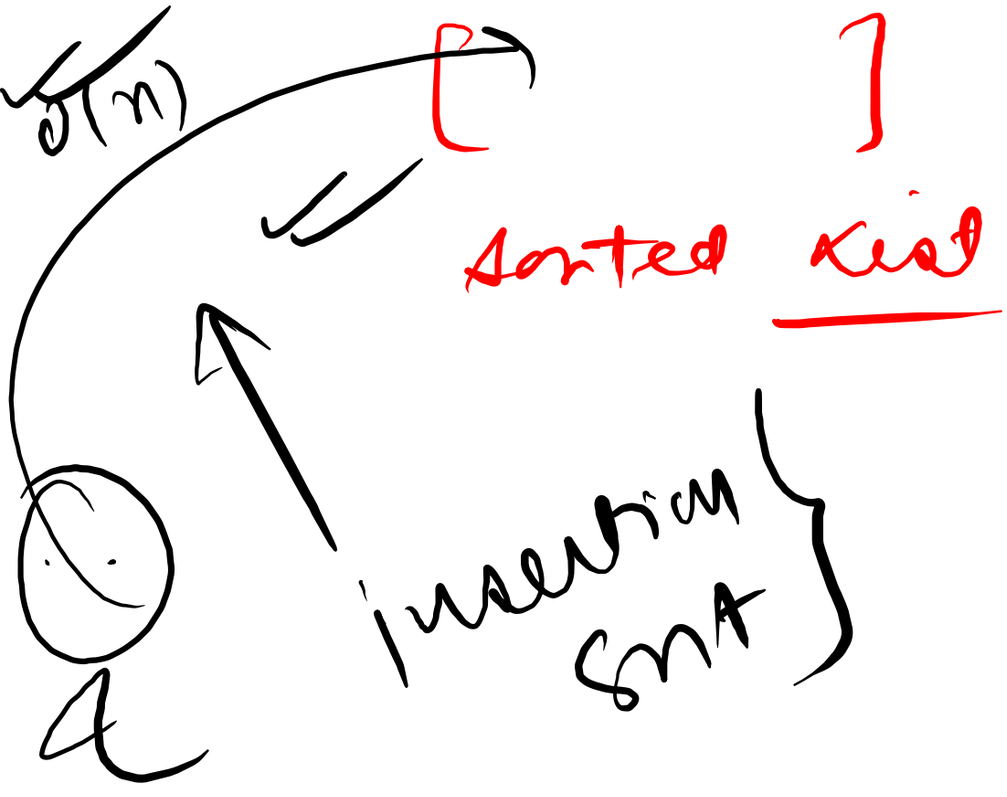
n

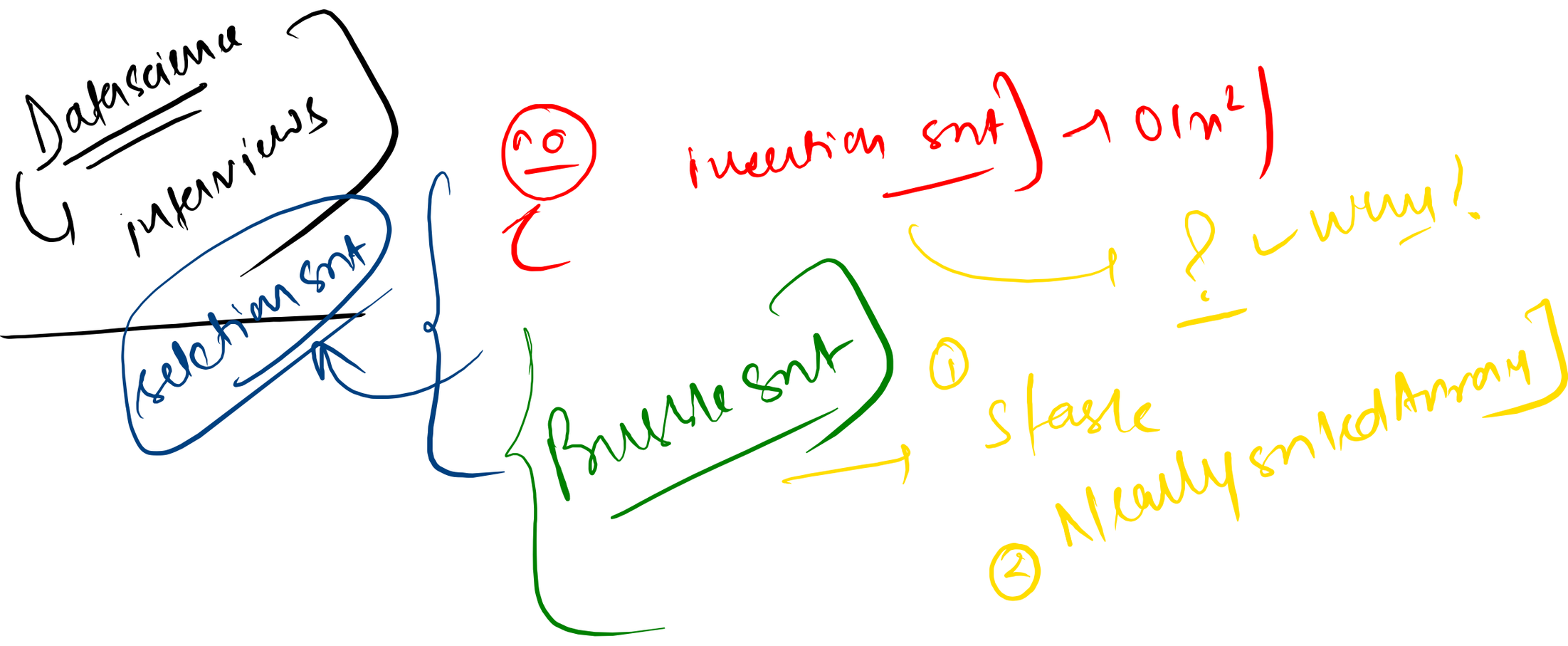
```python
def bubbleSort(arr):
    for i in range(len(arr)-1, 0 , -1):
        for j in range(i):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
```

$O(n^2)$

n^2

n

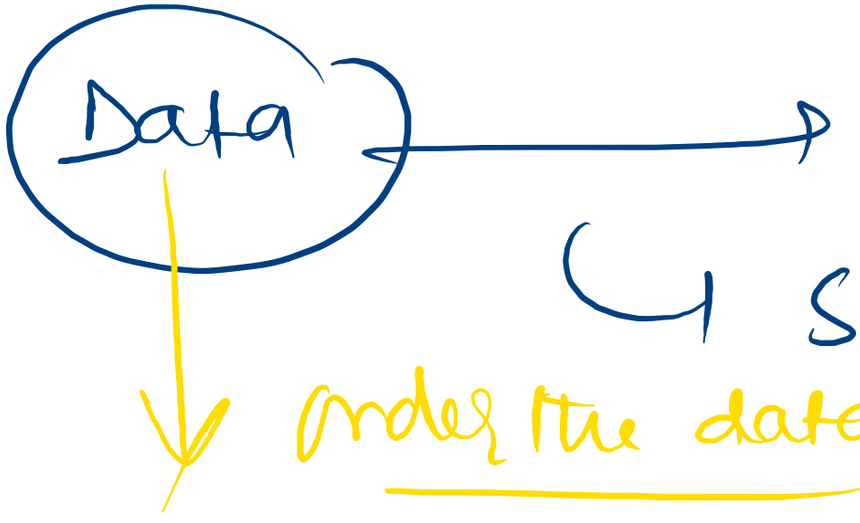

sorted list

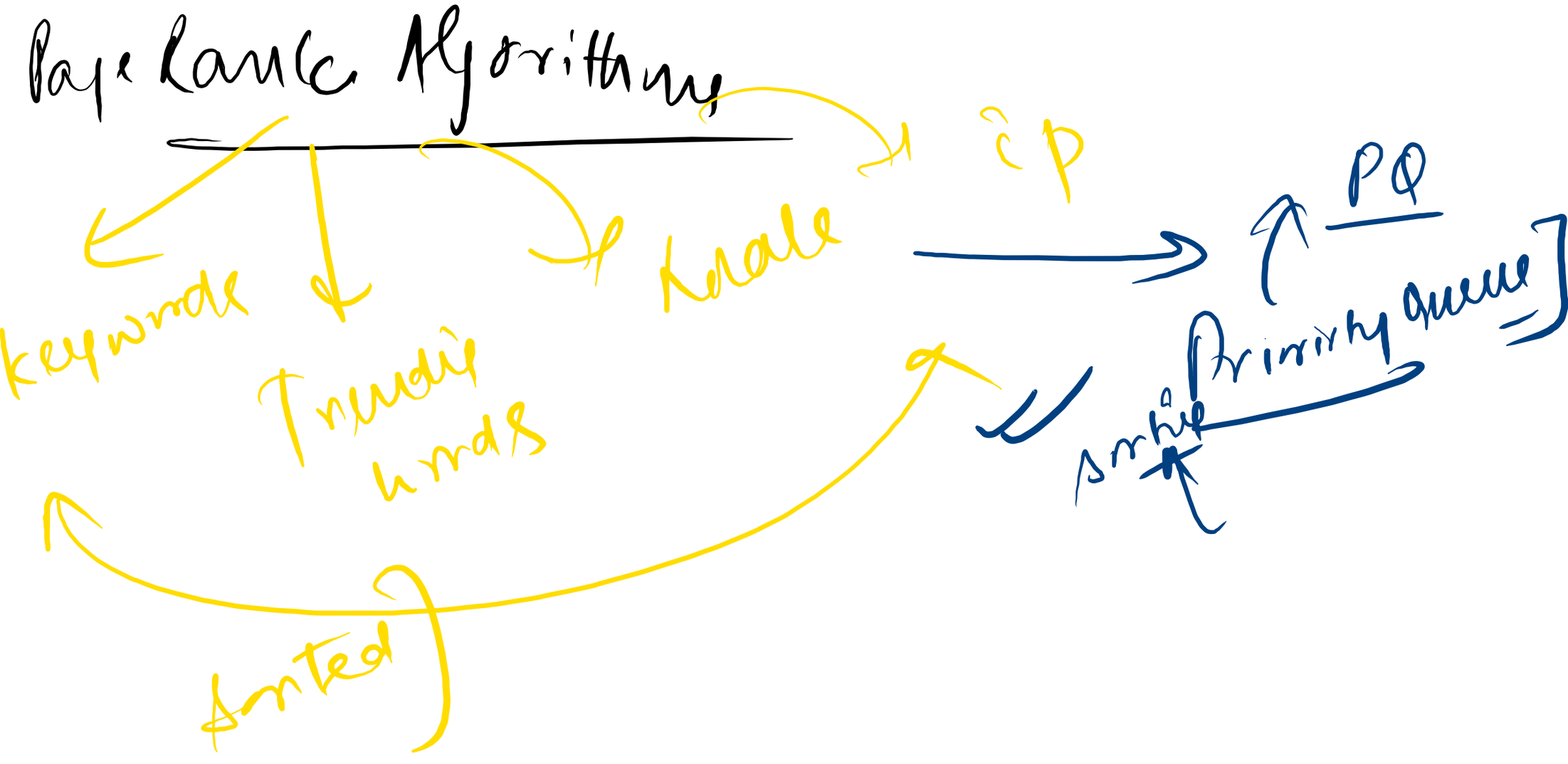

Datascience interviews
selection sort
Bubble sort
insertion sort → O(n²)
? - why!
① stable
② Nearly sorted Array

Stable smt

why?

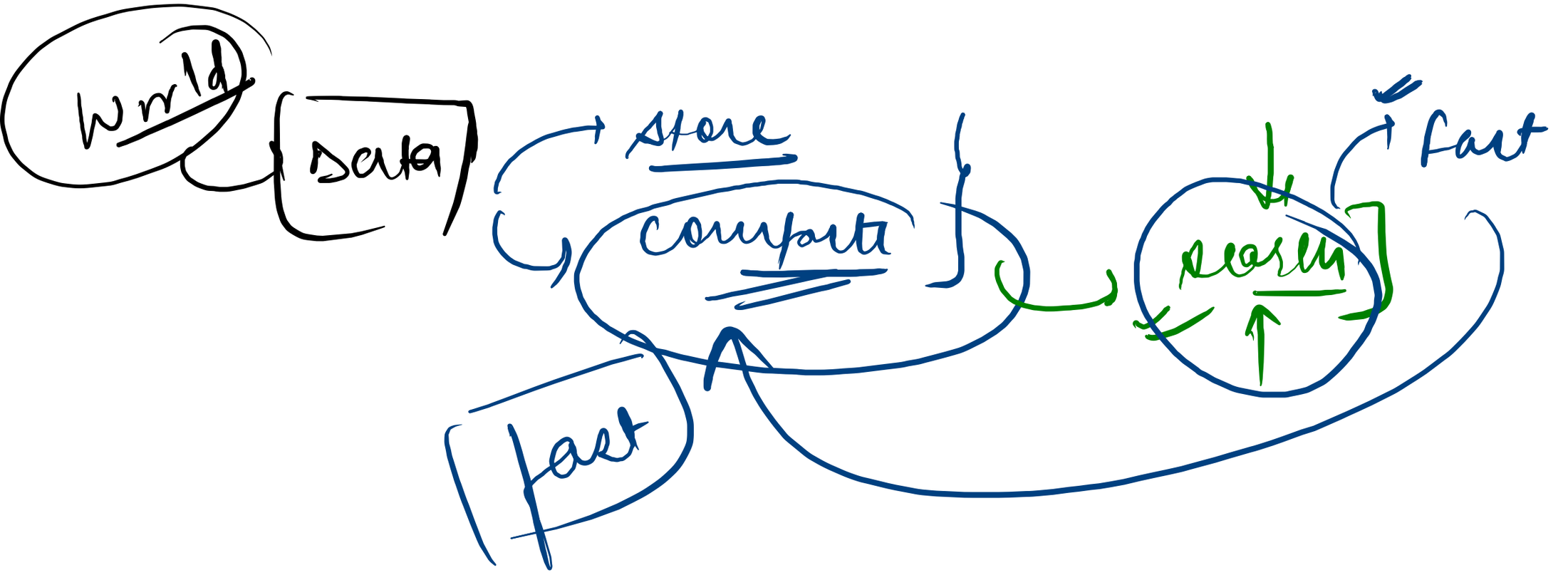
Sorting


Data
information
Statistical application
order the data
sorting


Page Rank Algorithm
keywords
Trends
words
Scale
CP
PQ
Priority queue
sorted

30 mins

100x


World
Data
store
compute
past
learn
fast

sorting

search } super fast]

Normal search approach

Linear search

10 elements → 10 seconds

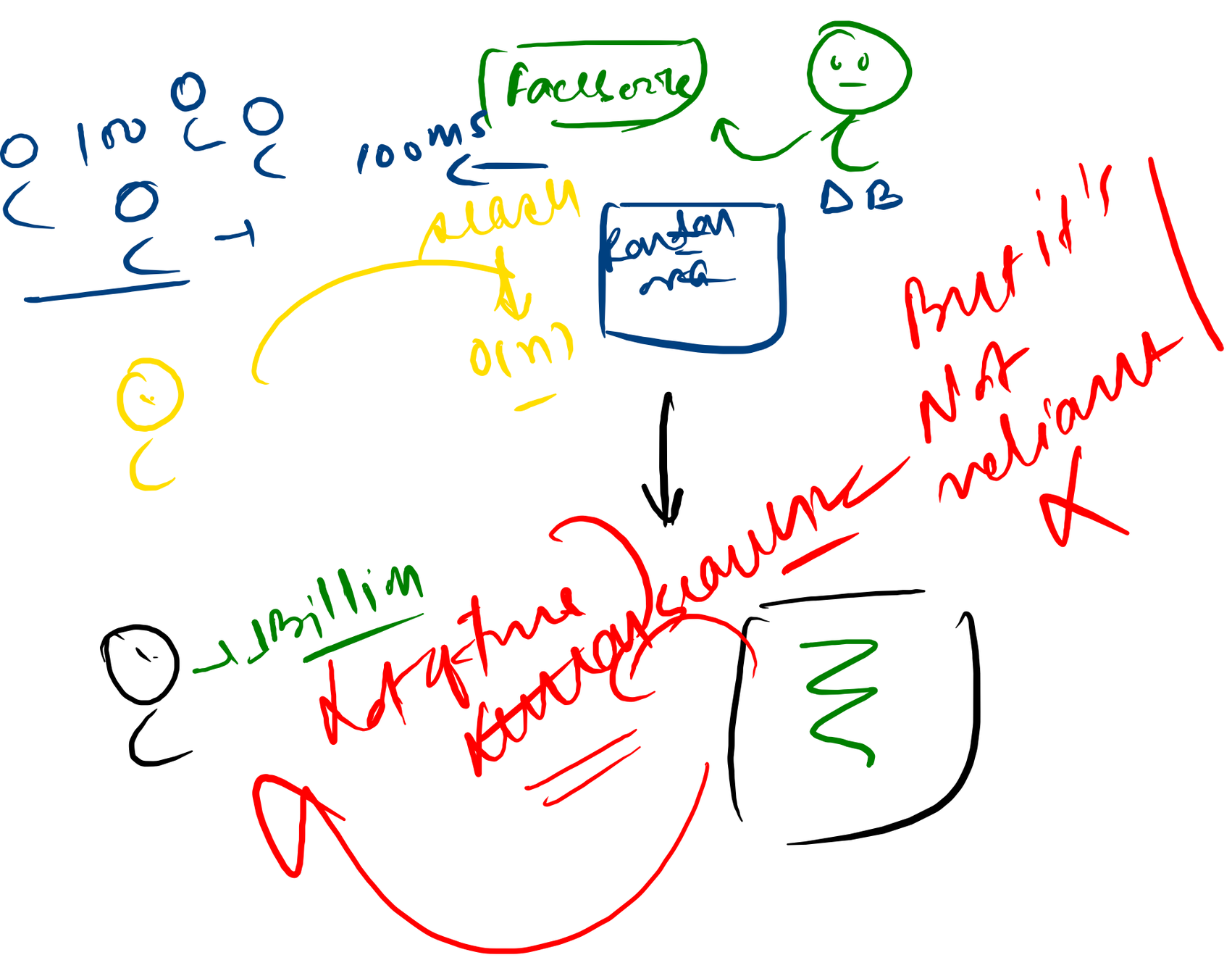
100 elements → 100 seconds

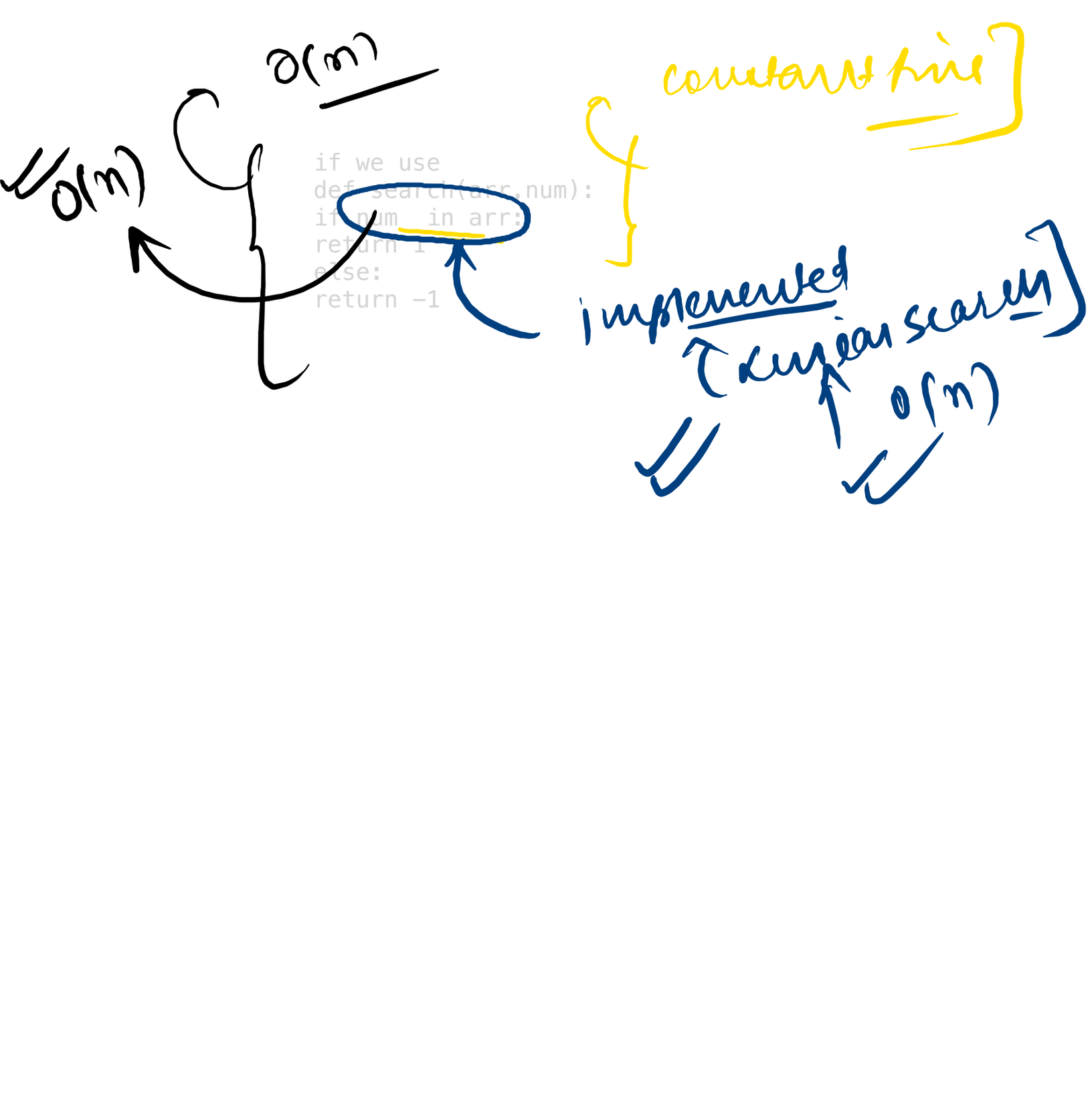
1 Million → 1 Million seconds

```python
def search(arr, num):
  for i in range(len(arr)):
    if arr[i]==num:
      return i
  return -1
```

$f(n) \rightarrow TC = O(n)$

More the number of items $\rightarrow$ search function will take


100ms
Facebook
DB
O(n)
But it's NOT reliable
1billion


O(n)
O(n)
if we use
def search(arr,num):
if num in arr:
return 1
else:
return -1
constant time
implemented
linear search
O(n)

(1) Linear search } → Python inbuilt { X NA and enough]

(2)

Sorting]

Pharmacy Shop

Room + Medicines

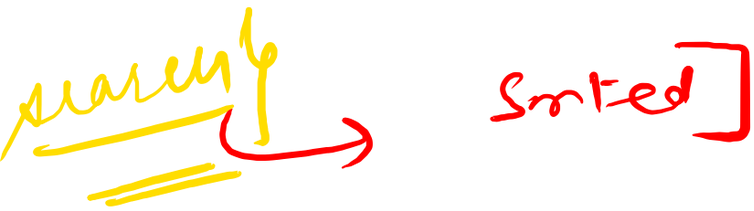
↓ Alphabetically Arranged

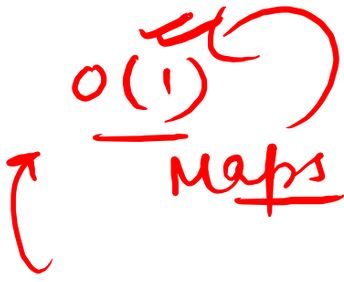
Dictionary → Missing words

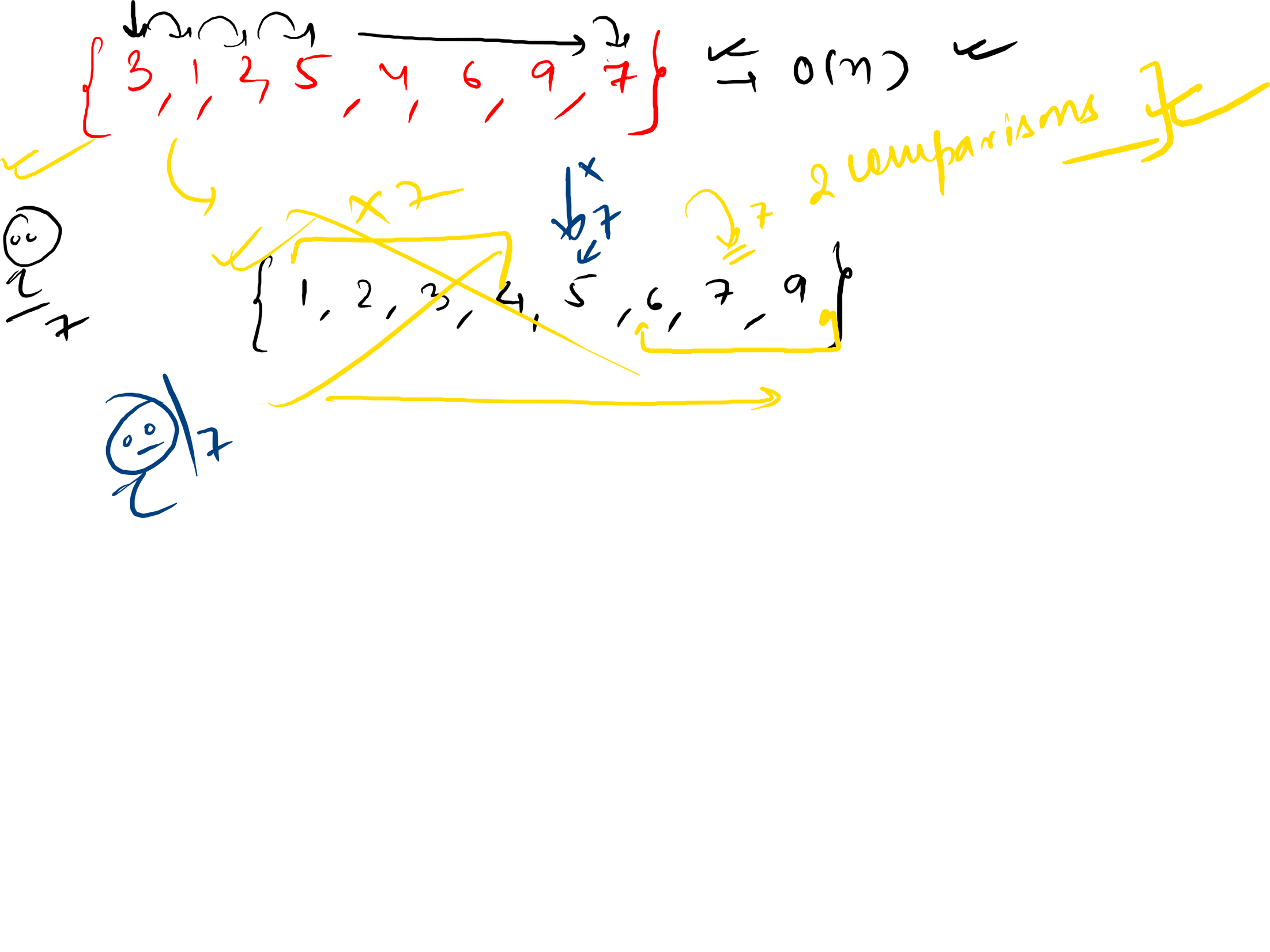
↳ Arranged alphabetically

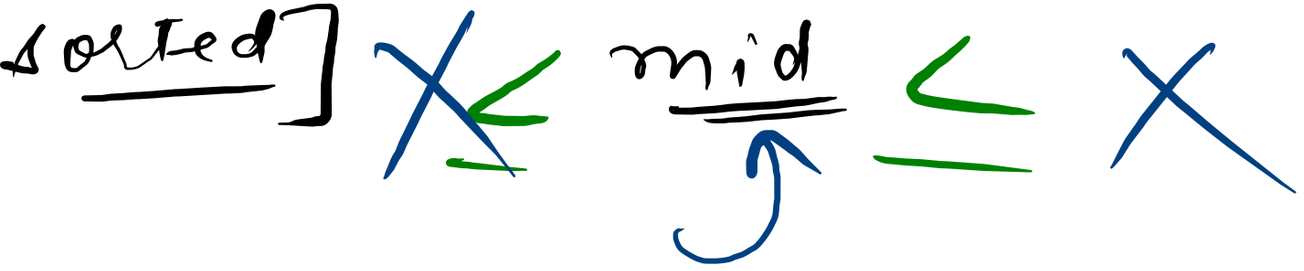
Real world

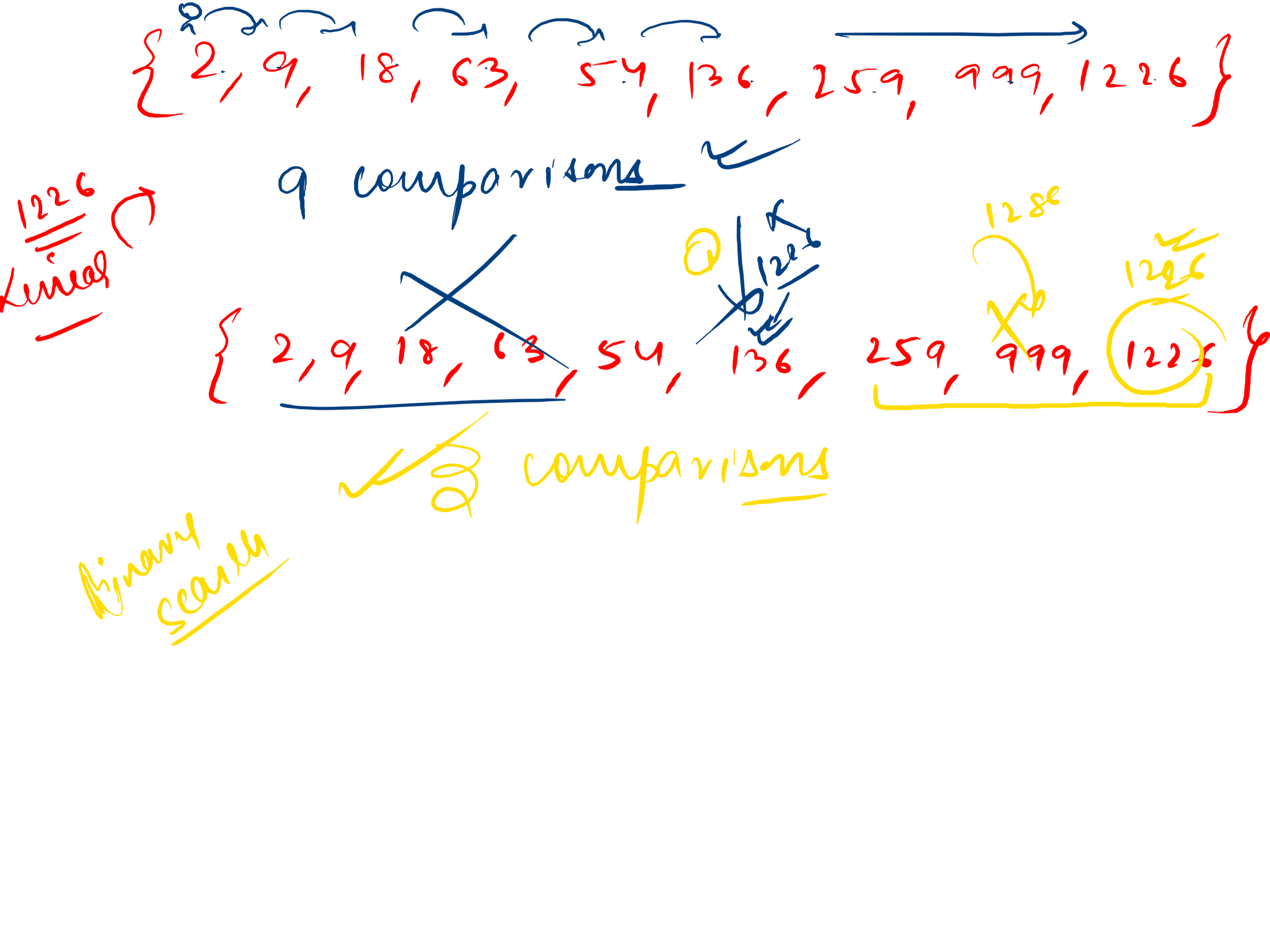

out
Bag
Main
Kelly
Tuli


search
sorted


O(1)
Maps

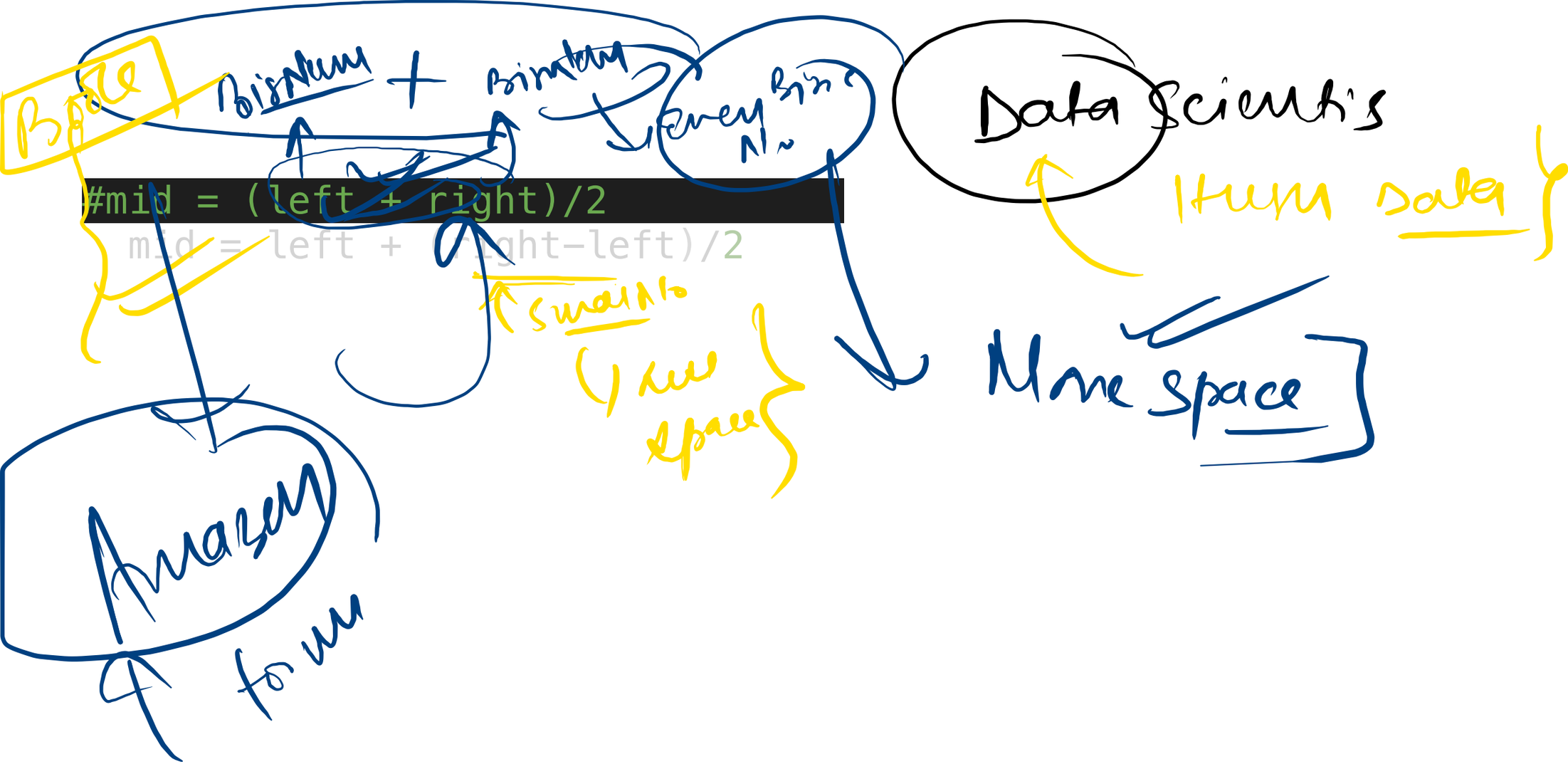

{3, 1, 2, 5, 4, 6, 9, 7}
O(n)
2 comparisons
x 7
x
7
7
{1, 2, 3, 4, 5, 6, 7, 9}
7
7

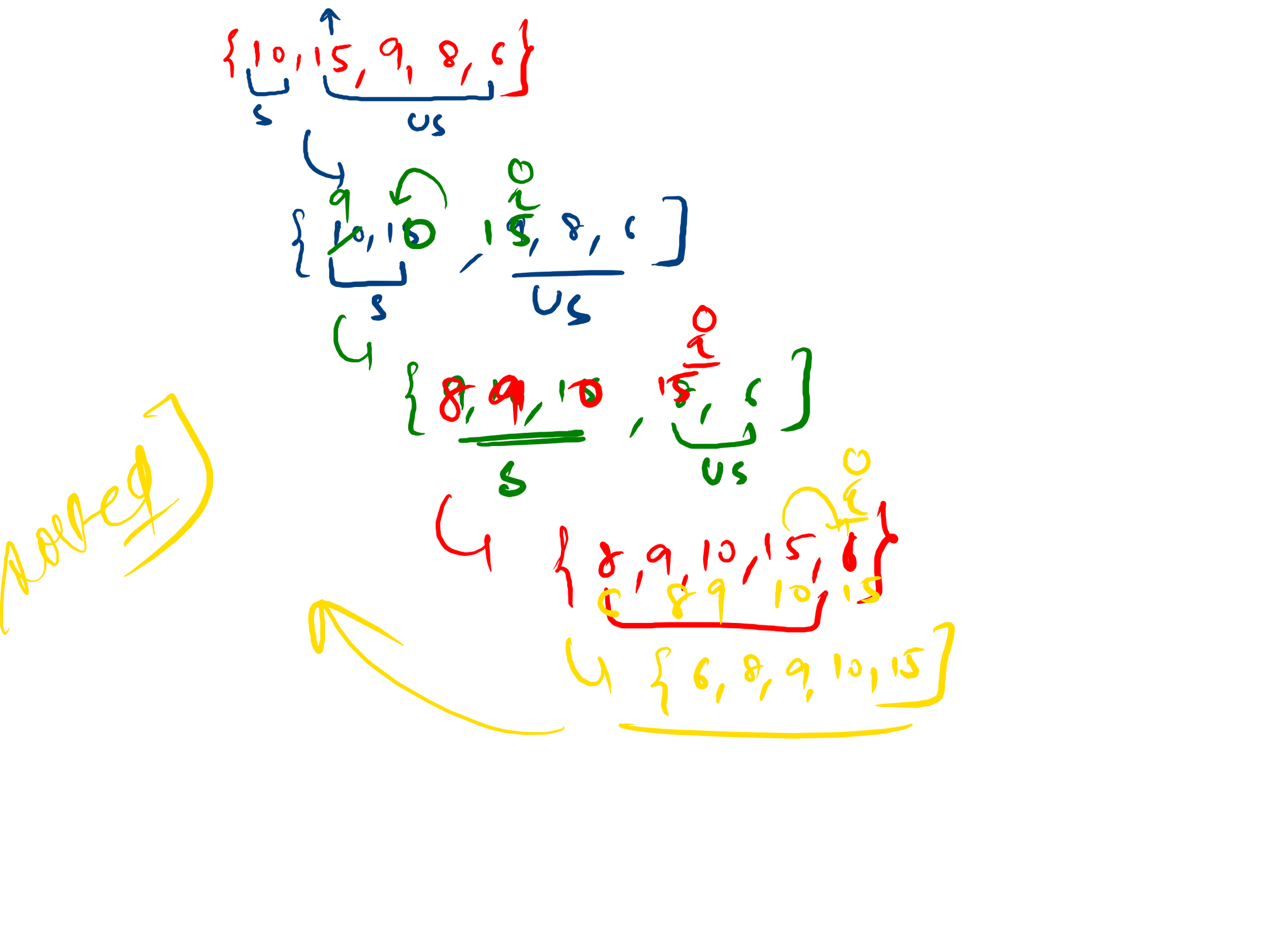

sorted
x
mid
x

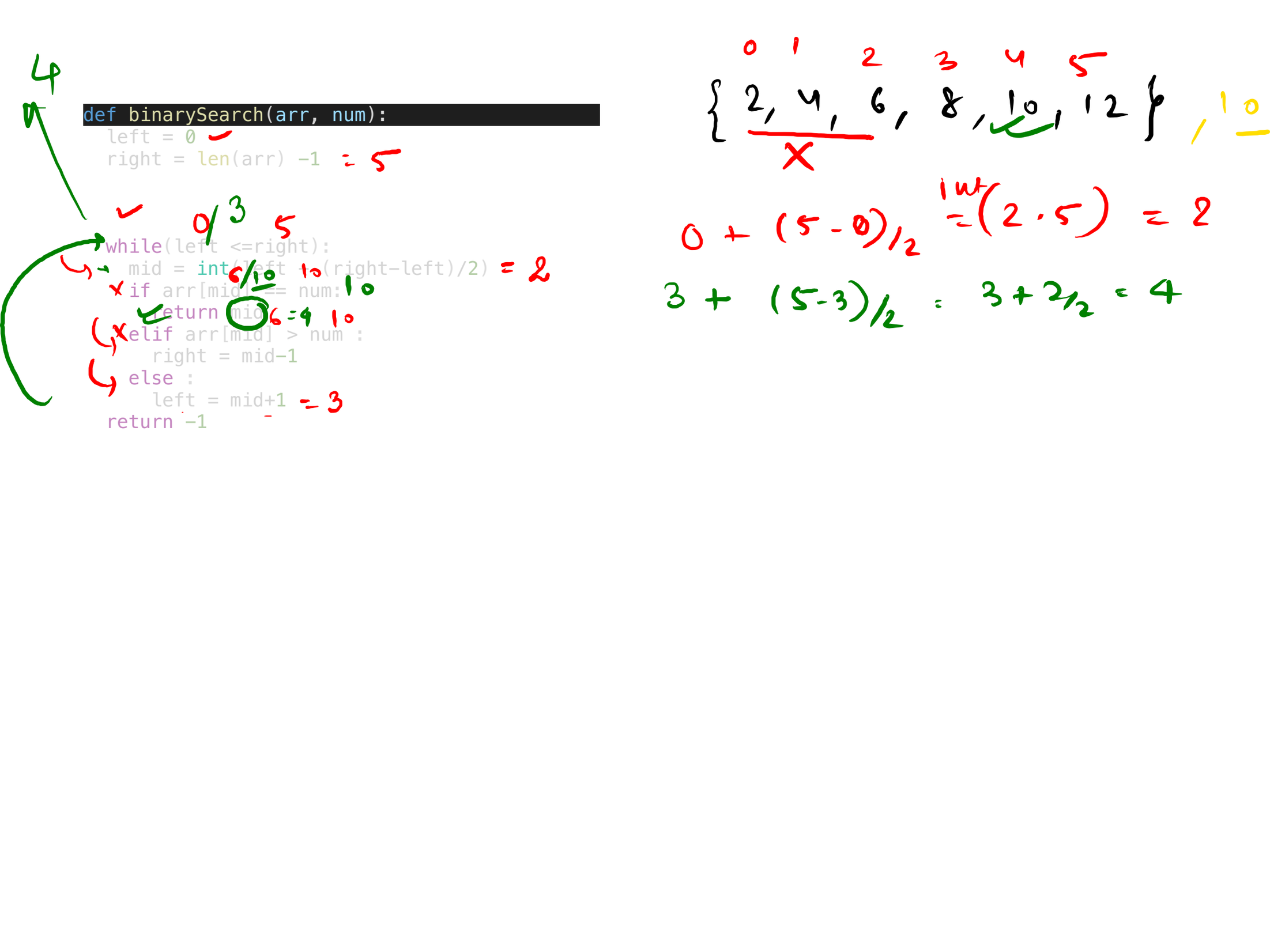

{ 2, 9, 18, 63, 54, 136, 259, 999, 1226 }
9 comparisons
1226
Linear
{ 2, 9, 18, 63, 54, 136, 259, 999, 1226 }
128
1226
3 comparisons
Binary search

Binary search

↳ Data should be sorted Already ✗


Space
Binary
Binary
Every Bin
#mid = (left + right)/2
mid = left + (right-left)/2
Data Scientists
Huge Data
Small
Low space
More Space
Array
form


{10, 15, 9, 8, 6}
s
us
{10, 10, 15, 8, 6}
s
us
{8, 9, 10, 15, 6}
s
us
{6, 8, 9, 10, 15}
{6, 8, 9, 10, 15}
Sorted


4
def binarySearch(arr, num):
left = 0
right = len(arr) -1 = 5
while(left <=right):
mid = int(left + (right-left)/2) = 2
if arr[mid] == num:
return mid
elif arr[mid] > num :
right = mid-1
else :
left = mid+1 = 3
return -1
0 1 2 3 4 5
{ 2, 4, 6, 8, 10, 12 } / 10
X
0 + (5 - 0)/2 int (2.5) = 2
3 + (5-3)/2 : 3 + 2/2 = 4

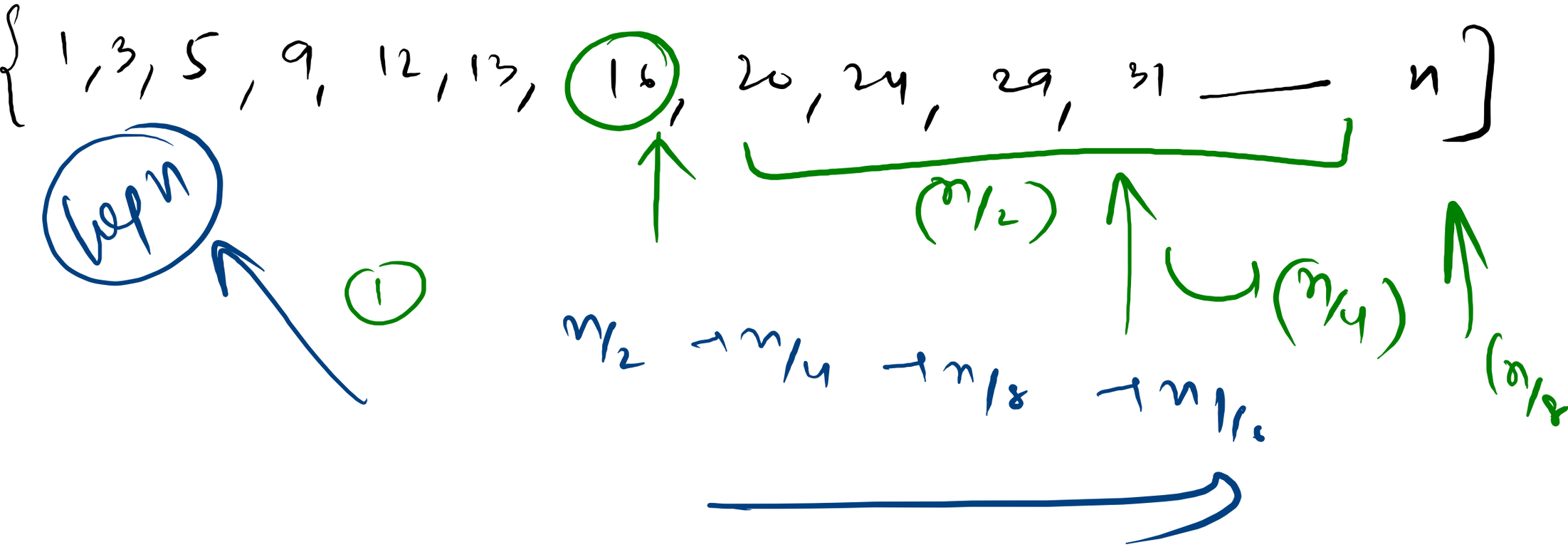
```python
def binarySearch(arr, num):
  left = 0
  right = len(arr) -1


  while(left <=right):
    mid = int(left + (right-left)/2)
    if arr[mid] == num:
      return mid
    elif arr[mid] > num :
      right = mid-1
    else :
      left = mid+1
  return -1
```

Time comparisons

1, 3,

O(log n)


{ 1, 3, 5, 9, 12, 13, 16, 20, 24, 29, 31 —— n]
log n
1
n/2 → n/4 → n/8 → n/16
(n/2)
(n/4)
(n/8

range (5) → 0, 1, 2, 3, 4

range (1, 5) → 1, 2, 3, 4

range(5 , 0 , -1) → 5, 4, 3, 2, 1