Complete Sorting algorithm

- Bubble Sort
- Stable & Unstable sort
- Selection sort
- Online sorting

SWIPE

@KRISHAN KUMAR

|| follow || for more update

Sorting Algorithm

Sorting → When we have data and we arrange that data, where we bring the meaning of that data, that called → Sorting

↳ Sorting can be easy for human in short data like

[3, 4, 1, 5, 9, 19] —sorting→ [1, 3, 4, 5, 9, 19]

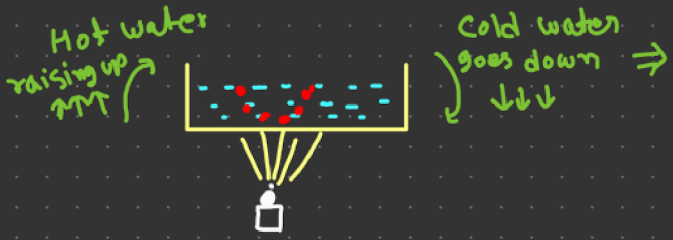
↳ But where Huge data like → in Millions for sorting we need computer. by this we can solved this problem Using python programing.

Types of Shorting algorith →

- Bubble sort algorithm
- Selection sort algorithm
- Insertion sort algorithm

Bubble sort algorithm

inspire by Bubble → when water heated



⇒ Untill the water boiled.

the same step, follow in the bubble sort algorithm.

Algorithm

[5, 3, 1, 2, 4] Apply bubble sort on this list

Step 1 :-

[5, 3, 1, 2] ⇒ Start moving towards right one by one and if we find a number which is bigger, we have to push him / Bubble him to the right.

Step 2 :- [[5, 3, 1, 2, 4]

comparing

Here when we comparing 5, 3 and i show that 5 is bigger than 3, so we swap 5 to the right. And the list is → [3, 5, 1, 2, 4] → Again go and compare (5, 1)

[3, 1, 5, 2, 4] → Again compare and swap

[3, 1, 2, 5, 4] → compare & swap

[3, 1, 2, 4, 5] → Now Repat for the Next (3, 1, 2, 4) element.

Rule:- range

① range(5) → 0, 1, 2, 3, 4

② range(1, 5) → 1, 2, 3, 4.

③ range(5, 0, -1) → 5, 4, 3, 2, 1 (-ve direction)

```
# bubble sorting                    Round
                                    ① ② ③
def bubbleSot(arr):                 3 / 2 / 1
    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]
```

outer loop → `for i in range(len(arr)-1, 0,-1):`

inner loop → `for j in range(i):`

Swaping → `arr[j], arr[j+1] = arr[j+1], arr[j]`

Let → arr → [4, 2, 3, 1] , len(arr) → 4

here → len(arr) - 1 → 4 - 1 ⇒ 3

Means → for i in range(len(arr)-1), here ⇒ i = 3

and for J in range(i): , here J = 0

Sorting

Step:2

J, i = 2

[2, 3, 1, 4] No swapping

[2, 3, 1, 4]

[2, 1, 3, 4]

{done}

J-

Step:3

J, i = 1

[2, 1, 3, 4] → No swapping

[1, 2, 3, 4]

Bubble sorting done

Step 1 :-

J, i = 3

{4, 2, 3, 1}

Swaping

[2, 4, 3, 1]

[2, 3, 4, 1]

[2, 3, 1, 4]

{Swaping for the 1st element done → going 4th outer loop}

Q.2

```
]: # bubble sorting                    Rounds of i taken 1 by 1

   def bubbleSot(arr):                  4 / 3 / 2 / 1
       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]
```

outter loop

inner loop

swapping

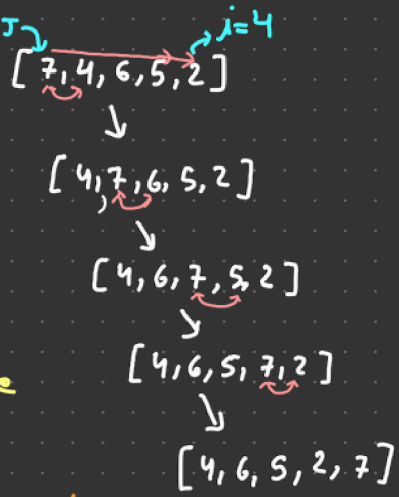
```
]: arr = [7,4,6,5,2]
   bubbleSot(arr)
   print(arr)

   [2, 4, 5, 6, 7]
```

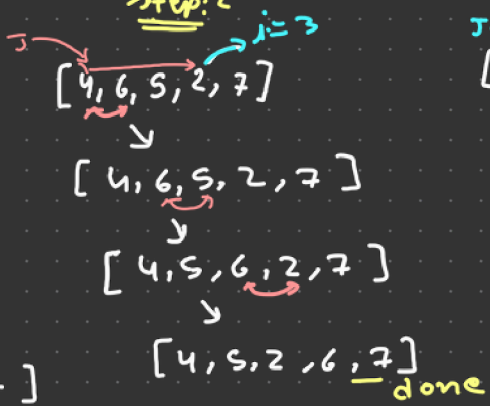
How this code work internally

Sol^n

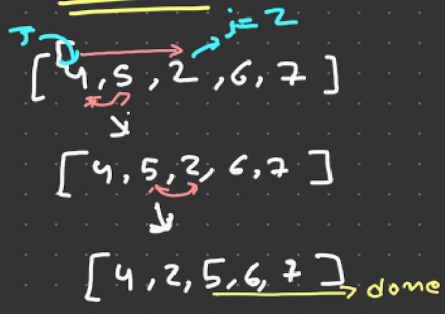
Step: 1



Step: 2

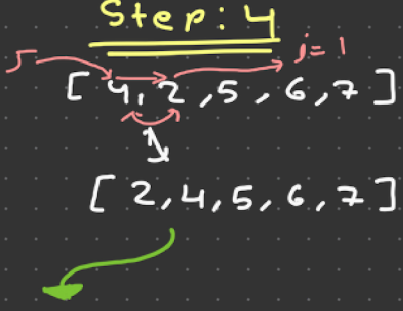


Step: 3



internal loop done go to outer loop and Repeat the same process until all value get Sorted

Step: 4



Bubble Sorting done

Q. What is the time complexity of this code.

```
# bubble sorting

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]
```

outer loop

inner loop

↳ outer loop runs ⇒ n times

inner loop runs ⇒ n times

So that time complexity of this code is

$T(n) = O(n \times n)$

$\boxed{T(n) = O(n^2)}$

Q. In the case of worst case How many times camparison will be happen in this code?

↳ In the worst case every time we will doing the comparison. So

the no. of loop is runing ⇒ n^2

So the in worst case camparision happens → n^2 time

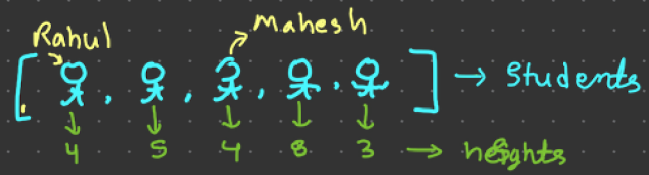
Q. what is the worst case in this situation?

↳ when array already is reversing or sorted in desending order. for example → {5,4,3,2,1} ⇒ Already Sorted (worst case)

Stable Sort & Unstable Sort

Stable Sort → if you have two elements with equal keys, and one comes before the other in the input, a stable sort ensures that the element that was initially before the other will be before it in the sorted output.

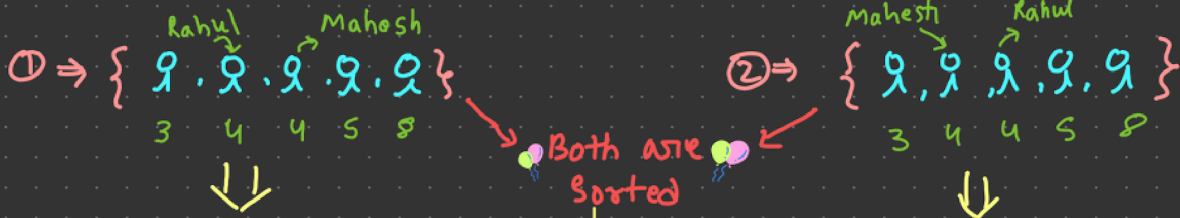
ex →



Problem

The class teacher said range all the students based on their height

↳ For this problem their could be two posibilities :-



Stable Sorting

before sorting ↴
Rahul stands before mahesh
After sorting ↴
Still Rahul Standing before mahesh
↓↓
This called as → Stable Sorting

UnStable Sorting

Before sorting ↴
Rahul Stands before mahesh
After sorting ↴
Rahul Standing after mahesh

This change called as unstable Sorting

Real world use case of stable & unstable sorting

Fear mongering → Spreading fake news.

Whatsapp → Spreading fake news that Rohit Sharma Retired

↓

Now people fowarding news.

↓

In all massages → Same keys → Rohit Sharma Retred

↓

Key

1 → 🔑 → RSR

2 → 🔑 → RSR

⋮

m → 🔑 → RSR

→ if i sort in → Unstable Sorting

{order will be Rearrange}

(not a good aproach)

→ if i sort → Stable Sorting

{order will be the same}

(Right aproach)

By Using Stable sorting and making data in arranging we will find the person who actually start this news.

Q. This sorting algorithm follow which one

```python
# bubble sorting

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]
```

swapping

(A) Stable Sorting

(B) Unstable Sorting

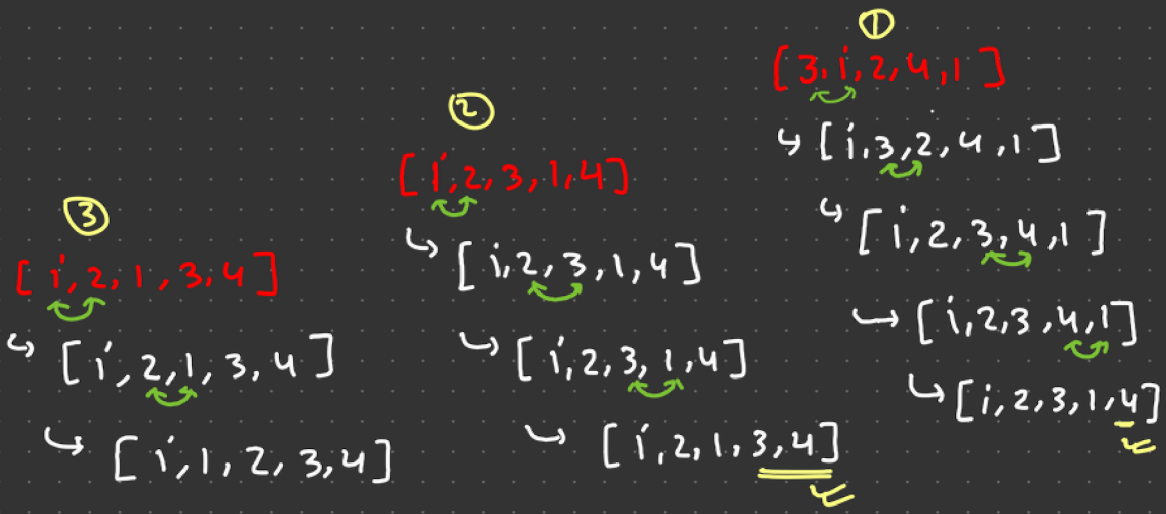
Ans → swapping happens only when → arr[j] > arr[j+1]

So,

if two value are same then we don't swapping here

So → this algorithm follow →~~> Stable Sorting.

Lets understand with an example:- arr → [3, i, 2, 4, 1]



Before sorting arr → [3, i, 2, 4, 1]

after sorting arr → [i, 1, 2, 3, 4]

↳ as we can see that before sorting [i] is before [1] and after the sorting [i] still before [1].

↳ So this is Stable sorting.

{Because order doesn't change here.}

Interview Q. 2

```python
# bubble sorting

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]
```

n^2 time

Q Apply bubble sorting on this code where $T(c) \neq O(n^2)$ array should be already sorted.

Ans in this code loop Runs → n^2 time

So the $T(c) \Rightarrow n^2$

Because bubble sort not that much smart who catch given array is sorted OR not → He run Both loops,

So,

we need to make some changes in the code in this way where if give sorted array to the code it detect himself the is it sorted OR not. and React differently when got sorted array.

Now the optimized code :)

```python
# bubble sorting

# stable sorting
def bubbleSot_optimized(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
```

0 time → (outer loop)

n time → (inner loop)

not go here ← (when isSorted = False)

Because already Sorted array

Let → [1,2,3,4,5]

T_2 : 1 → 2 → 3 → 4 → 5 (i)

isSorted = True

↳ [1,2,3,4,5]

↳ [1,2,3,4,5]

↳ [1,2,3,4,5]

Now in this case it will not go out from the outer For loop again and print the statement and Terminate there. in this way I don't have to run till n^2

So the $T(c) \neq O(n^2)$ where $T(c) = O(n \times 1) \Rightarrow O(n)$

What if we are use nearly sorted array ?

Let [1,2,3,5,4]

```python
# bubble sorting

# stable sorting
def bubbleSot_optimized(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
```

Ans →

(1)

[1,2,3,5,4]

is sorted = True

↳ [1,2,3,5,4]

↳ [1,2,3,5,4]

↳ [1,2,3,5,4]

is sorted = False

[1,2,3,4,5]

then it came to outer loop

(2)

[1,2,3,4,5]

is sorted = True

↳ [1,2,3,4,5]

↳ [1,2,3,4,5]

↳ [1,2,3,4,5]

↳ [1,2,3,4,5]

Now sorted fully

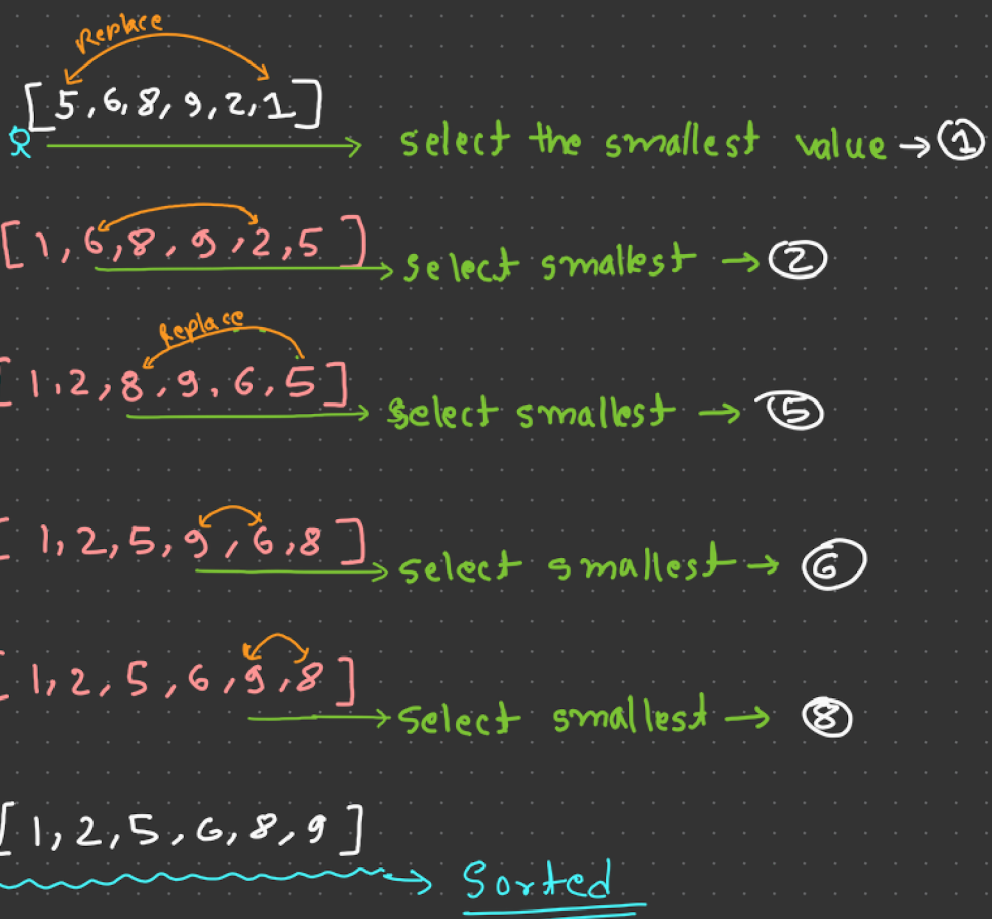
Now then loop Break

1 loop Runs only for ⇒ 2 times

Selection sorting

Think of arranging books on a shelf. You find the smallest book and put it first. Then, find the next smallest and put it next. Repeat until all books are in order.

Let,



in this example we find → smallest
we also find the largest value as value with selection sort technique

ex. 2

[3, 1, 5, 4, 2] (Replace) → Select largest value → (5)

↳ [3, 1, 2, 4, 5] → Select largest → (4)

↳ [3, 1, 2, 4, 5] (Replace) → Select largest → (3)

↳ [2, 1, 3, 4, 5] → Select largest → (2)

↳ [1, 2, 3, 4, 5] → Sorted

Selection Sort which technique follow stable OR Unstable sort

ex →

[3, 5, 2, 5', 1] → Select largest → (5)

↳ [3, 1, 2, 5', 5] → " → (5')

↳ [3, 1, 2, 5', 5] → " → (3)

↳ [2, 1, 3, 5', 5] → " → (2)

↳ [1, 2, 3, 5', 5] → Before sorting → 5 Before 5'

After sorting → 5' Before 5

Selection sorting follow Unstable sort ← change happen

Code for the selection sort

```python
# Selection sorting

def selection_sort(arr):
    for i in range(len(arr)):          ## iterating through range
        min = i
        for j in range(i+1, len(arr)):     ## finding the min values
            if arr[j] < arr[min]:

                min = j

        # find the smallest value
        arr[i], arr[min] = arr[min], arr[i]
```

Selection sort Vs Bubble sort

```python
# Selection sorting

def selection_sort(arr):
    for i in range(len(arr)):       ## iterating through range
        min = i
        for j in range(i+1, len(arr)):    ## finding the min values
            if arr[j] < arr[min]:

                min = j

        # find the smallest value
        arr[i], arr[min] = arr[min], arr[i]
```

V/s

```python
# bubble sorting

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]
```

1. $T(c) \Rightarrow O(n^2)$ — $T(c) = O(n^2)$
2. No. of Comparision $\Rightarrow 2n^2$ — Comparision $\hookrightarrow 2n^2$
3. No. of swapping $\rightarrow n$ time — Swapping $\hookrightarrow 2n^2$
4. Stability $\rightarrow$ Unstable — Stability $\hookrightarrow$ Stable

selection sort better Performance than bubble sort because, selection sort having less swapping Compare to bubble sort.

↳ In real world, (Practicaly)

Bubble, Selection } - Sort →
① Very expensive
② Slow

Use → fast → like → ① Merge sort
② Quick Sort

Insertion Sorting/ Online Sorting

Whenever you have incoming stream of data and regularly data coming in and you want to keep that data in sorted order. Here Insertion sort becames very handy.

for ex → zoom chat ↴

arrange the chat → Based on students name

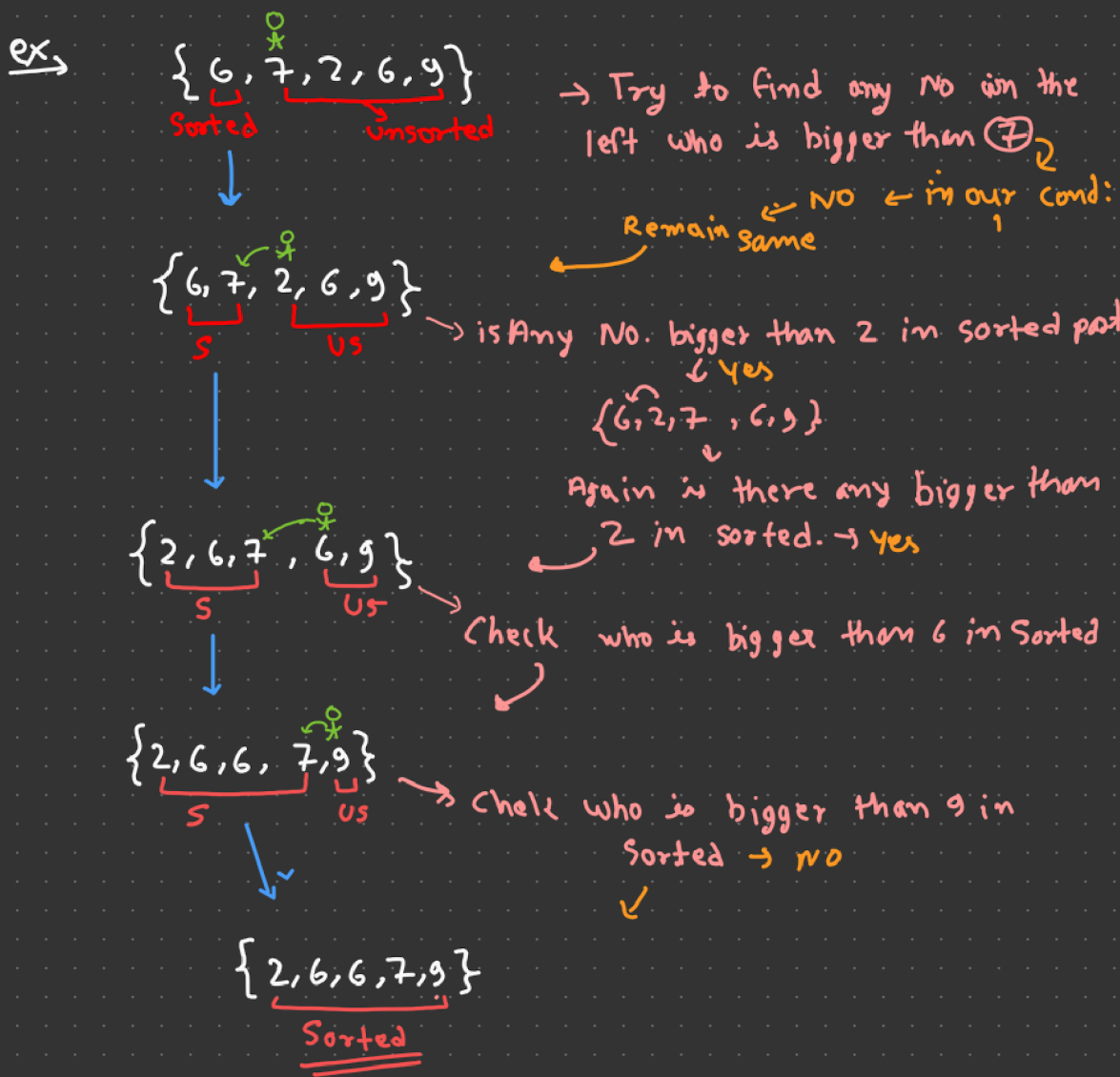
Algorithm that being used

insertion sort

definition:- Insertion Sort builds the sorted part of the list by repeatedly picking elements from the unsorted part and inserting them in the correct position within the sorted part.

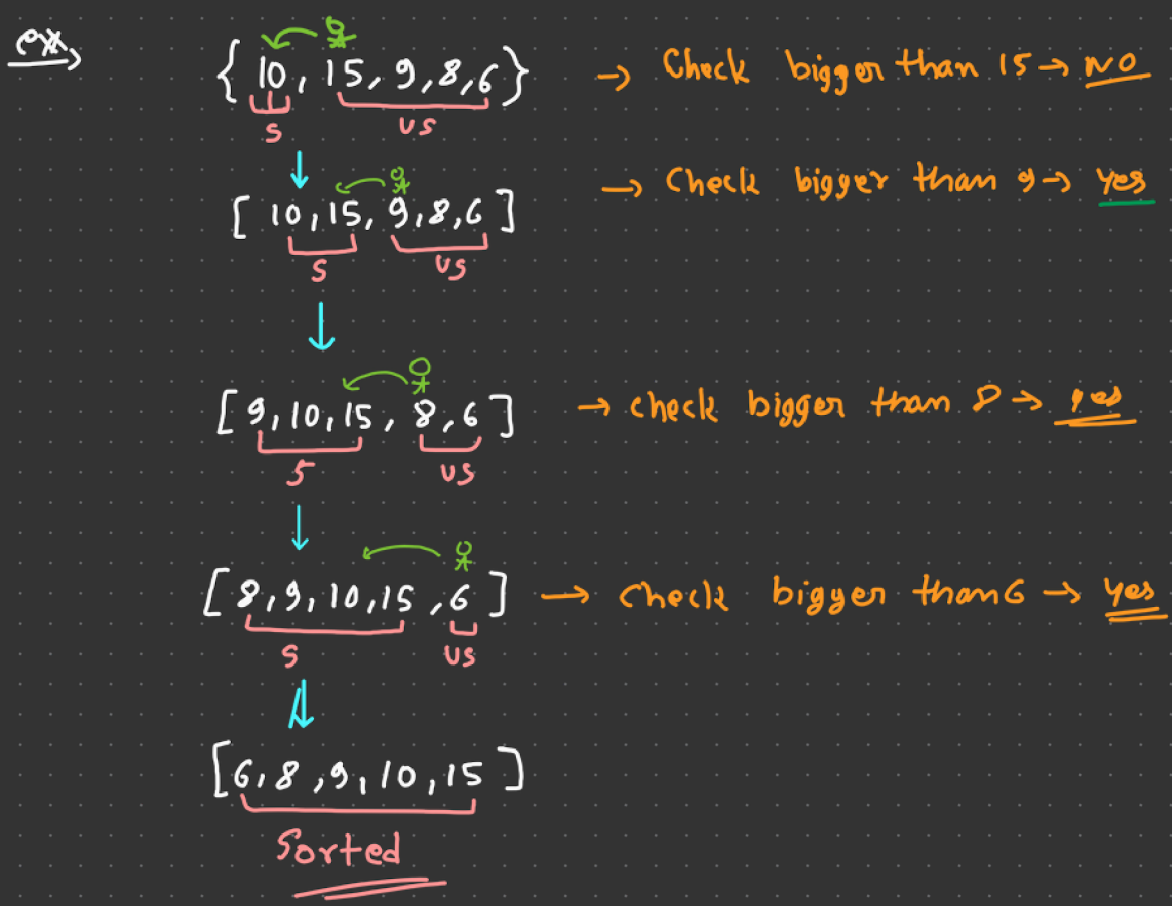
It is much less efficient on large list than more advanced algorithm such as quicksort, heapsort, mergesort, However it provide several advantages like :-

1. Simple implementation
2. Efficient for small data sets or nearly sorted.



Here we keep trying to inserting the element so that the array get sorted in spacific order

↳ Insertion / online sorting follow ⇒ Stable sorting



Code for insertion sort :

3. Online Sorting / insertion sort

```
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
```