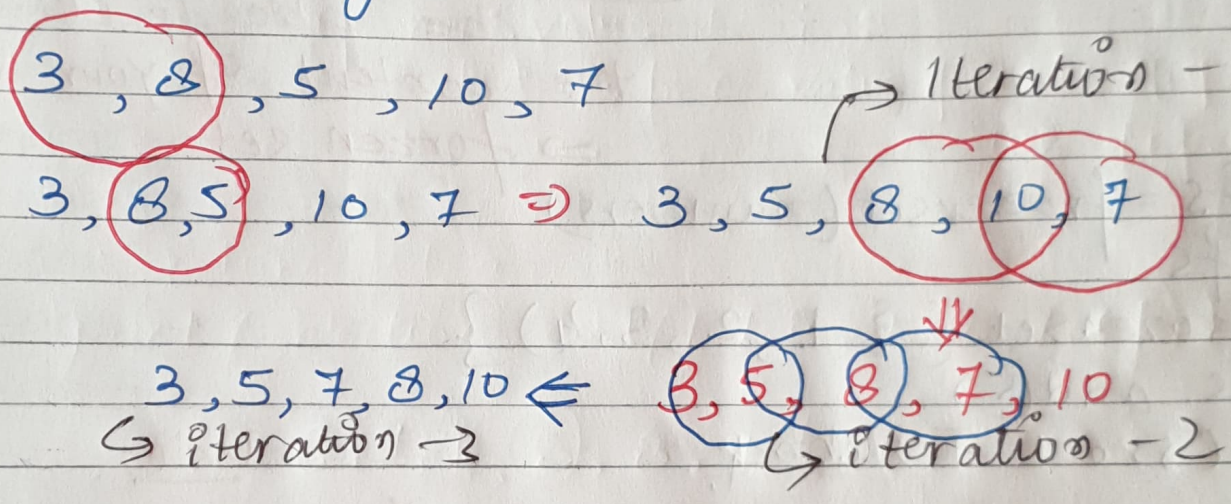
Data structure :-

Bubble Sort :- Takes an unsorted list & order them in ascending values. We have lowest no. at the begin & highest no. over the end of the list.

- We do this by comparing 2 elements in the given list at a time.



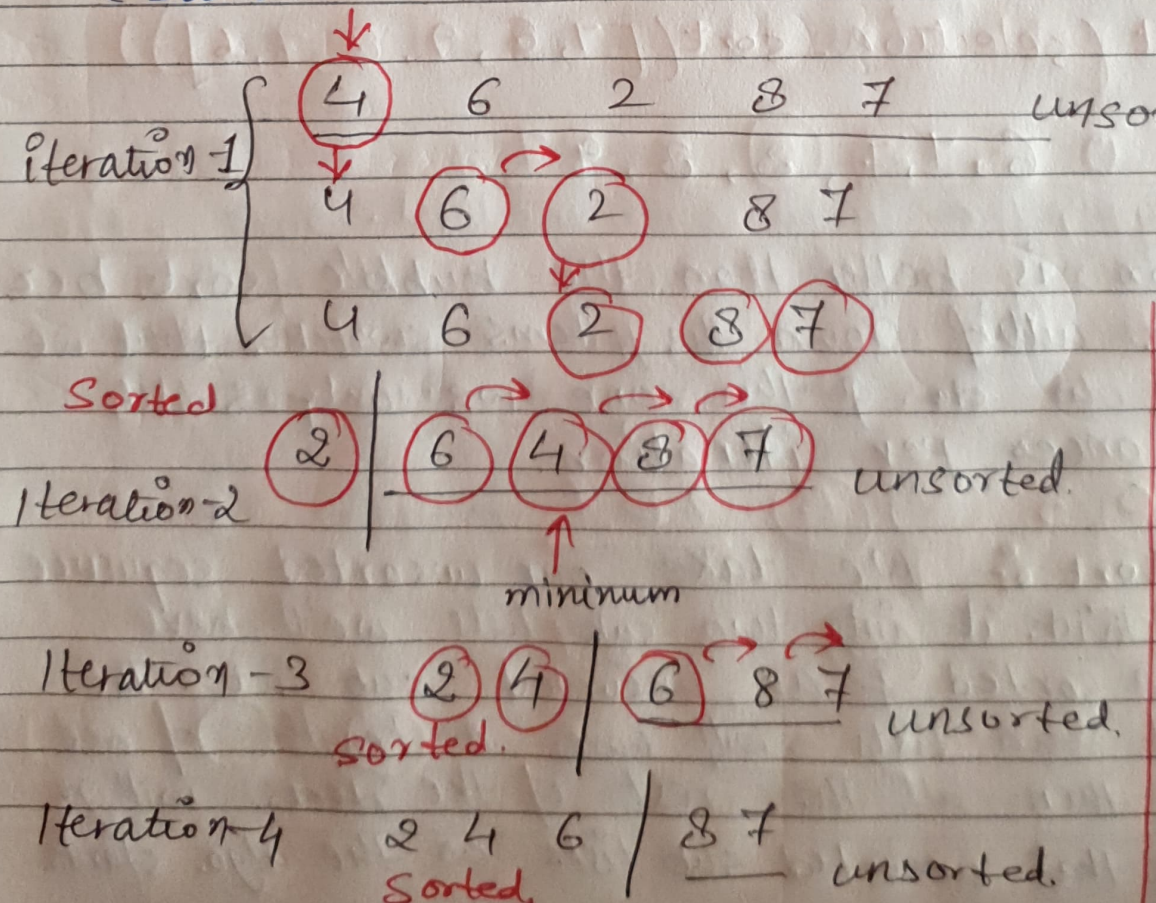
No switches = sorted.

```
>> def bubble(list_a):
       indexing_length = len(list_a) - 1
       sorted = False
       while not sorted:
           sorted = True
           for i in range(0, indexing_length):
               if list_a[i] > list_a[i+1]:
                   sorted = False
                   list_a[i], list_a[i+1] = list_a[i+1], list_a[i]
       return list_a
```

>> print(bubble([4, 6, 8, 3, 2, 5, 7, 8,]))

[2, 3, 4, 5, 6, 7, 8, 8, 9]

Selection Sort Algorithm:- We're trying to find the minimum value in a list. So, we are setting the first number as minimum & then comparing it to every number to its right. As soon as we come across the no. that is lower than the minimum, we assign that as the new minimum & continue on ~~the~~ at the end of ~~the~~ our iteration, our min. value will be moved to the left once we have that min. value we can divide our list into 2 sublists.



Sorting Algorithm :-

sort() : built-in list method
sorted() : built-in function

Selection sort Algo :- in-place comparision based algorithm.

Steps: (1.) Search the list & find out the minimum value.

5	15	3	12	17	0

- We are considering the no(s) in ascending order
- We need to find the min. value from list.
  - ↳ 1.) min()
    2.) take first no. as min. no. [0] & compare it with other values in the list. (update)

(2.) Swap the smallest number to 0th index.

0	15	3	12	17	5

unsorted.

↳ repate step-1 & then step-2

* if (descending order)
  ↳ then instead of finding min. value, find max. value & place that in the [0] position.

  max() or compare & update

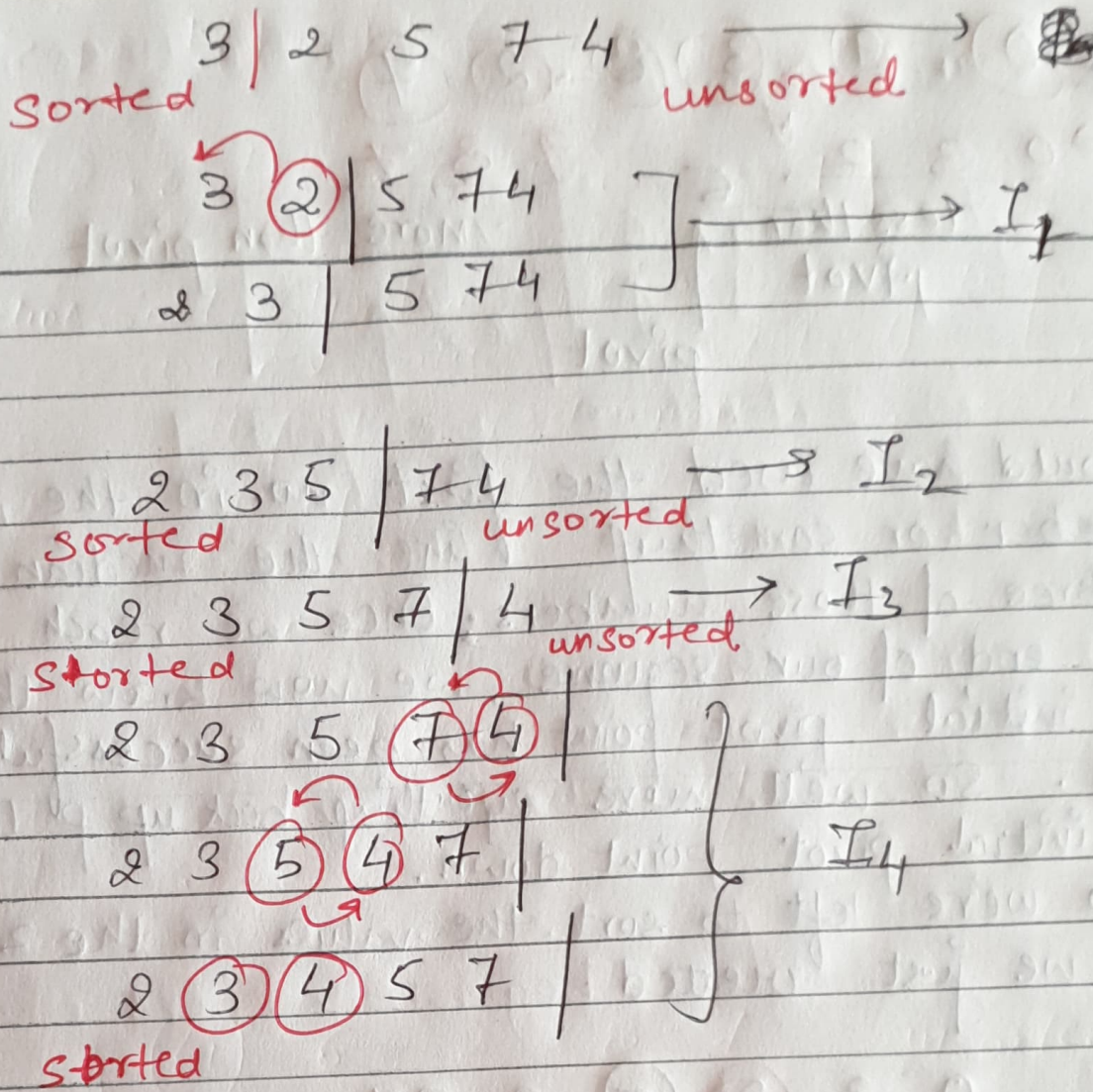
(3.) Repeate step 1 & 2 until list is completely sorted.

```
>> def selection_sort(list_a):
    indexing_length = range(0, len(list_a)-1)

    for i in indexing_length:
        min_value = i
        for j in range(i+1, len(list_a)):
            if list_a[j] < list_a[min_value]:
                min_value = j

        if min_value != i:
            list_a[min_value], list_a[i] =
                list_a[i], list_a[min_val]

    return list_a
>> print(selection_sort([7, 8, 9, 8, 7, 6]))
```

6, 7, 7, 8, 8, 9

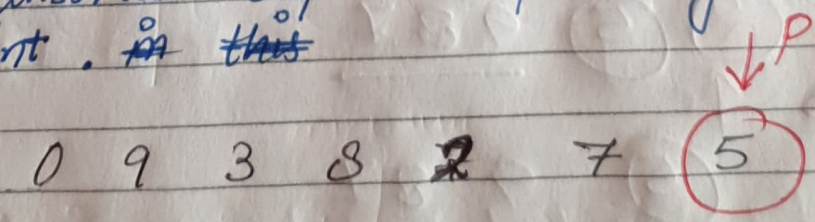
Selection is better than the bubble sort bez we're cutting down the #items switches that we need to do & this one we only switch onces per iteration.

Insertion Sort :- We take our unsorted sequence & divide it into two sub lists we'll have a sorted & an unsorted sub list. The sorted sub lists will have a length of one & then all the rest of the items will go into that unsorted sub lists as the algo. starts.

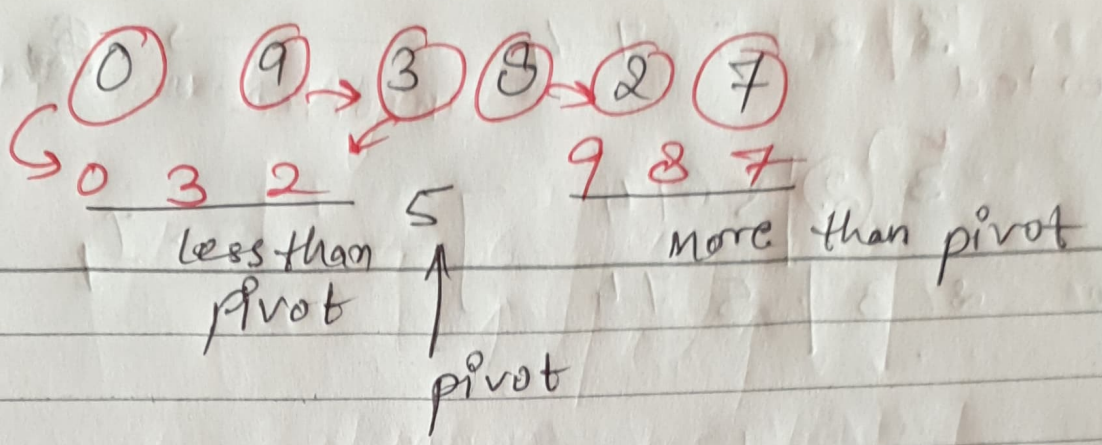


Quick Sort:- Quick sort algorithm takes an unsorted sequence & sorts it out in either ascending or descending order.

- We start off by taking one item in our ~~unorder~~ unsorted sequence & making that our pivot point. ~~in this~~



The pivot is just the no. that we ~~want~~ to base the comparisons of all the other numbers off of. ~~we~~

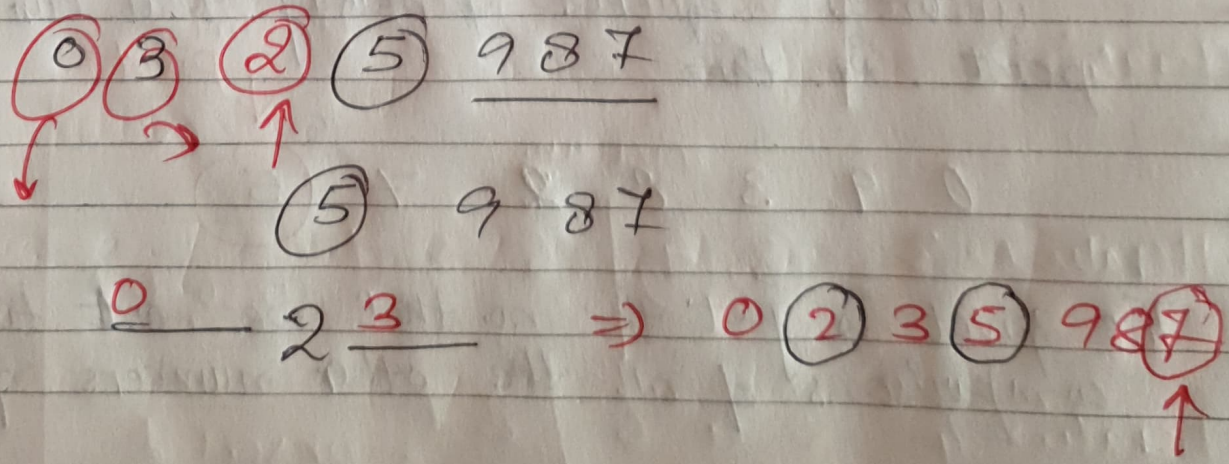


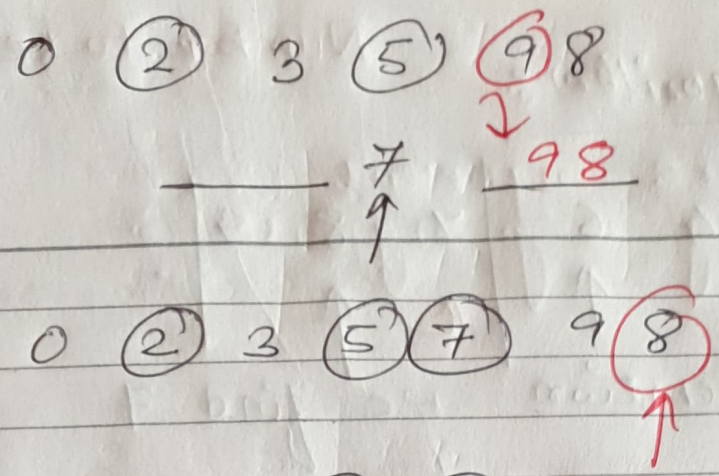
We could have used the first no. in the middle number or something like the median of all three of those numbers; Once we reach ~~add~~ the end of our sequence, we now know that the initial pivot point has been successfully sorted. So we'll move everything back up & lock that initial pivot point down.

- Now we're left to sort the values in the 2 lists that we just created.

0 3 (2) (5) 9 8 7

We can apply the above same method. Since we're are using the last no. from each of these two new lists & use that as the pivot point to sort out ~~this~~ its own list value. ~~were~~





We see that 7 was unable to break up those two higher values in the list.

⇒ another iteration

⇒ 0, 2, 3, 5, 7, 8, 9

inside ( ), we will take seq. of unsorted values

```
>> def quick_sort(sequence):
    length = len(sequence)
    if length <= 1
        return sequence
    else:
        pivot = sequence.pop()

    items_greater = []
    items_lower = []
    for item in sequence:
        if item > pivot:
            items_greater.append(item)
        else:
            item_lower.append(item)
    return quick_sort(item_lower) + [pivot] + quick_sort
                                              (item_greater)
>> print(quick_sort([5,6,7,8,9,8,7,6,5,6,7,8,9,0]))
```

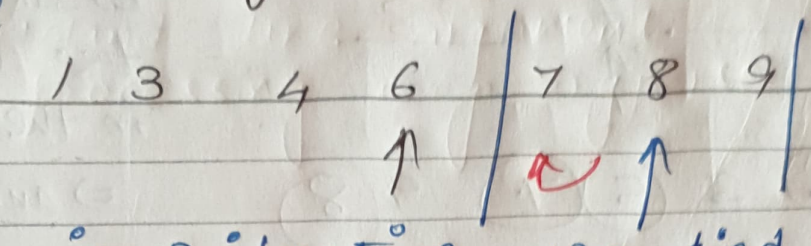
Result:

[0, 5, 5, 6, 6, 6, 7, 7, 8, 8, 8, 9, 9]

Binary Search Algorithm :-

Number to search : 7

↳ sorted



in 3 iteration we find 7

Search = = Midpoint = Return Position

```
>> def binary_search(sequence, item):
       begin_index = 0
       end_index = len(sequence) - 1

       while begin_index <= end_index:
           midpoint = begin_index + (end_index - begin_index)//2

           midpoint_value = sequence[midpoint]
           if midpoint_value == item:
               return midpoint
           elif item < midpoint_value:
               end_index = midpoint - 1
           else:
               begin_index = midpoint + 1
       return None

>> sequence_a = [2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14]
>> item_a = 12
>> print(binary_search(sequence_a, item_a))
   8
```