Insertion Sort

Algorithm InsertionSort(A):
Input: An array A of n comparable elements
Output: The array A with elements rearranged in nondecreasing order
for k from 1 to n − 1 **do**
Insert A[k] at its proper location within A[0], A[1], . . ., A[k].

Code Fragment 5.9: High-level description of the insertion-sort algorithm.

Sequence → unsorted ← 3 2 5 7 4

- We will divided it into two sub-list

3 | 2 5 7 4
"Sorted" " unsorted List"

Visually

As the Algorithm starts, we will take the item in the very Left position of the unsorted sequence

3 | (2) 5 7 4
Sorted Unsorted List

- and move it into the sorted sub list

3 2 | 5 7 4
Sorted Unsorted List

- once it's in the sorted sublist we will Compare that Value to the Value of it's Left

(3)(2) | 5 7 4
Sorted Unsorted List

If the Value is higher, then we change position of these two items

Compare

(3)(2) | 5 7 4
Sorted Unsorted List

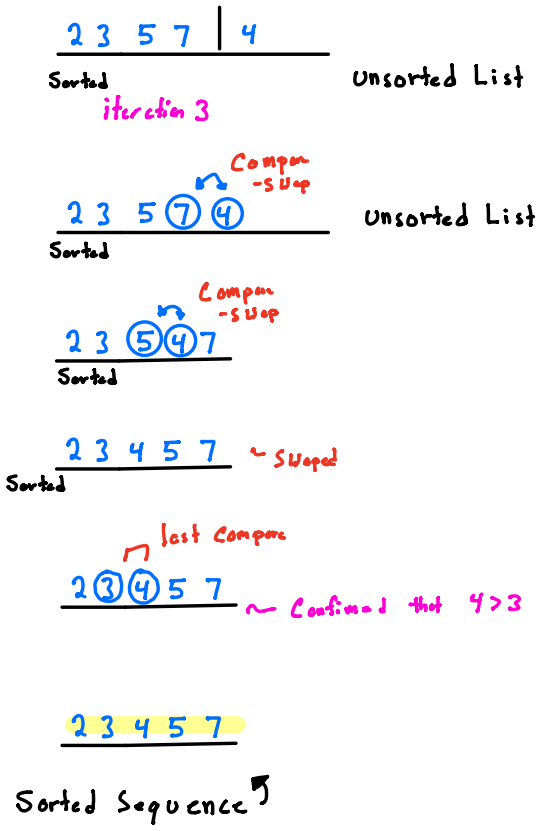
Changed position of two items

2 3 | 5 7 4
Sorted Unsorted List

iteration 1

- We will Continue down the sorted sublist doing this over and over until we find an item that is not higher than the item were trying to sort

2 3 5 | 7 4 ✓
Sorted Unsorted List
iteration 2



Python Code:

list to be sorted

```
def insertion_sort(list_a):

    indexing_length = range(1, len(list_a))

    for i in indexing_length:

        value_to_sort =list_a[i]

        while list_a[i-1] > value_to_sort and i>0: # python allows negative indexing

            list_a[i], list_a[i-1] = list_a[i-1], list_a[i]

            i = i -1

    return list_a

print(insertion_sort([2,12,123,122,42,312,42,124,9,10,1,2,3,45,9,0]))
```

length of List passed thru

Basically how many iterations we need to go thru the unsorted list

i will start @ 1, since we started our indexing from 1 to end of the list_a.

Check each Value in our Un sorted list

Value will switch position

Output:

```
[0, 1, 2, 2, 3, 9, 9, 10, 12, 42, 42, 45, 122, 123, 124, 312]
```

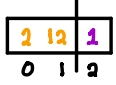
How the Algorithm works.

```
def insertion_sort(list_to_sort):
    indexing_length = range(1, len(list_to_sort))
    for i in indexing_length:
        value_to_sort = list_to_sort[i]

        while list_to_sort[i-1] > value_to_sort and i>0: # python allows negative indexing
            list_to_sort[i], list_to_sort[i-1] = list_to_sort[i-1], list_to_sort[i]
            i = i -1  # move down the sorted list to check if number is less than the previous number
    return list_to_sort


print(insertion_sort([2,12,1,0,42,312,42,124,9,10,1,2,3,45,9,0]))
'''output'''
[0, 0, 1, 1, 2, 2, 3, 9, 9, 10, 12, 42, 42, 45, 124, 312]
```

Index 0 1 2 3 4

iteration 2

Sorted list RN



i in indexing_length = 2

i = 2

```
value_to_sort = list_to_sort[i]
```

value_to_sort = 1

```
while list_to_sort[i-1] > value_to_sort and i>0:
```

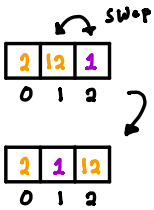
i-1 = 2

12 > 1 and i > 0 TRUE

```
list_to_sort[i], list_to_sort[i-1] = list_to_sort[i-1], list_to_sort[i]
```

i = 2, i-1 = 1

swap




```
i = i -1
```

i = 2-1

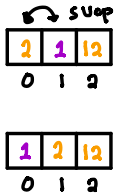
i = 1

```
while list_to_sort[i-1] > value_to_sort and i>0: # python
```

2 > 1 and i > 0 ✓

```
list_to_sort[i], list_to_sort[i-1] = list_to_sort[i-1], list_to_sort[i]
```

swap




```
i = i -1
```

i = 1-1 = 0

Fails

```
while list_to_sort[i-1] > value_to_sort and i>0: # python
```

..... Next iteration