Terms, Concepts, and Examples

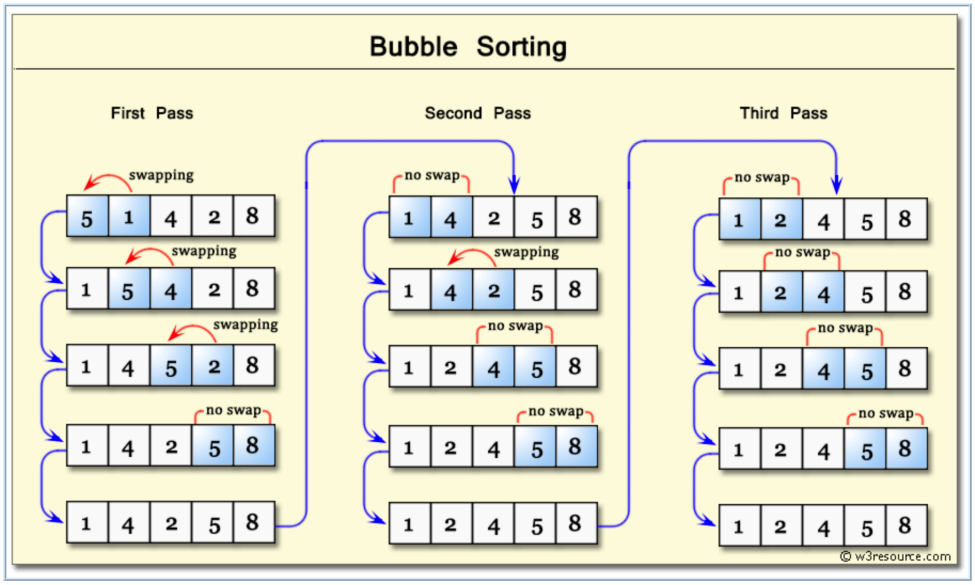
- **Sorting Algorithms** - Remember when using Binary Search the list had to be ordered already? Another reason for sorting is producing a directory of songs available for downloading requires that their titles be put in alphabetic order. More than 100 sorting algorithms have been devised, and it is surprising how often new sorting algorithms are developed. People study sorting algorithms for lots of reasons; some algorithms are easier to implement, some algorithms are more efficient (either in general, or when given input with certain characteristics, such as lists slightly out of order), some algorithms take advantage of particular computer architectures, and some algorithms are particularly clever.

  - The **Bubble Sort** is one of the simplest sorting algorithms, but not one of the most efficient. It puts a list into increasing order by successively comparing adjacent elements and swapping them if they are in the wrong order.

```
def BubbleSort(a):
    n = len(a)
    for i in range(0, n):
        for j in range(0, n - i - 1):
            if a[j] > a[j+1] :
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
    return a
```

Bubble Sort Dance - A visual example of performing a bubble sort.
Bubble Sort on Python Tutor

Example: Use Bubble Sort to order the list $[85, 12, 59, 45, 72, 51]$.
Image from w3resource



Notice the important pieces:

- It passes through the entire list swapping when necessary.
- It starts again at the beginning on the next pass.
- The list at the end is sorted.

Video Example of Tracing Bubble Sort

- The **Insertion Sort** is also a simple sorting algorithm, but again is not very efficient. To sort a list, the insertion sort begins with the second element. The insertion sort compares this second element with the first element and inserts it before the first element if it does not exceed the first element and after the first element if it does. At this point, the first two elements are in the correct order. The third element is then compared with the first element, and if it is larger than the first element, it is compared with the second element; it is inserted into the correct position among the first three elements.

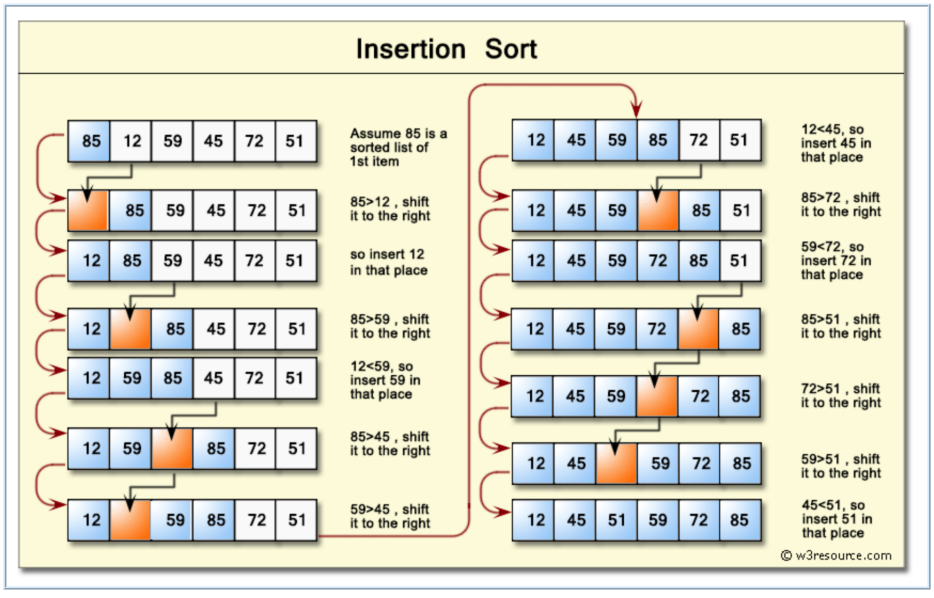
```
def insertionSort(a):
    for i in range(1,len(a)):

        currentvalue = a[i]
        position = i

        while position>0 and a[position-1]>currentvalue:
            a[position]=a[position-1]
            position = position-1

        a[position]=currentvalue
```

Insertion Sort Dance - A visual example of performing an insertion sort

Insertion Sort Dance - Another visual example of performing an insertion sort

Example: Use Insertion Sort to order the list $[85, 12, 59, 45, 72, 51]$.

Image from w3resource



Notice the important pieces:

- At the beginning it is puts the first element in holding.
- It shows which elements are being compared each time.
- It gives the state of the list after each new element gets sorted.
- The list at the end is sorted.

Video Example of Tracing Insertion Sort

Practice Problems

1. Use Bubble sort and "trace the algorithm" to sort the following lists.
   (a) $[6, 2, 3, 1, 5, 4]$
   (b) $[38, 12, 53, 75, 49]$
2. Use Insertion sort and "trace the algorithm" to sort the following lists.
   (a) $[6, 2, 3, 1, 5, 4]$
   (b) $[38, 12, 53, 75, 49]$
3. Give the output of the following algorithm when the list L = [6, 9, 10, 17, 23] is received as input. (Hint - use Python Tutor and run the algorithm to see the return value.)

```
def MyProcess(L):
    for i in range(0,len(L)):
        k = L[i]
        j = i - 1
        while j >= 0 and L[j] < k:
            L[j+1] = L[j]
            j = j - 1
        L[j+1] = k
    return L
```