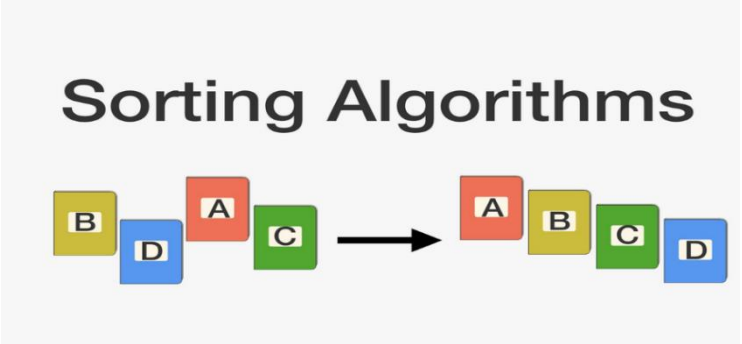
Sorting Algorithms



This is a topic found in almost **all introductory computer programming courses**. Through this section you will learn common algorithms used to sort numerical data.

"Sorting" is really just: **arranging the data in ascending or descending order**; however, this can be done in many different ways. Each way can be applied in different situations so that you can minimize:

a) the **time** taken to sort the given data.
b) **Memory Space** required to do so.

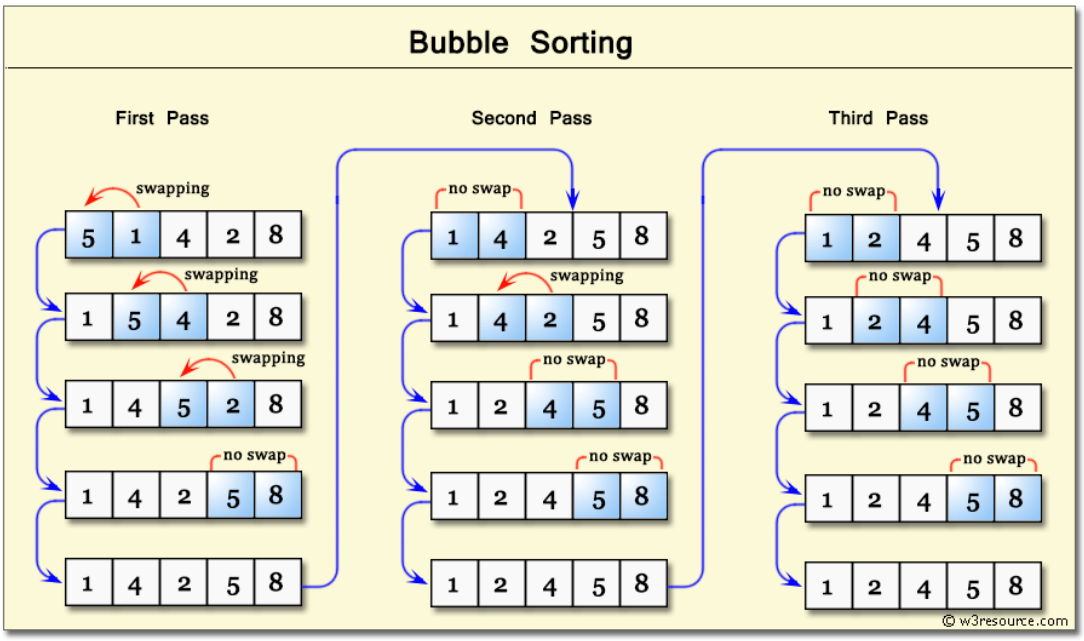
In today's world, developing a sorting algorithm from scratch could be replaced by cutting and pasting from other code or by using a simple one line function such as `sort()` ...So why is this usually a required section of introductory computer programming courses?

It's critical to look under the hood and understand how stuff works. Learning sorting algorithms exposes you to important ideas and techniques essential to becoming a better problem solver in programming.

Integrating, modifying, or referencing sorting algorithms is very common in computer programming.

Here are the names of common sorting algorithms.

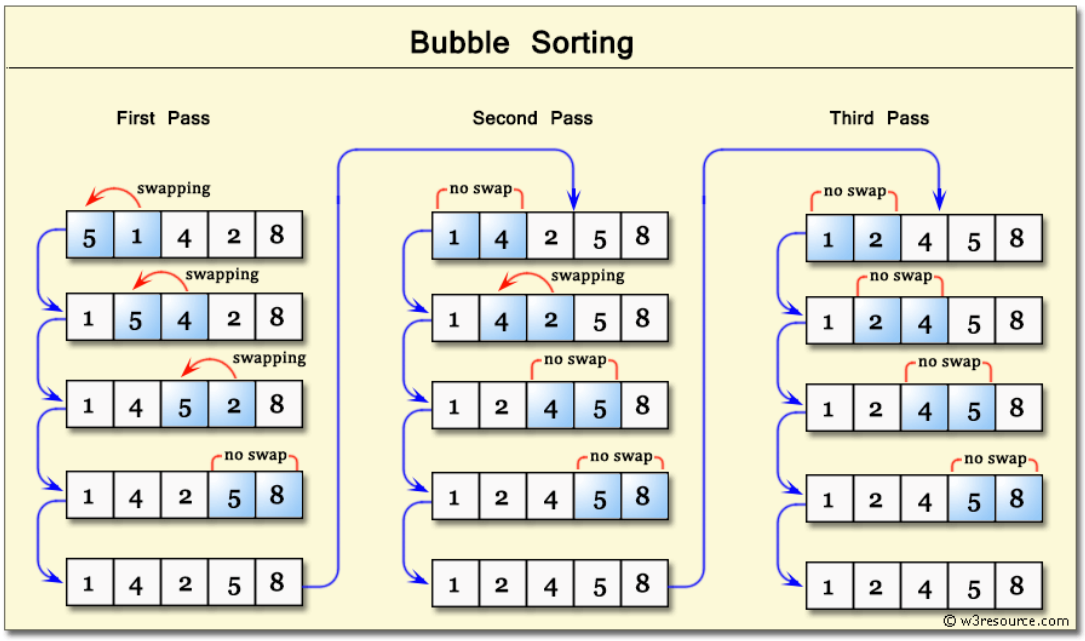
Bubble Sort
Insertion Sort
Selection Sort
Quick Sort
Merge Sort
Heap Sort



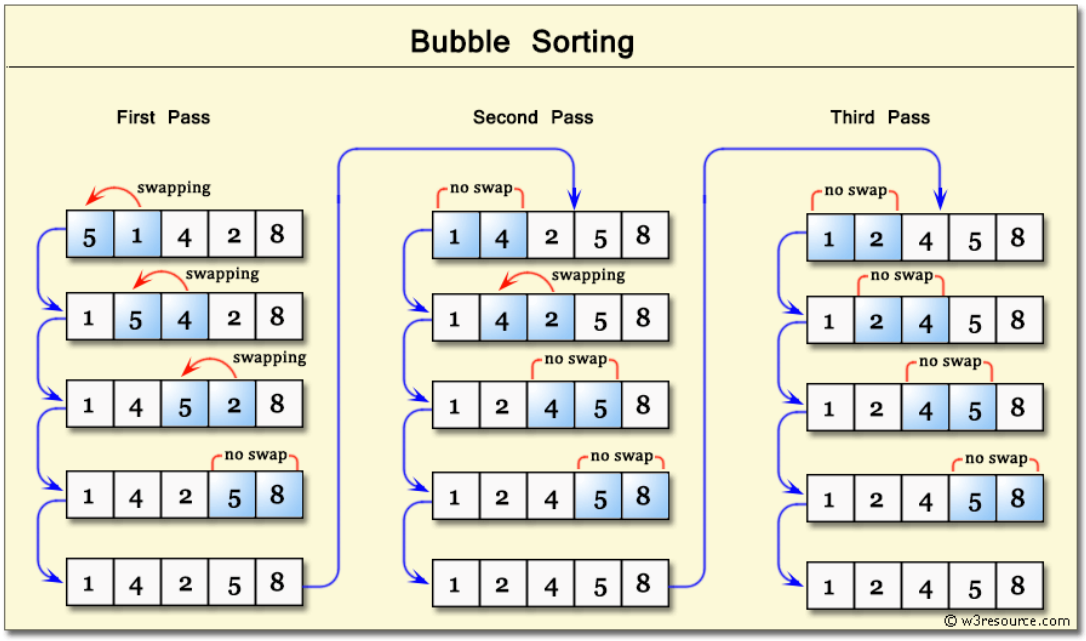
Bubble Sort

Probably the easiest sorting algorithm. Given a list of data a bubble sort is done by:

1. compares the first two numbers (which one is bigger)
2. swaps them if they need to be
3. compares the next pair of numbers
4. this is repeated until no swaps need to take place



This algorithm is called a **Bubble sort**, because with each iteration the largest element in the list *"bubbles up"* towards the last place, just like a water bubble rises up to the water surface!



PLEASE NOTE

Number of **Passes** and **Comparisons** needed:

If you are able to interpret the algorithm correctly you will see that:

1. After the first **pass** through the list, the **largest** element will **always** be moved all the way to the end. This means a maximum of **passes** will be necessary to sort *any* list is (the length of the list -1).

2. Also, if each **pass** pushes one more element into its correct position in the list, each **pass** will require 1 less **comparison**.

Result:

1. The maximum number of **passes** required is equal to:

[number of elements in the list] - 1 `len(listA)-1`

2. The number of **comparisons** necessary *during* each **pass** will decrease by 1 after every pass.

```
comparisons = comparisons - 1
```

Exercise #1

Look at the following code carefully. Notice:

1. There is a `for` loop within another `for` loop (a "nested" for loop)…what does this mean?
2. Each **pass** is represented by **x**
3. The total number of passes is: `(len(nlist)-1)`
4. For each **pass** we do another `for` loop of several **comparisons**.
5. During each pass we reduce the number of comparisons by 1
   `comparisons=comparisons-1`

```
nlist = [14,46,43,5,57,41,45,21,100]

comparisons=(len(nlist)-1)

for x in range(len(nlist)-1):
  for i in range(comparisons):
    if nlist[i]>nlist[i+1]:
      nlist[i],nlist[i+1]=nlist[i+1],nlist[i]
      comparisons=comparisons-1

print(nlist)
```

Each **pass** through the array

Each **comparison** *during* each pass

The actual comparison

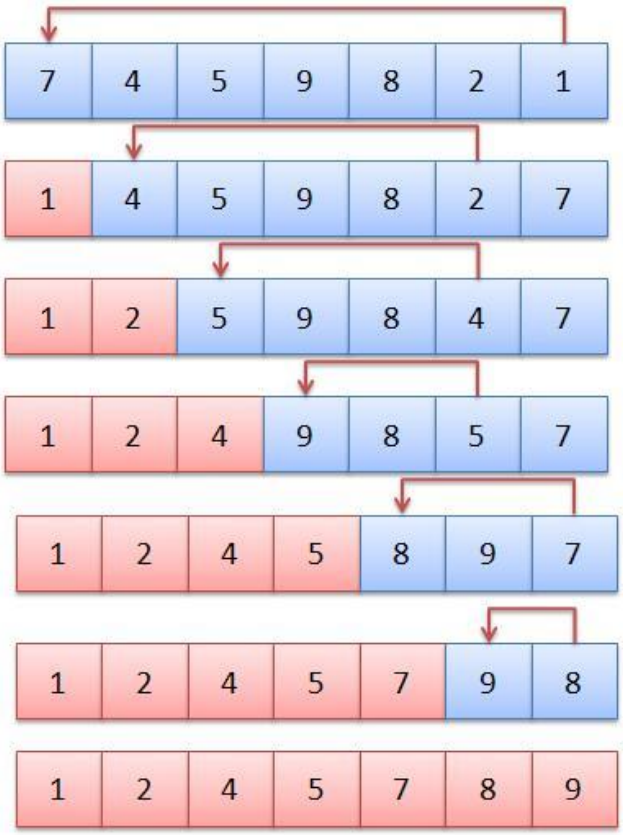
The **"swap"**

Now… it is your turn: *Exercise#1*

- Put it into an IDE to test it out.
- **Carefully review the code above** to make sure you fully understand the all steps and correct method of a bubble sort.
- ***Create your own bubble sort program from scratch without looking at the above example***
- Use different variable names.
- Obviously, your program should be identical or very similar to the one above.

Selection Sort Algorithm

(grab the smallest put it at the end, repeat)



A **Selection Sort** algorithm puts a list of data in order by doing the following:

1. **Finds the smallest element in an array** and then **puts it in the first position** of the array.
2. The **value that *was* in the first position is now placed where the smallest number came from**.
3. The first step is repeated (in the **unsorted data that remains**). The smallest value is always placed next to the previously designated smallest number.

Possible Solution:

```
A = [1,3,5,56,89,34,23,8]

for i in range(len(A)):

    min= i

    for j in range(i+1, len(A)):
        if A[min] > A[j]:
            min = j

    #swap values
    A[i], A[min] = A[min], A[i]
print(A)
```

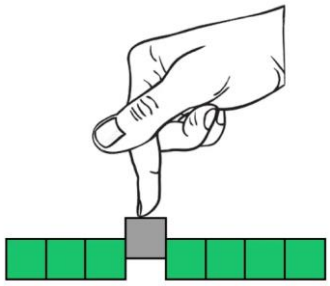
Nested For loop
(One For loop *inside* another)

Exercise #2

Now… it is your turn:

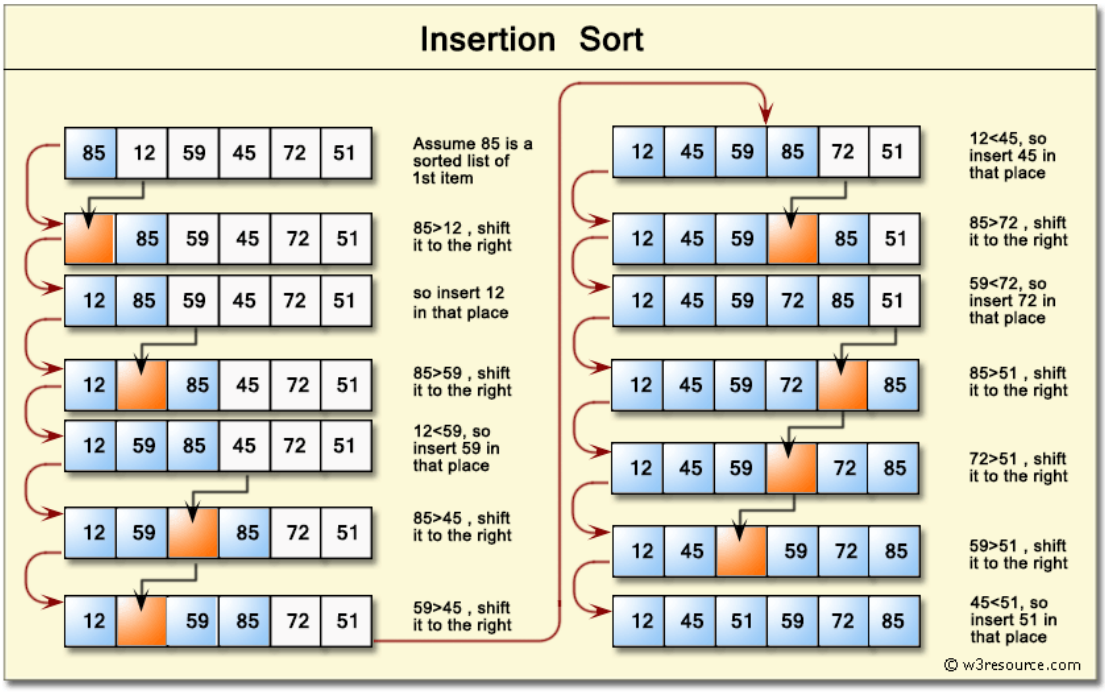
- Put it into an IDE to test it out.
- **Carefully review the code above** to make sure you fully understand the all steps and correct method of a bubble sort.
- ***Create your own Selection Sort program from scratch without looking at the above example***
- Use different variable names.
- Obviously, your program should be identical or very similar to the one above.

Insert sort



This Sorting Algorithm works as follows:

1. Assume the first element in the list is sorted.
2. Grab the next element in the list.
3. Compare the next element to ***all sorted*** elements and place it to the left of the highest value in the sorted elements.
4. Repeat steps 2 and 3 until you reach the last element in the list



Python Insert Sort

```
alist=[8,78,2,9,24,50,101,17]

for i in range(1,len(alist)):    # for loop grabs each element

        current = alist[i]       # set each element as current element

        while i>0 and alist[i-1]>current: # run each element through
            alist[i] = alist[i-1]         # the following while loop:
            i = i-1
            alist[i] = current

print(alist)
```

While loop:

If the current element is greater than the element to it's left, then shift the current element to the left again.
When you reach the left end of the list `i=0` or the current element's value is **greater** then the one to it's left, then jump out of the while loop and move on the next unsorted element.

Exercise #3

Now… it is your turn:

- Put it into an IDE to test it out.
- Carefully review the code above to make sure you fully understand the all steps and correct method of a bubble sort.
- ***Create your own insert sort program from scratch without looking at the above example***
- Use different variable names.
- Obviously, your program should be identical or very similar to the one above.

Exercise #4

There are dozens of other common sorting algorithms. Use the list on the first page of this assignment or search up a 4th sorting algorithm and create a python version of it. Save it and submit it. In the comments section of the code please see if you can explain the algorithm and how the code is working.