# Algorithms for Data Science
# Lab 2
# Sort Running Times

In this lab, you will be testing the running times of three sort implementations: mergeSort, insertionSort, and bubbleSort. Start by creating a single Python file. In it, define the three functions given above. Each should take in a single list of integers as a sole parameter. For example:

```
def mergeSort(L):
```

To test your sorts, create a list of *n* numbers, and put them in random unsorted order:

```
A = [i for i in range(n)]
random.shuffle(A)
```

Pass this list to each sort, and make sure the results come back sorted. An *n* of 10 is a good testing length. You don't need to re-create the list each time, but you certainly should reshuffle between tests (some sorts work much faster than normal if the list is already sorted!).

In the video/PowerPoint slides, you will find code for mergeSort and insertionSort. You will, however, need to code bubbleSort on your own. A bubble sort works as follows: On a single bubble pass, you run through all the elements in your list from front to back. At each index you compare its value to the value of the next one. For example, if i = 4, then you would compare the value at 4 to the value at 5. If they are out of order, then you swap the values. At the end of one bubble pass, the biggest element will have "bubbled" to the end of the list. To get all elements in order, you make *n* bubble passes.

After you have coded all three and made sure they work, you will perform a timing test. To time a sort, call the time() function (from time import time) right before and right after the sort call. Then subtract the values and multiply by 1000 to get the results in milliseconds. For example:

```
t1 = time()
B = mergeSort(A)
t2 = time()
mtime = (t2-t1)*1000
```

You are to time all three sorts with *n* values from 100 to 5000, in 100 increments. Display these values in a table like the following (you might want to display with tabs ("\t") and round your results):

```
N       Merge   Insert  Bubble
100      0.3     0.4     0.7
200      0.6     1.4     2.7
300      0.8     2.7     5.5
400      1.1     4.3     9.8
```

| | | | |
|---|---|---|---|
| 500 | 1.4 | 9.3 | 16.5 |
| 600 | 1.7 | 12.1 | 22.5 |
| 700 | 2.1 | 14.5 | 31.0 |
| 800 | 2.3 | 18.2 | 48.3 |
| 900 | 2.5 | 22.6 | 56.4 |
| 1000 | 2.8 | 28.2 | 62.0 |
| … | | | |