Week 07 Part Searching and Sorting

Sorting

You are giving a list `lst` of `n` numbers. If you are given an index `i` for `0 < i < n`. We find out which of the two numbers `lst[i-1]` and `lst[i]` is bigger. And, we will swap the bigger one to the right, such that `lst[i] > lst[i-1]`. If they are equal, just let it be.

Write a simple loop to do the above task from `i = 1` to `i = n-1`. (Note that the number of times is `n-1` but not `n`.) And let's call this function `bubble(lst)`. Your function should run like this:

```
>>> L = [4,5,6,7,1,2,3,9,8]
>>> L1 = bubble(L)
>>> L1
[4, 5, 6, 1, 2, 3, 7, 8, 9]
>>> L2 = bubble(L1)
>>> L2
[4, 5, 1, 2, 3, 6, 7, 8, 9]
>>> L3 = bubble(L2)
>>> L3
[4, 1, 2, 3, 5, 6, 7, 8, 9]
```

Did you notice something? What happens if this `bubble()` is applied to a list many times?

How many times do I need to apply `bubble()` to a list in order to make the list totally sorted?

Write a function `bubbleSort(lst)` to return a list that is sorted. Here is some sample output, in which, you should be able to change n to a larger number and the sorting still work.

```
>>> from random import randint
>>> n = 20
>>> L = [randint(0,10000) for i in range(n)]
>>> print(L)
[8753, 4935, 9379, 7034, 515, 854, 7747, 3661, 9932, 1590, 8123, 3924, 9565, 469
9, 6735, 1109, 9955, 1600, 2481, 9363]
>>> print(bubbleSort(L))
[515, 854, 1109, 1590, 1600, 2481, 3661, 3924, 4699, 4935, 6735, 7034, 7747, 812
3, 8753, 9363, 9379, 9565, 9932, 9955]
```

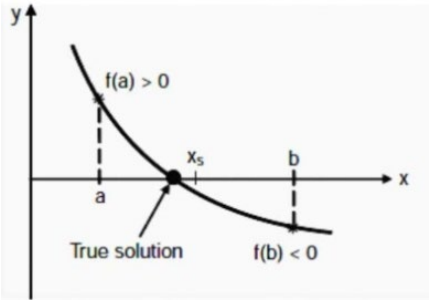
Final thoughts

Do you really need to:

- Apply so many times?
- For the whole list?

Searching: Bisection Method

Note that in this question, you cannot use the 'Newton Method' in the lecture slides. You have to follow the bisection method.



The ***bisection method*** in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. Given a function f(x), you want to solve for x when f(x) = 0.

In this question, we assume that the function f is continuous. And you are given two numbers a and b such that f(a) > 0 and f(b) < 0. So you repeat the following

- Compute x_s = (a + b) /2
- If $f(x_s) < 0$ then the solution lies on the of x_s. So, b = x_s.
    - Otherwise, a = x_s.
- Repeat these until the absolute value of $f(x_s)$ is smaller than a constant 'error'

After the program exits the loop, the value of x_s should be a very close answer for f(x) = 0. And, if f(a) < 0 and f(b) > 0, you have to reverse the sign for the above algorithm. Also, if both f(a) and f(b) have the same sign, you can just "give up" because there may be no solution that the bisection method may enter an infinite loop.

You are given two functions f and g to solve by bisection method.

```
def g(x):
    return (x/40)**5 - 8*(x/40)**3 + x/4 + 6

f = lambda x : x**3  - 10*x + 4
```

Write a function '`bisection(start,end)`' to solve x for f(x) = 0 and g(x) = 0 with bisection method. (In general, you should be able to solve any continuous function with f(a) and f(b) giving different signs.) The value of the '`error`' is given to you. The roots for f(x) = 0 and g(x) = 0 are -3.3459632955491543 and 63.39649252593517 respectively. (Your numbers could be a bit off. However, `abs(f(x))` should be less than '`error`'

Advance Sorting (Extra)

This one is totally extra to this course but it would be interesting to try different sorting methods.

You are given a list `lst` of n numbers. To simplify the problem, we assume we have no duplicate element in the list. (However, even so, it is not difficult to solve.)

We will pick any element of the list, say `x`. And for the rest of the element in `lst`, we will separate them into two lists, one list `lsta` contains all the element smaller than `x`, and `lstb`, otherwise. Let's give a name to this functionality as "partition".

Then we apply some "magic" to `lsta` and `lstb` such that they are sorted after the magic. Finally, we output the list `lsta + [x] + lstb`.

What is that magic? And what is the magic we have done?