Topics for this week

- Searching: linear search, binary search and their time complexities

- Sorting - iterative: InsertSort, BubbleSort (on the homework), more sorts to come next quarter

- More on complexity analysis: $O(1)$, $O(\log n)$, $O(n)$, $O(n^2)$
Problem: Sort a list of numbers

Write a function that takes as input a list of numbers and it outputs the same list with the numbers listed in increasing order.

Sort the list in place (i.e., do not use other lists).

idea: given a sorted list, we’ll insert the next element at the proper position

\( \text{e.g. } L = [1, 3, 5, 6, 9, 10] \) insert 4

we’ll do this in-place, without creating a new list: \( L = [1, 3, 5, 6, 9, 10, 4, \ldots] \)

i.e. we’ll shift all the elements that come before 4 by one position to the right:

\( [1, 3, 5, 6, 9, 10, 4, \ldots] \)

pseudo code:

```python
def InsertSort(L):
    for i going through 0, 1, 2, ..., size of L - 1:
        k = i - 1, copy = L[i]
        while k >= 0 and L[k] > copy:
            L[k+1] = L[k]
            k = k - 1
        L[k+1] = copy
```
Problem: Sort a list of numbers

Time complexity: \(O(n^2)\)

let \(n\) be the size of \(L\)

```python
def InsertSort(L):
    for i going through 0,1,2,..., size of L-1:
        k = i-1, copy = L[i]
        while k \geq 0 and L[k] > copy:
            L[k+1] = L[k]
            k = k-1
        L[k+1] = copy
```

\(\text{TOTAL number of steps} \leq 5n^2 = O(n^2)\)

in the worst case (when the list is in the reversed order),
we need \(O(n^2)\) steps

Testing:

- test with lists:
  - [0]
  - [really long list of numbers]
  - [list of numbers in the reversed order]
  - [list of numbers in the sorted order]
  - [include negative numbers]
  - [duplicate numbers]
  - [3,1]
Problem: Search for a number in a list

[Warm-up: linear search.]

Write a function that takes as input a number x and a list of numbers L and it outputs the position of x in L or -1 if x is not in L.

e.g.  \[ L = [5, 2, 1, 7, 3, 8] \]  \hspace{1cm}  x = 3

\hspace{1cm}  x = 0

```python
def linear_search(L, x):
    i = 0
    while i < len(L):
        if L[i] == x:
            return i
        i = i + 1
    return -1
```

Time complexity: \( O(n) \) if the number x is not in the list L.
Problem: Search for a number in a list

[Binary search.]

Write a function that takes as input a number \( x \) and a sorted list of numbers \( L \) and it outputs the position of \( x \) in \( L \) or -1 if \( x \) is not in \( L \).

\[
L = [1, 3, 4, 7, 8, 9, 20, 23]
\]

\[
x = 20
\]

\[
x = 8
\]

def binSearch ( L, x, left, right ):
    if left \geq right:
        return -1
    middle = average of left and right (rounded down)
    if L[middle] == x:
        return middle
    elif L[middle] > x:
        return binSearch (L, x, left, middle - 1)
    else:
        return binSearch (L, x, middle + 1, right)

we're starting with:
left = 0
right = len(L) - 1
Problem: Search for a number in a list

Time complexity:

suppose we have 1024 elems.
in the first recursive call we eliminate at least half of the elems - remaining ≤ 512
second - remaining ≤ 256
third - ≤ 128
fourth - ≤ 64
...

if we have \( 2^m \) elems, we need \( m \) recursive calls, each call takes about 5 steps

\[ \rightarrow \text{TOTAL} \leq 5m \text{ steps: } O(m) \]

thus, if the number of elems is \( n \), then the running time is \( O(\log n) \)

Testing:

read the posted notes

- idea: test a few values in the list, e.g. the first elem, the last, the middle, and a couple random ones

- test values not in the list - one smaller than the first
- one larger than the last
- one between the first and the last elem.