Insertion Sort Analysis

for k from 0 to len(a)-2:
    assert_the_first_k_positions_are_already_sorted

    # Pick up item k+1 (call it a[j]) and let it sink
    j = k+1

    while j > 0 and a[j-1] > a[j]:
        swap(a[j-1], a[j])
        j = j-1

O(n)

\( \sum_{x=1}^{N} x = \frac{N(N+1)}{2} \)

\( \sum_{\text{loops}} = \sum_{i=1}^{n-1} i \)

\[ = \frac{(n-1)(n+1-1)}{2} \]

\[ = \frac{(n-1)n}{2} \]

\[ \in O(n^2) \]
Given $n$ digits $\Rightarrow n!$.

Items $\Rightarrow \log_2 M$ comparisons

$\log_2 n! = \lg n!$

$\lg n! = \lg 1 + \lg 2 + \lg 3 + \ldots + \lg n$

$\leq \lg n + \lg n + \lg n + \ldots + \lg n$

$= n \lg n$

$O(n \lg n)$
There are lots of these around. They tend to be easy-ish to think up. Be careful not to reinvent the wheel!

We're going to look at a few more so that you have been exposed to the common ones and you also get more practice analysing complexity.

You will find a lot of CS types scoff at these algorithms as being a waste of time. But they have definite advantages:

- They are usually concise and understandable (means fewer implementation bugs)
- For small n, they might be the quickest method!
Selection Sort: Idea

- Search for smallest
Selection Sort: Example

3 6 4 5 1 2
1 6 4 5 3 2
1 2 4 5 3 6
1 2 3 4 5 6
1 2 3 4 5 6
Selection Sort

for k from 0 to len(a)-1:
    assert (the positions before a[k] are already sorted)
    # Find the smallest element in a[k..end] and swap it into a[k]
    iMin = k
    for j from iMin+1 to len(a)-1:
        if a[j] < a[iMin]:
            iMin = j
    swap(a[k], a[iMin])

\[ \sum_{j=1}^{n} j = \frac{n(n+1)}{2} \in O(n^2) \]
Binary Insertion Sort: Idea

1. Binary chop to find where it will end up
2. Exchanges to get it there
**Binary Insertion Sort: Analysis**

**Comparisons**

Sorted

\[ \sum_{k=1}^{n} \frac{1}{k^2} \]

\( k^{th} \) element could end up in any of \( k \) positions.

\[ \log k \text{ comparisons} \]

\[ \sum_{k=1}^{n} \log k = \log 1 + \log 2 + \ldots + \log n \]

\[ \leq n \log n \]

\( O(n \log n) \)

**Swaps**

\[ \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \in O(n^2) \]
Binary Insertion Sort: Analysis
Bubble Sort

9    repeat
10       # Go through all the elements once, swapping any that are out of order
11           didSomeSwapsInThisPass = false
12       for k from 0 to len(a)-2:
13           if a[k] > a[k+1]:
14               swap(a[k], a[k+1])
15           didSomeSwapsInThisPass = true
16    until didSomeSwapsInThisPass == false

\[(n-1)n = n^2 - n \in O(n^2)\]
Better Algorithms

- Can we actually hope to get close to our theoretical $O(n \log n)$ limit?

- You know we can because you've met mergesort before.
  - In ML it was a lovely short algorithm that did a great job sorting lists
  - It has some drawbacks when we look at arrays but we'll get to that shortly...
Mergesort: Array Example

3 6 4 5 1 2
3 6
3 6 4 5 1 2
3 4 6
3 4 6 5 1 2
1 5
3 4 6 1 5 2
1 2 5
3 4 6 1 2 5
1 2 3 4 5 6
1 2 3 4 5 6
Mergesort: Analysis

\[ f(n) = 2f \left( \frac{n}{2} \right) + kn \]

\[ n = 2^m \]

\[ f(2^m) = 2f(2^{m-1}) + k \cdot 2^m \]

\[ = 2 \left[ f(2^{m-2}) + k \cdot 2^{m-1} \right] + k \cdot 2^m \]

\[ = 2^a f(2^{m-a}) + a \cdot k \cdot 2^m \]

\[ f(n) = nf(1) + m \cdot k \cdot 2^m \]

\[ = nf(1) + n \log n \]

\[ O(n \log n) \]

\[ O(n) \text{ space} \]