CS 173 Lecture 17b: Analyzing Iterative Algorithms

input: array of length n, indexed 1 ... n

```
MysteryOne(A[1..n])
1. for j from 2 to n:
2.     m ← A[j]
3.     if (m < A[j-1]):
4.         A[j] ← A[j-1]
5.         A[j-1] ← m
```

```
MysteryTwo(A[1..n]):
1. for j from 2 to n:
2.     m ← A[j]
3.     i ← j-1
4.     while (A[i] > m & i > 0):
5.         A[i+1] ← A[i]
6.         A[i] ← m
7.         i ← i-1
```

step: assignment ←
comparison (<, >, =)
booleans (or, and)

disappears into big O

assumption: a step takes some constant amt of time

Running time of MysteryOne (in Big O)?

```
MysteryOne(A[1..n])
1. for j from 2 to n:
2.     m ← A[j]
3.     if (m < A[j-1]):
4.         A[j] ← A[j-1]
5.         A[j-1] ← m
```

lines 2-4 execute at most n-1 times, due to for-loop in line 1.

So running time is (n-1)(running time of 2-4),

lines 2-3 always execute.

lines 4-5, if they do execute, add const running time for each execution of the loop.

no matter what, lines 2-5 give constant running time in each execution of for loop.

on input [1,2,3,4,5]
line 3 is always "false"

on input [5,4,3,2,1]
line 3 is always "true"

Running time of MysteryOne is in the worst case,

Running time of MysteryOne is in the worst case, at most $c(n-1)$ for some constant $c \in \mathbb{R}$.

and so $O(n)$.

```
MysteryTwo(A[1..n]):
1. for j from 2 to n:
2.     m ← A[j]
3.     i ← j-1
4.     while (A[i] > m & i > 0):
5.         A[i+1] ← A[i]
6.         A[i] ← m
7.         i ← i-1
```

Running time of MysteryTwo?

Lines 2-7 execute n-1 times.

Lines 2-3 execute every time and take constant time.

Lines 5-7 take constant time for each execution, and execute up to $j-1$ times

Running time is at most

(lines 5-7) (lines 2-3)

$$\sum_{j=2}^{n} (c(j-1)+d)$$

$$= c\sum_{j=2}^{n} j-1 + \sum_{j=2}^{n} d$$

leading term n^2

$$= c\frac{n(n-1)}{2} + d(n-1)$$

$$= O(n^2).$$

Consider the following problem:

a bunch of nuts & bolts

Say that they're all different sizes & jumbled up.

But know that every bolt has a matching nut.

The following algorithm finds a matching nut-bolt pair:

(Nut/Bolt size, unsorted)

```
FindMatchingNutBolt(Nuts[1..n], Bolts[1..n]):
```

same size, unsorted

```
Find Matching Nut Bolt (Nuts[1..n], Bolts[1..n]):
1.  for i from 1 to n:
2.      for j from 1 to n:
3.          if (Nuts[i] = Bolts[j]):
4.              return (i, j)
```

Lines 3 & 4 get executed upto n times for each i.

In the worst case, very possible

But! Lines 2-4 only execute for i=1, because guaranteed a matching bolt for Nut[1].

(worst-case)

Running time of Find Matching Bolt Nut

$\leq cn = O(n)$.

(even though I had nested loops).