CSCA48 Winter 2016

Week 10 – Complexity & Regular Expression

Bo(Kenny) Zhao

University of Toronto Scarborough

March 16, 2016

UNIVERSITY OF TORONTO SCARBOROUGH

Announcements

- Term Test #2

  All tests are marked.

  Marks will appear on MarkUs by the end of this week.

  Tests will be returned in next week's tutorial.

- Assignment #2

  Due March 27, 2016

Today's Plan

- Complexity of Insertion Sort
  - Counting number of steps
  - Big-Oh complexity
- Regular Expression
  - Valid regexes
  - Matching a regex with a string

Insertion Sort

```
1  def insertion_sort(L):
2      ''' (list of int) -> NoneType
3      Sort L in decreasing order.
4      >>> L = [5, 7, 1, 9, 3, 2, 0, 4, 6, 8]
5      >>> insertion_sort(L)
6      >>> expected = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
7      >>> L == expected
8      True
9      '''
10     # go through 2nd to last element
11     for i in range(1,len(L)):
12         current_val = L[i]
13         j = i
14         # sort first to current element
15         while (j > 0 and L[j-1] < current_val):
16             L[j] = L[j-1]
17             j = j - 1
18         L[j] = current_val
```

Insertion Sort

```
1  def insertion_sort(L):
2      # go through 2nd to last element
3      for i in range(1,len(L)):
4          current_val = L[i]
5          j = i
6          # sort first to current element
7          while (j > 0 and L[j-1] < current_val):
8              L[j] = L[j-1]
9              j = j - 1
10         L[j] = current_val
```

```
          0  1  2  3  4  5  6  7  8  9
L = [5, 7, 1, 9, 3, 2, 0, 4, 6, 8]
```

Steps

How many steps `j = j - 1` takes?

A. 1 step

B. 2 steps

C. 3 steps

D. 4 steps

Steps

How many steps `L[ j ] = L[ j - 1 ]` takes?

A. 3 steps

B. 4 steps

C. 5 steps

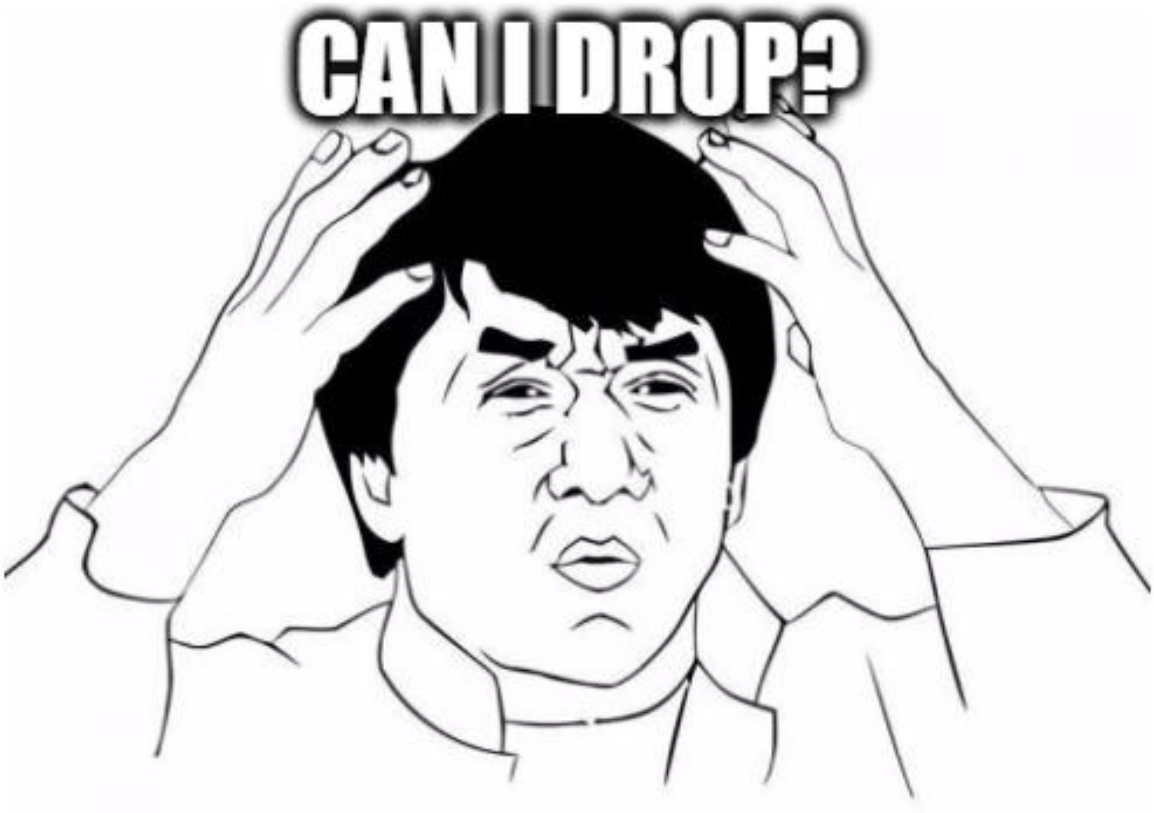
D. 6 steps

Steps

How many steps each of the following operations takes?

1. current_val = L[i]
2. while(j > 0 and L[j-1] > current_val):
3. for i in range(1,len(L)):
4. j = i
5. result = is_nearly_sorted_helper(L, 1, 1)
6. While (I < len(L)) and result):

Steps



Big-Oh Complexity

```
1  def insertion_sort(L):
2      # go through 2nd to last element
3      for i in range(1,len(L)):
4          current_val = L[i]
5          j = i
6          # sort first to current element
7          while (j > 0 and L[j-1] < current_val):
8              L[j] = L[j-1]
9              j = j - 1
10         L[j] = current_val
```

The first loop will run _____ times.

The second loop will run _____ times. (worst-case)

Big-Oh Complexity

```
1  def insertion_sort(L):
2      # go through 2nd to last element
3      for i in range(1,len(L)):
4          current_val = L[i]
5          j = i
6          # sort first to current element
7          while (j > 0 and L[j-1] < current_val):
8              L[j] = L[j-1]
9              j = j - 1
10         L[j] = current_val
```

The first loop will run _n - 1_ times.

The second loop will run _at most n_ times. (worst-case)

Big-Oh Complexity

The first loop will run _n - 1_ times.

The second loop will run _at most n_ times. (worst-case)

Since we are calculating big-oh complexity, then we can claim that the first loop will run n times, and each time the first loop is run, the second loop will also run n times.

e.g. If n = 1,001, then there is no significant difference between 1,000 and 1,001

Big-Oh Complexity

```
1   def insertion_sort(L):
2       # go through 2nd to last element
3       for i in range(1,len(L)):
4           current_val = L[i]
5           j = i
6           # sort first to current element
7           while (j > 0 and L[j-1] < current_val):
8               L[j] = L[j-1]
9               j = j - 1
10          L[j] = current_val
```

The first loop will run __n__ times.

The second loop will run __n__ times.

Regular Expression

Valid Regexes	Matching Strings	Interpretation
'0'	'0'	
'1'	'1'	
'2'	'2'	
'e'	'' the empty string	It doesn't match letter e
'0*'	'', '0', '0000', '00', etc.	Empty string or strings made up of 0's
'1*'	'', '1111', '11', etc.	Empty string or strings made up of 1's

Regular Expression

Valid Regexes	Matching Strings	Interpretation
‘2*’	‘’, ‘2’, ‘222’, ‘2222’, etc.	Empty string or strings made up of 2’s
‘2*****’	‘’, ‘2’, ‘222’, ‘2222’, etc.	Same as ‘2*’
‘e*’	‘’	No other strings match
‘e***’	‘’	Same as ‘e*’ and ‘e’
‘(0|1)’	‘0’, ‘1’	
‘(1.2)’	‘12’	

Regular Expression

Valid Regexes	Matching Strings	Interpretation
‘(e|0)’	‘’, ‘0’	
‘(2.e)’	‘2’	
‘(0*|2*)’	‘’, ‘0’, ‘0000’, ‘2’, ‘222’, etc.	
‘((0.1).2)’	‘012’	
‘(0|2)*’	‘’, ‘0202200’, ‘0000’, ‘22’, etc.	
‘((1.(0|2)*).0)’	‘102022000’, ‘10000’, ‘1220’, ‘10, etc.	