CSC340:
Complexity Analysis—for-loops

Main Topics:
1. The main idea
2. For-loops
3. Selection sort and insertion sort

Readings:
1. Limited coverage: Chapter 10 in the 6th edition

Hui Yang
Computer Science Department
San Francisco State University
http://www.cs.sfsu.edu/~huiyang/

Analyzing the complexity of loops

- Focus on for-loops
- Main ideas
  - It takes a constant amount of time to execute primitive operations, e.g., assignment and addition
  - It's critical to identify the total number of times each line needs to be executed in a for-loop
  - The total run time is the sum of run time consumed by each line of code.
  - The run time is a polynomial function of the upper bounds of the indexing variables.
- Examples
  - Please take notes

For-loops

- How many times will line 1 and line 2 be executed respectively in the following code snippets?

```
1.  for (i=0; i<n; i++)
2.          sum += i;
```

```
1.  for (i=0; i<n; i++)
2.          for (j=0; j<n; j++)
3.                  sum = i+j;
```

```
1.  for (i=0; i<n; i++)
2.          for (j=i; j<n; j++)
3.                  sum = i+j;
```


Selection Sort

```
SelectionSort(A)
// GOAL: place the elements of A in ascending order
1   n := length[A]
2   for i := 1 to n
3       // GOAL: place the correct number in A[i]
4       j := FindIndexOfSmallest( A, i, n )
5       swap A[i] with A[j]
        // L.I. A[1..i] the i smallest numbers sorted
6   end-for
7   end-procedure


FindIndexOfSmallest( A, i, n )
// GOAL: return j in the range [i,n] such
//       that A[j]<=A[k] for all k in range [i,n]
1   smallestAt := i ;
2   for j := (i+1) to n
3       if ( A[j] < A[smallestAt] ) smallestAt := j
        // L.I. A[smallestAt] smallest among A[i..j]
4   end-for
5   return smallestAt
6   end-procedure
```

Insertion Sort

InsertionSort(A[0 .. n-1])

```
1. for i ← 1 to length(A)-1
2.     x ← A[i]
3.     j ← i
4.     while j > 0 and A[j-1] > x
5.         A[j] ← A[j-1]
6.         j ← j - 1
7.     A[j] ← x
```