How to Analyze Code?

- Analysis case: worst-case, best-case, average-case
  - Our focus: worst-case execution (longest code path)

- Basic operations take (roughly) “constant” time – O(1)
  - Arithmetic
  - Assignment
  - Access one Java field or array index

- More complex operations:
  - Consecutive statements – sum of times
  - Conditionals – time to test + time of slower branch
  - Loops – sum of (time of) iterations
  - Calls – time of method body
  - Recursion – depends… (more later)

Code Analysis Example

- Code for finding integer in sorted array:
  ```java
  boolean findInteger(int[] arr, int k) {
      for(int i=0; i < arr.length; i++)
          if (arr[i] == k)
              return true;
      return false;
  }
  ```

- What is the run-time complexity for this method?
  - Worst-case:

- Can you think of ways to make this more efficient?
  - Implementation?
  - Algorithm?
Code Analysis: Nested Loop Example

- POSTED ON PIAZZA
- What is the run-time complexity for this code?

```c
for(int i = 0; i < n; i++)
    sum = sum + 1;
for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        sum2 = sum2 + 1;
```

Code Analysis: Nested Loops Again

- What is the run-time complexity for this code?

```c
for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        if (cond)
            run_a_loopless_routine(sum);
        else
            for (int k = 0; k < n*n; k++)
                sum = sum + 1;
```
Comparing Two Algorithms

- Two algorithms for finding the nth value in an array:
  
  ```java
  for(int i=0; i < n; i++)
      tmp = v[i];
  return tmp;
  ```
  
  Or
  
  ```java
  return v[n-1];
  ```

- Which is more efficient?
Revisit: Array or Linked List?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STACK:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>push</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>find_kth</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QUEUE:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enqueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dequeue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert_at_kth</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEQUE (double-ended queue):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert_front</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove_front</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert_back</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove_kth</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>