Big O: Order of Algorithm
How Fast is my Algorithm?

• There can be many algorithms to solve any problem – like linear search, binary search.

1. How do we choose the most efficient?
2. What is efficient?

• One measure is **how fast** our algorithm can determine the solution.
  ▫ This is not the only measure, nor is it always the best measure.
  ▫ How do we measure ‘how fast’.
‘How Fast’

- What contributes to how fast a program runs?
  - The speed the CPU can process operations.
  - The efficiency of your code (the number of operations needed to complete your calculations).
    - This depends on the algorithm used.
    - This may depend on the size of the data set being analyzed.
    - This depends on the particular implementation of the algorithm. (processor speed; instruction set, disk speed, brand of compiler and etc.)
  - How many other things your computer is doing at the same time.
Measuring ‘How Fast’

• Two approaches:

1. **Analyze** your algorithm/code
   - Determine an upper limit on the number of operations needed
   - Know your CPU speed (cycles per second)

2. **Implement** your algorithm then make **measurements** of how long it takes to run for data sets of varying sizes
   - Create a common baseline, run tests on same machine with same background load
   - Disadvantage: you already have spent the time coding and testing if the algorithm is not practical this may have been wasted.
Counting operations

- Consider the operations used in your code.
  - +, -, *, /, %, <, <=, >, >=, ==, =, !=, &, !, &&, || ...
  - Make a simplifying assumption that each of these operations take the same length of time to execute.
  - Now we just need to count the operations in your program to get an estimate of ‘how fast’ it will run.
  - This estimate is independent of the machine on which the code runs.
    - Machine-dependent: Once we know the time taken by an ‘operation’ on our machine we know how long our code will take.
Example: counting operations(1)

• Simple linear or branching code:

```python
if( neighborcount > 3 or neighborcount < 2 ):
    nextGenBoard[indexrow] = '.
```

• The first if executes 3 operations, >, or, and <

• If the first if is true then the block of code above executes with 2 operation: [ ] and =
Example: Counting Operations

• While loop

```python
count = 0
while (count < n):
    localSum = dataArray[count] + 2 * localSum
    count++;
```

• Total operations each time through loop is 6
• The initialization of count takes one operation before the loop begins executing
• The loop is executed n times
• The number of operations is $6n + 1$
Missed operation!!!

- **While loop**
  
  ```
  count = 0;
  while (count < n):
      localSum = dataArray[count] + 2 * localSum;
      count++;
  ```

- The number of operations is $6n + 1$

- The test in the while is executed **one additional time** at the end of the loop.
- The number of operations is $6n + 2$
Big O

- Estimate the order of the number of calculations needed
  - Order is the largest power of \( n \) in the estimated upper limit of the number of operations.
- For most \( n \) (amount of data) it is generally true that an order \( n^k \) algorithm is significantly faster than an order \( n^{k+1} \) algorithm.
- An algorithm with order \( n \) operations is said to run in linear time.
- An algorithm with order \( n^2 \) operations is said to run in quadratic time.
Estimate of how fast

- Looking for a ‘good’ upper limit
- Just consider the Order.
  - The order is the largest power of n
- First example: 9 operations
  - $\mathcal{O}(9) = 0$ Order 0 (not a function of n)
- Second example: $6n + 1$ operations
  - $\mathcal{O}(6n + 1) = n$ Order 1 (largest power of n is 1)
- Third example: $1 + 3n + 11n^2$
  - $\mathcal{O}(1 + 3n + 11n^2) = n^2$ Order 2 (largest power of n is 2)
Measuring ‘how fast’

• How good are our estimates

• The estimates we have made are worst case estimates.
  ▫ In some cases algorithms will finish much faster if input data has particular properties
  ▫ Be careful the measurement is only as good as the assumptions.

• We can directly measure ‘how fast’ for particular types of data sets of particular sizes
  ▫ You are doing this is your lab
  ▫ This is still a way to approximate performance in a general case on a wider variety of sizes.
Questions?