# Whiteboarding Practice

# In-class Whiteboarding Practice (1-A)

1. Write a method that takes an array of integers as a parameter and returns the smallest and the second smallest element.

2. Given an integer *n* as input, write a program to read the integer *n* and perform the following conditional actions:
   - If *n* is odd, print *Weird*
   - If *n* is even and in the inclusive range of 2 to 5, print *Not Weird*
   - If *n* is even and in the inclusive range of 6 to 20, print *Weird*
   - If *n* is even and greater than 20, print *Not Weird*

# In-class Whiteboarding Practice (1-B)

1. Write a method that takes an array of integers as a parameter and returns the difference between the maximum and minimum

2. Write a method that returns a String of given length n that has random lowercase characters from a - z

```
public String randomString (int  n) { … }
```

# In-class Whiteboarding Practice (2-A)

1. **Check Permutation**: Given two strings, write a method to decide if one is a permutation of the other. What is the runtime of your solution?

# In-class Whiteboarding Practice Hints (2-A)

**Check Permutation**. Hints for Interviewer to use:

1. Describe what it means for two strings to be a permutations of each other. Can you check the strings against that definition?
2. Come up with a brute-force solution. How long does it take?
3. There is one solution that is *O(n log n)*. Another solution uses some space but is *O(n)* time (on an average)
4. Could a hash table be useful?

# In-class Whiteboarding Practice (2-B)

1. **Point of intersection**: Given two singly linked lists A (size *n*) and B (size *m*), determine if the two lists intersect. Return the intersecting node. Note that the intersection is based on reference, not value. That is, if the kth node of the first list is the exact same node (by reference) as the jth node of the second list, then they are intersecting.

# In-class Whiteboarding Practice Hints (2-B)

**Point of intersection**. Hints for interviewer to use:

1. What is the brute force solution? It should take *O(1)* space. What is the run-time for your solution?
2. See if you can come up with a faster algorithm using an auxiliary data structure that takes space *O(n+m)*?
3. Draw some examples. How can we determine if two lists intersect at all? Can you walk through the lists and determine the intersecting node without an auxiliary data structure?

# In-class Whiteboarding Practice (3-A)

1. **Finding the boundary.** Design a method that finds the index of the first 1 from a sorted array of 0's and 1s. What is the time complexity of your solution in big-Oh notation?

# In-class Whiteboarding Practice (3-B)

1. **Big Sort**. Imagine that you have a 20GB file with one string per line. How would sort the file without using more than 1GB of memory at a time?