01-0: **Syllabus**

- Office Hours
- Course Text
- Prerequisites
- Test Dates & Testing Policies
  - Check dates now!
- Grading Policies

01-1: **How to Succeed**

- Come to class. Pay attention. Ask questions.

01-2: **How to Succeed**

- Come to class. Pay attention. Ask questions.
  - A question as vague as “I don’t get it” is perfectly acceptable.
  - If you’re confused, there’s a good chance someone else is confused as well.

01-3: **How to Succeed**

- Come to class. Pay attention. Ask questions.
  - A question as vague as “I don’t get it” is perfectly acceptable.
  - If you’re confused, there’s a good chance someone else is confused as well.
- Come by my office
  - I am *very* available to students.

01-4: **How to Succeed**

- Come to class. Pay attention. Ask questions.
  - A question as vague as “I don’t get it” is perfectly acceptable.
  - If you’re confused, there’s a good chance someone else is confused as well.
- Come by my office
  - I am *very* available to students.
- Start the homework assignments and projects early
  - Projects in this class are significantly harder than CS112

01-5: **How to Succeed**

- Come to class. Pay attention. Ask questions.
  - A question as vague as “I don’t get it” is perfectly acceptable.
  - If you’re confused, there’s a good chance someone else is confused as well.
- Come by my office

  - I am *very* available to students.
- Start the homework assignments and projects early
  - Projects in this class are significantly harder than CS112
- Read the notes.
  - Ask Questions!

01-6: **Brief Commercial**

- Most interview questions will be based on material for this claass
- If you go into software development, you will use this material *Every Day*!

01-7: **What is an algorithm?**

- An algorithm is a step-by-step method for solving a problem.
- Each step must be well defined.
- Algorithm $\neq$ Computer Program.
- A program is an *implementation* of an algorithm.
- Can have different implementations of the same algorithm
  - Different Languages
  - Different Coding Styles

01-8: **Example: Selection Sort** Algorithm:

- Examine all n elements of a list, and find the smallest element
- Move this element to the front of the list
- Examine the remaining $(n - 1)$ elements, and find the smallest one
- Move this element into the second position in the list
- Examine the remaining $(n - 2)$ elements, and find the smallest one
- Move this element into the third position in the list
- Repeat until the list is sorted

01-9: **Example: Selection Sort**

Java Code:

```
for (int i=0; i<A.length - 1; i++) {
   smallest = i;
   for (j=i+1; j<A.length; j++) {
     if (A[j] < A[smallest])
       smallest = j;
   tmp = A[i];
   A[i] = A[smallest];
   A[smallest] = tmp;
}
```

01-10: **Problem I: Bridge**

Four people want to cross a bridge

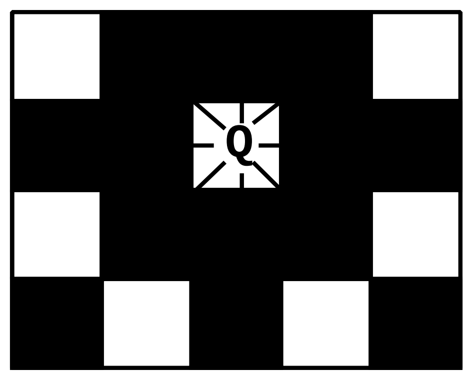
- The bridge can only hold two people at the same time
- Each person requires a different amount of time to cros the bridge (1, 2, 5, and 8 minutes)
- It is pitch black, and they have only 1 flashlight which they need to shuttle back and forth across the bridge

01-11: **Purpose of the Problem**

- Obvious solution is not always optimal
  - "Obviously" want fastest person to shuttle flashlight
- Optimial solution is often not obvious
  - *Second*-fastest person needs to do some of the shuttling

01-12: **Problem II: 8 Queens**

- Standard 8x8 Chessboard
- Place 8 Queens on this board, so that no Queen attacks another Queen.



  - Queens can move horizontally or diagonally any number of squares

01-13: **8-Queens Data Structures**

- Two-dimensional array of characters.
- List of x-y coordinates of each queen.
- Array of integers:
  - Each element in this array represents a column
  - Value stored in element `i` represents the row in which the queen at column `i` is located.

01-14: **8-Queens Data Structures**

- Choice of data structure can influence how we solve the problem
  - Two dimensional array of characters: $\binom{64}{8}$ potential solutions (around 10^{14})
  - List of 8 x-y coordinates: 64^8 potential solutions (also around 10^{14})
  - Array of rows: 8^8 potential solutions (around 10^7)

01-15: **9 Coins**

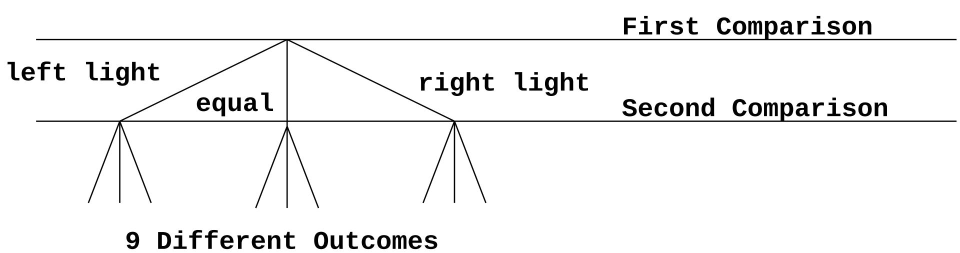
- There are 9 coins. 8 are good, but one is counterfeit. The counterfeit coin is ligher than the other coins.
- You have a balance scale, that can compare the weights of two sets of coins
- Can you determine which coin is counterfeit, using the scale only 2 times?
- If there are 27 coins, one lighter and counterfeit, can you find it using the scale 3 times?

01-16: **9 Coins**

- First, let's ensure that it's possible
- There are 9 possible cases: Coin 1 is bad, coin 2 is bad, coin 3 is bad, etc.
- Can we distinguish between 9 different cases using two weighings?

01-17: **9 Coins**

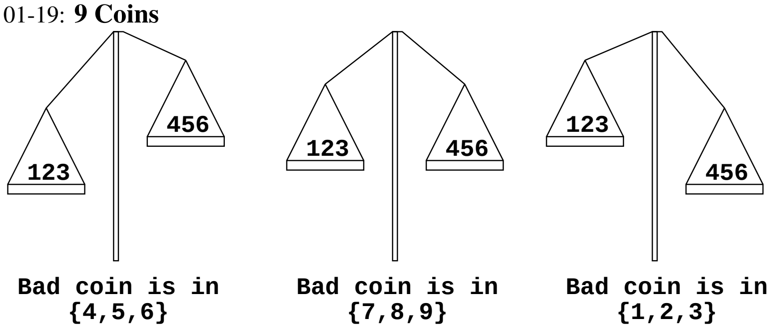
- First, let's ensure that it's possible
- There are 9 possible cases: Coin 1 is bad, coin 2 is bad, coin 3 is bad, etc.
- Can we distinguish between 9 different cases using two weighings?



01-18: **9 Coins**

- Weigh Coins 1,2,3 against 4,5,6
- What are the possible outcomes?
- What will they tell us?

01-19: **9 Coins**

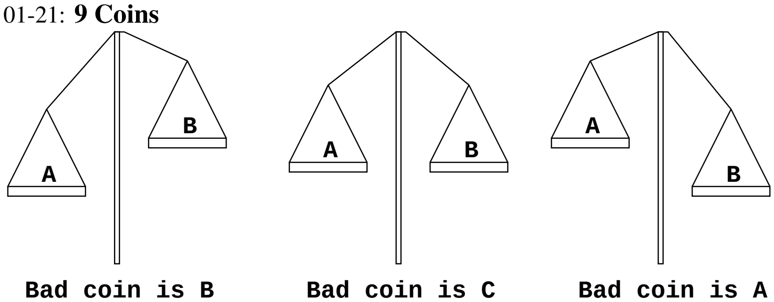


01-20: **9 Coins**

- We now have a set of 3 coins, we know one of them is bad

- Call these coins A, B, C
- Weigh A against B.
- What are the outcomes?
- What will they tell us?

01-21: **9 Coins**



01-22: **9 Coins**

- Things to think about
  - How would we extend this to 27 coins?
  - We decided which coins to weigh second, *after* we had the results of the first weighing. Could we decide which coins to weigh *before* getting any results, and still solve the problem in 2 weighings?
  - The "classic" vesion of this problem is 12 coins, 3 weighings, *and* the counterfeit coin could be either heavy or light. Can you solve that problem? (The classic problem is a little harder ...)