Comp 160: Algorithms, **Recitation 3**

1. (review of recurrences + preparation for lower bound homework) You have n coins that are all supposed to be gold coins of the same weight, but you know that one coin is fake and weighs less than the others. You have a balance scale: you can put any number of coins on each side of the scale, and it will tell you if the two sides weigh the same or which side is lighter if they do not weigh the same.

   (a) Describe a recursive algorithm that will identify the fake coin.

   Hint: although we want an algorithm that works for any number n of coins, first think what you would do with a small number of coins, say 60.

   (b) Write down the recurrence formula that describes the number of weighings needed to find the fake coin. Solve the recurrence giving matching upper and lower bounds.

2. If you flip a coin 10 times, what is the expected number of times you will see the pattern `HHH` (that is, three heads in a row)?

3. A group of k Vikings independently set out to make a new home. Each Viking has a copy of the same map, showing n islands. Each Viking decides to set sail for some random island. If two or more Vikings land on the same island, they have a battle. (No matter how many Vikings land on that island, it counts as one battle.)

   (a) How many islands do we expect will be visited by Vikings?

4. Let's do an exercise to understand the limitation of decision trees. Consider the following algorithm:

Algorithm 1 NotVeryEfficientSort (Array A of n numbers)

```
1: Generate all permutations of n numbers store them in S
2: for each permutation s in S do
3:     if CheckThisPermutation(A,s) then
4:         Return A[s(1)], A[s(2)], ..., A[s(n)]
```

Where CheckThisPermutation is a function that checks if the given permutation returns the array to sorted order (that is, if the array is $[6, 8, 4]$ and permutation is $(3, 1, 2)$ it will return true because $A[3] \leq A[1] \leq A[2]$). The check is done in the naive fashion (check if $A[s(1)] \leq A[s(2)]$, then $A[s(2)] \leq A[s(3)]$, and so on. If any check returns false you return false. Otherwise you return true).

   (a) Run the algorithm for $n = 2$ and $n = 3$ on a random instance (your choice, just something that is not sorted)

   (b) Is this algorithm correct? Why?

   (c) Can you give an upper bound for the runtime of this algorithm? You can assume that all permutations of n numbers can be generated in time proportional to the size of S (no need for the bound to be tight).

(d) Draw the decision tree of this algorithm for $n = 3$

(e) Draw the decision tree of this algorithm for $n = 50$

(f) Say that we add an extra step (Step 0) at the beginning of the algorithm that says: *For i from* 1 *to* $(n!^2)$ *do (nothing).*
Let NotVeryEfficientSort2.0 be the modified algorithm. Would this modification affect correctness of the algorithm? What about runtime?

(g) Draw the decision tree of the modified algorithm (for $n = 3$)

Additional practice questions:

5. Review McGuffin problem from lecture (try to answer it yourself without looking at solution).

6. Draw the decision tree of your favorite algorithm (for $n = 3$).

7. Review the coin flipping problem discussed in class. Now answer how many appearances of the `THT` pattern do we expect to see (also, when flipping 10 coins)? Can you answer the question without doing any calculations? Why?

8. What if we ask about a pattern of 5 coins (say, `HHHHH`) when flipping 10 coins? How many appearances do we expect?

9. How many people do we need in a room so that we can expect to have at least one birthday match?

   Note: The number is surprisingly small and defies intuition. This is why this problem is often called the birthday paradox