CS 161 **Problem Set 3**
Autumn 2017 **Due:** Friday October 20, 2017, at 3pm on Gradescope

---

Exercises

Exercises should be completed **on your own.**

---

1. **(2 pt.)** Siggi and Ollie are playing the following game of chance. Siggi will roll a fair six-sided die. If Siggi rolls a 1, 2, 3, or 4, then Ollie will flip a fair coin. If the coin comes up heads, Siggi pays Ollie a dollar. Otherwise, Ollie pays Siggi a dollar. On the other hand, if Siggi rolls a 5 or a 6, then Ollie pays Siggi a dollar immediately.

   (a) How much money does Siggi make in expectation? How much money does Ollie make in expectation?

   (b) What is the probability that Ollie makes money? What is the probability that Siggi makes money?

   (c) What is the probability that both Ollie and Siggi make money?

   (d) Suppose you know that Siggi made money. Conditional on that event, what is the probability that Siggi rolled a 1?

2. **(4 pt.)** In this exercise, we'll explore different types of randomized algorithms. We say that a randomized algorithm is a **Las Vegas Algorithm** if it is always correct, but the running time is a random variable. We say that a randomized algorithm is a **Monte Carlo Algorithm** if there is some probability that it is incorrect. For example, QuickSort (with a random pivot) is a Las Vegas algorithm, since it always produces a sorted array (but if we get very unlucky QuickSort may be slow).

   Consider the following task. The population of n tricky and trustworthy toads from HW1 is back. As before, you are guaranteed that there are strictly more than $n/2$ trustworthy toads in the population. Your goal is to find a single trustworthy toad. However, this time you've arrived on the island with a toad expert. The expert can determine whether a given toad is tricky or trustworthy, but she takes time $\Theta(n)$ to do it.

   The algorithms given in Figure 1 all attempt to identify a single trustworthy toad. Fill in the chart below. You may use asymptotic notation for the running times; for the probability of returning a truthful toad, give the tightest bound that you can.

Algorithm	Monte Carlo or Las Vegas?	Expected running time	Worst-case running time	Probability of returning a truthful toad
Algorithm 1				
Algorithm 2				
Algorithm 3				

   If it helps in writing up your solution, the LaTeXcode for the table (which you can copy and paste and fill in) has been reproduced at the end of this problem set. **[We are expecting: Your answers, and for each algorithm, a short informal justification of your answers.]**

Algorithm 1: findTrustworthyToad1

```
Input: A population of $n$ toads
while true do
    Choose a random index $i \in \{0, \ldots, n-1\}$;
    Ask the expert if toad $i$ is trustworthy;
    /* Asking the expert takes time $\Theta(n)$                              */
    if the expert says toad $i$ is trustworthy then
        return toad $i$
```

Algorithm 2: findTrustworthyToad2

```
Input: A population of $n$ toads
for 100 iterations do
    Choose a random index $i \in \{0, \ldots, n-1\}$;
    Ask the expert if toad $i$ is trustworthy;
    /* Asking the expert takes time $\Theta(n)$                              */
    if the expert says toad $i$ is trustworthy then
        return toad $i$
return toad 0
```

Algorithm 3: findTrustworthyToad3

```
Input: A population of $n$ toads
Put the toads in a random order ;
/* Assume it takes time $O(n)$ to put the $n$ toads in a random order        */
for $i = 0, \ldots, n-1$ do
    Ask the expert if toad $i$ is trustworthy;
    /* Asking the expert takes time $\Theta(n)$                              */
    if the expert says toad $i$ is trustworthy then
        return toad $i$
```

Figure 1: Three algorithms for finding a truthful toad

3. **(3 pt.)** This exercise references the IPython notebook `HW3.ipynb`, available on the course website.

   In our implementation of radixSort in class, we used bucketSort to sort each digit. Why did we use bucketSort and not some other sorting algorithm? There are several reasons, and we'll explore one of them in this exercise.

   - **(1 pt.)** One reason we chose bucketSort was that it makes radixSort work correctly! In `HW3.ipynb`, we've implemented four different sorting algorithms—bucketSort, quickSort, and two versions of mergeSort—as well as radixSort. Modify the code for radixSort to use each one of these four algorithms as an inner sorting algorithm. Which ones work?
     - Does using bucketSort work correctly?
     - Does using quickSort work correctly?
     - Does using mergeSort (with merge1) work correctly?
     - Does using mergeSort (with merge2) work correctly?

     [We are expecting: Yes or no for each part.]
   - **(2 pt.)** Explain what you saw above. What was special about the algorithms which worked? Why does this matter? (You may wish to play around with `HW3.ipynb` to "debug" the incorrect cases.)[1]

     Make sure that the algorithms that you observed to work do have the property, and that those that you observed not to work don't have the property.

     [We are expecting: A clear definition of the special property and a short but convincing explanation of why it matters. You do not need to justify why each of the algorithms do or do not have the property but you should understand why they do or do not.]

---

[1]**Note:** Yes, we talked about this a bit in class—that's why it's an exercise and not a problem!

Problems

You may talk with your fellow CS161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

---

1. **(4 pt.)** Suppose that n flamingos are standing in a line.

Each flamingo has a political leaning: left, right, or center. You'd like to sort the flamingos so that all the left-leaning ones are on the left, the right-leaning ones are on the right, and the centrist flamingos are in the middle. You can only do two sorts of operations on the flamingos:

Operation	Result
`poll(j)`	Ask the flamingo in position j about its political leanings
`swap(i,j)`	Swap the flamingo in position j with the flamingo in position i

However, in order to do either operation, you need to pay the flamingos to co-operate: each operation costs one brine shrimp.[2] Also, you didn't bring a piece of paper or a pencil, so you can't write anything down and have to rely on your memory. Like many humans, you can remember up to seven[3] integers between 0 and $n - 1$ at a time.

Design an algorithm to sort the flamingos which costs $O(n)$ brine shrimp, and uses no extra memory other than storing at most seven[4] integers between 0 and $n - 1$.

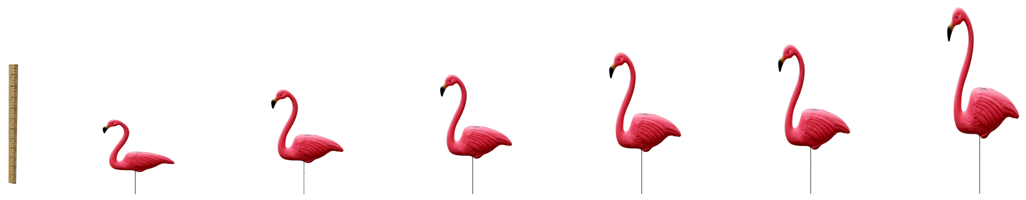
[We are expecting: Pseudocode with a short explanation of the idea of your algorithm; an informal justification that it works, uses only $O(n)$ brine shrimp and not too much memory.]

[2]According to Wikipedia, flamingos eat brine shrimp. Yum!

[3]`https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two`

[4]You don't need to use all seven storage spots, but you can if you want to. Can you do it with only two?

2. **(4 pt.)** Suppose that n flamingos are standing in a line, ordered from shortest to tallest.



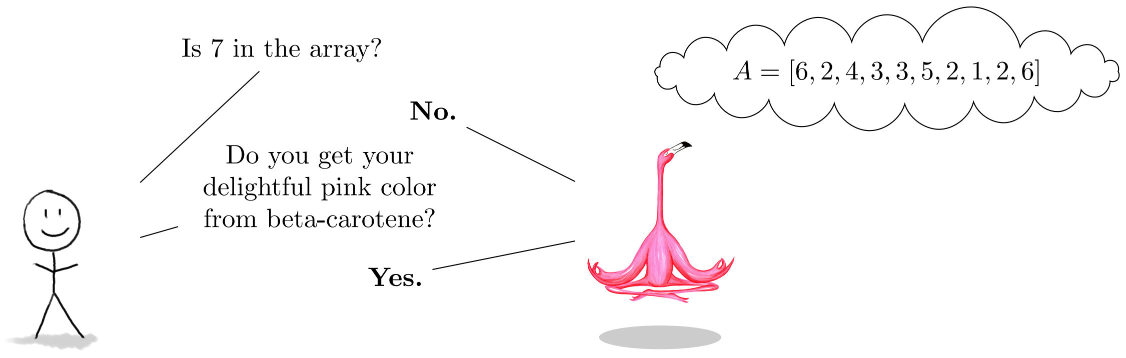
You have a measuring stick of a certain height, and you would like to identify flamingo which is the same height as the stick, or else report that there is no such flamingo. The only operation you are allowed to do is `compareToStick(f)`, where f is a flamingo, which returns `taller` if f is taller than the stick, `shorter` if f is shorter than the stick, and `the same` if f is the same height as the stick. As in Problem 1, you forgot to bring a piece of paper, so you can only store up to seven integers in $\{0, \ldots, n-1\}$ at a time. And, as in Problem 1, you have to pay a Flamingo a brine shrimp in order to compare it to the stick.

(a) **(1 pt.)** Give an algorithm which either finds a flamingo the same height as the stick, or else returns "No such flamingo," in the model above which uses $O(\log(n))$ brine shrimp.
[We are expecting: Pseudocode and an English description. You do not need to justify the correctness or brine shrimp usage.]

(b) **(3 pt.)** Prove that any algorithm in this model of computation must use $\Omega(\log(n))$ brine shrimp.
[We are expecting: A short but convincing argument.]

3. **(3 pt.)** A wise flamingo has knowledge of an array A of length n, so that $A[i] \in \{1, \ldots, k\}$ for all i. (Note that the elements of A are not necessarily distinct). You don't have direct access to the list, but you can ask the wise flamingo *any* yes/no questions about it. For example, you could ask "If I remove $A[5]$ and swap $A[7]$ with $A[8]$, would the array be sorted?" or "do molecular and morphological studies support a relationship between grebes and flamingos?"

This time you did bring a paper and pencil, and your job is to write down all of the elements of A in sorted order.[5] As usual, the wise flamingo charges one brine shimp per question.


Is 7 in the array?
No.
Do you get your delightful pink color from beta-carotene?
Yes.
$A = [6, 2, 4, 3, 3, 5, 2, 1, 2, 6]$

(a) **(3 pt.)** Give a procedure to output a sorted version of A which uses $O(k \log(n))$ brine shrimp. You may assume that you know n and k, although this is not necessary.
[We are expecting: Pseudocode, with an English explanation of what it is doing, why it is correct, and why it only uses $O(k \log(n))$ brine shrimp.]

(b) **(1 bonus pt.)** Prove that any procedure to solve this problem must use $\Omega(k \log(n/k))$ brine shrimp.
[We are expecting: A short but convincing argument.]

[5] Note that you don't have any ability to change the array A itself, you can only ask the wise flamingo about it.

Here is LaTeX code for the table in Exercise 3:

```
\begin{tabular}{|c|p{3cm}|p{2cm}|p{2cm}|p{4cm}|}
\hline
Algorithm & Monte~Carlo or Las~Vegas? & Expected running time &
Worst-case running time & Probability of returning a truthful toad \\
\hline
\textbf{Algorithm 1} &  &  &  & \\
\hline
\textbf{Algorithm 2} &  &  &  & \\
\hline
\textbf{Algorithm 3} &  &  &  & \\
\hline
\end{tabular}
```