CS 161 **Problem Set 3**
Winter 2021 **Due:** Wed, Feb 3 at 11:59 pm PST

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.

What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises

We suggest you do these on your own. As with any homework problem, though, you may ask the course staff for help.

[**NOTE:** These exercise questions **will not** be graded **for this homework only**, but we strongly recommend doing them as it will be good practice for the upcoming exam.]

---

1. **[OPTIONAL]** [**Majority Element**] Suppose we are given an array A of length n with the promise that there exists a majority element (i.e an element that appears $> \frac{n}{2}$ times). Additionally, we are only allowed to check whether two elements are equal (no $>$ or $<$ comparisons). Design an $O(n \log n)$ algorithm to find the majority element, using divide and conquer. Informally, explain the correctness and runtime of your algorithm.

   [**We are expecting:** pseudocode, an english description of the main idea of the algorithm, as well as an informal explanation of correctness and runtime]

2. **[OPTIONAL]** [**Median of Two**] Given two arrays of length n, find the median of all elements of the two arrays.

   (a) If the arrays are unsorted, what is the best you can do?

   [**We are expecting:** A runtime and a brief description of your algorithm.]

   (b) If the arrays are sorted, can you do better?

   [**We are expecting:** A short English description, Pseudocode, runtime analysis.]

Problems

You may talk with your fellow CS 161-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.

[**NOTE:** The following questions **will** be graded.]

---

3. **(12 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 (that is, it returns the right answer with probability 1), 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 returns a sorted array, but it might be slow if we get very unlucky.

   We will revisit the Majority Element problem to get more insight on randomized algorithms.

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

   [**We are expecting:** Your filled in-table, and a short justification for each entry of the table. You may use asymptotic notation for the running times; for the probability of returning a majority element, give the tightest bound that you can given the information provided. Fill in the table below, and justify your answers.]

Algorithm 1: findMajorityElement1

```
Input: A population $P$ of $n$ elements
while true do
    Choose a random $p \in P$;
    if isMajority(P, p) then
        return $p$;
```

Algorithm 2: findMajorityElement2

```
Input: A population $P$ of $n$ elements
for 100 iterations do
    Choose a random $p \in P$;
    if isMajority(P, p) then
        return $p$;
return $P[0]$;
```

Algorithm 3: findMajorityElement3

```
Input: A population $P$ of $n$ elements
Put the elements in $P$ in a random order.;
/* Assume it takes time $\Theta(n)$ to put the $n$ elements in a random order
   */
for $p \in P$ do
    if isMajority(P, p) then
        return $p$;
```

Algorithm 4: isMajority

```
Input: A population $P$ of $n$ elements and a element $p \in P$
Output: True if $p$ is a member of a majority species
count $\leftarrow$ 0;
for $q \in P$ do
    if $p = q$ then
        count ++;
if $count > n/2$ then
    return True;
else
    return False;
```

4. (10 pt.) [**Counting Inversions**] Given an array A of n elements, we call (i, j) an inversion if $0 \leq i < j < n$ and $A[i] > A[j]$.

(a) **(3 pt.)** Describe an $O(n^2)$ algorithm to count the number of inversions in a given array.

[**We are expecting:** Pseudocode, and a short English description explaining the main idea of the algorithm. No justification of the correctness or running time is required.]

(b) **(7 pt.)** Describe an $O(n \log n)$ algorithm to count the number of inversions in a given array. [*Hint:* Think about how you can modify MergeSort to solve this problem.]

[**We are expecting:** Pseudocode, and a short English description explaining the main idea of the algorithm.We are also Expecting an informal justification of correctness and of the running time.]

5. (10 pt.) [**Untitled Duck Problem**] Suppose that the n ducks are standing in a line.

Each duck has a favorite activity: **honking**, **eating**, or simply doing nothing (**idling**). And since they are ducks, they do whatever they like to do all day long. You'd like to sort the ducks so that all the honking ducks are on the left, the eating ducks are on the right, and the idling ducks are in the middle.[1] You can only do two types of operations on the ducks:

Operation	Result
`ask(j)`	Ask the duck in position j about its favorite activity
`swap(i,j)`	Swap the duck in position j with the duck in position i

You want to sort the ducks as soon as possible, but each of the above operations takes a constant time to execute. 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 integers[2] between 0 and $n-1$ at a time.

(a) **(7 pt.)** Design an algorithm to sort the ducks which takes $O(n)$ seconds, and requires you to remember no more than seven integers[3] between 0 and $n-1$ at a time.

[**We are expecting:** Pseudocode **AND** a short English description of your algorithm.]

(b) **(3 pt.)** Justify why your algorithm is correct, why it takes only $O(n)$ seconds, and why it requires you to remember no more than seven integers at a time.

[**We are expecting:** Informal justifications of the correctness, runtime, and memory usage of your algorithm that are both clear and convincing to the grader. If it's easier for you to be clear, you can give a formal proof of correctness, but you do not have to. It is okay to appeal to the correctness of an algorithm that we have seen in class, as long as you clearly explain the relationship between the two algorithms.]

[1] Having honking and eating ducks close would be disastrous. Don't ask why.

[2] see, e.g., `https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two`

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