**CS 344: Design and Analysis of Computer Algorithms** **Rutgers: Spring 2021**

# Homework #2

Deadline: Monday, February 22, 11:59 PM

## Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question. This however does not apply to the extra credit questions.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (video recordings, notes, textbook, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use a hash table of size $m$ with chaining on an array of length $n$ to get expected worst-case runtime of $O(1 + \frac{n}{m})$ for searching each element".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question).
- The "Challenge yourself" and "Fun with algorithms" are both extra credit. These problems are significantly more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.

---

**Problem 1.** Suppose we have an array $A[1:n]$ of $n$ *distinct* numbers. For any element $A[i]$, we define the **rank** of $A[i]$, denoted by $rank(A[i])$, as the number of elements in $A$ that are strictly smaller than $A[i]$ plus one; so $rank(A[i])$ is also the correct position of $A[i]$ in the sorted order of $A$.

Suppose we have an algorithm **magic-pivot** that given any array $B[1:m]$ (for any $m > 0$), returns an element $B[i]$ such that $m/3 \leq rank(B[i]) \leq 2m/3$ and has worst-case runtime $O(n)$[1].

**Example:** if $B = [1, 7, 6, 2, 13, 3, 5, 11, 8]$, then **magic-pivot**$(B)$ will return one arbitrary number among $\{3, 5, 6, 7\}$ (since sorted order of $B$ is $[1, 2, 3, 5, 6, 7, 8, 11, 13]$)

(a) Use **magic-pivot** as a black-box to obtain a deterministic quick-sort algorithm with worst-case running time of $O(n \log n)$. **(10 points)**

(b) Use **magic-pivot** as a black-box to design an algorithm that given the array $A$ and any integer $1 \leq r \leq n$, finds the element in $A$ that has rank $r$ in $O(n)$ time[2]. **(15 points)**

*Hint:* Suppose we run **partition** subroutine in quick sort with pivot $p$ and it places it in position $q$. Then, if $r < q$, we only need to look for the answer in the subarray $A[1:q]$ and if $r > q$, we need to look for it in the subarray $A[q+1:n]$ (although, what is the new rank we should look for now?).

[1] Such an algorithm indeed exists, but its description is rather complicated and not relevant to us in this problem.

[2] Note that an algorithm with runtime $O(n \log n)$ follows immediately from part (a)—sort the array and return the element at position $r$. The goal however is to obtain an algorithm with runtime $O(n)$.

**Problem 2.** Suppose we have an array $A[1:n]$ which consists of numbers $\{1,\ldots,n\}$ written in some arbitrary order (this means that $A$ is a *permutation* of the set $\{1,\ldots,n\}$). Our goal in this problem is to design a very fast randomized algorithm that can find an index $i$ in this array such that $A[i] \mod 3 = 0$, i.e., $A[i]$ is divisible by 3. For simplicity, in the following, we assume that $n$ itself is a multiple of 3 and is at least 3 (so a correct answer always exist). So for instance, if $n = 6$ and the array is $A = [2, 5, 4, 6, 3, 1]$, we want to output either of indices 4 or 5.

(a) Suppose we sample an index $i$ from $\{1,\ldots,n\}$ uniformly at random. What is the probability that $i$ is a correct answer, i.e., $A[i] \mod 3 = 0$? **(5 points)**

(b) Suppose we sample $m$ indices from $\{1,\ldots,n\}$ uniformly at random and with repetition. What is the probability that none of these indices is a correct answer? **(5 points)**

Now, consider the following simple algorithm for this problem:

**Find-Index-1**$(A[1:n])$**:**

- Let $i = 1$. While $A[i] \mod 3 \neq 0$, sample $i \in \{1,\ldots,n\}$ uniformly at random. Output $i$.

The proof of correctness of this algorithm is straightforward and we skip it in this question.

(c) What is the **expected** worst-case running time of **Find-Index-1**$(A[1:n])$? Remember to prove your answer formally. **(7 points)**

The problem with **Find-Index-1** is that in the worst-case (and not in expectation), it may actually never terminate! For this reason, let us consider a simple variation of this algorithm as follows.

**Find-Index-2**$(A[1:n])$**:**

- For $j = 1$ to $n$:
  - Sample $i \in \{1,\ldots,n\}$ uniformly at random and if $A[i] \mod 3 = 0$, output $i$ and terminate; otherwise, continue.
- If the for-loop never terminated, go over the array $A$ one element at a time to find an index $i$ with $A[i] \mod 3 = 0$ and output it as the answer.

Again, we skip the proof of correctness of this algorithm.

(d) What is the **worst-case running time** of **Find-Index-2**$(A[1:n])$? What about its **expected** worst-case running time? Remember to prove your answer formally.

**(8 points)**

**Problem 3.** Given an array $A[1:n]$ of a combination of $n$ positive and negative integers, our goal is to find whether there is a sub-array $A[l:r]$ such that

$$\sum_{i=l}^{r} A[i] = 0.$$

**Example.** Given $A = [13, 1, 2, 3, -4, -7, 2, 3, 8, 9]$, the elements in $A[2:8]$ add up to zero. Thus, in this case, your algorithm should output *Yes*. On the other hand, if the input array is $A = [3, 2, 6, -7, -20, 2, 4]$, then no sub-array of $A$ adds up to zero and thus your algorithm should output *No*.

*Hint:* Observe that if $\sum_{i=l}^{r} A[i] = 0$, then $\sum_{i=1}^{l-1} A[i] = \sum_{i=1}^{r} A[i]$; this may come handy!

(a) Suppose we are promised that every entry of the array belongs to the range $\{-5, -4, \ldots, 0, \ldots, 4, 5\}$. Design an algorithm for this problem with worst-case runtime of $O(n)$. **(15 points)**

*Hint:* Counting sort can also be used to efficiently sort arrays with negative entries whose absolute value is not too large; we just need to "shift" the values appropriately.

(b) Now suppose that there is no promise on the range of the entries of $A$. Design a randomized algorithm for this problem with expected worst-case runtime of $O(n)$. **(10 points)**

**Problem 4.** We want to purchase an item of price $n$ and for that we have an unlimited (!) supply of three types of coins with values 5, 9, and 13, respectively. Our goal is to purchase this item using the *smallest* possible number of coins or outputting that this is simply not possible. Design a dynamic programming algorithm for this problem with worst-case runtime of $O(n)$. **(25 points)**

**Example.** A couple of examples for this problem:

- Given $n = 17$, the answer is "not possible" (try it!).
- Given $n = 18$, the answer is 2 coins: we pick 2 coins of value 9 (or 1 coin of value 5 and 1 of value 13).
- Given $n = 19$, the answer is 3 coins: we pick 1 coin of value 9 and 2 coins of value 5.
- Given $n = 20$, the answer is 4 coins: we pick 4 coins of value 5.
- Given $n = 21$, the answer is "not possible" (try it!).
- Given $n = 22$, the answer is 2 coins: we pick 1 coin of value 13 and 1 coin of value 9.
- Given $n = 23$, the answer is 3 coins: we pick 1 coin of value 13 and 2 coins of value 5.

---

**Challenge Yourself.** Suppose we have two arrays $A[1:n]$ and $B[1:m]$ which are both in the sorted order and are consisting of distinct numbers. Design an algorithm that given an integer $1 \leq r \leq m+n$, find the element with rank $r$ in the union of arrays $A$ and $B$. Your algorithm should run in only $O(\log(n+m))$ time.

**(+10 points)**

**Example.** Suppose $A = [1, 5, 7, 9]$ and $B = [2, 4, 6, 12]$ and so $n = m = 4$. Then, the answer to $r = 3$ is 4 and the answer to $r = 7$ is 9 because the union of arrays $A$ and $B$ in the sorted order is $[1, 2, 4, 5, 6, 7, 9, 12]$.

**Fun with Algorithms.** Recall that Fibonacci numbers form a sequence $F_n$ where $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$. The standard algorithm for finding the $n$-th Fibonacci number takes $O(n)$ time. The goal of this question is to design a significantly faster algorithm for this problem. **(+10 points)**

(a) Prove by induction that for all $n \geq 1$:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}.$$

(b) Use the first part to design an algorithm that finds $F_n$ in $O(\log n)$ time.