CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Homework #2

Deadline: Tuesday, 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 "sort the array in $\Theta(n \log n)$ time using merge sort".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question). See "Practice Homework" for an example.
- 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 index i such that $rank(B[i]) = m/4$ and has worst-case runtime of $O(n)$[1].

Example: if $B = [1, 7, 6, 3, 13, 4, 5, 11]$, then **magic-pivot**$(B)$ will return index 4 as rank of $B[4] = 3$ is 2 which is $m/4$ in this case (rank of $B[4] = 3$ is 2 since there is only one element smaller than 3 in B).

(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 $A[i] \mod 8 \in \{1,2\}$, i.e., the reminder of dividing $A[i]$ by 8 is either 1 or 2. For simplicity, in the following, we assume that n itself is a multiple of 8 and is at least 8 (so a correct answer always exist).

For instance, if $n = 8$ and the array is $A = [8,7,2,5,4,6,3,1]$, we want to output either of indices 3 or 8.

(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., $A[i] \mod 8 \in \{1,2\}$? **(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 $A[i] \mod 8 \notin \{1,2\}$, 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 worse-case **expected** 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 modification to 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 $A[i] \mod 8 \in \{1,2\}$, 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 $A[i] \mod 8 \in \{1,2\}$ 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 worst-case **expected** running time? Remember to prove your answer formally.

(8 points)

Problem 3. Given an array $A[1:n]$ of n positive integers (possibly with repetitions), your goal is to find whether there is a sub-array $A[l:r]$ such that

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

Example. Given $A = [13,1,2,3,4,7,2,3,8,9]$ for $n = 10$, the answer is *Yes* since elements in $A[2:5]$ add up to $n = 10$. On the other hand, if the input array is $A = [3,2,6,8,20,2,4]$ for $n = 7$, the answer is *No* since no sub-array of A adds up to $n = 7$.

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

(a) Design an algorithm for this problem with worst-case runtime of $O(n)$. **(15 points)**

(b) Now suppose our goal is to instead find whether there is a sub-array $A[l, r]$ such that

$$\sum_{i=l}^{r} A[i] = n^2.$$

Design a randomized algorithm for this case with worst-case expected runtime of $O(n)$. **(10 points)**

Problem 4. You are given an array of characters $s[1 : n]$ such that each $s[i]$ is a small case letter from the English alphabet. You are also provided with a black-box algorithm *dict* that given two indices $i, j \in [n]$, $dict(i, j)$ returns whether the sub-array $s[i : j]$ in array s forms a valid word in English or not in $O(1)$ time. (Note that this algorithm is provided to you and you do not need to implement it in any way).

Design and analyze a dynamic programming algorithm to find whether the given array s can be partitioned into a sequence of valid words in English. The runtime of your algorithm should be $O(n^2)$.

Example: Input Array: $s = [maythef orcebewithyou]$.

Assuming the algorithm *dict* returns that $may, the, f orce, be, with$ and *you* are valid words (this means that for instance, for *may* we have $dict(1, 3) =$ True), this array can be partitioned into a sequence of valid words.

Challenge Yourself. You are given an array A of non-negative integers. Every element except one has a duplicate. Design an algorithm that finds the element with no duplicate in $O(n)$ time (You are promised that there is exactly one element with no duplicate).

Example. A couple of examples for this problem:

1. $A = [4, 3, 1, 1, 4]$
   Output: 3
   Explanation: 3 is the only number that does not have a duplicate
2. $A = [3, 1, 4, 5, 1, 4, 3]$
   Output: 5
   Explanation: 5 is the only number that does not have a duplicate

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.