Homework 4: Due February 21 (11:55 a.m.)

Instructions

- **Answer each question on a separate page.**
- **Honors questions are optional. They will not count towards your grade in the course. However you are encouraged to submit your solutions on these problems to receive feedback on your attempts. Our estimation of the difficulty level of these problems is expressed through an indicative number of stars ($'*' =$ easiest) to ($'*****' =$ hardest).**
- **You must enter the names of your collaborators or other sources as a response to Question 0. Do NOT leave this blank; if you worked on the homework entirely on your own, please write "None" here. Even though collaborations in groups of up to 3 people are encouraged, you are required to write your own solution.**

Question 0: List all your collaborators and sources: ($-\infty$ points if left blank)

Question 1: (4+2+2+2=10 points)

Define the notation $[n] = \{1, 2, \ldots, n\}$, and let S_n be the set of all possible permutations of $[n]$. The size of S_n is given by $|S_n| = n! = n \cdot (n-1) \cdots 1$. Recall that $n! = O(n^n)$ and $2^n = O(n!)$. Now, each input in S_n can serve as an input for a sorting algorithm. Then, we say that a sorting algorithm is ε-correct if the algorithm produces the correct result (i.e., produces a sorted array as output) on exactly ε fraction of the set of inputs in S_n. In other words, an ε-correct sorting algorithm is one which produces correct result on $\varepsilon \cdot (n!)$ possible inputs. We have, so far, tried to set $\varepsilon = 1$, i.e., produce the correct result on every possible input. In this question, we will see if lowering ε can yield a saving in the number of comparisons.

1. Let $\varepsilon = 1/2$. Show that for any constant $C \geq 0$, there is no comparison-based ε-correct sorting algorithm that can sort using less than Cn comparisons. This shows that taking $\varepsilon = 1/2$ does not help us reduce the number of comparisons to linear.
2. Consider $\varepsilon = 1/n$. In this setting, are we able to achieve a sorting algorithm for S_n with $O(n)$ comparisons?
3. Consider $\varepsilon = \frac{1}{2^n}$. In this setting, are we able to achieve a sorting algorithm for S_n with $O(n)$ comparisons?
4. Consider $\varepsilon = \frac{2^n}{n!}$. In this setting, are we able to achieve a sorting algorithm for S_n with $O(n)$ comparisons?

Question 2: (1+3+6=10 points)

We present another asymptotic notation that is used in the analysis of algorithms: For functions f, g, we say $f = o(g)$ ("f is little-o of g") if and only if the ratio $\frac{f(n)}{g(n)}$ goes to zero as n tends to infinity. For example, $\log n = o(n)$ but $n \neq o(n)$.

1. Is $\sqrt{n} = o(n)$?

2. Is it possible that $f = o(g)$ and $g = O(f)$? If yes, give an example. Otherwise prove that it is not possible.
3. Show that the worst case number of comparisons needed to merge two sorted lists, each of size n, is at least $2n - o(n)$.
(Hint: How many ways can we write down $2n$ numbers as two sorted lists of n numbers each? It may also be useful to look up the **central binomial coefficient (click here)** for its asymptotic growth.)

Question 3: (5+5=10 points)

In the algorithm to find the median in linear time, we grouped the elements into groups of 5 and used the medians of these groups. However what would happen if we had used groups of a different size? Consider the following cases:

1. we use groups of size 3
2. we use groups of size 7

In both cases write the recurrence for the running time of the algorithm and solve the recurrences to get the running times. You do not need to prove the correctness of your solution to the recurrences.

Question 4: (6+2=8 points)

Consider an array A with n elements where it is guaranteed that every element appears exactly twice in A, e.g., $A = (9, 7, 7, 1, 9, 1, 3, 5, 3, 5)$. For any two elements $A[i], A[j]$ in the array, we may only compare the elements by testing equality, i.e., $A[i] \overset{?}{=} A[j]$. With this in mind,

1. Give an algorithm that returns two positions in A that have the same element using at most $n - 2$ comparisons/equality tests.

   Note: These may be any two positions, so for example $(1, 5), (2, 3), (4, 6), (7, 9)$ or $(8, 10)$ are all valid outputs of this algorithm on A.
2. Prove that your algorithm really needs $n - 2$ comparisons in the worst case (i.e., there are inputs where it uses that many comparisons before terminating). Specifically, give an example of a worst-case input for $n = 10$.

Honors Questions

Question 5: Honors

In Question 4 we asked you to give an algorithm to find two positions with the same element.

- **(****)** We conjecture that any correct comparison-based deterministic algorithm must use *at least* $n - 2$ comparisons. Is the conjecture true? Can you prove it?
- **(**)** However, it is possible to construct a *randomized* algorithm which in expectation uses fewer comparisons. Suggest a randomized algorithm and state (try to justify) how many comparisons it requires in expectation. (Hint: suppose you just pick a few positions of the array at random. How many do you need to pick before there is a good chance that some element occurs at two of those positions?)