Foundations of Algorithms, Fall, 2025
Homework 2 FA25

Certain homework problems marked ***Collaborative Problem*** are to be worked by all members of the group, with each member *participating fully* and substantially contributing to the solution concept. **Each student must do the final write-up of his/her answers independently; do not copy either text or images from another group member into your own homework submission.** Individual participation will be evaluated via assessments submitted by each group member when the homework is due. Remember that all homework must include a personal statement of academic integrity, as noted in the Getting Started module.

1. [30 points total] Suppose that instead of swapping element $A[i]$ with a random element from the subarray $A[i...n]$, we swapped it with a random element from anywhere in the array.

Algorithm 1. Permute with All

```
function PERMUTEWITHALL(A)
    n ← A.length
    for i ← 1 to n do
        swap A[i] with A[RANDOM(1,n)]
```

Does this code produce a uniform random permutation? Why or why not?

2. [30 points total] Prove the correctness of the Twenty-One Card Trick. This neat little trick was taught to one of your professors when he was a kid, and is an example of oblivious compare/exchange sorting.

A *compare-exchange* operation on two array elements $A[i]$ and $A[j]$, where $i < j$, has the form

Algorithm 2. Compare and Exchange Values

```
function COMPAREEXCHANGE(A,i,j)
    if A[i] > A[j] then
        exchange A[i] with A[j]          ▷ i and j are determined ahead of time
```

After the compare-exchange operation, we know that $A[i] \leq A[j]$.

An *oblivious compare-exchange algorithm* operates solely by the sequence of pre-specified compare-exchange operations. We tell it the indices of the positions to be compared in advance, and although the indices can depend on the number of elements being sorted, they cannot depend on the values being sorted, nor can they depend on the result of any compare-exchange algorithm. If you're interested, here is an example of insertion sort using compare-exchange operations:

Algorithm 3. Insertion Sort with CompareExchange Operations

```
function INSERTIONSORT(A)
    for j ← to A.length do
        for i ← j − 1 downto 1 do
            COMPAREEXCHANGE(A,i,i + 1)
```

In the Twenty-One Card Trick, you start with 21 playing cards, fan them out, then ask your friend to choose a card of interest. (They do not remove or otherwise touch the card, they just remember what it is.) Then, begin the *deal phase* where you deal the cards face-down into three piles of seven cards each. Show each pile to your friend, asking, "is your card in this pile?" Take the pile with the chosen card—call it *the chosen pile*—and sandwich it in between the other two, making a single stack.[1] Repeat the deal phase two more times, for a total of three times. Then, take the stack, remove cards from the top one by one, and hold up the 11th card—like magic, it will be the chosen card.

[1] **IMPORTANT:** do not reorder the cards in the piles as you place them back into the stack.

The trick works because the chosen card will *migrate* toward the 11th position over the three deal phases of the trick. Your job will be to prove that this is the case, considering what position the chosen card starts out in, and what position it occupies in its pile in the next deal phase.

(a) [6 points] Prove that the top card in the chosen pile advances two positions in the next pile. That is, when you form the first three piles, suppose that the chosen card is the top card of its pile; when you deal out the three piles in the next phase, the chosen card will have moved by two positions away from the edge and in toward the center of its pile. Make a logical and/or mathematical argument that this is the case. (Hint: you may number the cards to help keep track of their positions in the piles and in the stack.)

(b) [6 points] Prove that the second and third positions advance by one.

(c) [6 points] Prove that once the chosen card appears at position 11 in the stack, it stays at position 11 through subsequent deals.

(d) [6 points] Prove that the chosen card appearing in positions 5, 6, or 7 within its pile will make it to the middle position after three deals. (Hint: relate these to the cases above that you already proved.)

(e) [6 points] The young person that taught this trick remarked, "it seems that about half the time, you only need to go through two rounds of deals–not three–and the trick still works" (the chosen card appears in position 11 in the deck). Why is this?

3. [40 points total] ***Collaborative Problem:*** A number of *peer-to-peer systems* on the internet are based on *overlay networks*. Rather than using the physical internet as the network on which to perform computation, these systems run protocols by which nodes choose collections of virtual "neighbors" so as to define a higher-level graph whose structure may bear little or no relation to the underlying physical network. Such an overlay network is then used for sharing data and services, and it can be extremely flexible compared with a physical network, which is hard to modify in real time to adapt to changing conditions.

   Peer-to-peer networks tend to grow through the arrival of new participants who join by linking into the existing structure. This growth process has an intrinsic effect on the characteristics of the overall network. Recently, people have investigated simple abstract models for network growth that might provide insight into the way such processes behave in real networks at a qualitative level.

   Here is a simple example of such a model. The system begins with a single node v_1. Nodes then join one at a time; as each node joins, it executed a protocol whereby it forms a directed link to a single other node chosen uniformly at random form those already in the system. More concretely, if the system already contains nodes $v_1, ..., v_{k-1}$ and node v_k wishes to join, it randomly selects one of $v_1, ..., v_{k-1}$ and links to that node.

   Suppose we run this process until we have a system consisting of nodes $v_1, ..., v_n$; the random process described above will produce a directed network in which each node other than v_1 has exactly one outgoing edge. On the other hand, a node may have multiple incoming links, or none at all. The incoming links to a node v_j reflect all the other nodes whose access into the system is via v_j; so if v_j has many incoming links, this can place a large load on it. Then to keep the system load-balanced, we would like all the nodes to have a roughly comparable number of incoming links. That is unlikely to happen, however, since nodes that join earlier in the process are likely to have more incoming links than nodes that join later. Let us try to quantify this imbalance as follows.

   (a) [20 points] Given the random process described above, what is the expected number of incoming links to node v_j in the resulting network? Give an exact formula in terms of n and j, and also try to express this quantity asymptotically (via an expression without large summations) using $\Theta(\cdot)$ notation.

   (b) [20 points] Part (a) makes precise a sense in which the nodes that arrive early carry an "unfair" share of connections in the network. Another way to quantify the imbalance is to observe that, in a run of this random process, we expect many nodes to end up with no incoming links. Give a formula for the expected number of nodes with no incoming links in a network grown randomly according to this model.