Keshav Dasu

For all questions (with the exception of 4) I am assuming the number of cards in the deck is 52.

Note:

For the iPad application I am only showing roughly every 100th shuffle for 3 seconds and quickly displaying the intermediary shuffles. I have chosen to implement it this way since, while testing sequences of 2 with my C++ implementation, I found it would take between 400 - 10,000 shuffles before finding two shuffles that are different. Even if I let each shuffle displayed for only 1 second it would take 2.7hrs to finish running. So I decided that this way would save everyone time. The console is printing out the card ID’s which can be used to confirm that the app is working.

Please complete the following task in C, C++, or Objective-C. You are free to use whatever data structures and design you prefer.  
  
1) create a deck of cards, and print/display the cards  
2) shuffle the deck of cards, and display the cards (in their shuffled order)  
3) reshuffle the deck, and display the cards (in their shuffled order)  
4) look for any 2-card sequence that was also present in the previous shuffle. If any are found, go back to step 3  
5) when two back to back shuffles do not contain a duplicate sequence, you're done.  
  
Questions:  
**1) How might you adapt your code to check for sequences of 3, 4, or more cards from the previous shuffle?**

I am now keeping track for each card (1 to K), which card was immediately after it. In the modification I can keep track of the next 3 or 4 cards that follow each card. Then I can check for each card if the attached sequence is the same before and after the shuffle. I can do that is in O(KN). K is the deck size.

**2) How might you adapt your code to check for combinations of N cards, rather than sequences?**

Before we shuffle the deck, suppose card # a(j) is in position j and after shuffling the deck card # b(j) is in position j.

I will calculate two sets of numbers:

For

And

For

If the set and the set have a number in common, then a sequence of length N is common to both the decks. Otherwise the two decks have no sequence of size N.

This is because v (j) and z (j) are uniquely determined by the specific N integers that go into their calculations.

I can check if set Z and V have a common number in O( K ln(K ). I will sort Z in O(K ln (K)) and then in O( ln (K)) check if *a* number in V is in Z.

(If for each shuffle I keep track of these numbers then I can also check if the combination ever repeated).

**3) How might you adapt your code to check for card sequences that were present in any previous shuffle? (i.e. comparing against all previous shuffles, not just the most recent one)**

I can use two different approaches.

One approach is for sequence of length 2. For length 2 for each card I will keep track of whether or not any of the other K-1 cards came after it. For this I will use a 2D array. The element (I,J) = 0 if card # J never came after card #I, and is 1 if card #J comes after card #I.

I can use this 2D array to check if in the current shuffle for each card whether the card that is following it also followed it in a previous shuffle.

If N >2 then the I can still use the 2 D array , but the number of columns will grow exponentially in N.

An alternative is to calculate a number that is unique to a sequence. For example if a sequence of length 5 is : {a(1), a(2), a(3), a(4), a(5)},

then the associated value is : .

I will use surds as multipliers. This will make sure that the sum is unique for a sequence.

For a sequence of length 5 I will calculate the following numbers for each shuffle:

for

If a v(j) value repeats then we know that the same sequence has repeated.

**4) If necessary, how might you change your code to accommodate more than 4 suits, or more than 13 different card values.**

The suits/values are only relevant in the visualization of the cards. I would have to rescale the images to fit on the screen properly. The method maps the deck to integer 1 to K.

**5) Would you do things any differently if you were optimizing purely for speed? How about if you were trying to minimize memory usage?**

The first way I approached finding a duplicate sequence of size 2 took an order of N^2 which is quite slow. I substituted that approach out for the approach described in question 1.

To speed up the iPad version of card shuffle, I would prefer to only show the two shuffles with nothing in common rather than showing every shuffle up to that point.

For memory usage I used a sprite sheet to generate all the cards instead of 52 different images being stored. I would like to set all of my array data structures to be the size of the deck. That way all memory allocated is being used.