Data Structures and Algorithms

Tutorial 2: Shuffling an Array

Naive Shuffle

Consider the following algorithm to Shuffle an array: draw elements randomly from the original array A, and mark taken elements using an array C so we don't take them away. To chose an element, just call RANDOM to generate a new index as long as the corresponding element has already been taken. We will assume that calls to RANDOM use a constant time to return values following a uniform distribution.

Input: an array A
Output: a shuffled copy of A

```
NAIVESHUFFLE(A)
n ← length(A)
for i ← 1 to n do C[i] ← 0
for i ← 1 to n do
    do
        j ← RANDOM(1, n)
    until C[j] = 0
    C[j] ← 1
    B[i] ← A[j]
return B
```

Questions:

1. Explain why this algorithm may not terminate.
2. Among the cases where the algorithm do terminate, what is the run-time complexity of the best case scenario to shuffle an array of size n with this algorithm?
3. Let t_i be a *random variable* (or *stochastic variable*) denoting the number of calls to RANDOM during the i-th iteration.
   (a) The probability that the second iteration makes only one call to RANDOM is
   $$\Pr\{t_2 = 1\} = \frac{n-1}{n}$$
   because there are $n-1$ possible free values to choose from the n. What is the probability $\Pr\{t_{i+1} = k+1\}$ to make $k+1$ calls to RANDOM (that means k unlucky random calls followed by 1 good call) after i values have already been taken?
   (b) Deduce the expected value $\mathrm{E}[t_{i+1} - 1]$.
   (c) Finally give the order (using Θ notation) of the average number of calls to RANDOM: $\sum_{i=1}^{n} \mathrm{E}[t_i]$.
4. Among all scenarios what is the probability of getting a best case?

The Modern Fisher-Yates Shuffle

This shuffle is also known as the *Knuth Shuffle*.

Input: an array A
Output: the array A shuffled in place

FisherYatesShuffle(A)
$n \leftarrow length(A)$
for $i \leftarrow 1$ to $n - 1$ do
    $j \leftarrow$Random(i, n)
    $A[i] \leftrightarrow A[j]$

Questions:

1. Explain why this algorithm always terminates.
2. What is the run-time complexity of this algorithm?
3. How can we justify that this algorithm is *unbiased*? (i.e., That it can generate all permutation with equal chance.)

Inside-Out Fisher-Yates Shuffle

Sometimes you do not want an array to be shuffle in place. This version of the algorithm will shuffle the array as it is being copied.

Input: an array A
Output: a shuffled copy of A

InsideOutFisherYatesShuffle(A)
$n \leftarrow length(A)$
$B[1] \leftarrow A[1]$
for $i \leftarrow 2$ to n do
    $j \leftarrow$Random$(1, i)$
    $B[i] \leftarrow B[j]$
    $B[j] \leftarrow A[j]$
return B

Questions:

1. Is this algorithm is *unbiased*?
2. What is its run-time complexity?
3. Such a shuffle could also be done by copying A on B and then calling FisherYatesShuttle(B). What would be the complexity of doing that?
4. Why would we prefer InsideOutFisherYatesShuffle over copy+FisherYatesShuttle?