Shuffling

Goldsmiths Computing

Motivation

Random permutations are useful for many applications:

- games with chance
- work distribution across a computational cluster
- component of randomized algorithms

Definition

Shuffling is the operation of taking a linear collection of items, and returning the collection with the items reordered according to a (uniformly) random permutation.

Shuffling by sort, broken version

```
function RandomComparison(x,y)
    return random() − 0.5
end function
function BadShuffle1(A)
    return sort(A,RandomComparison)
end function
```

Shuffling by sort, better version

```
function AttachRandom(A,T)
    for $0 \le i <$ length(A) do
        lookup(T,A[i]) ⟵ random()
    end for
end function
function IndexedRandomComparison(x,y)
    return lookup(T,x) - lookup(T,y)
end function
function ShuffleBySort(A)
    T ⟵ new HashTable()
    AttachRandom(A,T)
    return sort(A,IndexedRandomComparison)
end function
```

Complexity

Space

hash table with N entries, plus whatever space sort needs

$$\Rightarrow \Omega(N)$$

Shuffling by swap, broken version

```
function BadShuffle2(A)
    N ← length(A)
    for $0 \le i < L$ do
        r ← random()
        j ← $\lfloor N \times r \rfloor$
        swap(A[i],A[j])
    end for
end function
```

Fisher-Yates shuffle

```
function FisherYates(A)
    for N > i > 0 do
        r ← random()
        j ← ⌊(i+1) × r⌋
        swap(A[i],A[j])
    end for
end function
```

Complexity

Space

Only temporary variable space needed

$$\Longrightarrow \Theta(1)$$

Time

- N−1 iterations;
- constant work at each iteration

$$\Longrightarrow \Theta(N)$$

Work

1. Find out why BadShuffle1 and BadShuffle2 are bad:
    - implement BadShuffle1 and BadShuffle2;
    - run them each 60000 times on a test input of `[1,2,3]`, and record how often each possible output comes up;
    - compare against how often each possible output *should* come up