MSP 2024 - Tutorial 3

Exercise 1. Consider a stream of n elements where n is not known in advance.

1. Write a program in $\mathcal{O}(1)$ space that returns a random stream element, where all elements have equal probability of being picked.
2. Show correctness.

Solution

```
stream_random_element(stream):
  item = stream.next()
  item_count = 1
  while not stream.empty():
    next = stream.next()
    item_count += 1
    r = random_float(0,1)
    if r < 1/item_count:
      item = next
  return item
```

Correctness:

$$P(\text{i-th element is returned}) = P(\text{i-th element is stored to variable } \texttt{item}) \cdot P(\texttt{item} \text{ is not rewritten afterwards}) = \frac{1}{i} \cdot \left(\frac{i}{i+1} \cdot \frac{i+1}{i+2} \cdots \frac{n-2}{n-1} \cdot \frac{n-1}{n} \right) = \frac{1}{n}$$

Exercise 2. Assume an array with n elements.

1. Write a program that generates a random permutation of this array in $\mathcal{O}(n)$ time.
2. Show correctness.

Solution

```
array_shuffle(arr, n):
1.  for i from 1 to n-1:
2.     j = random_int(i,n)
3.     swap arr[i] and arr[j]
```

Correctness: $P(\texttt{j}\text{-th element ends up on position } \texttt{k}) =$
$P(\texttt{arr[j]} \text{ is not swapped first } (\texttt{k}\text{-}1) \text{ times}) \cdot$
$P(\texttt{arr[j]} \text{ is swapped during } \texttt{k}\text{-th iteration}) =$
$\left(\frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdots \frac{n-k+1}{n-k+2}\right) \cdot \frac{1}{n-k+1} = \frac{1}{n}$

Exercise 3. Assume the following modification of the hiring problem, where the company must pay severance to the fired worker if they have just been hired:

```
Hire-Assistant(n)
hire 1, best = 1
for i from 2 to n:
  if candidate i is better than best:
    fire best
    if best == i-1:
      pay severance to best
    hire i, best = i
```

Assume that candidates arrive at the interview in random order. Compute how many times the severance will be paid:

a) in the best case, b) in the worst case, c) on average.

Solution

(a) 0 times, e.g. if candidate 1 is the best one

(b) $n-1$ times iff the candidates are sorted from worst to best

(c) Let X_i, $2 \leq i \leq n$, be the indicator variable that attains value 1 if the severance is paid during i-th iteration, and 0 otherwise. The severance is paid during i-th iteration when (i-1)-th as well as i-th candidates were hired, that is, if candidate #(i-1) is the best among first (i-1) candidates and candidate #i is the best among first i candidates. Thus, $P(X_i = 1) = \frac{1}{i-1} \cdot \frac{1}{i}$.

The expected total number of paid severances is then

$$\mathrm{E}\left[\sum_{i=2}^{n} X_i\right] = \sum_{i=2}^{n} \mathrm{E}\left[X_i\right] = \sum_{i=2}^{n} P(X_i = 1) = \sum_{i=2}^{n} \frac{1}{i-1} \cdot \frac{1}{i} = \sum_{i=2}^{n} \frac{1}{i-1} - \frac{1}{i} =$$

$$= \left(1 - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \cdots + \left(\frac{1}{n-2} - \frac{1}{n-1}\right) + \left(\frac{1}{n-1} - \frac{1}{n}\right) =$$

$$= 1 - 1/n$$

Exercise 4. Consider the following algorithm describing one pass of the *bubble-sort* algorithm. Assume that the input array a (indexed from 1) contains numbers from 1 to n in random order (all permutations have equal probability).

```
bubble-sort-one-pass(a,n):
   for i from 1 to n-1:
      if a[i] > a[i+1]:
         swap a[i] and a[i+1]
```

Determine:

1. Probability that no elements are swapped (best-case behaviour).
2. Probability that $n-1$ swaps are executed (worst-case behaviour).
3. The expected number of swaps (average-case behaviour) – it is sufficient to give the asymptotic number of swaps including an explanation.

Solution

1. No elements are swapped when array is sorted: $P = \frac{1}{n!}$
2. $n-1$ swaps are executed when `a[1]` contains n: $P = \frac{1}{n}$
3. Let X_i, $1 \leq i \leq n-1$, be the indicator variable that attains value 1 if a swap is performed during i-th iteration, and 0 otherwise. Right before the i-th iteration, `a[i]` contains the maximum value between `a[1]`...`a[i]`. Then, X_i is 0 when `a[i+1]` is the largest number between `a[1]`...`a[i+1]`, i.e. with probability $\frac{1}{i+1}$. Thus, $P(X_i = 1) = 1 - \frac{1}{i+1}$.

   The expected total number of swaps is then

$$\mathrm{E}\left[\sum_{i=1}^{n-1} X_i\right] = \sum_{i=1}^{n-1} \mathrm{E}\left[X_i\right] = \sum_{i=1}^{n-1} P(X_i = 1) = \sum_{i=1}^{n-1} 1 - \frac{1}{i+1} =$$
$$= \sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} \frac{1}{i+1} = n - 1 - \sum_{i=1}^{n-1} \frac{1}{i+1} =$$
$$= \mathcal{O}(n) - \mathcal{O}(\log n) = \mathcal{O}(n)$$

Exercise 5. Consider the following randomised algorithm `all_even` that tests whether array `arr` of size `len` > 0 (indexed from 1) contains only even numbers. Function `rand_int(1,len)` returns a random integer in range `1` to `len` with the uniform probability.

```
all_even(arr,len,n):
for i from 1 to n:
    k = rand_int(1,len)
    if arr[k] is odd:
        return false
return true
```

1. Decide and justify whether `all_even` is a Las Vegas or a Monte Carlo algorithm.
2. Construct function $\mathcal{P}(\mathtt{len}, \mathtt{n})$ that returns, for the given `len` and $\mathtt{n} > 0$, the upper bound on the probability (with respect to all input arrays of size `len`), that `all_even(arr,len,n)` returns a wrong result.
3. Determine the smallest `n` such that the worst-case probability that `all_even(arr,10,n)` returns a wrong result is smaller than 50%.

Solution

1. It is a Monte Carlo algorithm because there is a non-zero probability that for a given input the algorithm returns a wrong result.
2. Clearly, if $\texttt{len} = 1$, then $\mathcal{P}(\texttt{len}, \texttt{n}) = 0$. It is easy to see that the worst-case input contains only a single odd number. For such inputs $\mathcal{P}(\texttt{len}, \texttt{n}) = \left(\frac{len-1}{len}\right)^n$.
3. Since $P(10, \texttt{n}) = \left(\frac{9}{10}\right)^n$, we require that $\left(\frac{9}{10}\right)^n < 0.5$, i.e. $\log_{0.9}(0.9^n) > \log_{0.9}(0.5) \Rightarrow n > 6.5$. Hence $n = 7$.