

# Stream Sampling

1. I'm going to start listing numbers off to you. I'll stop at some arbitrary point and ask you to give me a uniformly sampled number from the ones I listed.
  - a. Can you design a simple algorithm to do this?
  - b. Now what if I ask you to give me a uniform sample of  $k$  numbers without replacement from the ones I gave (I'll tell you  $k$  up front). Does your solution easily adapt?
  - c. Let's think about our solution and get a sense of how efficient it is. In computer science, we measure efficiency in few different metrics. Two of the most important are *time* and *space*.

*Time* is a question of how many steps an algorithm needs to compute an output. *Space* is a question of how much information an algorithm needs to “write down” while computing this out.

Specifically, we care about the *complexity* of these resources, which is a measurement of how much of the resource in question an algorithm uses as a function of the input size.

Consider as an example an algorithm that looks for a target number in a list of  $n$  numbers. In the worst case, it'll have to look at every number in the list—of which there are  $n$ —once, so it takes  $n$  steps. We say this algorithm<sup>1</sup> is  $O(n)$ , or linear.

On the other hand, the algorithm only has to write down the thing it's looking for. Assuming we can write a number down in a constant amount of space<sup>2</sup>, we say it takes  $O(1)$ , or “constant”, space.

We usually consider algorithms time-efficient and/or space-efficient if they use  $O(n)$  time and/or  $O(\log n)$  space respectively. Suppose I ultimately list off  $n$  numbers. Does our algorithm meet these standards?

---

<sup>1</sup> which is called linear search

<sup>2</sup> In particularly strict theoretical analyses, this doesn't quite hold but it's quite accurate to real-world computing.

2. Let's think about why we use as much space as we do and how we can use less.

- a. Let's simplify the problem for a moment by only sampling a single number—i.e. fixing  $k = 1$ .

If I ask you to give me the minimum number I list instead of a random one, can you come up with a more space-efficient solution?

- b. Can we turn the random-sampling problem into a min-finding problem?

*Hint: instead of just storing the numbers as they come in, what if we also store a random “tag” sampled uniformly from  $[0, 1)$  ? You may assume that we can take such samples at will.*

- c. Can we scale this solution into one that works for a sample of  $k$  numbers (again without replacement)? What are the time and space costs of this solution?

For time, you should note that it is possible to maintain a collection of  $k$  numbers using  $O(k)$  space<sup>3</sup> such that it costs:

- $O(1)$  time to find the maximum
- $O(\log k)$  time to remove the maximum
- $O(\log k)$  time to add a new item to the collection.<sup>4</sup>

- d. Suppose that while I'm listing numbers, I can stop you multiple times to get up-to-then random samples. Is there a statistical difference between the samples given by solutions 1 and 2?

---

<sup>3</sup> I'm making the simplifying assumption here that it takes  $O(1)$ —that is, constant—space to write down a number.

<sup>4</sup> This can be done by arranging the  $k$  elements into a data structure called a binary heap.

## 3. Can we do even better?

- a. Let's again go back to the  $k = 1$  case. Let  $E_i$  denote the event in which we take the  $i$ th number as our current sample. We'll always take the first number we are given as the sample, so  $\mathbb{P}[E_1] = 1$ . What is  $\mathbb{P}[E_2]$ ?  $\mathbb{P}[E_1]$ ?
- b. How can we use this idea to make a new version of our sampler? What are the time and space efficiency of this new version?
- c. Can you prove that this solution gives a uniform random sample? I.e. can you prove that for a list of  $n$  numbers, for every number  $x$  in the list,  $\mathbb{P}[\text{solution returns } x] = \frac{1}{n}$ ?  
*Hint: try induction on  $i$ , the number of numbers we've run through the sampler.*
- d. Can we generalize this solution to work for a size  $k$  sample?
- e. This final solution is called *reservoir sampling*, and is deployed in the real world in order to take samples from massive real-time data sets. What are its time and space costs?