## 2.2 Binomial and Geometric Distribution

When we have $n$ copies $Y_1, \ldots, Y_n$ of the same Bernoulli random variable with parameter $p$ and add them, then we get a binomially distributed random variable $X = Y_1 + Y_2 + \ldots + Y_n$ with parameters $n$ and $p$. For example, $X$ could describe the number of successful runs of an algorithm or the number of times that we see heads in a series of $n$ coin flips. The random variable $X$ in Example 2.9 is binomially distributed with parameters 10 and 1/2. Analogously to that example, we can use linearity of expectation to confirm our suspicion that the expected value of a binomially distributed variable with parameters $n$ and $p$ should be $np$.

**Lemma 2.15.** *Let $X = Y_1, \ldots, Y_n$ be a binomially distributed random variable with parameters $n$ and $p$. Then $\mathbf{E}[X] = np$.*

*Proof.* We know that $Y_i$ is a Bernoulli random variable with parameter $p$ for all $i \in \{1, \ldots, n\}$. We already observed that that implies that $\mathbf{E}[Y_i] = \mathbf{Pr}(Y_i = 1) = p$. By linearity of expectation, we get that

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \mathbf{E}[Y_i] = np.$$

□

The expected value of a binomially distributed random variable is easy to memorize. The probabilities for obtaining a specific number describe the distribution more precisely, but are a little more complex to compute.

**Lemma 2.16.** *Let $X = Y_1, \ldots, Y_n$ be a binomially distributed random variable with parameters $n$ and $p$. Then*

$$\mathbf{Pr}(X = j) = \binom{n}{j} \cdot p^j (1-p)^{n-j}.$$

*Proof.* Observe that there are $m = \binom{n}{j}$ ways how $j$ ones can occur in the sum of the $n$ random variables. Each possibility has probability $p^j(1-p)^{n-j}$. Formally, we can define an event $A_i$ for each of the $m$ possibilities and observe that these events are disjoint and their union is the event $X = j$. Each event has probability $p^j(1-p)^{n-j}$, so we have

$$\mathbf{Pr}(X = j) = \cup_{i=1}^{m} \mathbf{Pr}(A_i) = \sum_{i=1}^{m} \mathbf{Pr}(A_i) = m \cdot p^j (1-p)^{n-j} = \binom{n}{j} \cdot p^j (1-p)^{n-j}.$$

□

Another related type of random variables are *geometrically distributed* random variables. A geometrically distributed random variable models how long it takes until a Bernoulli experiment returns 1 for the first time. Let $Y_1, Y_2, Y_3, \ldots$ be independent

identical copies of the same Bernoulli random variable with parameter $p$. Then the corresponding geometrically distributed random variable $X : \Omega \to \mathbb{N}$ with parameter $p$ has value $i$ iff $Y_j = 0$ for $j < i$ and $Y_i = 1$. With such a random variable we can for example model how long we have to wait until independent repetitions of the same randomized algorithms lead to the first success.

The probability that $X = i$ is exactly $(1-p)^{i-1}p$ because the event $Y_i$ happens if $i-1$ tries return 0 and the $i$th try returns 1. We know from Lemma 2.15 that the binomially distributed random variable with parameters $n$ and $p$ has expected value $np$, thus the number of tries that return 1 within the first $1/p$ tries is 1 (assuming that $1/p$ is an integer). This gives us the intuition that the corresponding geometrically distributed variable with parameter $p$ should have expected value $1/p$. We see that that is indeed the case.

**Lemma 2.17.** *Let $X$ be a geometrically distributed random variable with parameter $p > 0$. Then $\mathbf{E}[X] = 1/p$.*

*Proof.* We recall that Lemma 2.10 says that $\mathbf{E}[X] = \sum_{j=1}^{\infty} \mathbf{Pr}(X \geq j)$. What is the probability that at least $j$ tries are necessary to get the first 1? This happens if and only if the first $j-1$ tries fail, so $\mathbf{Pr}(X \geq j) = (1-p)^{j-1}$. Now we can use that $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$ for any $r \neq 1$ (this is a geometric *series*) to obtain that

$$\mathbf{E}[X] = \sum_{j=1}^{\infty} \mathbf{Pr}(Y \geq j) = \sum_{j=1}^{\infty} (1-p)^{j-1} = \sum_{j=0}^{\infty} (1-p)^j = \frac{1}{1-1+p} = 1/p.$$

□

Up to this point, we talked a lot about the fact that we expect to need $1/p$ tries until the first 1 occurs, that we expect one 1 among the first $1/p$ tries and that we can make the probability that no 0 occurs smaller and smaller by using even more independent tries. Of course, this does *not* mean that a 1 in the next try becomes more likely only because we have already seen a lot of 0s. On the contrary, an important property of the geometric distribution is that it is *memoryless*.

**Lemma 2.18.** *Let $X$ be a geometrically distributed random variable with parameter $p$. It holds for all $n \in \mathbb{N}$ and $k \in \mathbb{N}_0$ that*

$$\mathbf{Pr}(X = n + k \mid X > k) = \mathbf{Pr}(x = n).$$

*Proof.* By definition, we have that

$$\begin{aligned}\mathbf{Pr}(X = n+k \mid X > k) &= \frac{\mathbf{Pr}\Big((X = n+k) \cap (X > k)\Big)}{\mathbf{Pr}(X > k)} = \frac{\mathbf{Pr}(X = n+k)}{\mathbf{Pr}(X > k)} \\ &= \frac{(1-p)^{n+k-1} \cdot p}{(1-p)^k} = (1-p)^{n-1} \cdot p = \mathbf{Pr}(X = n).\end{aligned}$$

For the third equality, we observe that the probability that we need more than $k$ tries to get the first 1 is the probability that we get 0 in $k$ consecutive tries. □

## 2.3 Applications

Our application section features the powerful linearity of expectation in different contexts, mixed with the knowledge we aquired about integer valued random variables. We start with *randomized QuickSort*, showing how randomization can protect an algorithm from worst case input instances. Then we do an excursion to randomized approximation algorithms, where we consider algorithms for the maximum cut problem and the vertex cover problem.

### 2.3.1 Randomized QuickSort

In this section, we analyze a randomized version of the popular sorting algorithm `QuickSort`. Recall that `QuickSort` is a recursive Divide&Conquer algorithm. As long as there are at least two elements, it chooses a pivot (element) $x$, partitions the elements except $x$ in those smaller than $x$ and larger than $x$ and recursively sorts the two subsets. The results are then concatenated appropriately. The following pseudo code captures the essence of `QuickSort`. We assume that the input is a set of distinct numbers. Notice that we do not specify how the set is stored at the beginning or during the algorithm, we want to focus on the main algorithmic steps. We model the output as an ordered vector. The desired output is the (unique) vector where the input elements are sorted increasingly.

**`QuickSort`**$(S = \{x_1, \ldots, x_n\})$

1. **if** $(n = 0)$ **then return** $()$
2. **if** $(n = 1)$ **then return** $(x_1)$
3. Choose a *pivot element* $x \in S$
4. Compute $S_1 = \{x_i \mid x_i < x\}$ and $S_2 = \{x_i \mid x_i > x\}$
5. $A_1 =$ `QuickSort`$(S_1)$; $A_2 =$ `QuickSort`$(S_2)$
6. Concatenate $A_1$, $(x)$ and $A_2$ to obtain the vector $A$
7. **return** $A$

The running time of `QuickSort` depends crucially on the specification of step 3, the choice of the pivot element. We recall that the worst case running time of `QuickSort` is $\Theta(n^2)$ if the algorithm chooses $x_1$ as the pivot element in every step. The bad case that can then occur (in every call with $n \geq 2$) is that one of the sets $S_1$ and $S_2$ has $n-1$ elements and the other is empty because $x_1$ always happened to be the smallest / largest element in $S$. This happens if the array is sorted in the beginning.

Ideally, the pivot element is the median of the elements in $S$ because that splits $S$ into two sets of equal size (or nearly equal size, depending on whether $n$ is odd or even). Computing the median in time $\mathcal{O}(n)$ is possible (an algorithm to do so was published in [BFP+73]). We will also see an easier randomized algorithm to compute the median later in this lecture. However, for the purpose of a lean QuickSort implementation it is even easier to randomize the choice of the pivot element in a straightforward manner.

We analyze the algorithm `RQuickSort` that always chooses the pivot element uniformly at random from $S$ in Step 3.

**Theorem 2.19.** *The expected number of comparisons of `RQuickSort` is $2n \ln n + \Theta(n)$ for any input $S$ with $n$ distinct elements.*

*Proof.* Let $(y_1, \ldots, y_n)$ be the sorted vector containing the elements $x_1, \ldots, x_n$, i.e. the output of `RQuickSort`. For any pair $i, j \in \{1, \ldots, n\}$ with $i < j$ we define the (indicator) random variable $X_{ij}$ by

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared during the execution of } \texttt{RQuicksort} \\ 0 & \text{else.} \end{cases}$$

Observe that two numbers $y_i$ and $y_j$ are never compared twice during the execution of `RQuicksort`: Comparisons are only made between a pivot element and other numbers. Thus, when $y_i$ and $y_j$ are compared cor the first time, one of them is the pivot element. Thus, it is no longer present in the recursive calls and no further comparisons between $y_i$ and $y_j$ can occur. Thus, if we model the total number of comparisons by the random variable $X$, we have

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{Pr}(X_{ij} = 1)$$

where we use linearity of expectation. To compute $\mathbf{Pr}(X_{ij} = 1)$, we observe that $y_i$ and $y_j$ are compared iff and only if either $y_i$ or $y_j$ is the first element from the set $\{y_i, \ldots, y_j\}$ that is chosen as a pivot element. To see that this is true, let $x$ be the first element from $\{y_i, \ldots, y_j\}$ that is chosen as the pivot element.

- If $x \neq y_i$ and $x \neq y_j$, then $y_i$ and $y_j$ are compared with $x$ and not with each other. Since $x \in \{y_i, \ldots, y_j\}$, we have $y_i < x < y_j$. Thus, the two elements are then separated because $y_i \in S_1$ and $y_j \in S_2$. Thus, they are never compared.
- If $x = y_i$ or $x = y_j$, then $y_i$ and $y_j$ are compared. In the recursive calls, $x$ is no longer present and no further comparisons occur.

Let $P$ be the random variable that has the chosen pivot element as its value. The algorithm chooses an element uniformly at random from the current set $S$, so $\mathbf{Pr}(P = x) = \frac{1}{|S|}$ for all $x \in S$. As long as no element from $\{y_i, \ldots, y_j\}$ is chosen as a pivot element, a call of `RQuickSort` either has all or no elements from $\{y_i, \ldots, y_j\}$ in $S$. We observe that

$$\begin{aligned}
&\mathbf{Pr}((P = y_i) \cup (P = y_j) \mid P = x \text{ with } x \in \{y_i, \ldots, y_j\}) \\
=&\frac{\mathbf{Pr}(((P = y_i) \cup (P = y_j)) \cap (P = x \text{ with } x \in \{y_i, \ldots, y_j\}))}{\mathbf{Pr}(P = x \text{ with } x \in \{y_i, \ldots, y_j\})} \\
=&\frac{\mathbf{Pr}((P = y_i) \cup (P = y_j))}{\mathbf{Pr}(P = x \text{ with } x \in \{y_i, \ldots, y_j\})} \\
=&\frac{2/|S|}{(j - i + 1)/|S|} = \frac{2}{j - i + 1}.
\end{aligned}$$

This is intuitive: The first element from $\{y_i, \ldots, y_j\}$ that is chosen is chosen uniformly at random from a super set of $\{y_i, \ldots, y_j\}$, so *if* it is from $\{y_i, \ldots, y_j\}$, then it is a uniformly chosen element from $\{y_i, \ldots, y_j\}$. Since $y_i$ and $y_j$ are two elements, choosing one of them has a probability of $2/(j-i+1)$. We observe that

$$\mathbf{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{Pr}(X_{ij} = 1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k}.$$

For a specific $j \in \{2, \ldots, n\}$, the term $1/j$ occurs in the second sum iff and only if $j \leq n-i+1$. This is true for all $i \in \{1, \ldots, n-j+1\}$, thus the term $1/j$ occurs exactly $n-j+1$ times. We thus get that

$$\begin{aligned}\sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} = \sum_{j=2}^{n} \frac{2}{j}(n-j+1) &= (n+1)\left(\sum_{j=2}^{n} \frac{2}{j}\right) - 2(n-1) \\ &= (2n+2)(H_n - 1) - 2(n-1),\end{aligned}$$

where $H_n = \sum_{i=1}^{n}(1/i)$ is the *n-th harmonic number*. Since $H_n = \ln n + \Theta(1)$, we get that

$$\mathbf{E}[X] = (2n+2)(\ln n + \Theta(1)) - 2(n-1) = 2n \ln n + \Theta(n).$$

□