11 Randomness in Algorithms

Most data structures and algorithms in this book are deterministic: the code decides the execution steps exclusively on available data and hence, if the same code is run twice on the same input, it takes the same amount of time on an ideal computer and yields the exact same outcome. Randomized algorithms, on the other hand, use randomness (e.g., flip coins) inside their computation to decide the execution steps. As a result, if the same code is run twice on the same input, it may take different amounts of time and may yield different outcomes in some cases.

Why randomness?
Randomized algorithms are often simpler and have better performance. Part of the reason might be that while an adversary can craft inputs that bring out the worst of a deterministic algorithm, it is far more difficult to do the same for "unpredictable" algorithms that use randomness in their decisions. However, their analysis tends to be more involved.

We will begin with a brief discussion of concepts in discrete probability, typically covered in an undergraduate discrete mathematics course. Following that, we will develop the machinery needed to analyze algorithms that make use of randomness. We will revisit randomized variants of previously discussed algorithms and study their performance characteristics.

11.1 The Rules of Discrete Probability

We start by discussing the notion of a probability space. Every statement involving probability and randomness either implicitly or explicitly refers to an underlying probability space.

A *probability space*, often denoted by a triplet $(\Omega, \mathcal{F}, \mathbf{Pr})$, consists of three components: First, we have a set of possible outcomes, known as a *sample space* Ω. Second, we have a family of sets $\mathcal{F}$ of allowable events, where each set in $\mathcal{F}$ is a subset of the sample space Ω. Third, we have a probability function $\mathbf{Pr} : \mathcal{F} \to [0, 1]$ satisfying

The notion of probability space was introduced in the 1930s by Russian mathematician Andrey Kolmogorov.

1. $\mathbf{Pr}[\Omega] = 1$; and
2. for any finite (or countably infinite) sequence of disjoint events $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$,
$$\mathbf{Pr}\left[\bigcup_{i=1}^{n} \mathcal{E}_i\right] = \sum_{i=1}^{n} \mathbf{Pr}[\mathcal{E}_i] .$$

This setup models a random process, which is often called a *random experiment*. When a random experiment is conducted, the nature picks a single outcome $\omega \in \Omega$ in a way that the probability that an outcome happens obeys the probability function $\mathbf{Pr}$.

Two examples are in order to illustrate the concept of a probability space. The first example is concerned with a fair coin. A fair coin has two sides and both sides are equally likely to turn up.

Fair Coin. Much can be derived from the properties of a sample space. The two sides of a fair coin are equally likely to turn up, i.e., $\mathbf{Pr}[H] = \mathbf{Pr}[T]$. Furthermore, the probability function totals to 1, i.e., $\mathbf{Pr}[H] + \mathbf{Pr}[T] = 1$. Hence, $\mathbf{Pr}[H] = \mathbf{Pr}[T] = \frac{1}{2}$.

Example. In a random process that flips a fair coin once, the sample space is $\Omega = \{H, T\}$, denoting heads and tails respectively. The allowable events are $\emptyset, \{H\}, \{T\}, \{H, T\}$, corresponding to the following events:

$\emptyset$ — the coin turned up nothing;

$\{H\}$ — the coin turned up heads;

$\{T\}$ — the coin turned up tails; and

$\{H, T\}$ — the coin turned up heads or tails.

The second example is concerned with a standard die. A standard die has 6 sides and each side is equally likely to turn up.

By a similar reasoning, the probability that a fair 6-sided die turns up k, where $k = 1, 2, \ldots, 6$, is exactly $\frac{1}{6}$.

Example. In a random process that tosses a standard die once, the sample space is $\Omega = \{1, 2, 3, \ldots, 6\}$, the numbers on the die's faces.
There are numerous allowable events. The size of 2^Ω is $2^6 = 64$. Example events include:

$\{3\}$ — the event that the die turns up 3;

$\{2, 4\}$ — the event that the tie turns up 2 or 4; and

$\{1, 3, 5\}$ — the event that the die turns up odd.

Not all events can happen, however. As an example, the event $\emptyset$ (the die did not turn up anything) is not a possibility. Still, there is no harm in including it in $\mathcal{F}$.

Throughout this book, we are only concerned with discrete probability where the sample space Ω is finite and the set of allowable events $\mathcal{F}$ satisfies $\mathcal{F} \subseteq 2^\Omega$. In this common scenario, when each outcome in the sample space is equally likely to happen, the probability of an event $\mathcal{E} \in \mathcal{F}$ can be found by counting the number of outcomes in the event and dividing by the total number of possible outcomes; that is,

$$\mathbf{Pr}[\mathcal{E}] = \frac{|\mathcal{E}|}{|\Omega|} = \frac{\text{\# of outcomes that meet the condition}}{\text{\# of total outcomes}}$$

allowing counting techniques from combinatorics to be used. As an example, consider rolling two dice.

Rolling two dice has the following 36 outcomes:

$(1,1), (1,2), \ldots, (1,6)$
$(2,1), (2,2), \ldots, (2,6)$
$(3,1), (3,2), \ldots, (3,6)$
$\vdots$
$(6,1), (6,2), \ldots, (6,6)$

Example. The probability that the sum of the two numbers is 5 can be calculated by counting the combinations that sum to 5. There are 4 combinations: $(1,4), (2,3), (3,2), (4,1)$. Hence, the probability is $\frac{4}{36} = \frac{1}{9}$. By the same reasoning, there are 6 combinations that sum to 7 and so the probability that the sum of the two numbers is 7 is $\frac{6}{36} = \frac{1}{6}$.

It is possible to generalize this to settings in which the outcomes have different probabilities. A probability distribution on Ω is given by a function

$p : \Omega \to \mathbb{R}_+$, known as a *probability mass function (pmf)*, which satisfies $\mathbf{Pr}[\omega] = p(\omega)$ for all $\omega \in \Omega$. Then, the probability of an event $\mathcal{E}$ can be found by adding the probabilities $p(\omega)$ of all outcomes $\omega \in \mathcal{E}$, i.e., $\mathbf{Pr}[\mathcal{E}] = \sum_{\omega \in \mathcal{E}} p(\omega)$.

Basic Properties. Two properties follow directly from the properties of a probability space. The proof is simple and is left as an exercise to the reader.

Theorem 11.1. Let A and B be events defined on a probability space $(\Omega, \mathcal{F}, \mathbf{Pr})$. Then,

- $\mathbf{Pr}\left[\overline{A}\right] = 1 - \mathbf{Pr}[A]$, where $\overline{A}$ denotes the set complement of A.
- If $A \subseteq B$, then $\mathbf{Pr}[A] \leqslant \mathbf{Pr}[B]$.

Independence. The concept of independence formalizes the idea that the occurrence of an event does not affect the odds of another occurring. Two events A and B are *independent* if and only if $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B]$. When we have multiple events, we say that $A_1, \ldots, A_k$ are *mutually independent* if and only if for any nonempty subset $I \subseteq \{1, \ldots, k\}$,

$$\mathbf{Pr}\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \mathbf{Pr}[A_i] \,.$$

Two Dice. Roll two dice. Let A be the event that the first die comes up even and B be the event that the second die comes up odd. We find that $\mathbf{Pr}[A \cap B] = \frac{1}{4}$, $\mathbf{Pr}[A] = \frac{1}{2}$, and $\mathbf{Pr}[B] = \frac{1}{2}$. We expect the outcome of the first die to have no influence on the second die. Indeed,

$$\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B] \,,$$

confirming our intuition.

11.2 Random Variables and Expectation

A *random variable* is a function $f : \Omega \to \mathbb{R}$. That is to say, a random variable assigns a numerical value to an outcome $\omega \in \Omega$. Random variables are typically denoted by capital letters $X, Y, \ldots$.

Example. Consider one toss of a 6-sided fair die. We can define a random variable X to be the number on the face that turns up. Hence, X belongs to the set $\{1, 2, 3, \ldots, 6\}$. As another example, consider one flip of a fair coin. We can define a random variable Y so that it will be 1 if the coin turns up heads and 0 otherwise.

As it turns out, the random variable Y in the example above is a particular type of random variables that comes up often and is called an indicator random variable:

Definition 11.2. For an event $\mathcal{E}$, the *indicator random variable* $\mathbb{I}\{\mathcal{E}\}$ takes on the value 1 if $\mathcal{E}$ occurs and 0 otherwise.

For a (discrete) random variable X and a number $a \in \mathbb{R}$, the event "$X = a$" is the set $\{\omega \in \Omega : X(\omega) = a\}$. Therefore,

$$\mathbf{Pr}[X = a] = \mathbf{Pr}[\{\omega \in \Omega : X(\omega) = a\}]$$

We can similarly define events such as $X \leqslant a$ or $X > a$. The concept of independence also carries over to random variables: two random variables X and Y are independent if for every two sets $A, B \subseteq \mathbb{R}$,

$$\mathbf{Pr}[X \in A \wedge Y \in B] = \mathbf{Pr}[X \in A]\,\mathbf{Pr}[Y \in B]\,.$$

Expectation

The expectation (also known as the expected value) of a random variable is the weighted average of the values of the function over all outcomes, where the weight is the probability of each outcome.

Expected value of $\mathbb{I}\{\mathcal{E}\}$?

$$\begin{aligned}\mathbf{E}[\mathbb{I}\{\mathcal{E}\}] &= 0 \cdot \mathbf{Pr}\left[\overline{\mathcal{E}}\right] + 1 \cdot \mathbf{Pr}[\mathcal{E}] \\ &= \mathbf{Pr}[\mathcal{E}]\,.\end{aligned}$$

Definition 11.3 (Expected Value)**.** The expected value of a random variable X, denoted by $\mathbf{E}[X]$, is

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} X(\omega)\,\mathbf{Pr}[\omega] = \sum_{k} k \cdot \mathbf{Pr}[X = k]$$

Two examples below illustrate how to compute the expected value of a random variable straight from the definition.

Six-sided Fair Die. Because each side of a fair die has the same probability of turning up, $\mathbf{Pr}[X = 1] = \mathbf{Pr}[X = 2] = \cdots = \mathbf{Pr}[X = 6] = \frac{1}{6}$.

Example. The expectation of the variable X representing the value of a fair die is

$$\begin{aligned}\mathbf{E}[X] &= \sum_{i=1}^{6} i \cdot \mathbf{Pr}[X = i] \\ &= 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} \\ &= \frac{7}{2}.\end{aligned}$$

Flip a fair coin 3×. There are $2^3 = 8$ possible outcomes, so each outcome has a probability of $\frac{1}{8}$.

Example. A fair coin is flipped *three* times. Let X be the random variable that gives the number of heads. What's the expected value of X?

$$\begin{aligned}\mathbf{E}[X] &= \frac{1}{8}\Big(X(\mathsf{HHH}) + X(\mathsf{HHT}) + X(\mathsf{HTH}) + X(\mathsf{HTT}) \\ &\qquad + X(\mathsf{THH}) + X(\mathsf{THT}) + X(\mathsf{TTH}) + X(\mathsf{TTT})\Big) \\ &= \frac{1}{8}\,(3+2+2+1+2+1+1+0) \\ &= \frac{12}{8} = \frac{3}{2}.\end{aligned}$$

Linearity of Expectations

One of the most important theorems in probability is *linearity of expectations*. As stated and proved below, it says that given two random variables, the sum of their expected values is the same as the expected value of their sum.

Theorem 11.4 (Linearity of Expectations)**.** Let X and Y be random variables. Then, for any constant $a \in \mathbb{R}$,

1. $\mathbf{E}[a \cdot X] = a \cdot \mathbf{E}[X]$; and
2. $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$.

Manipulation of random variables. Random variables can be added, multiplied, or applied to a function. For example, it is common to write $Z = X + Y$, which means $Z(\omega) = X(\omega) + Y(\omega)$ for all ω. Furthermore, if $f : \mathbb{R} \to \mathbb{R}$, we can write $X' = f(X)$, which means $X'(\omega) = f(X(\omega))$ for all $\omega \in \Omega$.

Proof. Let $a \in \mathbb{R}$. Then, by the definition of expectation,

$$\mathbf{E}[a \cdot X] = \sum_{\omega \in \Omega} a \cdot X(\omega)\, \mathbf{Pr}[\omega] = a \sum_{\omega \in \Omega} X(\omega)\, \mathbf{Pr}[\omega] = a \cdot \mathbf{E}[X]\,,$$

which proves the first clause. For the second clause, let $Z = X + Y$, so $Z(\omega) = X(\omega) + Y(\omega)$ for every $\omega \in \Omega$. Then,

$$\begin{aligned}
\mathbf{E}[X + Y] = \mathbf{E}[Z] &= \sum_{\omega \in \Omega} Z(\omega) \cdot \mathbf{Pr}[\omega] \\
&= \sum_{\omega \in \Omega} (X(\omega) + Y(\omega))\, \mathbf{Pr}[\omega] \\
&= \left(\sum_{\omega \in \Omega} X(\omega) \cdot \mathbf{Pr}[\omega] \right) + \left(\sum_{\omega \in \Omega} Y(\omega) \cdot \mathbf{Pr}[\omega] \right) \\
&= \mathbf{E}[X] + \mathbf{E}[Y]\,,
\end{aligned}$$

which completes the proof. □

The theorem is widely applicable because it does not require the events to be independent. It can be applied to any random variables—with no restrictions. Furthermore, applying mathematical induction on the theorem, we have the following more general form:

Corollary 11.5. If $X_1, X_2, X_3, \ldots, X_n$ are random variables and $a_1, a_2, a_3, \ldots, a_n \in \mathbb{R}$ are scalars, then

$$\mathbf{E}[a_1 X_1 + a_2 X_2 + \cdots + a_n X_n] = a_1 \cdot \mathbf{E}[X_1] + a_2 \cdot \mathbf{E}[X_2] + \cdots + a_n \cdot \mathbf{E}[X_n]\,.$$

Applications of Linearity of Expectations

Example. If a fair coin is flipped *three* times and X is the random variable that gives the number of heads, what's the expected value of X? To use linearity of expectations, we will let

$$\begin{aligned}
X_1 &= \#\text{ of heads from the first flip} \\
X_2 &= \#\text{ of heads from the second flip} \\
X_3 &= \#\text{ of heads from the third flip}
\end{aligned}$$

The total number of heads is clearly $X = X_1 + X_2 + X_3$. But we also know

Our direct computation gave $\mathbf{E}[X] = \frac{3}{2}$. Linearity of expectations will make the computation much easier.

that X_1 is either 0 or 1, and

$$X_1 = \begin{cases} 1 & \text{if it turns up heads} \\ 0 & \text{if it turns up tails} \end{cases}$$

Observe that this X_1 is exactly the indicator random variable $X_1 = \mathbb{I}\{\text{the first flip turns up heads}\}$. Its expectation is as follows:

$$\mathbf{E}[X_1] = 1 \cdot \mathbf{Pr}[\text{turns up heads}] + 0 \cdot \mathbf{Pr}[\text{turns up tails}] = 1 \times \frac{1}{2} + 0 \times \frac{1}{2} = \frac{1}{2}.$$

We also know that X_1, X_2, and X_3 behave in the same way, so $\mathbf{E}[X_i] = \frac{1}{2}$ for $i = 1, 2, 3$. By linearity of expectations,

$$\mathbf{E}[X] = \mathbf{E}[X_1 + X_2 + X_3] = \mathbf{E}[X_1] + \mathbf{E}[X_2] + \mathbf{E}[X_3] = \tfrac{1}{2} + \tfrac{1}{2} + \tfrac{1}{2} = \tfrac{3}{2}.$$

As another example, we will now explore a more general setting of n general coins. Suppose we toss n coins, where each coin has a probability p of coming up heads. What is the expected value of the random variable X denoting the total number of heads?

An Identity. Below is an algebraic proof. Can you prove it using counting methods?

$$\begin{aligned} \binom{n}{k} &= \frac{n!}{k!(n-k)!} \\ &= \frac{n}{k} \cdot \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{n}{k}\binom{n-1}{k-1} \end{aligned}$$

Binomial Theorem. For $n \geqslant 1$,

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}.$$

Example. As our first attempt, we will apply the definition of expectation directly. This will rely on our strength to compute the probability and work out the sum:

$$\begin{aligned} \mathbf{E}[X] &= \sum_{k=0}^{n} k \cdot \mathbf{Pr}[X = k] \\ &= \sum_{k=1}^{n} k \cdot p^k (1-p)^{n-k} \binom{n}{k} \\ &= \sum_{k=1}^{n} k \cdot \frac{n}{k}\binom{n-1}{k-1} p^k (1-p)^{n-k} && [\text{ because } \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} \text{ }] \\ &= n \sum_{k=1}^{n} \binom{n-1}{k-1} p^k (1-p)^{n-k} \\ &= n \sum_{j=0}^{n-1} \binom{n-1}{j} p^{j+1} (1-p)^{n-(j+1)} && [\text{ because } k = j+1 \text{ }] \\ &= np \sum_{j=0}^{n-1} \binom{n-1}{j} p^j (1-p)^{(n-1)-j} \\ &= np(p + (1-p))^n && [\text{ Binomial Theorem }] \\ &= np \end{aligned}$$

Tips. The trick is often to express the quantity of interest as a sum of simple random variables.

Our first attempt yielded an answer but was pretty tedious. We will now try again using linearity of expectations.

Example. To use linearity of expectations, we proceed like we did for fair coins. Let $X_i = \mathbb{I}\{i\text{-th coin turns up heads}\}$. That is, it is 1 if the i-th coin turns up heads and 0 otherwise. Clearly, $X = \sum_{i=1}^{n} X_i$. So then, by

linearity of expectations,

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

What is the probability that the i-th coin comes up heads? This is exactly p, so $\mathbf{E}[X_i] = 0 \cdot (1-p) + 1 \cdot p = p$, which means

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} p = np.$$

We now turn to a more involved example. A coin has a probability p of coming up heads. What is the expected value of Y representing the number of flips until we see a head? (The flip that comes up heads counts too.) In probability theory, the random variable Y is said to be distributed according to the well-known geometric random variable with success probability p.

Example. We will directly apply the definition of expectation:

$$\begin{aligned}
\mathbf{E}[Y] &= \sum_{k=1}^{\infty} k \cdot \mathbf{Pr}[Y = k] \\
&= \sum_{k=1}^{\infty} k(1-p)^{k-1}p \\
&= p\sum_{k=1}^{\infty} -\frac{d}{dp}(1-p)^k \\
&= -p \times \frac{d}{dp}\sum_{k=1}^{\infty}(1-p)^k \\
&= -p \times \frac{d}{dp}\left(\frac{1-p}{p}\right) \\
&= \frac{1}{p}
\end{aligned}$$

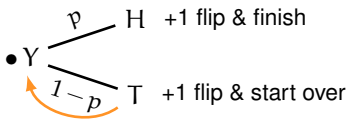
Direct Probability Calculation. It takes exactly k flips if the first $k-1$ flips were tails and the final flip was heads. Hence, the probability that it takes exactly k flips is

$$\mathbf{Pr}[Y = k] = (1-p)^{k-1}p.$$

Furthermore, it takes at least k flips if the first $k-1$ flips were all tails, so $\mathbf{Pr}[Y \geqslant k] = (1-p)^{k-1}$.

Alternatively, we can compute $\mathbf{E}[Y]$ using a recurrence relation. We have to make a leap of faith that $\mathbf{E}[Y]$ is well-defined and is a real number, but the result is a short, simple derivation:

Example. Consider carrying out the first coin toss in this random process. With probability p, the coin will turn up heads and the process is over—and with probability $1-p$, the coin will turn up tails and it is as if the process is starting over. So, if we assume that $\mathbf{E}[Y]$ is a real number,



$$\begin{aligned}
\mathbf{E}[Y] &= p \cdot 1 + (1-p)\Big(1 + \mathbf{E}[Y]\Big) \\
&= 1 + (1-p)\,\mathbf{E}[Y]
\end{aligned}$$

By solving for $\mathbf{E}[Y]$, we have $\mathbf{E}[Y] = \frac{1}{p}$.

Alternative View. If X is a nonnegative integer-valued random variable, then

$$\mathbf{E}[X] = \sum_{k=1}^{\infty} \mathbf{Pr}[X \geqslant k]\,.$$

See Exercise 11.2. for more detail. As a quick example, this means

$$\begin{aligned}
\mathbf{E}[Y] &= \sum_{k=1}^{\infty}(1-p)^{k-1} \\
&= \frac{1}{1-(1-p)} = \frac{1}{p}.
\end{aligned}$$

11.3 Analysis of Randomized Algorithms

Traditional algorithms are most commonly analyzed for running time and space usage with respect to the worst-case scenario, the best-case scenario, and sometimes an average-case scenario (if there is a compelling notion of what an average case means). The study of randomized algorithms—algorithms that make use of randomness—considers practically the same questions. But instead of looking for the worst-case running time, we are most interested in the following variants:

- What is the *expected worst-case* running time (or space requirement) of an algorithm? In this case, the expectation is over the internal randomization of the algorithm and it is with respect to the worst-case scenario (i.e., the most difficult input possible of that size).
- What is the most likely running time (or space requirement)? This is typically presented in terms of a *high-probability* bound. For example, we might claim that with probability nearing 1 (e.g., with probability at least $1 - \frac{1}{n}$), the algorithm runs in time $O(n \log n)$. This helps show that the algorithm almost always runs in the specified time.

In this chapter, we will focus almost exclusively on deriving expected worst-case bounds, leaving high-probability analysis to more advanced lessons. This will require a different mindset and toolset from analyzing traditional algorithms. We will now put to use the tools developed earlier in this chapter.

Example: Three-Card Monte

Three Card Monte, Jaffa, Israel. https://www.flickr.com/photos/ziodave/25510393 CC BY-SA 2.0

As a toy example, we will consider the three-card monte game. A dealer places three cards (face down) on a table. We are asked to find a specific card among the three cards (e.g., the Queen of Hearts). We have to pay $1 for each card that we turn over; the dealer allows us to turn over as many as cards as we want. If we locate the specific card (the Queen), we are paid $2.

What strategy should we use to maximize our gain? If our strategy is deterministic (e.g., it turns over the cards from left to right), the dealer—knowing our strategy ahead of time—can always force the worst out of our strategy. In this particular example strategy, the dealer would place the Queen on the far right and we will have to pay $3 \times \$1$ and earn $2, for a net loss of $1. In other words, we are surely losing money and are better off not playing this game!

Alternatively, we can bring in randomness: Pick a card to turn over at random and repeat until we find the target Queen. It is not difficult to see that with probability $\frac{1}{3}$, we find the Queen the first time around. With probability $\frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$, we will turn over two cards, and with the same $\frac{2}{3} \times \frac{1}{2} \times \frac{1}{1} = \frac{1}{3}$ probability, we will turn over three cards. In all cases, we are paid $2. Hence, our expected net gain is

$$2 - \frac{1}{3}(1+2+3) = 0.$$

Using this strategy, we are at least not losing money, but the reward is not at all an enticing prospect.

Example: Finding The Second Largest Number

We are given an array of n distinct numbers. *How many comparisons do we need to find the second largest number?* Without randomness, the naïve algorithm requires about $2n-3$ comparisons. It is possible to write a divide-and-conquer solution that needs about $3n/2$ comparisons. We will develop a randomized algorithm that requires far fewer comparisons than these in expectation. To begin, we will revisit the naïve algorithm:

In the `secondMax` code, we assume that the input array has length at least 2. Here, `top` is an array of size 2, keeping the maximum and the second maximum, respectively.

```
1  int secondMax(int[] a) {
2      int[] top;
3
4      if (a[0] > a[1]) top = new int[]{a[0], a[1]};
5      else             top = new int[]{a[1], a[2]};
6
7      for (int i=2;i<a.length;i++) {
8          if (a[i] > top[1]) {
9              if (a[i] > top[0]) top = new int[]{a[i], top[0]};
10             else               top = new int[]{top[0], a[i]};
11         }
12     }
13
14     return top[1];
15 }
```

How many comparisons does this naïve algorithm take? We will be meticulous about constants. At the start, we make one comparison. Then, the loop runs for $n-2$ times. Each iteration of the loop makes one comparison for sure—but it can potentially make a second comparision, depending on the outcome of the first comparison. This means in the worst case, each iteration makes two comparisons, leading to a total of $2(n-2)+1 = 2n-3$ comparisons.

Worst-Case Input. We can craft an input array so that this comparison is made in every iteration: if the input is an increasing sequence of n, e.g., 1, 2, 3, . . ., n, then each `a[i]` is the new maximum hence triggering comparisons at both the `if`'s statements. Hence, this input causes the algorithm to expend $2n-3$ comparisons.

It should be noted that we were pessimistic about the number of comparisons in the analysis above. While $n-2+1 = n-1$ comparisons are always made, the comparison at the inner-most if-statement may or may not be made. In a worst-case analysis, we look for the worst-possible scenario and hence always treat each of these as a comparison made.

As turns out, such a nice—but undesirable—input is unlikely. In fact, if the input is a sequence of random numbers, there is only a 1 in $n!$ chance that the input will give the worst-case behavior. But we have no control over the given input. This is where randomness comes in: We can make the input look like a random sequence without changing the answer. For a sequence of n numbers a, pick a random permutation π on n elements (i.e., we choose one of the $n!$ permutations) and permute a according to π.

Shuffling Imagined. There is no need to explicitly shuffle the array. Simply pick a random element that has not been considered and repeat until we have considered the whole sequence.

This transformation only shuffles the input array, but the elements are still the same. It therefore does not affect the answer. We were hoping that this would help avoid worst-case inputs. But how much does it help?

To answer this question, let X be the total number of comparisons made in the algorithm when run on a randomly permuted input array. As we noted

earlier, $n-2+1 = n-1$ comparisons are always made (at the start and at the outer `if` in every iteration). To analyze the number of comparisons due to the inner `if`, let X_i be an indicator variable denoting whether the inner `if` (Line 9) was made in the iteration that `i` $= i$. In other words, it indicates whether `a[i]` was compared to `top[0]` in that iteration. Then,

$$X = n - 1 + \sum_{i=2}^{n-1} X_i$$

This expression is always true, regardless of the random choice made. In this case, all random decisions are made when a (random) permutation is chosen. We seek to derive the expected value of X, representing the expected worst-case bound for the number of comparisons on input of length n. By linearity of expectations, we have

$$\mathbf{E}[X] = n - 1 + \sum_{i=2}^{n-1} \mathbf{E}[X_i]\,. \tag{11.1}$$

This means that we will be able to determine $\mathbf{E}[X]$ if we can compute each $\mathbf{E}[X_i]$ for $i = 2, \ldots, n-1$. Our task therefore boils down to these computations. Remember that each X_i is an indicator random variable. This means that $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$, which amounts to computing the probability for the event

$$\mathcal{E}_i = \texttt{a[i]} \text{ was compared to } \texttt{top[0]}.$$

Calculating this probability is not difficult once we observe the following: $\mathcal{E}_i$ happened if `a[i]` is the top two among `a[0]`, `a[1]`, ..., `a[i]`. To put it differently, it is the probability that the largest or the second largest number is the rightmost number in a randomly-shuffled sequence of length $i+1$.

If $7, 2, 4$ are randomly-shuffled, each position has an equal probability to be 7. The same is true about 2 and 4.

We proceed by noting that each number in the sequence is equally likely to be anywhere in the randomly-permuted sequence. If we focus our attention on the first $i+1$ numbers of the sequence, this means

- the largest number has a $\frac{1}{i+1}$ probability of appearing at the rightmost end; and
- the second largest number has a $\frac{1}{i+1}$ probability of appearing at the rightmost end.

Therefore, the probability that `a[i]` is the largest or the second largest number among `a[0]`, `a[1]`, ..., `a[i]` is $\mathbf{Pr}[\mathcal{E}_i] = \frac{1}{i} + \frac{1}{i} = \frac{2}{i}$. This ultimately means

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = \mathbf{Pr}[\mathcal{E}_i] = \frac{2}{i}.$$

Plugging this back into the expression for $\mathbf{E}[X]$ in equation (11.1) yields

$$\begin{aligned}\mathbf{E}[X] &= n-1+\sum_{i=2}^{n-1}\frac{2}{i+1}\\ &= n-1+2\left(\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}\right)\\ &\leqslant n-1+2H_n,\end{aligned}$$

where H_n is the n-th Harmonic number. But $H_n \leqslant 1+\log_2 n$, so we have $\mathbf{E}[X] \leqslant n-1+2\log_2 n$. Notice that this is a marked improvement from $2n-3$ or even $3n/2$.

Approximating H_n. The Harmonic sum $H_n = 1+\frac{1}{2}+\cdots+\frac{1}{n}$ satisfies:

$$H_n = \ln n+\gamma+\varepsilon_n,$$

where γ is the Euler-Mascheroni constant, which is approximately $0.57721\cdots$, and $\varepsilon_n \sim \frac{1}{2n}$, which tends to 0 as n approaches ∞.

Remarks. In reality, the performance difference between the $2n-3$ algorithm and the $n-1+2\log n$ algorithm is unlikely to matter unless the comparison function is super expensive. For most real-world cases, the $2n-3$ algorithm might in fact be faster due to better cache locality.

The point of this example is to demonstrate the power of randomness in achieving something that otherwise seems impossible. More importantly, the analysis hints at why on a typical "real-world" instance, the $2n-3$ algorithm does much better than what we analyzed in the worst case. Studies have found that real-world instances are usually not adversarial.

11.4 Quickselect

In the previous example, we developed an efficient randomized algorithm for finding the largest two numbers in an array of numbers. Most often, however, we are interested in a more general version of the problem: for a given integer $k \geqslant 1$, find the smallest (or largest) k numbers in an unsorted input array.

Here we will work with the version that finds the smallest k numbers. To solve this problem, we could maintain a priority-queue data structure that takes $O(\log \eta)$ time for a priority queue of size η. Using this method, finding the smallest k numbers would take a total of $O(n \log k)$ time because the size of the priority queue never exceeds $O(k)$. Alternatively, we could first run a good sorting algorithm, which takes $O(n \log n)$ time, and read off the first k numbers. For both methods, when k is large (e.g., $k = \Theta(n)$), we need $O(n \log n)$ time overall.

Via randomization, we will derive a simple algorithm that takes $O(n)$ worst-case expected time, independent of the value of k. For ease of exposition, we will assume that the numbers in the input array are unique.

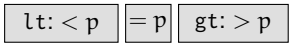
The quickselect algorithm was invented by Tony Hoare, who also invented quicksort. At a high level, the algorithm—just like quicksort—begins by picking a random pivot p from the input array and splitting the input into

(1) `lt` = the input numbers that are less than p;

(2) `eq` = the input numbers that are equal to p; and

(3) `gt` = the input numbers that are greater than p.

Quickselect Algorithm. Quickselect, like quicksort, picks a random pivot p and splits the input array into three pieces—less than the pivot, equal to the pivot, and greater than the pivot.



What should the algorithm do next? Consider the following example. Suppose $k = 5$, the input array is `a = {32, 1, 91, 43, 5, 22, 14, 29}`, and it happens that the randomly-chosen pivot is $p = 43$. Hence, we have

```
lt = { 32, 1, 5, 22, 14, 29 }  // all less than p = 43
eq = { 43 }                    // exactly p = 43
gt = { 91 }                    // all greater than p = 43
```

Case I: $|\texttt{lt}| \geqslant k$. Hence, the whole k will come from `lt`.

By construction, every number in `lt` is smaller than `eq` and `gt`, so `lt` has the smallest numbers. Because we want the smallest $k = 5$ numbers and `lt` has at least k numbers, they will all come from `lt`. We can drill down by asking the algorithm recursively on `lt` for k numbers.

In another scenario, suppose we have the same input array but the randomly-chosen pivot turns out to be $p = 29$. Then, we'll have

```
lt = { 1, 5, 22, 14 }  // all less than p = 29
eq = { 29 }            // exactly p = 29
gt = { 32, 91, 43 }    // all greater than p = 29
```

Case II: $|\texttt{lt}| + |\texttt{gt}| \geqslant k$. The answer will come from the whole of `lt` and $k - |\texttt{lt}|$ from `eq`.

What should the algorithm do next? It turns out `lt` does not have enough (only 4 out of $k = 5$). We need $k - |\texttt{lt}| = 1$ more. It turns out that `eq` has exactly what we needed, so we don't need to recursively solve it in this case.

Finally, suppose we have the same input parameters but the randomly-chosen pivot turns out to be $p = 22$. Then, we'll have

```
lt = { 1, 5, 14 }          // all less than p = 22
eq = { 22 }                // exactly p = 22
gt = { 32, 91, 43, 29 }    // all greater than p = 22
```

Case III: $|\texttt{lt}| + |\texttt{gt}| < k$. The answer will come from `lt` and `eq`, plus the remaining $k - |\texttt{lt}| - |\texttt{eq}|$ from `gt`.

What should the algorithm do next? Together, `lt` and `eq` give the smallest $3 + 1$ numbers. But we want $k = 5$ numbers; we need $k - |\texttt{lt}| - |\texttt{eq}| = 1$ more. The remaining will have to come from `gt`. Conveniently, we can ask the algorithm recursively on `gt` for the remaining numbers.

There is one more scenario to consider: What if the input array has length at most k? Then, the input array itself is the answer. Altogether, we have the following algorithm:

Algorithm 11.1: QUICKSELECT(k, a), return the smallest k numbers in an array a[]

```
if |a| ⩽ k then
    return a
Pick p ∈ a uniformly at random
lt ← ⟨x | x ∈ a and x < p⟩
eq ← ⟨x | x ∈ a and x == p⟩
gt ← ⟨x | x ∈ a and x > p⟩
if |lt| ⩾ k then
    return QUICKSELECT(k, lt)
else if |lt| + |eq| ⩾ k then
    return lt + eq[:k − |lt|]
else
    return lt + eq + QUICKSELECT(k − |lt| − |eq|, gt)
```

Running Time Analysis

Correctness of this algorithm is clear and can be easily established using induction. The next question is, *how fast is this algorithm?* We will begin by attempting to write a running time recurrence:

$$T(n) = \begin{cases} T(|\texttt{lt}|) + O(n) & \text{if the algorithm recurses on } \texttt{lt} \\ T(|\texttt{gt}|) + O(n) & \text{if the algorithm recurses on } \texttt{gt} \\ O(n) & \text{if the algorithm doesn't recurse} \end{cases}$$

where $O(n)$ stems from the running time of partitioning the input array `a`.

Since the algorithm is randomized, we will look to analyze the worst-case expected time, as given by $\mathbf{E}[T(n)]$. Evidently, there are several obvious obstacles: We have no idea how big the parts will be; they are probabilistically determined. Furthermore, we cannot easily write a recurrence that knows whether we are recursing on `lt` or `gt`—or not at all.

If we are willing to make a pessimistic estimate, what is true is that, of the two possible ways to recurse, QUICKSELECT is called with an array of length at most $\max(|\texttt{lt}|, |\texttt{gt}|)$. Hence, by letting $X_n = \max(|\texttt{lt}|, |\texttt{gt}|)$, we can update our recurrence to

$$T(n) \leqslant T(X_n) + O(n),$$

where we note that X_n is a random variable that provides an upper bound on the size of the array the algorithm recurses into.

Tackling this recurrence requires some new techniques. We will start by gathering some intuition—what is the expected value of X_n? A closer look at X_n reveals that $X_n = \max(|\texttt{lt}|, n-1-|\texttt{leq}|)$, assuming still that the numbers are unique. By construction, we know $0 \leqslant |\texttt{lt}| \leqslant n-1$. Further examination shows that

- X_n always ranges between $\lfloor n/2 \rfloor$ and $n-1$ (inclusive).
- There are two ways $X_n = t$ can happen: $|\texttt{lt}| = t$ or $n-1-|\texttt{lt}| = t$.
- It follows that $\mathbf{Pr}[X_n = t] \leqslant \frac{2}{n}$ as every element is equally likely to be chosen as `p`.

We will work out the expectation for when n is even to simply calculations (the odd case can be similarly handled). By the definition of expectation,

$$\begin{aligned} \mathbf{E}[X_n] &\leqslant \sum_{t=n/2}^{n-1} \frac{2}{n} \times t = \frac{2}{n} \times \frac{1}{2}\left(n-1-\frac{n}{2}+1\right)\left(n-1+\frac{n}{2}\right) \\ &\leqslant \frac{1}{n} \times \frac{n}{2} \times \frac{3n}{2} = \frac{3n}{4} \end{aligned}$$

where we have used the summation formula.

We learn from this computation that in expectation, X_n is a constant fraction smaller than n. It gives us hopes that the problem size will be geometrically decreasing, à la $f(n) = f(3n/4) + O(n)$, leading to $f(n) = O(n)$. But, in general, $\mathbf{E}[f(X_n)]$ isn't the same as—or even bounded by—$f(\mathbf{E}[X_n])$, so we have more work to do.

Trivia. Is $\mathbf{E}[\max(X, Y)]$ equal to $\max(\mathbf{E}[X], \mathbf{E}[Y])$? This is, in fact, not true.

Example. For $n = 6$:

$\ell = \lvert\texttt{lt}\rvert$	X_n
0	5 (i.e., $n-1$)
1	4 (i.e., $n-2$)
2	3 (i.e., $n/2$)
3	3 (i.e., $n/2$)
4	4 (i.e., $n-2$)
5	5 (i.e., $n-1$)

To make this idea more concrete, we will try to compute the probability that $X_n \leqslant \frac{3n}{4}$. It helps to start out with an example.

Example. Consider an input array `a = {7, 3, 2, 1, 4, 6, 5, 8}`, which is a permutation of $\{1, 2, \ldots, 8\}$. The consequence of choosing a number p as the pivot is shown below:

p	1	2	3	4	5	6	7	8
$\lvert lt \rvert$	0	1	2	3	4	5	6	7
$n-1-\lvert lt \rvert$	7	6	5	4	3	2	1	0
X_n	7	6	5	4	4	5	6	7

The crucial observation here is that if $(r+1)$-th smallest number is chosen as the pivot, then $X_n = \max(r, n-1-r)$. With this observation, we will show in the following lemma that we are recursing into a problem of size at most $3n/4$ with probability at least $\frac{1}{2}$.

Lemma 11.6. For $n \geqslant 2$,

$$\mathbf{Pr}\left[X_n \leqslant \frac{3n}{4}\right] \geqslant \frac{1}{2}.$$

$\max(i, n-1-i) \leqslant a$ means $i \leqslant a$ and $n-1-i \leqslant a$, which, in turn, means $n-1-a \leqslant i \leqslant a$.

Proof. Let σ_i be the i-th smallest number in the input array. This is defined to aid the proof; we do not need to compute this in the algorithm at all. The pivot is chosen uniformly at random, so for each $0 \leqslant i \leqslant n-1$, the pivot is is σ_{i+1} with probability $\frac{1}{n}$. When σ_{i+1} is chosen, we have $\lvert lt \rvert = i$ and so $X_n = \max(i, n-1-i)$. Let m be the number of settings of i such that $X_n \leqslant \frac{3n}{4}$. Observe that

$$X_n \leqslant \frac{3n}{4} \iff \frac{n}{4} - 1 \leqslant i \leqslant \frac{3n}{4},$$

But i has to be an integer between 0 and $n-1$ (inclusive), so $m = \lfloor 3n/4 \rfloor - \lceil n/4 - 1 \rceil + 1$. This works out to

$$m \geqslant \left(\frac{3n}{4} - 1\right) - \left(\frac{n}{4} - 1 + 1\right) + 1 = \frac{n}{2}.$$

We conclude that $\mathbf{Pr}\left[X_n \leqslant \frac{3n}{4}\right] = \frac{m}{n} \geqslant \frac{n/2}{n} = \frac{1}{2}$. □

We will now put all the pieces together. As we begin, note that given an input array of size n, how the algorithm performs in the future is irrespective of what it did in the past. Also, to simplify notation, let $\overline{T}(n) = \mathbf{E}[T(n)]$ denote the expected time performed on input of size n. Furthermore, we will make a simplifying assumption that $\overline{T}(n)$ is nondecreasing, i.e., for all $x \leqslant y$, $T(x) \leqslant T(y)$. Then, by the definition of expectation, for some constant $c \in \mathbb{R}_+$

$$\overline{T}(n) \leqslant \sum_{\ell} \mathbf{Pr}[X_n = \ell] \cdot \overline{T}(\ell) + c \cdot n.$$

Using Lemma 11.6, together with the assumption that $\overline{T}$ is nondecreasing, we can upper-bound $\overline{T}(n)$ as follows:

- With probability $\frac{1}{2}$, QUICKSELECT is called with problem size at most $3n/4$.
- With the remaining probability (i.e., $1 - 1/2 = 1/2$), QUICKSELECT is called with problem size at most n.

This leads to

$$\overline{T}(n) \leqslant \frac{1}{2} \cdot \overline{T}\left(\frac{3n}{4}\right) + \frac{1}{2} \cdot \overline{T}(n) + c \cdot n$$

which rearranges to

$$\overline{T}(n) \leqslant \overline{T}\left(\frac{3n}{4}\right) + 2c \cdot n$$

by collecting similar terms and multiplying through by 2.

We now unravel this recurrence using the standard technique and by letting the summation, which is geometrically decreasing, continues to infinity, we have

$$\overline{T}(n) \leqslant 2c\left(n + \alpha \cdot n + \alpha^2 \cdot n + \alpha^3 \cdot n + \ldots\right) = \frac{2c}{1-\alpha} \cdot n$$

where $\alpha = \frac{3}{4}$. Hence, we conclude in the following theorem that $\overline{T}(n)$ is $O(n)$.

Theorem 11.7. Let $k \geqslant 1$. On input an array a of n arbitrarily-ordered numbers, the quickselect algorithm takes $O(n)$ worst-case expected time and returns the smallest $\min(k, n)$ numbers.

11.5 Randomized Quicksort

Earlier in Chapter 8, we saw that quicksort is among the best sorting algorithms and the choice of pivots crucially determines its performance. Specifically, deterministic choices of pivots are prone to adversarial inputs and picking an element to be the pivot at random yields good performance. Now that we are armed with tools to analyze randomized algorithms, we are ready to analytically study the performance of randomized quicksort.

To remind ourselves how the algorithm works, below is randomized quicksort, stripped down to a bare minimum:

Algorithm 11.2: QUICKSORT(xs), return the sorted version of the input

```
if |xs| ⩽ 1 then
    return xs
else
    Pick p ∈ xs uniformly at random
    lt ← ⟨x | x ∈ xs and x < p⟩
    eq ← ⟨x | x ∈ xs and x == p⟩
    gt ← ⟨x | x ∈ xs and x > p⟩
    return QUICKSORT (lt) + eq + QUICKSORT (gt)
```

Notice that the bulk of the work done in the algorithm happens on the lines that select elements into `lt`, `eq`, and `gt`. As has been established before, all

of these take $O(n)$ time, where n is the length of the input sequence to that recursive call.

Running Time Analysis

We will begin by writing a recurrence for the running time:

$$T(n) = T(|\texttt{lt}|) + T(|\texttt{gt}|) + O(n),$$

where the $O(n)$ term stems from selecting elements into respective sequences as discussed earlier.

Since the algorithm is randomized, we will look to analyze the worst-case expected time, as given by $\mathbf{E}[T(n)]$. The same reasoning we did for quickselect can be used to show the following:

Lemma 11.8. For $k = 0, \ldots, n-1$, $|\texttt{lt}| = k$ and $|\texttt{gt}| = n-1-k$ with probability $\frac{1}{n}$.

More specifically, with probability $1/n$, the random pivot p happens to be the $(k+1)$-th smallest element of the input sequence to that recursive call and therefore, there are k elements smaller than p (i.e., $|\texttt{lt}| = k$) and there are $n-1-k$ elements larger than p (i.e., $|\texttt{gt}| = n-1-k$).

From the definition of expectation, we have:

$$\begin{aligned}\mathbf{E}[T(n)] &= O(n) + \sum_{k=0}^{n-1} \mathbf{Pr}[\texttt{lt} = k] \cdot \Big(\mathbf{E}[T(k)] + \mathbf{E}[T(n-1-k)]\Big) \\ &= O(n) + \frac{1}{n}\sum_{k=0}^{n-1} \Big(\mathbf{E}[T(k)] + \mathbf{E}[T(n-1-k)]\Big),\end{aligned}$$

We can solve this recurrence relation directly (Exercise 11.7.) or can verify that the solution is indeed $O(n \log n)$ via induction.

which by symmetry, implies the following equation:

$$\mathbf{E}[T(n)] = \frac{2}{n}\sum_{k=0}^{n-1} \mathbf{E}[T(k)] + O(n) \tag{11.2}$$

with $T(0) = T(1) = O(1)$. This solves to $\mathbf{E}[T(n)] = O(n \log n)$. Hence, we have the following theorem:

Theorem 11.9. On input an array of length n, the randomized quicksort algorithm takes $O(n \log n)$ expected worst-case time.

Exercises

Exercise 11.1. If a fair coin is flipped 16 times, what is the probability that the number of heads is the same as the number of tails? What about the probability that there are more heads than tails?

Exercise 11.2. Let X be a nonnegative integer-valued random variable. Prove the following alternative characterization of expectation

$$\mathbf{E}[X] = \sum_{k=1}^{\infty} \mathbf{Pr}[X \geqslant k] .$$

Exercise 11.3. Give an example of two random variables X and Y such that $\mathbf{E}[\max(X, Y)] \neq \max(\mathbf{E}[X], \mathbf{E}[Y])$.

Exercise 11.4. Let $f(x) = x^2$. Is it true that

$$\mathbf{E}[f(X)] = f\left(\mathbf{E}[X]\right)$$

for any random variable X? Prove or disprove the statement. If the statement is *not* true in general, is there an example of a random variable X such that it is true?

Exercise 11.5. Let $p : [n] \to [n]$ be a random permutation on $[n] = \{1, 2, \ldots, n\}$ chosen uniformly among the $n!$ permutations. Consider the following code:

```
int minSoFar = Integer.MAX_VALUE;
int numUpdate = 0;
for (int i=1;i<=n;i++) {
    if (p[i] < minSoFar){
        minSoFar = p[i];
        numUpdate++;
    }
}
```

Prove that at the end of this code,

$$\ln(n+1) \leqslant \mathbf{E}[\texttt{numUpdate}] \leqslant 1 + \ln n.$$

Exercise 11.6. A p-biased coin, $0 \leqslant p \leqslant 1$, turns up heads with probability p and tails with the remaining probability. Write probabilistic recurrences and solve them to answer the following questions in terms of p:

(i) If you keep on tossing this coin until you see a tails followed directly by a tails (TT), what is the expected number of coin tosses, including the final TT?

(ii) If you keep on tossing this coin until you see three consecutive tails (TTT), what is the expected number of coin tosses, including the final TTT?

(iii) If you keep on tossing this coin until you see THT consecutively, what is the expected number of coin tosses, including the final THT?

Exercise 11.7. The recurrence

$$f(n) = n + 1 + \frac{2}{n}\left(f(n-1) + f(n-2) + \cdots + f(1)\right), \qquad \text{where } f(0) = 0.$$

represents the expected running time of randomized quicksort. We will solve this recurrence for a closed form in a few steps:

(i) Consider $f(n)$ and $f(n-1)$, where $n \geqslant 2$. We have

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

By multiplying the first equation by n and the second equation by $(n-1)$, we have

$$n \cdot f(n) = (n+1)n + 2\left(f(n-1) + f(n-2) + \cdots + f(1)\right) \quad (11.3)$$
$$(n-1)f(n-1) = n(n-1) + 2\left(f(n-2) + f(n-3) + \cdots + f(1)\right) \quad (11.4)$$

Subtracting equation (11.4) from equation (11.3), we'll get

$$n \cdot f(n) - (n-1)f(n-1) = 2n + 2f(n-1)$$

In other words,

$$n \cdot f(n) = 2n + (n+1)f(n-1) \quad (11.5)$$

Your task in this step is to understand the derivation we just made. Other than that, no actions are required on your part.

(ii) Let $g(n) = \frac{f(n)}{n+1}$. Can you write what you have in terms of the function g? (*Hint:* divide equation (11.5) by $n(n+1)$.)

(iii) Your task in this step is to find a closed form for g. The recurrence g that we have isn't a familiar one, but you can easily solve for a closed form. To get started, let's look at how to solve a related recurrence: $h(n) = h(n-1) + \frac{1}{n}$, with $h(0) = 0$

We start out by unraveling $h(n)$. By the definition of $h(n)$,

$$\begin{aligned} h(n) &= h(n-1) + \frac{1}{n} && \text{expand it one more time} \\ &= h(n-2) + \frac{1}{n} + \frac{1}{n-1} && \text{expand it one more time} \\ &= h(n-3) + \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} \end{aligned}$$

It is pretty clear that if we keep on expanding the recurrence, we'll get

$$h(n) = \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{3} + \frac{1}{2} + 1.$$

In Math, this has a name: the n-th *Harmonic number*, denoted by H_n, is given by $H_n = \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{3} + \frac{1}{2} + 1$, so the closed form for $h(n)$ is $h(n) = H_n$.

(iv) Now that you can express $g(n)$ in terms of some expression involving Harmonic numbers, you can proceed to derive a closed form for $f(n)$. Finally, use the following fact to conclude that $f(n)$ is $O(n \ln n)$:

> **Fact:** $H_n \leqslant 1 + \ln(n)$, where ln denotes the natural logarithm.

Exercise 11.8. Let X be a nonnegative random variable where $\mathbf{E}[X]$ is well-defined. Prove that for any $t > 0$,

$$\mathbf{Pr}[X \geqslant t] \leqslant \frac{\mathbf{E}[X]}{t}.$$

This is a well-known inequality in probability theory commonly called the Markov's inequality.

Exercise 11.9. Let X be a random variable where both $\mathbf{E}[X]$ and $\mathbf{E}\left[X^2\right]$ are well-defined. Define the variance of X to be $\sigma^2 = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right]$. Prove that for any $t > 0$,

$$\mathbf{Pr}\left[(X - \mathbf{E}[X])^2 \geqslant t^2\right] \leqslant \frac{\sigma^2}{t^2}.$$

This is the well-known Chebyshev's inequality in probability theory. (*Hint:* Use Exercise 11.8.)

Exercise 11.10. Consider the following code:

```
def permute(a: List[int]):
    for i in range(len(a)):
        # draw a number between 0 and n-1 (incl) randomly
        j = randint(0, n)
        a[i], a[j] = a[j], a[i] # swap them
```

Does this function produce a uniformly-random permutation of `a`? Justify your answer.

Chapter Notes

Probabilistic analysis and randomized algorithms are considered modern tools in computer science, compared to the other results in this book. For an excellent introduction to basic discrete probability, check out the book by Lehman et al. [LLM15], who also discuss the topic in further depth than here. The analyses of quickselect and quicksort presented here are mechanical. There are several other more "slick" analyses; see, for example, the book by Motwani and Raghavan [MR95]. For in-depth treatment of the subject, readers are referred to textbooks by Motwani and Raghavan [MR95], and Mitzenmacher and Upfal [MU05].