5 Probabilistic Analysis and Randomized Algorithms

5.1 The hiring problem

5.1-1

> Show that the assumption that we are always able to determine which candidate is best in line 4 of procedure HIRE-ASSISTANT implies that we know a total order on the ranks of the candidates.

A total order is a partial order that is a total relation $(\forall a, b \in A : aRb \text{ or } bRa)$. A relation is a partial order if it is reflexive, antisymmetric and transitive.

Assume that the relation is good or better.

- **Reflexive:** This is a bit trivial, but everybody is as good or better as themselves.
- **Transitive:** If A is better than B and B is better than C, then A is better than C.
- **Antisymmetric:** If A is better than B, then B is not better than A.

So far we have a partial order.

Since we assume we can compare any two candidates, then comparison must be a total relation and thus we have a total order.

5.1-2 ⋆

> Describe an implementation of the procedure $\text{RANDOM}(a, b)$ that only makes calls to $\text{RANDOM}(0, 1)$. What is the expected running time of your procedure, as a function of a and b?

As $(b - a)$ could be any number, we need at least $\lceil \lg(b - a) \rceil$ bits to represent the number. We set $\lceil \lg(b - a) \rceil$ as k. Basically, we need to call $\text{RANDOM}(0, 1)$ k times. If the number represented by binary is bigger than $b - a$, it's not valid number and we give it another try, otherwise we return that number.

```
RANDOM(a, b)
    range = b - a
    bits = ceil(log(2, range))
    result = 0
    for i = 0 to bits - 1
        r = RANDOM(0, 1)
        result = result + r << i
    if result > range
        return RANDOM(a, b)
    else return a + result
```

The expectation of times of calling procedure $\mathrm{RANDOM}(a, b)$ is $\frac{2^k}{b-a}$. $\mathrm{RANDOM}(0, 1)$ will be called k times in that procedure.

The expected running time is $\Theta(\frac{2^k}{b-a} \cdot k)$, k is $\lceil \lg(b-a) \rceil$. Considering 2^k is less than $2 \cdot (b-a)$, so the running time is $O(k)$.

5.1-3 ⋆

> Suppose that you want to output 0 with probability $1/2$ and 1 with probability $1/2$. At your disposal is a procedure $\mathrm{BIASED\text{-}RANDOM}$, that outputs either 0 or 1. It outputs 1 with some probability p and 0 with probability $1-p$, where $0 < p < 1$, but you do not know what p is. Give an algorithm that uses $\mathrm{BIASED\text{-}RANDOM}$ as a subroutine, and returns an unbiased answer, returning 0 with probability $1/2$ and 1 with probability $1/2$. What is the expected running time of your algorithm as a function of p?

There are 4 outcomes when we call $\mathrm{BIASED\text{-}RANDOM}$ twice, i.e., 00, 01, 10, 11.

The strategy is as following:

- 00 or 11: call $\mathrm{BIASED\text{-}RANDOM}$ twice again
- 01: output 0
- 10: output 1

We can calculate the probability of each outcome:

- $\Pr\{00|11\} = p^2 + (1-p)^2$
- $\Pr\{01\} = (1-p)p$
- $\Pr\{10\} = p(1-p)$

Since there's no other way to return a value, it returns 0 and 1 both with probability $1/2$.

The pseudo code is as follow:

```
UNBIASED-RANDOM
    while true
        x = BIASED-RANDOM
        y = BIASED-RANDOM
        if x != y
            return x
```

This algorithm actually uses the equivalence of the probability of occurrence of 01 and 10, and subtly converts the unequal 00 and 11 to 01 and 10, thus eliminating the probability that its probability is not equivalent.

Each iteration is a Bernoulli trial, where "success" means that the iteration does return a value.

We can view each iteration as a Bernoulli trial, where "success" means that the iteration returns a value.

$$\begin{aligned}\Pr\{\text{success}\} &= \Pr\{0 \text{ is returned}\} + \Pr\{1 \text{ is returned}\} \\ &= 2p(1-p).\end{aligned}$$

The expected number of trials for this scenario is $1/(2p(1-p))$. Thus, the expected running time of UNBIASED-RANDOM is $\Theta(1/(2p(1-p))$.

5.2 Indicator random variables

5.2-1

> In HIRE-ASSISTANT, assuming that the candidates are presented in a random order, what is the probability that you hire exactly one time? What is the probability you hire exactly n times?

You will hire exactly one time if the best candidate is at first. There are $(n-1)!$ orderings with the best candidate being at first, so the probability that you hire exactly one time is $\frac{(n-1)!}{n!} = \frac{1}{n}$.

You will hire exactly n times if the candidates are presented in increasing order. There is only an ordering for this situation, so the probability that you hire exactly n times is $\frac{1}{n!}$.

5.2-2

> In HIRE-ASSISTANT, assuming that the candidates are presented in a random order, what is the probability that you hire exactly twice?

Note that

- Candidate 1 is always hired
- The best candidate (candidate whose rank is n) is always hired
- If the best candidate is candidate 1, then that's the only candidate hired.

In order for HIRE-ASSISTANT to hire exactly twice, candidate 1 should have rank i, where $1 \le i \le n - 1$, and all candidates whose ranks are $i + 1, i + 2, \ldots, n - 1$ should be interviewed after the candidate whose rank is n (the best candidate).

Let E_i be the event in which candidate 1 have rank i, so we have $P(E_i) = 1/n$ for $1 \le i \le n$.

Our goal is to find for $1 \le i \le n - 1$, given E_i occurs, i.e., candidate 1 has rank i, the candidate whose rank is n (the best candidate) is the first one interviewed out of the $n - i$ candidates whose ranks are $i + 1, i + 2, \ldots, n$.

So,

$$\sum_{i=1}^{n-1} P(E_i) \cdot \frac{1}{n-i} = \sum_{i=1}^{n-1} \frac{1}{n} \cdot \frac{1}{n-i}.$$

5.2-3

> Use indicator random variables to compute the expected value of the sum of n dice.

Expectation of a single dice X_i is

$$\begin{aligned} E[X_k] &= \sum_{i=1}^{6} i \Pr\{X_k = i\} \\ &= \frac{1+2+3+4+5+6}{6} \\ &= \frac{21}{6} \\ &= 3.5. \end{aligned}$$

As for multiple dices,

$$ \begin{aligned} ✅ & = \text E\Bigg[\sum_{i = 1}^n X_i \Bigg] \\ & = \sum_{i = 1}^n \text E[X_i] \\ & = \sum_{i = 1}^n 3.5 \\ & = 3.5 \cdot n. \end{aligned} $$

5.2-4

> Use indicator random variables to solve the following problem, which is known as the ***hat-check problem***. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their hat?

Let X be the number of customers who get back their own hat and X_i be the indicator random variable that customer i gets his hat back. The probability that an individual gets his hat back is $\frac{1}{n}$. Thus we have

✅ = E\Bigg[\sum_{i = 1}^n X_i\Bigg] = \sum_{i = 1}^n E[X_i] = \sum_{i = 1}^n \frac{1}{n} = 1.$$

5.2-5

> Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A. (See Problem 2-4 for more on inversions.) Suppose that the elements of A form a uniform random permutation of $\langle 1, 2, \ldots, n \rangle$. Use indicator random variables to compute the expected number of inversions.

Let $X_{i,j}$ for $i < j$ be the indicator of $A[i] > A[j]$. We have that the expected number of inversions

$$\begin{aligned}
\mathrm{E}\left[\sum_{i<j} X_{i,j}\right] &= \sum_{i<j} \mathrm{E}[X_{i,j}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{A[i] > A[j]\} \\
&= \frac{1}{2} \sum_{i=1}^{n-1} n - i \\
&= \frac{n(n-1)}{2} - \frac{n(n-1)}{4} \\
&= \frac{n(n-1)}{4}.
\end{aligned}$$

5.3 Randomized algorithms

5.3-1

> Professor Marceau objects to the loop invariant used in the proof of Lemma 5.5. He questions whether it is true prior to the first iteration. He reasons that we could just as easily declare that an empty subarray contains no 0-permutations. Therefore, the probability that an empty subarray contains a 0-permutation should be 0, thus invalidating the loop invariant prior to the first iteration. Rewrite the procedure $\text{RANDOMIZE-IN-PLACE}$ so that its associated loop invariant applies to a nonempty subarray prior to the first iteration, and modify the proof of Lemma 5.5 for your procedure.

Modify the algorithm by unrolling the $i = 1$ case.

```
swap(A[1], A[RANDOM(1, n)])
for i = 2 to n
    swap(A[i], A[RANDOM(i, n)])
```

Modify the proof of Lemma 5.5 by starting with $i = 2$ instead of $i = 1$. This resolves the issue of 0-permutations.

5.3-2

> Professor Kelp decides to write a procedure that produces at random any permutation besides the identity permutation. He proposes the following procedure:
>
> ```
> PERMUTE-WITHOUT-IDENTITY(A)
>     n = A.length
>     for i = 1 to n - 1
>         swap A[i] with A[RANDOM(i + 1, n)]
> ```

> Does this code do what Professor Kelp intends?

The code does not do what he intends. Suppose $A = [1, 2, 3]$. If the algorithm worked as proposed, then with nonzero probability the algorithm should output $[3, 2, 1]$. On the first iteration we swap $A[1]$ with either $A[2]$ or $A[3]$. Since we want $[3, 2, 1]$ and will never again alter $A[1]$, we must necessarily swap with $A[3]$. Now the current array is $[3, 2, 1]$. On the second (and final) iteration, we have no choice but to swap $A[2]$ with $A[3]$, so the resulting array is $[3, 1, 2]$. Thus, the procedure cannot possibly be producing random non-identity permutations.

5.3-3

> Suppose that instead of swapping element $A[i]$ with a random element from the subarray $A[i..n]$, we swapped it with a random element from anywhere in the array:
>
> ```
> PERMUTE-WITH-ALL(A)
>     n = A.length
>     for i = 1 to n
>         swap A[i] with A[RANDOM(1, n)]
> ```
>
> Does this code produce a uniform random permutation? Why or why not?

Consider the case of $n = 3$ in running the algorithm, three IID choices will be made, and so you'll end up having 27 possible end states each with equal probability. There are $3! = 6$ possible orderings, these should appear equally often, but this can't happen because 6 does not divide 27.

5.3-4

> Professor Armstrong suggests the following procedure for generating a uniform random permutation:
>
> ```
> PERMUTE-BY-CYCLIC(A)
>     n = A.length
>     let B[1..n] be a new array
>     offset = RANDOM(1, n)
>     for i = 1 to n
>         dest = i + offset
>         if dest > n
>             dest = dest - n
>         B[dest] = A[i]
>     return B
> ```
>
> Show that each element $A[i]$ has a $1/n$ probability of winding up in any particular position in B. Then show that Professor Armstrong is mistaken by showing that the resulting permutation is not uniformly random.

Fix a position j and an index i. We'll show that the probability that $A[i]$ winds up in position j is $1/n$. The probability $B[j] = A[i]$ is the probability that $dest = j$, which is the probability that $i + offset$ or $i + offset - n$ is equal to j, which is $1/n$.

This algorithm can't possibly return a random permutation because it doesn't change the relative positions of the elements; it merely cyclically permutes the whole permutation.

For instance, suppose $A = [1, 2, 3]$,

- if $offset = 1$, $B = [3, 1, 2]$,
- if $offset = 2$, $B = [2, 3, 1]$,
- if $v = 3$, $B = [1, 2, 3]$.

Thus, the algorithm will never produce $B = [1, 3, 2]$, so the resulting permutation cannot be uniformly random.

5.3-5 ⋆

> Prove that in the array P in procedure $\text{PERMUTE-BY-SORTING}$, the probability that all elements are unique is at least $1 - 1/n$.

Let $\Pr\{j\}$ be the probability that the element with index j is unique. If there are n^3 elements, then the $\Pr\{j\} = 1 - \frac{j-1}{n^3}$.

$$\begin{aligned}
\Pr\{1 \cap 2 \cap 3 \cap \ldots\} & = \Pr\{1\} \cdot \Pr\{2 \mid 1\} \cdot \Pr\{3 \mid 1 \cap 2\} \cdots \\
& = 1(1 - \frac{1}{n^3})(1 - \frac{2}{n^3})(1 - \frac{3}{n^3}) \cdots \\
& \ge 1(1 - \frac{n}{n^3})(1 - \frac{n}{n^3})(1 - \frac{n}{n^3}) \cdots \\
& \ge (1 - \frac{1}{n^2})^n \\
& \ge 1 - \frac{1}{n},
\end{aligned}$$

where the last step holds for $(1 - x)^n \ge 1 - nx$.

5.3-6

> Explain how to implement the algorithm $\text{PERMUTE-BY-SORTING}$ to handle the case in which two or more priorities are identical. That is, your algorithm should produce a uniform random permutation, even if two or more priorities are identical.

```
PERMUTE-BY-SORTING(A)
    let P[1..n] be a new array
    for i = 1 to n
        P[i] = i
    for i = 1 to n
        swap P[i] with P[RANDOM(i, n)]
```

5.3-7

> Suppose we want to create a ***random sample*** of the set $\{1, 2, 3, \ldots, n\}$, that is, an m-element subset S, where $0 \le m \le n$, such that each m-subset is equally likely to be created. One way would be to set $A[i] = i$ for $i = 1, 2, 3, \ldots, n$, call $\text{RANDOMIZE-IN-PLACE}(A)$, and then take just the first m array elements. This method would make n calls to the RANDOM procedure. If n is much larger than m, we can create a random sample with fewer calls to RANDOM. Show that the following recursive procedure returns a random m-subset S of $\{1, 2, 3, \ldots, n\}$, in which each m-subset is equally likely, while making only m calls to RANDOM:

```
RANDOM-SAMPLE(m, n)
    if m == 0
        return Ø
    else S = RANDOM-SAMPLE(m - 1, n - 1)
        i = RANDOM(1, n)
        if i ∈ S
            S = S ∪ {n}
        else S = S ∪ {i}
        return S
```

We prove that it produces a random m subset by induction on m. It is obviously true if $m = 0$ as there is only one size m subset of $[n]$. Suppose S is a uniform $m - 1$ subset of $n - 1$, that is, $\forall j \in [n-1]$, $\Pr[j \in S] = \frac{m-1}{n-1}$.

If we let S' denote the returned set, suppose first $j \in [n-1]$,

$$\begin{aligned}
\Pr[j \in S'] &= \Pr[j \in S] + \Pr[j \notin S \wedge i = j] \\
&= \frac{m-1}{n-1} + \Pr[j \notin S] \Pr[i = j] \\
&= \frac{m-1}{n-1} + \left(1 - \frac{m-1}{n-1}\right)\frac{1}{n} \\
&= \frac{n(m-1) + n - m}{(n-1)n} \\
&= \frac{nm - m}{(n-1)n} = \frac{m}{n}.
\end{aligned}$$

Since the constructed subset contains each of $[n-1]$ with the correct probability, it must also contain n with the correct probability because the probabilities sum to 1.

5.4 Probabilistic analysis and further uses of indicator random variables

5.4-1

> How many people must there be in a room before the probability that someone has the same birthday as you do is at least $1/2$? How many people must there be before the probability that at least two people have a birthday on July 4 is greater than $1/2$?

The probability of a person not having the same birthday as me is $(n-1)/n$. The probability of k people not having the same birthday as me is that, squared. We apply the same approach as the text - we take the complementary event and solve it for k,

$$\begin{aligned} 1 - \left(\frac{n-1}{n}\right)^k &\geq \frac{1}{2} \\ \left(\frac{n-1}{n}\right)^k &\leq \frac{1}{2} \\ k \lg\left(\frac{n-1}{n}\right) &\geq \lg\frac{1}{2} \\ k = \frac{\log(1/2)}{\log(364/365)} &\approx 253. \end{aligned}$$

As for the other question,

$$\begin{aligned} \Pr\{2 \text{ born on Jul } 4\} &= 1 - \Pr\{1 \text{ born on Jul } 4\} - \Pr\{0 \text{ born on Jul } 4\} \\ &= 1 - \frac{k}{n}\left(\frac{n-1}{n}\right)^{k-1} - \left(\frac{n-1}{n}\right)^k \\ &= 1 - \left(\frac{n-1}{n}\right)^{k-1}\left(\frac{n+k-1}{n}\right). \end{aligned}$$

Writing a Ruby programme to find the closest integer, we get 115.

5.4-2

> Suppose that we toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

This is just a restatement of the birthday problem. I consider this all that needs to be said on this subject.

5.4-3 ⋆

> For the analysis of the birthday paradox, is it important that the birthdays be mutually independent, or is pairwise independence sufficient? Justify your answer.

Pairwise independence is enough. It's sufficient for the derivation after (5.6).

5.4-4 ⋆

> How many people should be invited to a party in order to make it likely that there are three people with the same birthday?

The answer is 88. I reached it by trial and error. But let's analyze it with indicator random variables.

Let X_{ijk} be the indicator random variable for the event of the people with indices i, j and k have the same birthday. The probability is $1/n^2$. Then,

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} X_{ijk} \\
&= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} \frac{1}{n^2} \\
&= \binom{n}{3} \frac{1}{n^2} \\
&= \frac{k(k-1)(k-2)}{6n^2}.
\end{aligned}$$

Solving this yields 94. It's a bit more, but again, indicator random variables are approximate.

Finding more commentary online is tricky.

5.4-5 ⋆

> What is the probability that a k-string over a set of size n forms a k-permutation? How does this question relate to the birthday paradox?

$$\begin{aligned}
\Pr\{\text{k-perm in n}\} &= 1 \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdots \frac{n-k+1}{n} \\
&= \frac{n!}{n^k (n-k)!}.
\end{aligned}$$

This is the complementary event to the birthday problem, that is, the chance of k people have distinct birthdays.

5.4-6 ⋆

> Suppose that n balls are tossed into n bins, where each toss is independent and the ball is equally likely to end up in any bin. What is the expected number of empty bins? What is the expected number of bins with exactly one ball?

Let X_i be the indicator variable that bin i is empty after all balls are tossed and X be the random variable that gives the number of empty bins. Thus we have

✅ = \sum_{i = 1}^n E[X_i] = \sum_{i = 1}^n \bigg(\frac{n - 1}{n}\bigg)^n = n\bigg(\frac{n - 1}{n}\bigg)^n.$$

Let X_i be the indicator variable that bin i contains exactly 1 ball after all balls are tossed and X be the random variable that gives the number of bins containing exactly 1 ball. Thus we have

✅ = \sum_{i = 1}^n E[X_i] = \sum_{i = 1}^n \binom{n}{1}\bigg(\frac{n - 1}{n}\bigg)^{n - 1} \frac{1}{n} = n\bigg(\frac{n - 1}{n}\bigg)^{n - 1},$$

because we need to choose which toss will go into bin i, then multiply by the probability that that toss goes into that bin and the remaining $n-1$ tosses avoid it.

5.4-7 ⋆

> Sharpen the lower bound on streak length by showing that in n flips of a fair coin, the probability is less than $1/n$ that no streak longer than $\lg n - 2\lg\lg n$ consecutive heads occurs.

We split up the n flips into n/s groups where we pick $s = \lg(n) - 2\lg(\lg(n))$. We will show that at least one of these groups comes up all heads with probability at least $\frac{n-1}{n}$. So, the probability the group starting in position i comes up all heads is

$$\Pr(A_{i,\lg n-2\lg(\lg n)}) = \frac{1}{2^{\lg n-2\lg(\lg n)}} = \frac{\lg n^2}{n}.$$

Since the groups are based of of disjoint sets of IID coin flips, these probabilities are independent. so,

$$\begin{aligned}\Pr(\bigwedge \neg A_{i,\lg n-2\lg(\lg n)}) &= \prod_i \Pr(\neg A_{i,\lg n-2\lg(\lg n)}) \\ &= \left(1-\frac{\lg n^2}{n}\right)^{\frac{n}{\lg n-2\lg(\lg n)}} \\ &\le e^{-\frac{\lg n^2}{\lg n-2\lg(\lg n)}} \\ &= \frac{1}{n}e^{\frac{-2\lg(\lg n)\lg n}{\lg n-2\lg(\lg n)}} \\ &= n^{-1-\frac{2\lg(\lg n)}{\lg n-2\lg(\lg n)}} \\ &< n^{-1}.\end{aligned}$$

Showing that the probability that there is no run of length at least $\lg n - 2\lg(\lg n)$ to be $< \frac{1}{n}$.

Problem 5-1 Probabilstic counting

> With a b-bit counter, we can ordinarily only count up to $2^b - 1$. With R. Morris's ***probabilistic counting***, we can count up to a much larger value at the expense of some loss of precision.
>
> We let a counter value of i represent that a count of n_i for $i = 0, 1, \ldots, 2^b - 1$, where the n_i form an increasing sequence of nonnegative values. We assume that the initial value of the counter is 0, representing a count of $n_0 = 0$. The INCREMENT operation works on a counter containing the value i in a probabilistic manner. If $i = 2^b - 1$, then the operation reports an overflow error. Otherwise, the INCREMENT operation increases the counter by 1 with probability $1/(n_{i+1} - n_i)$, and it leaves the counter unchanged with probability $1 - 1/(n_{i+1} - n_i)$.
>
> If we select $n_i = i$ for all $i \ge 0$, then the counter is an ordinary one. More interesting situations arise if we select, say, $n_i = 2^{i-1}$ for $i > 0$ or $n_i = F_i$ (the ith Fibonacci number—see Section 3.2).

> For this problem, assume that n_{2^b-1} is large enough that the probability of an overflow error is negligible.
>
> **a.** Show that the expected value represented by the counter after n INCREMENT operations have been performed is exactly n.
>
> **b.** The analysis of the variance of the count represented by the counter depends on the sequence of the n_i. Let us consider a simple case: $n_i = 100i$ for all $i \geq 0$. Estimate the variance in the value represented by the register after n INCREMENT operations have been performed.

a. To show that the expected value represented by the counter after n INCREMENT operations have been performed is exactly n, we can show that each expected increment represented by the counter is 1.

Assume the initial value of the counter is i, increasing the number represented from n_i to n_{i+1} is with a probability of $\frac{1}{n_{i+1}-n_i}$ and leaving the value not changed otherwise.

The expected increase:

$$\frac{n_{i+1} - n_i}{n_{i+1} - n_i} = 1.$$

b. For this choice of n_i, we have that at each increment operation, the probability that we change the value of the counter is $\frac{1}{100}$. Since this is a constant with respect to the current value of the counter i, we can view the final result as a binomial distribution with a p value of 0.01. Since the variance of a binomial distribution is $np(1-p)$, and we have that each success is worth 100 instead, the variance is going to be equal to $0.99n$.

Problem 5-2 Searching an unsorted array

> The problem examines three algorithms for searching for a value x in an unsorted array A consisting for n elements.
>
> Consider the following randomized strategy: pick a random index i into A. If $A[i] = x$, then we terminate; otherwise, we continue the search by picking a new random index into A. We continue picking random indices into A until we find an index j such that $A[j] = x$ or until we have checked every element of A. Note that we pick from the whole set of indices each time, so that we may examine a given element more than once.
>
> **a.** Write pseudocode for a procedure RANDOM-SEARCH to implement the strategy above. Be sure that your algorithm terminates when all indices into A have been picked.
>
> **b.** Suppose that there is exactly one index i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and RANDOM-SEARCH terminates?
>
> **c.** Generalizing your solution to part (b), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and RANDOM-SEARCH terminates? Your answer should be a function of n and k.

> **d.** Suppose that there are no indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we have checked all elements of A and RANDOM-SEARCH terminates?
>
> Now consider a deterministic linear search algorithm, which we refer to as $\text{DETERMINISTIC-SEARCH}$. Specifically, the algorithm searches A for x in order, considering $A[1], A[2], A[3], \ldots, A[n]$ until either it finds $A[i] = x$ or it reaches the end of the array. Assume that possible permutations of the input array are equally likely.
>
> **e.** Suppose that there is exactly one index i such that $A[i] = x$. What is the average-case running time of $\text{DETERMINISTIC-SEARCH}$? What is the worst-case running time of $\text{DETERMINISTIC-SEARCH}$?
>
> **f.** Generalizing your solution to part (e), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the average-case running time of $\text{DETERMINISTIC-SEARCH}$? What is the worst-case running time of $\text{DETERMINISTIC-SEARCH}$? Your answer should be a function of n and k.
>
> **g.** Suppose that there are no indices i such that $A[i] = x$. What is the average-case running time of $\text{DETERMINISTIC-SEARCH}$? What is the worst-case running time of $\text{DETERMINISTIC-SEARCH}$?
>
> Finally, consider a randomized algorithm SCRAMBLE-SEARCH that works by first randomly permuting the input array and then running the deterministic linear search given above on the resulting permuting array.
>
> **h.** Letting k be the number of indices i such that $A[i] = x$, give the worst-case and expected running time of SCRAMBLE-SEARCH for the cases in which $k = 0$ and $k = 1$. Generalizing your solution to handle the case in which $k \geq 1$.
>
> **i.** Which of the three searching algorithms would you use? Explain your answer.

a.

```
RANDOM-SEARCH(x, A, n)
    v = Ø
    while |Ø| != n
        i = RANDOM(1, n)
        if A[i] = x
            return i
        else
            v = v ∩ i
    return NIL
```

v can be implemented in multiple ways: a hash table, a tree or a bitmap. The last one would probabily perform best and consume the least space.

b. RANDOM-SEARCH is well-modelled by Bernoulli trials. The expected number of picks is n.

c. In similar fashion, the expected number of picks is n/k.

d. This is modelled by the balls and bins problem, explored in section 5.4.2. The answer is $n(\ln n + O(1))$.

e. The worst-case running time is n. The average-case is $(n+1)/2$ (obviously).

f. The worst-case running time is $n - k + 1$. The average-case running time is $(n+1)/(k+1)$. Let X_i be an indicator random variable that the ith element is a match. $\Pr\{X_i\} = 1/(k+1)$. Let Y be an indicator random variable that we have found a match after the first $n - k + 1$ elements ($\Pr\{Y\} = 1$). Thus,

$$\begin{aligned} E[X] &= E[X_1 + X_2 + \ldots + X_{n-k} + Y] \\ &= 1 + \sum_{i=1}^{n-k} E[X_i] = 1 + \frac{n-k}{k+1} \\ &= \frac{n+1}{k+1}. \end{aligned}$$

g. Both the worst-case and average case is n.

h. It's the same as $\text{DETERMINISTIC-SEARCH}$, only we replace "average-case" with "expected".

i. Definitelly $\text{DETERMINISTIC-SEARCH}$. SCRAMBLE-SEARCH gives better expected results, but for the cost of randomly permuting the array, which is a linear operation. In the same time we could have scanned the full array and reported a result.