Chapter 5

Michelle Bodnar, Andrew Lohr

January 20, 2017

Exercise 5.1-1

We may of been presented the candidates in increasing order of goodness. This would mean that we can apply transitivity to determine our preference between any two candidates

Exercise 5.1-2

Algorithm 1 RANDOM(a,b)

```
1: n = ⌈lg(b − a + 1)⌉
2: Initialize an array A of length n
3: while true do
4:     for i = 1 to n do
5:         A[i] = RANDOM(0, 1)
6:     end for
7:     if A holds the binary representation of one of the numbers in a through
   b then
8:         return number represented by A
9:     end if
10: end while
```

Each iteration of the while loop takes n time to run. The probability that the while loop stops on a given iteration is $(b-a+1)/2^n$. Thus the expected running time is the expected number of times run times n. This is given by:

$$n\sum_{i\geq 1} i\left(1-\frac{b-a+1}{2^n}\right)^{i-1}\left(\frac{b-a+1}{2^n}\right) = n\left(\frac{b-a+1}{2^n}\right)\left(\frac{2^n}{b-a+1}\right)^2 = n\frac{2^n}{b-a+1} = O(\lg(b-a)).$$

Exercise 5.1-3

Clearly since a and b are IID, the probability this algorithm returns one is equal to the probability it returns 0. Also, since there is a constant positive probability $(2p(p-1))$ that the algorithm returns on each iteration of the for

```
1: for all eternity do
2:     a = BiasedRandom
3:     b = BiasedRandom
4:     if a > b then
5:         return 1
6:     end if
7:     if a < b then
8:         return 0
9:     end if
10: end for
```

loop. This program will expect to go through the loop a number of times equal to:

$$\sum_{j=0}^{\infty} j(1-2p(p-1))^j(2p(p-1)) = \frac{2p(p-1)(1-2p(p-1))}{(2p(p-1))^2} = \frac{1-2p(p-1)}{2p(p-1)}$$

Note that the formula used for the sum of $j\alpha^j$ can be obtained by differentiawting both sides of the geometric sum formula for α^j with respect to α

Exercise 5.2-1

You will hire exactly one time if the best candidate is presented first. There are $(n-1)!$ orderings with the best candidate first, so, it is with probability $\frac{(n-1)!}{n!} = \frac{1}{n}$ that you only hire once. You will hire exactly n times if the candidates are presented in increasing order. This fixes the ordering to a single one, and so this will occur with probability $\frac{1}{n!}$.

Exercise 5.2-2

Since the first candidate is always hired, we need to compute the probability that that exactly one additional candidate is hired. Since we view the candidate ranking as reading an random permutation, this is equivalent to the probability that a random permutation is a decreasing sequence followed by an increase, followed by another decreasing sequence. Such a permutation can be thought of as a partition of $[n]$ into 2 parts. One of size k and the other of size $n-k$, where $1 \leq k \leq n-1$. For each such partition, we obtain a permutation with a single increase by ordering the numbers each partition in decreasing order, then concatenating these sequences. The only thing that can go wrong is if the numbers n through $n-k+1$ are in the first partition. Thus there are $\binom{n}{k} - 1$ permutations which correspond to hiring the second and final person on step $k+1$. Summing, we see that the probability you hire exactly twice is

$$\frac{\sum_{k=1}^{n-1}\binom{n}{k} - 1}{n!} = \frac{2^n - 2 - (n-1)}{n!} = \frac{2^n - n - 1}{n!}.$$

Exercise 5.2-3

Let X_j be the indicator of a dice coming up j. So, the expected value of a single dice roll X is

$$E[X] = \sum_{j=1}^{6} j \Pr(X_j) = \frac{1}{6} \sum_{j=1}^{6} j$$

So, the sum of n dice has probability

$$E[nX] = nE[X] = \frac{n}{6} \sum_{j=1}^{6} j = \frac{n6(6+1)}{12} = 3.5n$$

Exercise 5.2-5

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

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

Exercise 5.2-4

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 $\frac{1}{n}$. Then we have

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

Exercise 5.3-1

We modify the algorithm by unrolling the $i = 1$ case.

```
1: swap A[1] with A[Random(1,n)]
2: for i from 2 to n do
3:     swap A[i] with A[Random(i,n)]
4: end for
```

Modify the proof of the lemma by starting with $i = 2$ instead of $i = 1$. This entirely sidesteps the issue of talking about 0- permutations.

Exercise 5.3-2

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.

Exercise 5.3-3

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 shuld appear equally often, but this can't happen because 6 does not divide 27

Exercise 5.3-4

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 $i + \mathit{offset}$ or $i + \mathit{offset} - n$ 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 $\mathit{offset} = 1$, $B = [3, 1, 2]$. If $\mathit{offset} = 2$, $B = [2, 3, 1]$. 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.

Exercise 5.3-5

Let $X_{i,j}$ be the event that $P[i] = P[j]$. Then, the event that all are unique is the complement of there being some pair that are equal, so, we must show that $\Pr(\cup_{i,j} X_{i,j}) \leq 1/n$. We start by applying a union bound

$$\Pr(\cup_{i<k} X_{i,j}) \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr(X_{i,j}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{n^3}$$

Where we use the fact that any two indices will be equal with probability equal to one over the size of the probability space being drawn from, which is n^3.

$$= \sum_{i=1}^{n-1} \frac{n-i}{n^3} = \frac{n(n-1)}{n^3} - \frac{n(n-1)}{2n^3} = \frac{n-1}{2n^2} < \frac{1}{n}$$

Exercise 5.3-6

After the completion of the for loop, check whether or not two or more priorities are identical. Suppose that k of them are identical. Then call $RANDOM(1, k^3)$ k times to introduce a relative ordering on the k identical entries. Some entries m may still be identical, so simply call $RANDOM(1, m^3)$ that many times to introduce a relative ordering on these identical entries. For example, if the first sequence of numbers we get is $1, 1, 1, 3, 4, 5, 5$, we'll need to order the 1's and the 5's. For the 1's, we'll call $RANDOM(1, 27)$ 3 times. Suppose it produces $23, 14, 23$. Then we'll call $RANDOM(1, 8)$ twice to sort the 23's. Suppose it gives $3, 5$. Then the relative ordering of $23, 14, 23$ becomes $2, 1, 3$, so the new relative ordering on the original array is $2, 1, 3, 4, 5, 6, 6$. Now we call $RANDOM(1, 8)$ twice to sort the 6's. If the first number is larger than the second, our final array will be $2, 1, 3, 4, 5, 7, 6$.

Exercise 5.3-7

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, $\forall j \in [n-1], \Pr[j \in S] = \frac{m-1}{n-1}$. Then, if we let S' denote the returned set, suppose first $j \in [n-1]$, $\Pr[j \in S'] = \Pr[j \in S] + \Pr[j \notin S \wedge i = j] = \frac{m-1}{n-1} + \Pr[j \notin S] \Pr[i = j] = \frac{m-1}{n-1} + \left(1 - \frac{m-1}{n-1}\right) \frac{1}{n} = \frac{n(m-1)+n-m}{(n-1)n} = \frac{nm-m}{(n-1)n} = \frac{m}{n}$. 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.

Exercise 5.4-1

The probability that none of n people have the same birthday as you is $(1 - \frac{1}{365})^n = \frac{364^n}{365^n}$. This falls below zero when $n \geq \log_{\frac{364}{365}}(.5) \approx 252.6$ so, when $n = 253$. Since you are also a person in the room, we add one to get the final answer of 254.

The probability that k of the n people have july 4 as a birthday is $\binom{n}{k} \frac{364^{n-k}}{365}$. In particular, for $k = 0$ it is $\frac{364^n}{365^n}$ and for $k = 1$ it is $\frac{n364^{n-1}}{365^n}$. Adding these up and solving for n to have the sum drop less than a half, we get

$$\left(\frac{364}{365}\right)^n \left(1 + \frac{n}{364}\right) < .5$$

This is difficult to solve analytically, but because of the LHS's monotonicity, the answer can be found rather quickly by using a gallop search to be $n = 612$.

Exercise 5.4-2

We compute directly from the definition of expectation. Let X denote the

number of balls tossed until some bin contains two balls.

$$E[X] = \sum_{i=2}^{b+1} iP(X = i).$$

The probability that i tosses are required is the probability that the first $i-1$ tosses go into unique bins, so we have

$$P(X = i) = 1 \cdot \left(\frac{b-1}{b}\right)\left(\frac{b-2}{b}\right)\cdots\left(\frac{b-i+2}{b}\right)\left(\frac{i-1}{b}\right).$$

Thus

$$\begin{aligned} E[X] &= \sum_{i=2}^{b+1} i \left(\frac{b-1}{b}\right)\left(\frac{b-2}{b}\right)\cdots\left(\frac{b-i+2}{b}\right)\left(\frac{i-1}{b}\right) \\ &= \sum_{i=2}^{b+1} i \left(1-\frac{1}{b}\right)\left(1-\frac{2}{b}\right)\cdots\left(1-\frac{i-2}{b}\right)\left(\frac{i-1}{b}\right). \end{aligned}$$

Exercise 5.4-3

Pairwise independence is sufficient. All the independence is used for is to show that $\Pr(b_i = r \wedge b_j = r) = \Pr(b_i = r)\Pr(b_j = r)$. This is a result of pairwise independence.

Exercise 5.4-4

We can compute "likely" in two ways. The probability that at least three people share the same birthday is 1 minus the probability that none share the same birthday, minus the probability that any number of pairs of people share the same birthday. Let $n = 365$ denote the number of days in a year and k be the number of people at the party. As computed earlier in the section, we have

$$P(\text{all unique birthdays}) = 1 \cdot \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right)\cdots\left(\frac{n-k+1}{n}\right) \leq e^{-k(k-1)/2n}.$$

Next we compute the probability that exactly i pairs of people share a birthday. There are $\binom{k}{2}\binom{k-2}{2}\cdots\binom{k-2i+2}{2}$ ways to choose an ordered collection of i pairs from the k people, $\binom{n}{i}$ ways to select the set of birthdays the pairs will share, and the probability that any such ordered subset has these precise birthdays is $(1/n^2)^i$. Multiplying by the probability that the rest of the birthdays

are different and unique we have

$$P(\text{exactly } i \text{ pairs of people share birthdays}) = \frac{\binom{k}{2}\binom{k-2}{2}\cdots\binom{k-2i+2}{2}\binom{n}{i}}{n^{2i}}\left(\frac{n-i}{n}\right)\left(\frac{n-i-1}{n}\right)\cdots\left(\frac{n-k+i+}{n}\right.$$
$$\leq \frac{k!\binom{n}{i}}{2^i(k-2i)!n^{2i}}\left(1-\frac{i}{n}\right)\left(1-\frac{i+1}{n}\right)\cdots\left(1-\frac{k-i-1}{n}\right)$$
$$\leq \frac{k!\binom{n}{i}}{2^i(k-2i)!n^{2i}}e^{-(k-1)(k-2i-1)/2n}$$

Thus,

$$P(\geq 3 \text{ people share a birthday}) \leq 1-e^{-k(k-1)/2n}-\frac{k!\binom{n}{i}}{2^i(k-2i)!n^{2i}}e^{-(k-1)(k-2i-1)/2n}.$$

This is pretty messy, even with the simplifying inequality $1 + x \leq e^x$, so we'll do another analysis, this time on expectation. We'll determine the value of k required such that the expected number of triples (i, j, m) where person i, person j, and person m share a birthday is at least 1. Let X_{ijm} be the indicator variable that this triple of people share a birthday and X denote the total number of triples of birthday-sharers. Then we have

$$E[X] = \sum_{\text{distinct triples } (i,j,m)} E[X_{ijm}] = \binom{k}{3}\frac{1}{n^2}.$$

To make $E[X]$ exceed 1 we need to find the smallest k such that $k(k-1)(k-2) \geq 6(365)^2$, which happens when $k = 94$.

Exercise 5.4-5

Since to be a k-permutation, we need that no letter appears repeated, it is equivalent to the birthday problem with k people and n days. So, this probability is given on the top of page 132 to be:

$$\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)$$

Problem 5.4-6

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. Then we have

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n}\left(\frac{n-1}{n}\right)^n = n\left(\frac{n-1}{n}\right)^n.$$

Now 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. Then we have

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \binom{n}{1} \left(\frac{n-1}{n}\right)^{n-1} \frac{1}{n} = n \left(\frac{n-1}{n}\right)^{n-1}$$

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.

Exercise 5.4-7

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,

$$\Pr(\bigwedge_i \neg A_{i,\lg(n)-2\lg(\lg(n))}) = \prod_i \Pr(\neg A_{i,\lg(n)-2\lg(\lg(n))})$$

$$= \left(1 - \frac{\lg(n)^2}{n}\right)^{\frac{n}{\lg(n)-2\lg(\lg(n))}} \leq e^{-\frac{\lg(n)^2}{\lg(n)-2\lg(\lg(n))}} = \frac{1}{n} e^{\frac{-2\lg(\lg(n))\lg(n)}{\lg(n)-2\lg(\lg(n))}}$$

$$= n^{-1-\frac{2\lg(\lg(n))}{\lg(n)-2\lg(\lg(n))}} < n^{-1}$$

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

a. We will show that the expected increase from each increment operation is equal to one. Suppose that the value of the counter is currently i. Then, we will increase the number represented from n_i to n_{i+1} with a probability of $\frac{1}{n_{i+1}-n_i}$, leaving the value alone otherwise. Multiplying these together, we get that the expected increase is $\frac{n_{i+1}-n_i}{n_{i+1}-n_i} - 1$.

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 .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 $.99n$.

Problem 5-2

a. Assume that A has n elements. Our algorithm will use an array P to track the elements which have been seen, and add to a counter c each time a new element is checked. Once this counter reaches n, we will know that every element has been checked. Let $RI(A)$ be the function that returns a random index of A.

Algorithm 2 RANDOM-SEARCH

```
Initialize an array P of size n containing all zeros
Initialize integers c and i to 0
while c ≠ n do
    i = RI(A)
    if A[i] == x then
        return i
    end if
    if P[i] == 0 then
        P[i] = 1
        c = c + 1
    end if
end while
return A does not contain x
```

b. Let N be the random variable for the number of searches required. Then

$$\begin{aligned} E[N] &= \sum_{i \geq 1} iP(i \text{ iterations are required}) \\ &= \sum_{i \geq 1} i \left(\frac{n-1}{n}\right)^{i-1} \left(\frac{1}{n}\right) \\ &= \frac{1}{n} \frac{1}{\left(1 - \frac{n-1}{n}\right)^2} \\ &= n. \end{aligned}$$

c. Let N be the random variable for the number of searches required. Then

$$\begin{aligned} E[N] &= \sum_{i \geq 1} iP(i \text{ iterations are required}) \\ &= \sum_{i \geq 1} i \left(\frac{n-k}{n}\right)^{i-1} \left(\frac{k}{n}\right) \\ &= \frac{k}{n} \frac{1}{\left(1 - \frac{n-k}{n}\right)^2} \\ &= \frac{n}{k}. \end{aligned}$$

d. This is identical to the "How many balls must we toss until every bin contains at least one ball?" problem solved in section 5.4.2, whose solution is $b(\ln b + O(1))$.

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

f. Let X be the random variable which gives the number of elements examined before the algorithm terminates. Let X_i be the indicator variable that the i^{th} element of the array is examined. If i is an index such that $A[i] \neq x$ then $P(X_i) = \frac{1}{k+1}$ since we examine it only if it occurs before every one of the k indices that contains x. If i is an index such that $A[i] = x$ then $P(X_i) = \frac{1}{k}$ since only one of the indices corresponding to a solution will be examined. Let $S = \{i|A[i] = x\}$ and $S' = \{i|A[i] \neq x\}$. Then we have

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i \in S} P(X_i) + \sum_{i \in S'} P(X_i) = \frac{k}{k} + \frac{n-k}{k+1} = \frac{n+1}{k+1}.$$

Thus the average case running time is $\frac{n+1}{k+1}$. The worst case happens when every occurrence of x is in the last k positions of the array. This has running time $n - k + 1$.

g. The average and worst case running times are both n.

h. SCRAMBLE-SEARCH works identically to DETERMINISTIC-SEARCH, except that we add to the running time the time it takes to randomize the input array.

i. I would use DETERMINISTIC-SEARCH since it has the best expected runtime and is guaranteed to terminate after n steps, unlike RANDOM-SEARCH. Moreover, in the time it takes SCRAMBLE-SEARCH to randomly permute the input array we could have performed a linear search anyway.