1. Let $T$ be a treap with $n \geq 2$ vertices.

(a) What is the *exact* expected number of leaves in $T$?

**Solution:** Node $k$ is a leaf if and only if its priority is smaller than the priorities of its neighbors in rank order: node $k-1$ (unless $k=1$) and node $k+1$ (unless $k=n$). Thus,

$$\Pr[\text{node } k \text{ has no children}] = \begin{cases} 1 & \text{if } k = n = 1 \\ 1/2 & \text{if } n > 1 \text{ and } (k = 1 \text{ or } k = n) \\ 1/3 & \text{otherwise} \end{cases}$$

Linearity of expectation now implies that

$$\begin{aligned} &\mathrm{E}[\text{number of nodes with no children}] \\ &= \sum_{k=1}^{n} \Pr[\text{node } k \text{ has no children}] \\ &= \boxed{\frac{n+1}{3}} \end{aligned}$$

■

**Rubric:** 3 points = 1½ for answer + 1½ for proof. Max 1 point for $\Theta(n)$ solution with wrong (or missing) constant factor. Max 2 points for $n/3 \pm \Theta(1)$ solution with wrong constant term. No penalty for ignoring the special case $n = 1$.

(b) What is the *exact* expected number of nodes in $T$ that have two children?

**Solution:** Node $k$ has two children if and only if $1 < k < n$ and the priority of node $k$ is larger than the priorities of both nodes $k-1$ and $k+1$. Thus,

$$\Pr[\text{node } k \text{ has two children}] = \begin{cases} 0 & \text{if } k = 1 \text{ or } k = n \\ 1/3 & \text{otherwise} \end{cases}$$

Linearity of expectation now implies that

$$\begin{aligned} &\mathrm{E}[\text{number of nodes with two children}] \\ &= \sum_{k=1}^{n} \Pr[\text{node } k \text{ has two children}] \\ &= \boxed{\frac{n-2}{3}} \end{aligned}$$

■

**Rubric:** 3 points; same partial credit as part (a).

(c) What is the *exact* expected number of nodes in $T$ that have exactly one child?

**Solution:** By linearity of expectation, the expected number of nodes with one child is equal to $n$ (the total number of nodes) minus the expected number of nodes with zero or two children:

$$n - \frac{n+1}{3} - \frac{n-2}{3} = \boxed{\frac{n+1}{3}}$$

■

**Solution:** Node $k$ has one child if and only if $n > 1$ and the priority of node $k$ is smaller than exactly one of nodes $k-1$ and $k+1$. Thus,

$$\Pr[\text{node } k \text{ has one child}] = \begin{cases} 0 & \text{if } n = k = 1 \\ 1/2 & \text{if } n > 1 \text{ and } (k = 1 \text{ or } k = n) \\ 1/3 & \text{otherwise} \end{cases}$$

Linearity of expectation now implies

$$\begin{aligned} &\mathrm{E}[\text{number of nodes with one child}] \\ &= \sum_{k=1}^{n} \Pr[\text{node } k \text{ has one child}] \\ &= \boxed{\frac{n+1}{3}} \end{aligned}$$

■

**Rubric:** 4 points = 3 points for correct exact answer (same partial credit as part (a)) + 1 point for exact answer equal to $n$ — part (a) — part (b). No proof is required.

2. (a) Describe and analyze an efficient algorithm that either assigns each ghost a distinct house that they can haunt, or correctly reports that such an assignment is impossible.

**Solution:** We build a bipartite graph $G = (L \sqcup R, E)$, where $L$ is the set of $n$ ghosts, $R$ is the set of $m$ houses, and $ij \in E$ if and only if ghost $i$ can haunt house $j$. Then we compute a maximum matching in the graph, as described in class, in $O(VE) = \boldsymbol{O(n^3)}$ ***time***. Finally, if this matching has $n$ edges, we return TRUE; otherwise, we return FALSE. ■

**Rubric:** 5 points = 2 for setting up the graph + 2 for maximum matching + 1 for running time as a function of $m$ and $n$.

(b) Suppose every ghost is assigned to a distinct house that they cannot haunt. Describe and analyze an efficient algorithm to compute an exchange that results in a valid assignment of ghosts to houses.

**Solution (alternating cycle):** We set up the same bipartite graph $G$ as in part (a). Beetlejuice's assignment corresponds to a set of edges that are *missing* from $G$; call this set of non-edges $B$. Compute a maximum matching $M$ in $G$, in $O(VE) = O(n^3)$ time, as described in class.

Now consider the graph $M \cup B$. Every vertex in $G'$ is incident to exactly one edge in $M$ and exactly one edge in $B$. Thus, every vertex of $M \cup B$ has degree 2, which implies that $M \cup B$ consists entirely of disjoint simple cycles, which alternate between $M$ and $B$.

Direct the vertices in $B$ from ghost to house, and direct the vertices in $M$ from house to ghost. Finally, for each ghost $i$, let $GiveTo[i]$ be the ghost vertex reached from vertex $i$ by following two directed edges (first in $B$ and second in $M$).

The overall algorithm runs in $\boldsymbol{O(n^3)}$ ***time***. ■

**Rubric:** 5 points = 2 for building $M \cup B$ + 2 for pointer-jumping + 1 for running time as a function of $n$. +1 extra credit if the algorithm is still correct when $m \neq n$.

3. Suppose we hash a set of $n$ items into a table of size $m = 2n$, using a hash function $h$ chosen uniformly at random from some universal family. Assume $\sqrt{n}$ is an integer.

(a) ***Prove*** that the expected number of collisions is at most $n/4$.

**Solution:** Without loss of generality, assume the items to be hashed are the integers $1, 2, \ldots, n$, and let $X$ denote the total number of collisions.

$$\begin{aligned} \mathrm{E}[X] &= \sum_{1 \le i < j \le n} \Pr[h(i) = h(j)] && \text{[linearity of expectation]} \\ &\le \sum_{1 \le i < j \le n} \frac{1}{2n} && \text{[definition of “universal”]} \\ &= \frac{n(n-1)}{2}\frac{1}{2n} = \frac{n-1}{4} < \frac{n}{4} && \text{[algebra]} \end{aligned}$$

■

**Rubric:** 3 points

(b) ***Prove*** that the probability that there are at least $n/2$ collisions is at most $1/2$.

**Solution:** Part (a) implies $n/2 > 2\,\mathrm{E}[X]$ and thus $\Pr[X > n/2] \le \Pr[X > 2\,\mathrm{E}[X]]$, and Markov's inequality gives us $\Pr[X > 2\,\mathrm{E}[X]] \le 1/2$. ■

**Rubric:** 2 points

(c) ***Prove*** that the probability that any subset of more than $\sqrt{n}$ items all hash to the same address is at most $1/2$. *[Hint: Use part (b).]*

**Solution:** Every pair of items that hashes to the same address collides. Thus, if any subset of $k$ items hash to the same address, there are at least $\binom{k}{2} = k(k-1)/2$ collisions. In particular, if any subset of $\sqrt{n}+1$ items hash to the same address, the number of collisions is at least

$$\binom{\sqrt{n}+1}{2} = \frac{\sqrt{n}(\sqrt{n}+1)}{2} > \frac{n}{2}.$$

The claim now follows immediately from part (b). ■

**Rubric:** 2 points

(d) Now suppose we choose $h$ at random from a ***4-uniform*** family of hash functions, which means for all distinct items $w, x, y, z$ and all addresses $i, j, k, l$, we have

$$\Pr_{h \in \mathcal{H}}\big[h(w) = i \ \wedge\ h(x) = j \ \wedge\ h(y) = k \ \wedge\ h(z) = \ell\big] = \frac{1}{m^4}.$$

***Prove*** that the probability that any subset of more than $\sqrt{n}$ items all hash to the same address is at most $O(1/n)$.

**Solution:** 4-uniformity implies that $\mathrm{E}[X]$ is actually *equal to* $(n-1)/4$, not just *at most* $(n-1)/4$ as in part (a). Because the hash function is 4-uniform, the collision events $[h(i) = h(j)]$ are *pairwise* independent. (Every pair of collisions involves at most four different items!) Thus, Chebyshev's inequality implies

$$\Pr[X > 2\,\mathrm{E}[X]] < \frac{1}{\mathrm{E}[X]} = \frac{4}{n-1} = O(1/n).$$

■

**Rubric:** 3 points

4. (a) Describe an algorithm to simulate one roll of a fair 20-sided die using independent rolls of a fair 6-sided die *and no other source of randomness*.

   (b) What is the *exact* expected number of 6-sided-die rolls executed by your algorithm?

   (c) Derive an upper bound on the probability that your algorithm requires more than $N$ rolls. Express your answer as a function of $N$.

   (d) Estimate the smallest number $N$ such that the probability that your algorithm requires more than $N$ rolls is less than $\delta$. Express your answer as a function of $\delta$.

**Solution (simple rejection sampling):** Generate an integer $z$ uniformly between 1 and 36 using two die rolls. If $z \le 20$, then return $z$; otherwise, start over.

```
RollD20():
  x ← Random(6)
  y ← Random(6)
  z ← 6(x − 1) + y
  ⟨⟨z is uniform in 1..36⟩⟩
  if z ≤ 20
      return z
  else
      RollD20()
```

Each trial uses two die rolls and succeeds with probability $20/36 = 5/9$. So the expected number of die rolls satisfies the equation

$$X = 2 + \frac{4}{9}X \implies X = \boxed{\frac{18}{5} = 3.6}.$$

Let $P_N$ denote the probability that we need more than $N$ rolls; we immediately have

$$P_N = \boxed{\left(\frac{4}{9}\right)^{\lfloor N/2 \rfloor}} = \begin{cases} (\frac{2}{3})^N & \text{if } N \text{ is even} \\ (\frac{2}{3})^{N-1} & \text{if } N \text{ is odd} \end{cases}$$

So to succeed in at most $N$ rolls with probability at least $1 - \delta$, it suffices to set

$$\delta = \left(\frac{2}{3}\right)^N \implies N = \log_{2/3} \delta = \boxed{\log_{3/2} \frac{1}{\delta}} \approx 2.4663 \ln(1/\delta) = \Theta\left(\log \frac{1}{\delta}\right)$$

■

**Rubric:** 10 points = 4 for part (a) + 2 for part (b) + 2 for part (c) + 2 for part (d). +1 extra credit for an algorithm where the answer to part (b) is less than 3. Answers to (b), (c), and (d) must match the algorithm in part (a) to receive full credit. Only the expressions in boxes are required for full credit.

**Solution (factored rejection sampling):** Generate an integer $x$ uniformly between 1 and 4 and another integer $y$ uniformly between 1 and 5.

```
RollD20():
  ⟨⟨x ← Random(4)⟩⟩
  repeat
      x ← Random(6)
  until x ≤ 4
  ⟨⟨y ← Random(5)⟩⟩
  repeat
      y ← Random(6)
  until y ≤ 5
  ⟨⟨Combine x and y⟩⟩
  return 5(x − 1) + y
```

Let $X$ be the expected number of rolls in the first loop, and let $Y$ be the expected number of rolls in the second loop. We immediately have

$$X = 1 + \frac{X}{3} \implies X = \frac{3}{2} \qquad\qquad Y = 1 + \frac{Y}{6} \implies Y = \frac{6}{5}$$

So the total expected number of rolls is $3/2 + 6/5 = \boxed{27/10 = 2.7}$.

Let $P_N$ denote the probability that we need more than $N$ rolls. If we need more than $N$ rolls, then we need more than $N/2$ rolls in at least one of the two loops. It follows that

$$\boxed{P_N \le \frac{1}{3^{N/2}} + \frac{1}{6^{N/2}}}$$

So to succeed in at most $N$ rolls with probability at least $1 - \delta$, it suffices to set

$$\delta = \frac{1}{3^{N/2}} \implies N = \boxed{2\log_3\left(\frac{1}{\delta}\right)} \approx 1.8204 \ln(1/\delta) = \Theta\left(\log \frac{1}{\delta}\right)$$

■

**Solution (Optimal! better factored rejection sampling):** We can improve the previous solution by using a better algorithm to simulate a 4-sided die:

```
RollD20():
  ⟨⟨x ← Random(4)⟩⟩
  p ← Random(6)
  if p ≤ 4
      x ← p
  else
      x ← 1 + 2·[p = 6] + [Random(6) > 3]
  ⟨⟨y ← Random(5)⟩⟩
  repeat
      y ← Random(6)
  until y ≤ 5
  ⟨⟨Combine x and y⟩⟩
  return 5(x − 1) + y
```

Let $X$ be the expected number of rolls to generate $x$, and let $Y$ be the expected number of rolls to generate $y$. We immediately have

$$X = 1 + \frac{1}{3} = \frac{4}{3} \qquad\qquad Y = 1 + \frac{Y}{6} \implies Y = \frac{6}{5}$$

So the total expected number of rolls is $4/3 + 6/5 = \boxed{38/15 \approx 2.53333}$. **This expected number of rolls is actually optimal!**

Let $P_N$ denote the probability that we need more than $N$ rolls. For any $N > 1$, We need more than $N$ rolls if and only if one of the following disjoint events occurs:

- We generate $x$ with only one roll, and we need more than $N-1$ rolls to generate $y$.
- We generate $x$ with two rolls, and we need more than $N-2$ rolls to generate $y$.

It follows that $P_0 = P_1 = 1$

$$P_N = \frac{2}{3} \cdot \frac{1}{6^{N-1}} + \frac{1}{3} \cdot \frac{1}{6^{N-2}} = \boxed{\frac{16}{6^N}}$$

for all $N \geq 2$. **This algorithm is in fact optimal!** For any $N > 1$, exactly $6^N \bmod 20 = 16$ outcomes from the first $N$ rolls do not produce output, which is the least possible.

So to succeed in at most $N$ rolls with probability at least $1 - \delta$, it suffices to set

$$\delta = \frac{16}{6^N} \implies N = \boxed{\log_6\left(\frac{16}{\delta}\right)} \approx 0.5581 \ln N + 1.5474 = \Theta\left(\log \frac{1}{\delta}\right)$$

■

**Solution (Optimal! $4 \cdot 20 = 5 \cdot 16 = 80$):**

```
RollD20():
  z ← 6 · (Random(6) − 1) + Random(6)    ⟨⟨z is uniform in 1..36⟩⟩
  if z ≤ 20
      return z
  r ← z − 20                              ⟨⟨r is uniform in 1..16⟩⟩
  repeat forever:
      x ← Random(6)
      if x ≤ 5
          y ← 5(r − 1) + x                ⟨⟨y is uniform in 1..80⟩⟩
          return (y mod 20) + 1
```

Let $X$ be the expected total number of rolls executed by this algorithm, and let $Y$ be the expected number of rolls in the main repeat-forever loop. We immediately have

$$Y = 1 + \frac{Y}{6} \implies Y = \frac{6}{5}.$$

The main loop is executed with probability $16/36 = 4/9$, so

$$X = 2 + \frac{4}{9}Y = \boxed{\frac{38}{15} \approx 2.53333}.$$

**This expected number of rolls is actually optimal!**

Let $P_N$ denote the probability that we need more than $N$ rolls; we immediately have

$$\boxed{P_N = \begin{cases} 1 & \text{if } N = 1 \\ \frac{4}{9}\left(\frac{1}{6}\right)^{N-2} & \text{otherwise} \end{cases}}$$

Alternatively, we can observe that for any $N > 1$, there are $6^N$ possible outcomes for the first $N$ rolls, and exactly 16 of those outcomes do not produce output: the 16 possible rolls of the first two dice that don't immediately terminate, followed by $N-2$ 6s. Thus,

$$\boxed{P_N = \begin{cases} 1 & \text{if } N = 1 \\ \frac{16}{6^N} & \text{otherwise} \end{cases}}$$

**This algorithm is in fact optimal!** For any $N > 1$, exactly $6^N \bmod 20 = 16$ outcomes from the first $N$ rolls do not produce output, which is the least possible.

So to succeed in at most $N$ rolls with probability at least $1 - \delta$, it suffices to set

$$\delta = \frac{16}{6^N} \implies N = \boxed{\log_6\left(\frac{16}{\delta}\right)} \approx 0.5581 \ln N + 1.5474 = \Theta\left(\log \frac{1}{\delta}\right)$$

■