o. For each of the following conditions, compute the *exact* expected number of fair coin flips until that condition is met.

(a) Hamlet flips heads.

Solution: Let X_a denote the number of flips up to and including the first heads. If the first flip is heads, Hamlet only flips once; otherwise, Hamlet starts over after the first flip. Said differently: After the first flip, Hamlet starts over with probability $1/2$. Thus,

$$\mathrm{E}[X_a] = 1 + \frac{1}{2}\mathrm{E}[X_a]$$

We conclude that $\mathbf{E[X_a] = 2}$. ■

(b) Hamlet flips both heads and tails (in different flips, of course).

Solution: Let X_b denote the number of flips for this experiment. Without loss of generality, suppose the first flip is tails. Then the experiment ends after the first heads. Thus, linearity of expectation implies $\mathbf{E[X_b] = 1 + E[X_a] = 3}$. ■

(c) Hamlet flips heads twice.

Solution: Let X_c denote the number of flips for this experiment. Linearity of expectation immediately implies $\mathbf{E[X_c] = E[X_a] + E[X_a] = 4}$. ■

(d) Hamlet flips heads twice in a row.

Solution: Let X_d denote the number of flips to get two heads in a row, and let Y_d be the number of remaining flips to get two heads in a row if we just flipped heads. Then

$$\mathrm{E}[X_d] = 1 + \frac{1}{2}\mathrm{E}[X_d] + \frac{1}{2}\mathrm{E}[Y_d]$$
$$\mathrm{E}[Y_d] = 1 + \frac{1}{2}\mathrm{E}[X_d]$$

Solving these equations gives us $\mathbf{E[X_d] = 6}$ and $\mathrm{E}[Y_d] = 4$. ■

(e) Hamlet flips heads followed immediately by tails.

Solution: Let X_e denote the number of flips for this experiment. The problem is unchanged if we remove the word "immediately"; the first tails after the first heads occurs immediately after some heads. So linearity of expectation immediately implies $\mathbf{E[X_e] = E[X_a] + E[X_a] = 4}$. ■

(f) Hamlet flips more heads than tails.

Solution (Trust the Recursion Fairy): Let X_f denote the number of flips for this experiment. If the first flip is heads, the experiment ends immediately. Otherwise, after the first tails, Hamlet must perform a series of flips with one more heads than tails, and then perform another series of flips with one more

heads than tails. Thus, linearity of expectation implies

$$\mathrm{E}[X_f] = 1 + \frac{1}{2} \cdot 2\,\mathrm{E}[X_f].$$

This equation has no solution, which implies that $\mathbf{E}[X_f] = \infty$.[a] ■

[a] Here I'm relying on a subtle observation that every non-negative random variable has a well-defined expectation, which is either a non-negative real number or ∞. Random variables that can be both positive and negative may have **no well-defined expectation**! Consider a game where you flip a fair coin until it comes up heads, and your reward for flipping n tails in a row is $(-2)^n$ dollars and your head a splode. I'm also implicitly relying on the fact that X_f is finite (and therefore actually an integer) with probability 1; see part (i).

Solution (Do not trust the Recursion Fairy): Let X_f denote the number of flips for this experiment. We can compute the expectation directly from the definition $\mathrm{E}[X_f] = \sum_{x \geq 0} x \cdot \Pr[X_f = x]$.

- The first time the number of heads exceeds the number of tails, the total number of flips must be odd. Thus, $\mathrm{E}[X_f] = \sum_{n \geq 0} (2n+1) \cdot \Pr[X_f = 2n+1]$.
- $X_f = 2n+1$ if and only if the first $2n+1$ flips have more heads than tails, but no prefix has that property. Equivalently, the first $2n$ flips is isomorphic to a balanced string of parentheses, where tails are open parens and heads are closed parens, and the $(2n+1)$th flip is a heads.
- Thus, the number of possible flip sequences of length $2n+1$ that would imply $X_f = 2n+1$ is equal to the number of balanced strings of length $2n$, which is the nth Catalan number $\binom{2n}{n}\frac{1}{n+1}$.
- The binomial theorem implies $4^n = \sum_{k=0}^{2n} \binom{2n}{k}$. The middle binomial coefficient $\binom{2n}{n}$ is the largest term in this summation. It follows immediately that $\binom{2n}{n} \geq \frac{4^n}{2n+1}$. (Stirling's approximation implies tighter bounds, but this one is good enough.)
- Each flip sequence of length $2n+1$ has probability $1/2^{2n+1}$.

We conclude that

$$\begin{aligned}
\mathrm{E}[X_f] &= \sum_{n \geq 0} (2n+1) \cdot \Pr[X_f = 2n+1] \\
&= \sum_{n \geq 0} (2n+1) \cdot \binom{2n}{n} \frac{1}{n+1} \cdot \frac{1}{2^{2n+1}} \\
&\geq \sum_{n \geq 0} (2n+1) \cdot \frac{4^n}{2n+1} \cdot \frac{1}{n+1} \cdot \frac{1}{2^{2n+1}} \\
&= \sum_{n \geq 0} \frac{1}{2n+2} \\
&= \frac{1}{2} \sum_{k \geq 1} \frac{1}{k}
\end{aligned}$$

The final sum diverges, which implies $\mathrm{E}[X_f] = \infty$.

Yeesh. Maybe I should have trusted the Recursion Fairy. ■

(g) Hamlet flips the same number of heads and tails.

Solution: **Zero**, because $0 = 0$. ■

(h) Hamlet flips the same positive number of heads and tails.

Solution: Let X_h denote the number of flips for this experiment. Without loss of generality, suppose the first flip is tails. Then the remaining flips must have more heads than tails. Thus, linearity implies $\mathrm{E}[X_h] = 1 + \mathrm{E}[E_f] = \infty$. ■

(i) Hamlet flips more than twice as many heads as tails.

Solution: Let X_i denote the number of flips for this experiment (or ∞ if this experiment never ends). If Hamlet ever flips more than twice as many heads as tails, he must have already flipped more heads than tails. Thus, $X_i \geq X_f$, which implies $E[X_i] \geq E[X_f]$, and therefore $E[X_i] = \infty$.

Despite the superficial similarity, there is a significant difference between this experiment and the experiment in part (f).

- The *probability* that Hamlet *never* flips more heads than tails turns out to be exactly 0; said differently, the experiment ends after a finite number of flips *with probability* 1. This doesn't mean it's *impossible* for the experiment to run forever; it only means that the probability is (literally) vanishingly small. In fact, our infinite-series analysis of $\mathrm{E}[X_f]$ already implicitly assumed that X_f is finite with probability 1; recall that $\sum_{n\geq 0} f(n)$ is formal shorthand for $\lim_{N\to\infty} \sum_{n=0}^{N} f(n)$. So $\mathrm{E}[X_i]$ is a weighted average of finite values, but it's still infinite, because the infinite series does not converge.
- On the other hand, the probability that Hamlet *never* flips more than twice as many heads as tails turns out to be exactly $1/\phi^2 = 2 - \phi \approx 0.38197$, where $\phi = (1+\sqrt{5})/2$ is the golden ratio. (See this Mathematics StackExchange post for details.) This observation *immediately* implies that the expected value of X_i is infinite. On the other hand, X_i is no longer formally an integer random variable! ■

Rubric: Not graded!

1. Consider the following non-standard algorithm for shuffling a deck of n cards, initially numbered in order from 1 on the top to n on the bottom. At each step, we remove the top card from the deck and ***insert*** it randomly back into in the deck, choosing one of the n possible positions uniformly at random. The algorithm ends immediately after we pick up card $n-1$ and insert it randomly into the deck.

   (a) Prove that this algorithm uniformly shuffles the deck, meaning each permutation of the deck has equal probability.

Solution: First we need to prove a simple claim:

Claim 1. *For any positive integer n, inserting a new card uniformly at random into a uniformly shuffled deck of $n-1$ cards yields a uniformly shuffled deck of n cards.*

Proof: A deck of $n-1$ cards has $(n-1)!$ possible permutations, and a new card can be inserted into n possible locations. Each starting permutation and insertion location yield a different permutation of the n cards. Because the deck is uniformly shuffled, each starting permutation has probability $1/(n-1)!$. Because the nth card is inserted uniformly at random, each insertion location has probability $1/n$. Thus, each final permutation has probability $1/(n-1)! \cdot 1/n = 1/n!$. □

For purposes of analysis, we break the execution of the algorithm into $n-1$ *phases* as follows. Each phase except the last ends when a card is inserted under card $n-1$; the last phase consists of inserting card $n-1$ randomly into the deck.

Claim 2. *For all integers $0 \le k \le n-2$, the $k+1$ cards underneath card $n-1$ immediately after the kth phase are uniformly shuffled. (Here "immediately after the 0th phase" means "at the start of the algorithm".)*

Proof: Fix an arbitrary integer $0 \le k \le n-2$, and assume for all non-negative integers $j < k$ that the $j+1$ cards underneath card $n-1$ immediately after jth phase are uniformly shuffled. There are two cases to consider.

- When the algorithm begins, the single card under card $n-1$ (namely, card n) is in one of 1! permutations, each with probability $1/1!$.
- Suppose $k \ge 0$. The induction hypothesis implies that when the kth phase begins, the k cards underneath card $n-1$ are uniformly shuffled. The kth phase ends when some card (it doesn't matter which one) is inserted under card $n-1$. Because every insertion is uniformly distributed, each of the k possible insertion locations under card $n-1$ is equally likely. The result now follows from Claim 1.

In both cases the claim is proved. □

When the last phase begins, card $n-1$ is on top of the deck and by Claim 2, the other $n-1$ cards are uniformly shuffled. Claim 1 implies that randomly reinserting card $n-1$ yields a uniformly shuffled deck of n cards, as required. ■

Rubric: 5 points = 2 for proof of Claim 1 + 3 for other details. This is more detail than necessary for full credit, but a proof of Claim 1 is necessary.

(b) What is the *exact* expected number of steps executed by the algorithm? *[Hint: Split the algorithm into phases that end when card $n-1$ changes position.]*

Solution: As suggested by the hint, we break the execution of the algorithm into $n-1$ *phases* as in part (a). Let T_i denote the number of steps in the ith phase, and let $T = \sum_{i=1}^{n-1} T_i$ denote the total number of steps. We immediately have $T_{n-1} = 1$, but the other T_i's are random variables.

Consider any integer $1 \le i < n-1$. In any step of the ith phase, the top card is inserted uniformly at random into one of n locations, and exactly $i+1$ of these locations are under card $n-1$. Thus, we have

$$\mathrm{E}[T_i] = 1 + \frac{n-i-1}{n}\mathrm{E}[T_i],$$

which implies $\mathrm{E}[T_i] = n/(i+1)$. The last phase consists of moving a single card, so $T_{n-1} = 1$. We conclude that $\mathrm{E}[T_i] = n/(i+1)$ for all $1 \le i \le n-1$.

Linearity of expectation now implies that

$$\mathrm{E}[T] = \sum_{i=1}^{n-1}\mathrm{E}[T_i] = \sum_{i=1}^{n-1}\frac{n}{i+1} = n\cdot\sum_{j=2}^{n}\frac{1}{j} = \boldsymbol{nH_n - n}.$$

■

Rubric: 5 points for the exact answer; 3 points for $\Theta(n \log n)$. The expression "$nH_{n-1}+n+1$" is also correct and worth full credit. This is not the only correct derivation.

2. Suppose we are given a two-dimensional array $M[1..n, 1..n]$ in which every row and every column is sorted in increasing order and no two elements are equal.

   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M larger than $M[i, j]$ and smaller than $M[i', j']$.

Solution: We describe and analyze an algorithm NUMBERBETWEEN(M, x, y) that returns the number of elements M that are larger than x and smaller than y, for arbitrary numbers $x < y$. Our algorithm also computes two arrays $LessEqX[1..n]$ and $LessY[1..n]$ that will be useful in our later algorithms:

- $LessEqX[i]$ is the number of elements less than or equal to x in the ith row.
- $LessY[i]$ is the number of elements strictly smaller than y in the ith row.

Because the rows of M are sorted, the elements less or equal to x in any row of M define a *prefix* of that row. Because the columns of M are sorted, these prefixes either stay the same length or get shorter from one row to the next. Intuitively, all elements less than or equal to x lie above a "staircase" in the upper left corner of M. Similarly, elements greater than or equal to y define a *suffix* of each row, and these suffixes define a "staircase" in the lower right corner of M. We count the elements between these two staircases by walking along both staircases row by row.

```
NUMBERBETWEEN(M, x, y):
  jx ← n
  jy ← n
  count ← 0
  for i ← 1 to n
      while jx ≥ 1 and M[i, jx] > x
          jx ← jx − 1
      LessEqX[i] ← jx
      while jy ≥ 1 and M[i, jy] ≥ y
          jy ← jy − 1
      LessY[i] ← jy
      count ← count + LessY[i] − LessEqX[i]
  return count
```

The lines $jx \leftarrow jx - 1$ and $jy \leftarrow jy - 1$ are each executed at most n times; we conclude that NUMLESS runs in $O(n)$ time. ■

Rubric: 3 points = 1 for "staircase" intuition + 1 for other details + 1 for time analysis. This is not the only way to formulate this algorithm. Computing the arrays *LessY* and *LessEqX* is not required for full credit (but if you don't compute them here, you'll need to compute them in part (b)).

(b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i, j]$ and larger than $M[i', j']$. Assume the requested range is always non-empty.

Solution: We actually describe an algorithm to choose a random element of M between arbitrary numbers x and y. After running NUMBERBETWEEN(M, x, y), we choose a random number r between 1 and the number of relevant elements, and then scan for the row containing the rth relevant element using the auxiliary arrays *LessEqX* and *LessY*.

```
RANDOMBETWEEN(M, x, y):
  count ← NUMBERBETWEEN(M, x, y)
  ⟨⟨Also computes arrays LessEqX and LessY⟩⟩
  r ← RANDOM(count)
  i ← 1
  while r > LessY[i] − LessEqX[i]
      r ← r − (LessY[i] − LessEqX[i])
      i ← i + 1
  return M[i, r + LessEqX[i]]
```

The algorithm clearly runs in $O(n)$ time. ■

Rubric: 3 points = 2 for setting up *LessY* and *LessEqX* (or equivalent) + 1 for other details. This is not the only way to formulate this algorithm.

Solution (rejection sampling + extraction): We actually describe an algorithm to choose a random element of M between arbitrary numbers x and y. As in the previous algorithm, we start by computing $K =$ NUMBERBETWEEN(M, x, y)

If $K \geq n$, we use *rejection sampling*: We repeatedly draw elements at random from the *entire* array M, until we happen to draw an element between x and y.

```
RANDOMBETWEEN(M, x, y):
  repeat
      i ← RANDOM(n)
      j ← RANDOM(n)
  until x ≤ M[i, j] ≤ y
  return M[i, j]
```

Each iteration of this algorithm succeeds with probability K/n^2, so the expected number of iterations is $n^2/K \leq n$. Thus, this algorithm runs in $O(n)$ *expected* time.

On the other hand, if $K < n$, we extract the K relevant elements of M into a smaller array $R[1..K]$, using the following variant of our solution to part (a), and then return $R[$RANDOM$(K)]$. This algorithm also runs in $O(n)$ time.

```
CopyBetween(M, x, y):
  jx ← n
  jy ← n
  R ← empty list    ⟨⟨list of relevant entries⟩⟩
  for i ← 1 to n
      while jx ≥ 1 and M[i, jx] > x
          jx ← jx − 1
      while jy ≥ 1 and M[i, jy] ≥ y
          jy ← jy − 1
      append M[i, jx + 1 .. jy] to R
  return R
```

■

Rubric: 3 points = 1½ for rejection sampling when $K \geq n$ (including analysis) + 1½ for extraction when $K < n$. These are not the only correct solutions.

(c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n \log n)$ expected time.

Solution: The algorithm is a variant of binary search:

```
MEDIAN(M):
  lo ← −∞
  hi ← ∞

  while lo < hi
      pivot ← RANDOMBETWEEN(M, lo, hi)
      rank ← NUMBETWEEN(M, −∞, pivot)

      if rank = n²/2
          return pivot
      else if rank < n²/2
          lo ← pivot
      else if rank > n²/2
          hi ← pivot
```

Every iteration the loop requires $O(n)$ time, so to compute the expected running time, we only need to compute the expected number of iterations. For each index i, let $X_i = 1$ if MEDIAN uses the ith smallest element of M as a pivot; the number of iterations is exactly $X = \sum_i X_i$.

We can compute $\Pr[X_i = 1]$ by reformulating the algorithm as follows: Initialize $lo \leftarrow -\infty$ and $hi \leftarrow \infty$, and consider the elements $m \in M$ in random order. If $lo < m < hi$, then we use m as a pivot (and possibly adjust hi or lo). With this formulation, we observe that $X_i = 1$ if and only if the ith smallest element is the first element considered with rank between i and $n^2/2$. Because each of those elements is equally likely to be considered first, we have

$$\Pr[X_i = 1] = \frac{1}{|n^2/2 - i| + 1}$$

and therefore

$$\mathrm{E}[X] = \sum_{i=1}^{n^2} \frac{1}{|n^2/2 - i| + 1} < \sum_{j=1}^{n^2/2} \frac{2}{j} = 2H_{n^2/2} < 4 \ln n + 2 = O(\log n).$$

We conclude that the expected running time is $O(n \log n)$, as required. ■

Solution: We use the same algorithm from the previous solution. Again, every iteration the loop requires $O(n)$ time, so to compute the expected running time, we only need to compute the expected number of iterations.

Let $L(m)$ denote the expected number of remaining iterations when $hi - lo = m$. Following the crude analysis of randomized quicksort (or nuts and bolts), call an iteration of the loop *good* if $m/4 \le rank < 3m/4$, and *bad* otherwise; each iteration is good with probability $1/2$. If the trial is good, the rest of the algorithm requires at most $L(3m/4)$ iterations, and if the trial is bad, the rest of the

algorithm crudely requires at most $L(m)$ iterations. Thus, we have

$$L(m) \le 1 + \frac{1}{2}L\left(\frac{3m}{4}\right) + \frac{1}{2}L(m)$$

which implies $L(m) \le 2 + L(3m/4)$ and therefore $L(m) = O(\log m)$.

We conclude that the expected time to find the median of M is $O(n) \cdot L(n^2) = O(n \log n)$, as required. ■

Solution: We use the same algorithm from the previous solution. Again, every iteration the loop requires $O(n)$ time, so to prove that the expected running time is $O(n \log n)$, we only need to show that the expected number of iterations is $O(\log n)$.

The expected number of iterations is exactly the expected depth of node $n^2/2$ in a treap with n^2 leaves, which is $2H_{n^2/2} - 2 < 2\ln(n^2/2) = 4\ln n - 2\ln 2 = O(\log n)$, as required. ■

Rubric: 4 points = 1 for binary search (or one-armed quicksort) + 1 for details + 2 for analysis. These are not the only ways to either formulate or analyze the algorithm. This is not the only correct algorithm.

(The algorithm for part (c) can be used as a subroutine to solve HW2.2 in $O(n \log^2 n)$ expected time!)

Solution: I might add this later if enough people are interested. ■

Rubric: Not graded!

3. Death knocks on your door one cold blustery morning and challenges you to a game.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

Solution: The following algorithm determines whether we can win Death's game, starting at any arbitrary node v *with even depth*.

```
DeathGame(v):
  if v is a leaf
      return [v is white]
  ll ← DeathGame(v.left.left)
  lr ← DeathGame(v.left.right)
  rl ← DeathGame(v.right.left)
  rr ← DeathGame(v.right.right)
  return (ll ∧ lr) ∨ (rl ∧ rr)
```

The algorithm spends $O(1)$ time at every node with even depth and therefore runs in $\boldsymbol{O(4^n)}$ ***time***. ■

Solution: The problem becomes become slightly easier if we regard every node in the tree as a NAND gate; if the inputs to a NAND gate are x and y, the output of that gate is $\overline{x \wedge y} = \overline{x} \vee \overline{y}$. De Morgan's law inductively implies that a tree of NAND gates is logically equivalent to a tree of alternating AND and OR gates, as long as the depth of every leaf is even.

In this light, we can determine whether we can beat Death using a straightforward post-order traversal of the given binary tree. The following recursive algorithm determines whether we can win Death's game, starting at any arbitrary node v.

```
DeathGame(v):
  if v is a leaf
      return [v is white]
  return ¬(DeathGame(v.left) ∧ DeathGame(v.right))
```

The algorithm spends $O(1)$ time at every node, and therefore runs in $\boldsymbol{O(4^n)}$ ***time***. ■

Rubric: 3 points = 2 for algorithm + 1 for analysis.

(b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

Solution (randomly order grandchildren): Here I'll stick to the original view of the tree as alternating between AND and OR gates. To speed up the algorithm, we make two changes. First, we implement the logic more carefully to avoid redundant recursive calls. Second, we randomly decide whether to visit the grandchildren of v in left-to-right order (as in our deterministic algorithm) or in right-to-left order.

```
DEATHGAME(v):
  if v is a leaf
      return [v is white]
  with probability 1/2
      if DEATHGAME(v.left.left) = TRUE
          if DEATHGAME(v.left.right) = TRUE
              return TRUE
      if DEATHGAME(v.right.left) = TRUE
          return DEATHGAME(v.right.right)
      return FALSE
  else
      if DEATHGAME(v.right.right) = TRUE
          if DEATHGAME(v.right.left) = TRUE
              return TRUE
      if DEATHGAME(v.left.right) = TRUE
          return DEATHGAME(v.left.left)
      return FALSE
```

The left-to-right algorithm makes four recursive calls in exactly two cases:

- The grandchildren have values TRUE, FALSE, TRUE, FALSE. In this case, the right-to-left algorithm makes only two recursive calls (both returning FALSE).
- The grandchildren have values TRUE, FALSE, TRUE, TRUE. In this case, the right-to-left algorithm makes only two recursive calls (the rightmost TRUES).

Symmetrically, in both two cases where the right-to-left algorithm makes four recursive calls, the left-to-right algorithm makes only two. In every other case, both algorithms make at most three recursive calls. Thus, **no matter what values the grandchildren have**, the *expected* number of recursive calls is at most 3.

Let $T(n)$ denote the worst-case expected running time of this algorithm when v is the root of a tree with depth $2n$. Because random decisions at different levels of recursion are independent, this function satisfies the recurrence

$$T(n) \leq O(1) + 3 \cdot T(n-1).$$

We conclude that our algorithm runs in ***$O(3^n)$ expected time***, as required. (Moreover, this analysis is actually tight.) ■

Solution (randomly order children): This solution views every node in the tree as a NAND gate. We make two changes to our deterministic recursive algorithm. First, at each level of recursion, we flip a fair coin to decide whether to evaluate the left subtree first or the right subtree first. Second, if the first subtree evaluates to FALSE, we already know that the value of the whole tree is TRUE, so we do not evaluate the other subtree at all.

```
DEATHGAME(v):
  if v is a leaf
      return [v is white]
  with probability 1/2
      if DEATHGAME(left(v)) = FALSE
          return TRUE
      return ¬DEATHGAME(right(v))
  else
      if DEATHGAME(right(v)) = FALSE
          return TRUE
      return ¬DEATHGAME(left(v))
```

Let $T_0(d)$ denote the worst-case expected running time of our algorithm when the output is FALSE, and let $T_1(d)$ denote the worst-case expected running time when the output is TRUE, given a tree of depth d as input. Finally, let $T(d) = \max\{T_0(d), T_1(d)\}$. We need to find $T(2n)$.

If the output is FALSE, then both subtrees evaluate to TRUE, which means we must evaluate both subtrees.

$$T_0(d) = 2\,T_1(d-1) + \Theta(1).$$

On the other hand, if the output is TRUE, then at least one of the subtrees evaluates to FALSE. Thus, with probability *at least* $1/2$, the first subtree we evaluate has value FALSE, and therefore we don't evaluate the other subtree. Symmetrically, we evaluate both subtrees with probability *at most* $1/2$. So the expected number of children that we need to evaluate is at most $3/2$, which implies

$$T_1(d) \le \frac{3}{2}T(d-1) + \Theta(1).$$

Now we can express both $T_0(d)$ and $T_1(d)$ in terms of $T(d-2)$:

$$\begin{aligned} T_0(d) &= 2\,T_1(d-1) + \Theta(1) \le 3T(d-2) + \Theta(1), \\ T_1(d) &\le \frac{3}{2}T(d-1) + \Theta(1) \le 3T(d-2) + \Theta(1). \end{aligned}$$

We conclude that $\boldsymbol{T(d) \le 3T(d-2)+\Theta(1)}$. We can now easily verify by induction that $\boldsymbol{T(2n) = O(3^n)}$, as required. ■

Rubric: 7 points = 4 for algorithm + 3 for analysis. These are not the only correct solutions!

Solution (clever): See part (c). ■

Rubric: Full credit if and only if the algorithm is explained in detail in part (c) (4 points) and the analysis in part (c) is correct (3 points).

*(c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm from part (b) at all!]*

Solution: We use ***exactly*** the same NAND-tree algorithm from part (c), but we refine the recursive analysis of $T_1(d)$. We *must* evaluate a subtree whose value is FALSE. If we evaluate this subtree first, we're done; otherwise, we must evaluate a TRUE subtree first and a FALSE subtree second. The first case happens with probability *at least* $1/2$. Thus, we evaluate a TRUE subtree with probability *at most* $1/2$.

$$T_1(d) \le T_0(d-1) + \frac{1}{2}T_1(d-1) + \Theta(1)$$

Combining this inequality with our earlier recurrence $T_0(d) = 2T_1(d-1)$, we obtain a self-contained recurrence for T_1:

$$T_1(d) \le \frac{1}{2}T_1(d-1) + 2T_1(d-2) + \Theta(1).$$

The annihilator method (described in the recurrences notes) implies that $T_1(d) = O(\alpha^d)$, where $\alpha = \frac{1+\sqrt{33}}{4} \approx 1.68614$ is the larger root of the characteristic polynomial $x^2 - \frac{1}{2}x - 2$. We immediately have $T_0(d) = 2\,T_1(d-1) = O(\alpha^d)$, so $T(d) = \max\{T_0(d), T_1(d)\} = O(\alpha^d)$.

We conclude that the worst-case expected running time of our randomized algorithm is $T(2n) = O((\alpha^2)^n) = O\left(\left(\frac{17+\sqrt{33}}{8}\right)^n\right) = \boldsymbol{O(2.84308^n)} = o(3^n)$. ■

Rubric: 5 points = 2 points for the algorithm ("same as part (b)" is fine if that algorithm works) + 3 points for the refined analysis.