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 $\mathbf{E[}X_f\mathbf{]} = \boldsymbol{\infty}$.[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

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

$$= \sum_{n\geq 0} \frac{1}{2n+2}$$
$$= \frac{1}{2} \sum_{k\geq 1} \frac{1}{k}$$

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: Practice only! Not graded!

1. (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf.

Solution: We can use the following adversary strategy to assigns values to vertices of the tree (both leaves and internal nodes) as slowly as possible. Whenever the algorithm asks for the value of a leaf ℓ, the adversary executes the following recursive procedure:

```
QUERY(ℓ):
  if no sibling of ℓ has value 0
      value(ℓ) ← 0
  else if no sibling of ℓ has value 1
      value(ℓ) ← 1
  else if ℓ is the root
      value(ℓ) ← 0
  else
      value(ℓ) ← QUERY(parent(ℓ))
  return value(ℓ)
```

(Yep, it's another algorithm.) This procedure assigns a value to any node only when values have been determined for all three of its children, and therefore (inductively) all of its descendants, and therefore in particular all of its leaves. On the other hand, if a value has not yet been assigned to a node, then at most one of its children has value 1 and at most one of its children has value 0, so (again inductively) the value could be either 0 or 1, depending on the values of the unassigned leaves. ■

There is a subtle point here that many students missed. When we argue about *arbitrary* deterministic algorithms, we cannot make *any* assumptions about *how* the algorithm works. In particular, for this problem, we ***cannot*** assume that the algorithm ever computes the value of any node in the tree other than the root. I don't know how to solve the problem *without* evaluating the children of the root, but assuming that that's the *only* way to solve the problem is just like Kolmogorov assuming that multiplying two n-digit numbers requires computing each of the n^2 individual digit×digit products, or dozens of algorithms textbooks assuming that we can only sort by using pairwise comparisons. Who knows; maybe there's a clever algorithm that reverses the thaumic flow in the cthonic matrix of the optimized bi-direction octagonate.

We *can* assume (and in fact, my argument implies) that the leaves examined by the algorithm must *uniquely determine* the value of every node in the tree. But that does *not* mean that the algorithm actually computes those values. In particular, the QUERY procedure described here (which does compute a value for every node) is executed by the *adversary*, not by the algorithm we are trying to analyze.

Rubric: 5 points. This is not the only correct formulation of this proof.

(b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some explicit constant $c < 3$.

Solution: To determine the value of a node v, recursively evaluate two of its children, chosen uniformly at random. If and only if those two children has different values, recursively evaluate the third child.

- If all three children of v have the same value, the algorithm recurses twice.
- Otherwise, with probability $1/3$, the two children with the same value will be chosen first, and the algorithm will recurse only twice, and with probability $2/3$, the algorithm recurses three times. Thus, the expected number of recursive calls is $2 \cdot 1/3 + 3 \cdot 2/3 = 8/3$.

In both cases, the expected number of recursive calls is *at most* $8/3$.

It follows that the expected running time $T(n)$ for a tree of depth n obeys the recurrence $T(n) \leq (8/3) \cdot T(n-1)$, with the base case $T(0) = 1$. We conclude that our algorithm runs in $\boldsymbol{O((8/3)^n)}$ ***expected time***. ■

Rubric: 5 points. This is the best expected time possible.

2. Let A and B be two arrays of integers of length n and m respectively. Assume that all integers in the union of two arrays are distinct and we can compare whether $A[i] > B[j]$ or $A[i] < B[j]$ in $O(1)$ time, but cannot compare any two elements in A or B directly.

   (a) Show that with the allowed comparisons, it is not possible to perfectly sort the union of two arrays $A \cup B$, unless the two arrays perfectly interleave.

   Solution: Let A and B be two arrays that do not perfectly interleave. Let $A[1] < A[2] < \ldots < A[n]$ be the indices in the sorted order. Without loss of genrality, we may assume that there exists an index $i \in [n]$ such that either all elements of B are smaller than both $A[i]$ and $A[i+1]$ or all elements of B are larger than both $A[i]$ and $A[i+1]$.

   Consider any algorithm (deterministic or randomized) that given the two unsorted arrays as inputs, outputs $A \cup B$ in the sorted order. This algorithm must determine if $A[i] < A[i+1]$ or $A[i] > A[i+1]$. However, there is no element of the other array B that the algorithm can compare with to determine which case holds. Hence, it cannot sort the two arrays perfectly. In particular, the algorithm must err on one of the following two input cases: either we give the input arrays in the original unsorted order, or in the original unsorted input arrays, we swap $A[i]$ with $A[i+1]$. ■

   Rubric: Max 2 points. The justification should be valid for any two given input arrays; at most 1 point for only giving a specific example.

   (b) **Extra Credit:** Give a randomized algorithm that only uses the allowed comparisons and sorts the union of the two arrays $A \cup B$ when they perfectly interleave.

   Solution (near-linear time): Let us refer to the array A as the nuts, and to the array B as the bolts. Unlike the situation in the lecture, all the nuts and bolts are of different sizes and they perfectly interleave now. Note that the difference between the number of nuts and bolts can be at most one if they perfectly interleave.

   Consider the following randomized algorithm to sort all the *nuts*. The bolts can be sorted by repeating the same algorithm afterwards with the roles of nuts and bolts exchanged. The algorithm takes as inputs two unsorted arrays Nuts and Bolts. The top level recursive call is InterleavedSort(A, B). The algorithm uses a subroutine IsSmaller to determine if a given nut is smaller than the pivot nut by only using the allowed comparisons.

```
InterleavedSort(Nuts, Bolts):
  n ← length(Nuts)
  m ← length(Bolts)  ⟨⟨Note that |n − m| ≤ 1⟩⟩
  if n < 473
      use brute force
  p ← Random(n)        ⟨⟨choose a random pivot nut⟩⟩
  pivot ← Nuts[p]
  Partition Bolts into B< < pivot and B> > pivot
  ⟨⟨Partition the nuts into N< < pivot and N> > pivot⟩⟩
  for each i ← [n] \ {p}
      if IsSmaller(Bolts, Nuts[i], pivot)  ⟨⟨If Nuts[i] is smaller than pivot⟩⟩
          Add i to N<
      else
          Add i to N>
  ⟨⟨Recurse on the smaller and larger parts separately⟩⟩
  L ← InterleavedSort(N<, B<)
  R ← InterleavedSort(N>, B>)
  return L, pivot, R
```

The subroutine IsSmaller repeatedly compares both nuts with a random bolt until it finds a bolt that lies in between the two.

```
⟨⟨Is nut1 smaller than nut2?⟩⟩
IsSmaller(Bolts, nut1, nut2):
  m ← length(Bolts)
  while True
      r ← Random(m)
      if nut1 < Bolts[r] and Bolts[r] < nut2
          return True
      if nut1 > Bolts[r] and Bolts[r] > nut2
          return False
```

Runtime: First, we determine the expected running time of the IsSmaller subroutine. For this, we note that in each iteration of the while loop, we choose a bolt Bolts[r] uniformly at random. The probability (only over the choice of r) that it lands between the nut and the pivot is given by $|\text{rank(pivot)}-\text{rank(nut)}|/n$, where rank$(\cdot)$ denote the rank of a nut in the sorted order for the array Nuts. For a given nut and pivot, the subroutine makes $n/|\text{rank(pivot)} - \text{rank(nut)}|$ comparisons in expectation, since we want to compute the expected number of times we have to flip a biased coin until it comes up heads.

Now, consider the InterleavedSort algorithm. Note that partitioning the bolts into $B_<$ and $B_>$ takes at most $n+1$ comparisons, while for a fixed pivot nut with rank $s = \text{rank(pivot)}$, the expected number of comparions to partition the nuts into $N_<$ and $N_>$ is

$$\sum_{i=1}^{s-1} \frac{n}{r-i} + \sum_{i=s+1}^{n} \frac{n}{i-s} \;=\; nH_{s-1} + nH_{n-s} \;=\; O(n \log n),$$

where as usual H_k denotes the kth harmonic number.

Now, let $\overline{T}(n)$ denote the expected running time of InterleavedSort when the input array Nuts has size n. (Recall that the number of bolts is at most $n+1$.) This function satifies the recurrence

$$\overline{T}(n) = O(n \log n) + \mathbb{E}_s[\overline{T}(s-1) + \overline{T}(n-s)],$$

where $\mathbb{E}_s$ denotes expectation over the rank s of the randomly chosen pivot nut. This is the standard nuts and bolts recurrence that we saw in the lectures, except that the time for one recursive call is $O(n \log n)$ instead of $O(n)$. Thus, InterleavedSort runs in $\boldsymbol{O(n \log^2 n)}$ ***expected time***. ■

Solution ($O(n \log n)$ time): The main idea is to consider the nuts in random order, and use each nut to partition a subset of the bolts. Without loss of generality, assume that there are n nuts and $n+1$ bolts, so the smallest and largest objects are both bolts.

The algorithm maintains an alternating sequence $B_0 < n_1 < B_1 < n_2 < B_2 < n_3 < \cdots < n_k < B_k$, where each n_i is a nut and each B_i is a non-empty subset of the bolts, which we call a *block*. Every bolt is contained in exactly one block. Each nut n_i is larger than all bolts in B_{i-1} and smaller than all bolts in block B_i. It follows that for all $i < j$, every bolt in block B_i is smaller than every bolt in block B_j; howevre, we do not know the order of bolts within each block. Initially, we have only a single block B_0 containing all the bolts.

At each iteration, we choose a new random pivot nut n^* and then proceed in three phases:

- In the first phase, we perform a binary search to find two consecutive blocks B_i and B_{i+1}, one of which contains bolts both larger and smaller than n^*.

```
FindBlocks(n*) :
  lo ← 0
  hi ← k    ⟨⟨number of blocks⟩⟩
  while hi > lo + 1
      mid ← ⌊(lo + hi)/2⌋
      b ← any bolt in B_mid    (*)
      if b < n*
          hi ← mid
      else
          lo ← mid
  return lo, hi
```

  Regardless of which bolts we choose in line $(*)$, this binary search takes $O(\log k) = O(\log n)$ time ***in the worst case***..

- In the second phase, we determine which of the two blocks B_i or B_{i+1} contains both larger and smaller bolts than n^*. Initially, we know that B_i contains at least one smaller bolt, and B_{i+1} contains at least one larger bolt. The algorithm alternately scans through the two blocks.
  - If we find a bolt in B_i that is larger than n^*, then n^* must split B_i

- If we find a bolt in B_{i+1} that is smaller than n^*, then n^* must split B_{i+1}
- If we completely scan either block without meeting either of the previous conditions, then n^* must split the larger of the two blocks.

The following pseudocode assumes each block is stored in an array; the order that bolts are stored within each block is irrelevant.

```
WhichBlock(B_i, B_{i+1}, n*):
  for j ← 1 to min{|B_i|, |B_{i+1}|}
      if n* < B_i[j]
          return B_i
      if n* > B_{i+1}[j]
          return B_{i+1}
  if |B_i| > |B_{i+1}|
      return B_i
  else
      return B_{i+1}
```

- Finally, in the third phase, we partition whichever block B we found in phase two, by comparing every bolt in B with the pivot nut n^*. The overall running time for the second and third phases is $O(|B|)$.

The total ***worst-case*** time for all phase-one binary searches is $O(n \log n)$. It remains only to bound the expected total time for the second and third phases. Consider any block B that appears at any time during the algorithm. Because we consider nuts in random order, the first pivot nut that splits B is equally likely to be any of the nuts that split B. Thus, the total expected time spent on any block of size m (excluding the binary searches) satisfies the recurrence

$$\overline{T}(m) = \mathrm{E}_r[\overline{T}(r) + \overline{T}(m-r)] + O(m)$$

where E_r denotes that the expectations is taken over the rank r of the first nut that splits the block. This is the standard nuts and bolts recurrence, so $\overline{T}(n) = O(n \log n)$.

Adding the $O(n \log n)$ time for all binary searches, we conclude that this algorithm runs in $\boldsymbol{O(n \log n)}$ ***expected time***. ■

Rubric: 8 points = 5 for the algorithm + 3 for a correct time analysis. Any algorithm that runs in sub-quadratic expected time, that is, expected time $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$, is worth full credit. An $O(n^2)$-time algorithm is worth at most 2 points. An algorithm that runs in $O(n \log n)$ expected time is worth at most 13 points (full credit + 5 additional points).

These are not the only correct solutions. In particular, this is not the simplest $O(n \log n)$-time algorithm, but (as far as we can tell) simpler algorithms require more complex analysis.

3. Consider the following process for sampling a set of two distinct integers $\{i, j\}$ that both lie in the base interval $[1, n]$: first, we choose two non-overlapping sub-intervals $I \cup J$ of $[1, n]$. Then, we sample integers $i \in I$ and $j \in J$ uniformly at random from each sub-interval.

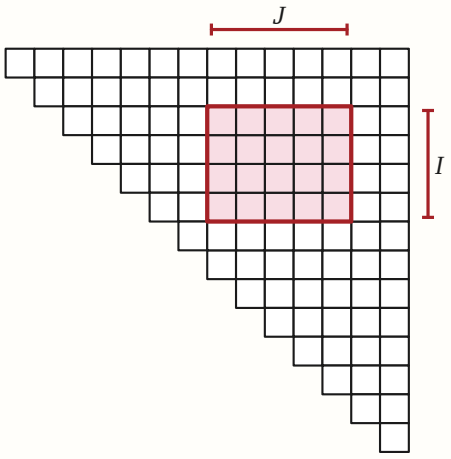
   (a) Prove that the set $\{i, j\}$ is is **not** uniformly distributed if we choose I and J to be fixed intervals and $n > 2$.

Solution: Note that if $n > 2$, at least one of the two sub-intervals I or J must be of size at least 2. Without loss of generality, say it is I. The probability that we sample a set $\{i, j\}$ where both i and j are in I, is zero for the above process. Thus, the sample can not be uniformly distributed among all set of distinct integers of $[1, n]$, since it must have probability $1/\binom{n}{2}$ otherwise. ■

Solution (rectangle ≠ triangle): Three are exactly $\binom{n}{2}$ unordered pairs $\{i, j\}$, each identified by an ordered pair (i, j) where $i < j$. These can be visualized as the set of all entries in an $n \times n$ array that lie above the main diagonal, where i is the row index and j is the column index.

Not consider ***arbitrary*** disjoint intervals I and J, where without loss of generality both endpoints of I are smaller than both endpoints of J. The ordered pairs $(i, j) \in I \times J$ comprise a rectangular subarray of our $n \times n$ array, which lies entirely above the main diagonal.

Assuming $n > 2$, the rectangle $I \times J$ cannot contain every cell above the main diagonal. In particular, $I \times J$ contains at most one of the $n - 1$ cells on the diagonal $j = i + 1$.



■

Rubric: 2 points. For full credit, the justification must be valid for any two given intervals; at most 1 point for giving a specific example.

(b) Prove that if we sample the integers $i \in I$ and $j \in J$ from the random sub-intervals $I \cup J$ given by the described algorithm, then the set $\{i, j\}$ is uniformly distributed among all sets of two distinct integers in $[1, n]$.

Solution: We first note that the algorithm does the following in each recursive call where the length of the input interval is ℓ (which is always a power of two):

- With probability $p_L = \frac{\ell-2}{4(\ell-1)} = \binom{\ell/2}{2}/\binom{\ell}{2}$ it recurses on the left sub-interval with $m = 2$. This case happens when $roll \leq \ell - 2$.
- With probability $p_{LR} = \frac{\ell}{2(\ell-1)} = \binom{\ell/2}{1}^2/\binom{\ell}{2}$, it outputs I to be the left sub-interval and J to be the right sub-interval of the current base interval $[lo, hi]$. This case happens when $\ell - 2 < roll \leq 4\ell - 2$.
- With probability $p_R = \frac{\ell-2}{4(\ell-1)} = \binom{\ell/2}{2}/\binom{\ell}{2}$ it recurses on the right sub-interval with $m = 2$. This case happens when $3\ell - 2 < roll \leq 4\ell - 4$.

Define the following induction hypothesis: for any base interval $[lo, hi]$ such that the length $\ell = hi - lo + 1$ is a power of two, and for any $m \in \{0, 1, 2\}$, sampling one point from each of the m non-overlapping intervals outputted by the algorithm SampleSubInterval(lo, hi, m) gives us a uniformly distributed set of m distinct integers in the base interval $[lo, hi]$.

The base case $m = 0$ is trivial. The case $m = 1$ is also trivial: since the algorithm has to output only one uniform point from the base interval $[lo, hi]$, it suffices to output the whole interval.

For the case $m = 2$, let us compute the probability of sampling any fixed set $\{i, j\}$ by first sampling the non-overlapping intervals and then by sampling a point from each interval uniformly. We consider three cases:

- **$\{i, j\}$ are both in left sub-interval of $[lo, hi]$.** The probability of choosing $\{i, j\}$ in this case is
$$p_L \cdot \frac{1}{\binom{\ell/2}{2}} = \frac{1}{\binom{\ell}{2}},$$
where we used the induction hypothesis for the left sub-interval.
- **i is in the left sub-interval and j is in in the right sub-interval.** The probability of choosing $\{i, j\}$ in this case is
$$p_{LR} \cdot \frac{1}{\binom{\ell/2}{1}} \cdot \frac{1}{\binom{\ell/2}{1}} = \frac{1}{\binom{\ell}{2}}.$$
- **$\{i, j\}$ are both in right sub-interval of $[lo, hi]$.** This case is symmetric with the first case.

Thus, $\{i, j\}$ is distributed uniformly among all sets of two distinct integers of $[lo, hi]$ and taking $lo = 1$ and $hi = n$ for the top-level recursive call, we get the desired statement. ■

Rubric: 4 points.

(c) Show that the *expected* score of the pair of sub-intervals given by the above algorithm is $O(\log n)$, which is close to optimal.

Solution: Note that the expected score only depends on the lengths of the base interval and the sub-intervals and not their end points. That is, shifting the sub-intervals or the base interval does not change the score. Let $s(\ell)$ denote the expected score when the length of the base-interval $[lo, hi]$ is ℓ. Consider the three cases as before,

- With probability p_L, the algorithm recurses on the left sub-interval with $m = 2$. The expected score in this case is $2s(\ell/2)$. The factor of 2 comes in because to compute the score $s(\ell/2)$, the size of the interval is halved.
- With probability p_{LR}, the algorithm outputs the left and right sub-intervals, so the score in this case is

$$\frac{\ell}{\ell/2} + \frac{\ell}{\ell/2} = 4.$$

- With probability p_R, the algorihm recurses on the right sub-interval with $m = 2$. The expected score in this case is $2s(\ell/2)$ similar to the first case.

Overall, the expected score $s(\ell)$ satisfies the recurrence:

$$\begin{aligned} s(\ell) &= 4p_{LR} + 2(p_L + p_R) \cdot s(\ell/2) \\ &= \frac{2\ell}{\ell-1} + \frac{\ell-2}{\ell-1} \cdot s(\ell/2) \\ &\leq 4 + s(\ell/2), \end{aligned}$$

with the base case $s(2) = 4$. (The last inequality uses the fact that $\ell \geq 2$, which implies $2\ell \leq 4(\ell-1)$.) Unrolling this recurrence gives us $s(\ell) \leq 4\log \ell$. Thus, for the initial interval $[1, n]$, the expected score is $O(\log n)$. ■

Solution: The algorithm SAMPLESUBINTERVAL implicitly splits the interval $1..n]$ into a balanced binary tree of $2n-1$ *canonical* intervals. Specifically, a canonical interval is either the entire interval $[1..n]$, or the left half or the right half of a canonical interval.

Fix two canonical intervals I and J whose union $I \cup J$ is a canonical interval of length $n/2^k \geq 2$. Let $P(k)$ denote the probability that SAMPLESUBINTERVAL$(1, n, 2)$ actually returns the pair $\{I, J\}$. (Symmetry in he algorithm implies that $P(k)$ does not depend on the specific intervals I and J, but only on their length.)

Returning $\{I, J\}$ requires the algorithm to recursively call itself k times, each time in either the first or third cases of the main recurrence. Each of those recursive calls has probability $\frac{\ell-2}{4(\ell-1)} \leq 1/4$, so

$$P(k) \leq \frac{1}{4^k}.$$

(In the last inequality, we are using the fact that $n/2^k \geq 2$, so $n - 2^k \geq n/2$.)

The definition of score implies

$$score(I \cup J) = \frac{n}{n/2^{k+1}} + \frac{n}{n/2^{k+1}} = 2^{k+2}.$$

Finally, there are exactly 2^k canonical intervals $I \cup J$ of length $n/2^k$, for each k from 0 to $\lg n - 1$. We conclude that the overall expected score is

$$\begin{aligned}
\mathrm{E}_{I,J}[score(I,J)] &= \sum_{I,J} \Pr[I \cup J] \cdot score(I \cup J) \\
&= \sum_{k=0}^{\lg n - 1} \sum_{I,J\,:\,|I \cup J| = n/2^k} \Pr[I \cup J] \cdot score(I \cup J) \\
&\le \sum_{k=0}^{\lg n - 1} 2^k \cdot \frac{1}{4^k} \cdot 2^{k+2} \\
&= \sum_{k=0}^{\lg n - 1} 4 \\
&= 4 \lg n.
\end{aligned}$$

■

Rubric: 4 points. These are not the only correct solutions.

(d) **Extra Credit:** Give a randomized algorithm to sample k non-overlapping intervals of $[1,n]$ such that (i) sampling one point from each interval given by the algorithm results in a uniformly distributed set of k distinct integers in $[1,n]$, and (ii) the *expected* score of the intervals is within a $O(\log n)$ factor of the optimal score.

Solution: The algorithm is a straightforward generalization of the proposed algorithm based on the following observation. Suppose we sample a set T of size k from $[1,n]$ distributed uniformly among all sets of size k. The probability of choosing m elements from the left sub-interval and $k-m$ elements from the right sub-interval is exactly

$$p_m = \frac{\binom{n/2}{m}\binom{n/2}{k-m}}{\binom{n}{k}},$$

for every $j \in \{0,1,\ldots,k\}$. The algorithm will first sample a random number $M \in \{0,1,\ldots,k\}$ with the probabilities $\Pr[M=m] = p_m$ as above, and then recursively sample a set of size M and $k-M$ from the left and right sub-intervals respectively. The base case is again when the sets are of size 0 or 1 as in the previous algorithm. Overall, the following is the pseudocode with the top-level recursive call being SAMPLESUBINTERVAL$(1,n,k)$.

```
SAMPLESUBINTERVAL(lo, hi, m):
  if m = 0
      return {}
  if m = 1
      return {[lo, hi]}
  ℓ ← hi − lo + 1          ⟨⟨length of base interval⟩⟩
  mid ← ⌊(lo + hi)/2⌋      ⟨⟨end of left half⟩⟩
  Sample a random variable M ∈ {0,...,k} with Pr[M = m] = p_m
  F_l ← SAMPLESUBINTERVAL(lo, mid, M)
  F_r ← SAMPLESUBINTERVAL(mid + 1, hi, k − M)
  return F_l ∪ F_r         ⟨⟨F_l and F_r are sets of disjoint intervals⟩⟩
```

A very similar inductive proof as in part (b) generalized to an arbitray m implies that the set obtained by sampling one element from each interval given by the above algorithm is a uniformly distributed set of size k.

Expected Score Analysis. Next we compute its expected score $s(\ell,k)$ which is given by the recurrence:

$$s(\ell,k) = 2\mathbb{E}_M[s(\ell/2,M) + s(\ell/2,k-M)].$$

The first equality carries a factor of 2 because the length of the score is measured with respect to a smaller interval of half the size when we recurse. Now note that $\Pr[M=m] = \Pr[k-M=m]$. Thus, $\mathbb{E}_M[s(\ell/2,M)] = \mathbb{E}_M[s(\ell/2,k-M)]$, and we can simplify the recurrence to

$$s(\ell,k) = 4\mathbb{E}_M[s(\ell/2,M)]. \tag{1}$$

We claim by induction that $s(\ell,k) \le (k^2-k)\log(\ell)$. The base case already follows from the case $k=2$ from part (c). So, from (1) and the induction hypothesis, we obtain

$$s(\ell,k) \le 4\mathbb{E}_M[M^2-M]\log(\ell/2). \tag{2}$$

Claim: $\mathbb{E}_M[M] = k/2$ and $\mathbb{E}_M[M^2] \le k(k+1)/4$.

Proof of Claim: Let T be a uniformly random set of k elements from the base interval of size ℓ. Note that M is the size of the intersection of T with the left sub-interval L. We can write M as a sum of indicator random variables as follows:

$$M = \sum_{i \in L} [i \in T].$$

Note that $\Pr[i \in T] = k/\ell$. Since $|L| = \ell/2$, we have

$$\mathbb{E}[M] \;=\; \sum_{i \in L} \Pr[i \in T] \;=\; |L| \cdot \frac{k}{\ell} \;=\; \frac{k}{2}.$$

Similarly,

$$\begin{aligned}
\mathbb{E}[M^2] &= \mathbb{E}\left[\left(\sum_{i \in L}[i \in T]\right)^2\right] \\
&= \sum_{i \in L} \Pr[i \in T] + \sum_{i \ne j \in L} \Pr[i,j \in T] \\
&= |L| \cdot \frac{k}{\ell} + |L|(|L|-1) \cdot \frac{k}{\ell} \cdot \frac{k-1}{\ell-1} \\
&\le \frac{k}{2} + \frac{k(k-1)}{4} \\
&\le \frac{k(k+1)}{4}.
\end{aligned}$$

□

Our claim and equation (2) imply that

$$s(\ell,k) \;\le\; (k(k+1)-2k)\log(\ell/2) \;\le\; (k^2-k)\log \ell.$$

Hence, the expected score of the algorithm for the initial recursive call on $[1,n]$ is $\boldsymbol{O(k^2 \log n)}$.

Optimal Score. We claim that this expected score is within a $O(\log n)$ factor of the optimal score. Consider any disjoint intervals $I_1, \ldots, I_k$ with lengths $\ell_!, \ldots, \ell_k$. Disjointness implies $\sum_{i=1}^k \ell_i \le n$, and by definition the score of these intervals is

$$n\left(\sum_{i=1}^{k} \frac{1}{\ell_i}\right).$$

This sum of reciprocals is minimized when the numbers ℓ_i are all equal. Otherwise, we can pick two arbitrary ℓ_i's that are not equal and replace them by their average and the obtain a smaller value, since

$$\frac{1}{a}+\frac{1}{b} \geq \frac{1}{\frac{a+b}{2}}+\frac{1}{\frac{a+b}{2}} \iff (a-b)^2 \geq 0.$$

Thus, the score of $I_1, \ldots, I_k$ must be at least

$$n\left(\sum_{i=1}^{k} \frac{k}{n}\right) \geq k^2.$$

Remark: This algorithm and its analysis are presented in the following paper:

[1] Shachar Lovett and Jiapeng Zhang. Streaming lower bounds and asymmetric set disjointness. *Proc. 64th FOCS*, 871–882, 2023. arXiv:2301.05658. ■

Rubric: 10 points: 3 for the algorithm + 4 for the expected score analysis + 3 for proving the expected score is within a $O(\log n)$ factor of optimal.