0. 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 $\mathbf{E}[X_b] = 1 + \mathbf{E}[X_a] = 3$. ■

   (c) Hamlet flips heads twice.

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

   (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

$$\begin{aligned}\mathrm{E}[X_d] &= 1 + \frac{1}{2}\,\mathrm{E}[X_d] + \frac{1}{2}\,\mathrm{E}[Y_d]\\ \mathrm{E}[Y_d] &= 1 + \frac{1}{2}\,\mathrm{E}[X_d]\end{aligned}$$

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

   (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 $\mathbf{E}[X_e] = \mathbf{E}[X_a] + \mathbf{E}[X_a] = 4$. ■

   (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 $\mathbf{E}[X_f] = \infty$.[1] ■

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 $\mathrm{E}[X_f] = \sum_{x \ge 0} x \cdot \Pr[X_f = x]$.

- The first time the number of heads exceeds the number of tails, the total number of flips must be odd. Thus, $\mathrm{E}[X_f] = \sum_{n \ge 0} (2n+1) \cdot \Pr[X_f = 2n+1]$.
- $X_f = 2n+1$ 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 $\binom{2n}{n} \ge \frac{4^n}{2n+1}$. (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

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

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

Yeesh. I should have trusted the Recursion Fairy. ■

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

Solution: Let X_g 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 $\mathrm{E}[X_g] = 1 + \mathrm{E}[E_f] = \infty$. ■

Rubric: Not graded!

[1] 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 expectation at all! 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 asplode.

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. *[Hint: Prove that at all times, the cards below card $n-1$ are uniformly shuffled.]*

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 choice of starting permutation and insertion location yields 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. This is more detail than necessary for full credit. Ignore the "weak induction" Deadly Sin.

(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 the ith phase, for some $i < n-1$. In any step in this 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 $\mathrm{E}[T_i] = n/(i+1)$.

Linearity of expectation now implies that

$$\mathrm{E}[T] \;=\; \sum_{i=1}^{n-1} \mathrm{E}[T_i] \;=\; \sum_{i=1}^{n-2} \frac{n}{i+1} + 1 \;=\; \boldsymbol{nH_{n-1} - n + 1}$$

■

Rubric: 5 points. 3 points for $\Theta(n \log n)$.

2. (a) Describe and analyze a deterministic algorithm to determine whether or not you can beat Death.

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]
    $ll \leftarrow$ DeathGame($v.left.left$)
    $lr \leftarrow$ DeathGame($v.left.right$)
    $rl \leftarrow$ DeathGame($v.right.left$)
    $rr \leftarrow$ DeathGame($v.right.right$)
    return $(ll \wedge lr) \vee (rl \wedge rr)$
```

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 $\neg$(DeathGame($v.left$) $\wedge$ 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 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

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

We conclude that our algorithm runs in ***$O(3^n)$ expected time***, as required. (Moreover, the 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 explicit constant $c < 3$. Your analysis should yield an exact value for the constant c.

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.

(a) State a recurrence for the expected running time of QuickSelect, as a function of n and k.

Solution: Let $T(n,k)$ denote the expected running time of QuickSelect, as a function of n and k. Partition runs in $O(n)$ time (specifically, using $n-1$ comparisons). Because the pivot element is chosen uniformly at random, the pivot rank r is equally likely to be any integer from 1 to n. Thus,

$$T(n,k) = O(n) + \frac{1}{n}\sum_{r=1}^{k-1} T(n-r, k-r) + \frac{1}{n}\sum_{r=k+1}^{n} T(r-1,k)$$

Each of the sums corresponds to one of the "else" cases in the algorithm. The range of each summation corresponds to the values of r that invoke that case. ∎

Rubric: 3 points.

(b) What is the *exact* probability that QuickSelect compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

Solution: For convenience, I'll assume the elements of the input array are the integers 1 through n, so that I can write "i" instead of "the ith smallest item". Let $P(i,j,k)$ denote the probability that QuickSelect compares i and j, given that k is the target rank.

Without loss of generality, we can assume that $i \le j$. The algorithm never compares an element to itself, so $P(i,i,k) = 0$ for all i and k.

For any $i < j$, consider the deepest recursive subproblem that includes items i, j, and k. The random pivot for this subproblem cannot be either smaller than i and k or larger than j and k; otherwise, there would be a deeper relevant subproblem. This is the only subproblem where items i and j can be compared. There are three cases to consider:

- If the pivot for this subproblem is either i or j, then the algorithm compares i and j.
- If the pivot lies strictly between i and j, then at most one of those items survives to the next subproblem, so the algorithm does not compare i and j.
- Otherwise, if the pivot is item k, then the algorithm does not recurse and thus does not compare i and j.
- Finally, if the pivot is between i and k, or between j and k, then neither i nor j survive to the next subproblem, so the algorithm does not compares i and j.

We conclude that

$$P(i,j,k) = \begin{cases} 0 & \text{if } i = j \\ \dfrac{2}{\max\{j,k\} - \min\{i,k\} + 1} & \text{otherwise} \end{cases}$$

∎

Rubric: 2 points.

(c) What is the *exact* probability that in one of the recursive calls to QuickSelect, the first argument is the subarray $A[i\,..\,j]$? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

Solution: Again, I'll assume without loss of generality that the elements of the input array are the integers 1 through n, so that I can write "i" instead of "the ith smallest item". For any indices $i \leq j$ and k, Let $R(i, j, k)$ denote the probability that some Recursive call partitions the subarray $A[i\,..\,j]$, where the *initial* target rank is k. We consider four cases.

- Because every recursive subproblem includes k, we immediately have $R(i, j, k) = 0$ either if $k < i$ or $k > j$.
- The subarray $A[1\,..\,n]$ is our initial call, so $R(1, n, k) = 1$.
- Suppose $i = 1$ and $k \leq j < n$. The subarray $A[1\,..\,j]$ is an active subproblem if and only if $j+1$ is chosen as a pivot before any element smaller than $j+1$. Since pivots are chosen in random order, we have
$$R(1, j, k) = \frac{1}{j+1}.$$
- Symmetrically, when $j = n$ and $1 < i \leq k$, the subarray $A[i\,..\,n]$ is an active subproblem if and only if $i-1$ is chosen as a pivot any element larger than $i-1$.
- Finally, suppose $1 < i \leq k \leq j < n$. Now the subarray $A[i\,..\,n]$ is an active subproblem if and only if $i-1$ and $j+1$ are *both* chosen as pivots before any integer between i and j. There are $j-i+3$ elements between $i-1$ and $j+1$, so there are $\binom{j-i+3}{2} = (j-i+3)(j-i+2)/2$ possibilities for the first two pivots in that range, and each of those possibilities is equally likely.

We conclude that
$$R(i, j, k) = \begin{cases} 1 & \text{if } i = 1 \text{ and } j = n \\ 0 & \text{if } k < i \text{ or } k > j \\ \dfrac{1}{j+1} & \text{if } i = 1 \text{ and } k \leq j < n \\ \dfrac{1}{n-i+2} & \text{if } j = n \text{ and } 1 < i \leq k \\ \dfrac{2}{(j-i+3)(j-i+2)} & \text{otherwise} \end{cases}$$

■

Rubric: 3 points.

(d) Show that for any n and k, the expected running time of QuickSelect is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b) or (c).

Solution (conservative recurrence): The algorithm always performs at least $n-1$ comparisons, so it's enough to prove that the expected running time is $O(n)$.

Let $T(n) = \max_k T(n,k)$ denote the expected worst-case time for QuickSelect on an input array of size n. If the random pivot is in the middle half of the array, we recurse in an array of size at most $3n/4$. Otherwise, we recurse in an array of size at most n. Each of these choices happens with probability $1/2$. Sowe have the following crude upper bound recurrence:

$$T(n) \le \frac{T(n)}{2} + \frac{T(3n/4)}{2} + n$$

Simple algebra simplifies the recurrence to $T(n) \le T(3n/4) + 2n$. This recurrence expands into a descending geometric series, which implies $\boldsymbol{T(n) = O(n)}$. ■

Rubric: 3 points. Yes, this is enough for full credit.

Solution (from part (a), by induction): The algorithm always performs at least $n-1$ comparisons, so it's enough to prove that the expected running time is $O(n)$.

We prove by induction that $T(n,k) \le \alpha n$ from the recurrence from part (a), for some constant $\alpha > 0$ that will fall out of the proof. The base case $n \le 1$ is trivial.

$$\begin{aligned}
T(n,k) &= n-1+\frac{1}{n}\sum_{r=1}^{k-1} T(n-r,k-r)+\frac{1}{n}\sum_{r=k+1}^{n} T(r-1,k) \\
&\le n-1+\frac{1}{n}\sum_{r=1}^{k-1}\alpha(n-r)+\frac{1}{n}\sum_{r=k+1}^{n}\alpha(r-1) && \text{[induction hypothesis]} \\
&= n-1+\frac{\alpha}{n}\left(\sum_{r=1}^{k-1}(n-r)+\sum_{r=k+1}^{n}(r-1)\right) \\
&= n-1+\frac{\alpha}{n}\left(\sum_{i=1}^{n-1} i-\sum_{i=1}^{n-k} i \;+\; \sum_{j=1}^{n-1} j-\sum_{j=1}^{k-1} j\right) && [i=n-r;\ j=r-1] \\
&= n-1+\frac{\alpha}{n}\left(n(n-1)-\frac{(n-k)(n-k-1)}{2}-\frac{(k-1)(k-2)}{2}\right)
\end{aligned}$$

The expression $(n-k)(n-k-1)+(k-1)(k-2)$ is the sum of two convex functions of k, one decreasing and the other increasing. Thus, the expression is *minimized* when $k=(n+1)/2$, making the two terms equal.[2] We conclude that

$$\begin{aligned}
T(n,k) &\le n-1+\frac{\alpha}{n}\left(n(n-1)-\frac{n-1}{2}\cdot\frac{n-3}{2}\right) \\
&\le n-1+\frac{\alpha}{n}\cdot\frac{(n-1)(n+3)}{2} \\
&\le n-1+\frac{\alpha}{n}\cdot\frac{3n^2}{4} \\
&< \left(\frac{3\alpha}{4}+1\right)n
\end{aligned}$$

This final upper bound is less than or equal to αn if and only if $\alpha \ge 4$. We conclude that $\boldsymbol{T(n,k) \le 4n}$. ■

[2]If you prefer a more formal proof, set the partial derivative $\frac{\partial}{\partial k}((n-k)(n-k-1)+(k-1)(k-2))$ to zero and solve for k.

Solution (from part (b)): The algorithm always performs at least $n-1$ comparisons, so it's enough to prove that the expected running time is $O(n)$.

The exact expected number of comparisons is the sum, over all pairs $i < j$ of the probability of comparing the ith and jth smallest elements:

$$T(n,k) = \sum_{1\le i<j\le n} P(i,j,k) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{\max\{j,k\} - \min\{i,k\} + 1}$$

This summation only considers pairs with $i < j$ because we only want to count each possible comparison once. The case $i = j$ doesn't contribute anything to the summation, so we ignore it.

The best way to bound this summation is to break it up into three cases: $k \le i < j$, $i < k < j$, and $i < j \le k$.

$$T(n,k) \;=\; \sum_{i=k}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-k+1} \;+\; \sum_{i=1}^{k-1}\sum_{j=k+1}^{n} \frac{2}{j-i+1} \;+\; \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} \frac{2}{k-i+1}$$

Let's bound each of these three sums separately.

- The last term is the easiest, because the fraction doesn't depend on the index of the inner sum; every term in the outer sum is less than 1.

$$\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} \frac{2}{k-i+1} \;=\; \sum_{i=1}^{k-1} \frac{2(k-i)}{k-i+1} \;<\; \sum_{i=1}^{k-1} 2 \;=\; 2(k-1)$$

- To evaluate the first term, we first reverse the order of summation. Then every term in the outer sum is less than 1, just like the previous case.

$$\sum_{i=k}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-k+1} \;=\; \sum_{j=k+1}^{n}\sum_{i=k}^{j-1} \frac{2}{j-k+1} \;=\; \sum_{j=k+1}^{n} \frac{2(j-k)}{j-k+1} \;<\; \sum_{j=k+1}^{n} 2 \;=\; 2(n-k)$$

- The middle term is the trickiest. The limits of the sum

$$\sum_{i=1}^{k-1}\sum_{j=k+1}^{n} \frac{2}{j-i+1},$$

  consider all values of i and j such that $1 \le i < k < j \le n$. The denominator $j-i+1$, which the length of the interval $[i\,..\,j]$, varies from 3 to n, and each interval length ℓ is considered *at most* $\ell-2$ times. Thus, we can overestimate the sum by considering each relevant length ℓ *exactly* $\ell-2$ times.

$$\sum_{i=1}^{k-1}\sum_{j=k+1}^{n} \frac{2}{j-i+1} \;<\; \sum_{\ell=3}^{n} \frac{2(\ell-2)}{\ell} \;<\; \sum_{\ell=3}^{n} 2 \;=\; 2(n-2)$$

Combining these three upper bounds gives us $\boldsymbol{T(n,k) < 4n-6 = O(n)}$.

(More careful analysis of this sum leads to the upper bound

$$\begin{aligned} T(n,k) \;&\le\; 2nH_n - 2(n-k+1)H_{n-k+1} - 2kH_k + 2n + o(n) \\ &\le\; 2n(H_n - H_{n/2} + 1) + o(n) \\ &\le\; 2n(\ln 2 + 1) + o(n) \;<\; 3.3863n + o(n) \end{aligned}$$

with the worst case occurring at $k = n/2$.) ■

Solution (from part (c)): The algorithm always performs at least $n-1$ comparisons, so it's enough to prove that the expected running time is $O(n)$.

We can decompose the expected number of comparisons $T(n,k)$ into a summation over all possible intervals of the expected time spent partitioning that interval. Partitioning the interval $[i\,..\,j]$ requires either $j-i+1$ comparisons (if that interval is the argument of some recursive call) or 0 comparisons (if the interval is never the argument of some recursive call). Thus, we have

$$\begin{aligned} T(n,k) &= \sum_{\text{intervals } I} \Pr[I \text{ appears}] \cdot \mathrm{E}[\text{time partitioning } I \mid I \text{ appears}] \\ &= \sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j,k)\cdot(j-i+1) \end{aligned}$$

Plugging in our closed-form expression for $R(i,j,k)$ from part (c) gives us

$$T(n,k) \;=\; n-1 \;+\; \sum_{j=k}^{n-1}\frac{j}{j+1} \;+\; \sum_{i=2}^{k}\frac{n-i+1}{n-i+2} \;+\; \sum_{i=2}^{k}\sum_{j=k}^{n-1}\frac{2(j-i+1)}{(j-i+3)(j-i+2)}$$

As in the previous solution, we consider each of these sums separately.

- The first sum is easy; every term is less than 1.

$$\sum_{j=k}^{n}\frac{j}{j+1} \;<\; n-k+1$$

- The second sum is also easy; every term is less than 1.

$$\sum_{i=1}^{k}\frac{n-i+1}{n-i+2} \;<\; k$$

- The third term is the trickiest. First let's conservatively simplify the fraction inside the sum.

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

  The limits of this duble summation consider all values of i and j such that $1 < i \le k \le j < n$. The denominator $j-i+2$, which is one more than the length of the interval $[i\,..\,j]$, varies from 2 to $n-1$, and each value ℓ of this denominator appears *at most* $\ell-1$ times. Thus, we can overestimate the sum as follows:

$$\sum_{i=2}^{k}\sum_{j=k}^{n-1}\frac{2}{j-i+2} \;<\; \sum_{\ell=2}^{n-1}\frac{2(\ell-1)}{\ell} \;<\; \sum_{\ell=2}^{n-1}2 \;=\; 2(n-2)$$

Combining these three upper bounds gives us $\boldsymbol{T(n,k) \le 4n-4}$.

(More careful analysis of this summation leads to the upper bound $2n(\ln 2+1) + o(n) < 3.3863n + o(n)$, just like part (c).) ■