1. Consider an n-node treap T. As in the lecture notes, we identify nodes in T by the ranks of their search keys. Thus, 'node 5' means the node with the 5th smallest search key. Let i, j, k be integers such that $1 \le i \le j \le k \le n$.

   (a) What is the *exact* probability that node j is a common ancestor of node i and node k?

   Solution: Recall from class that node j is an ancestor of node i if and only if node j has the smallest priority among all nodes between i and j. Similarly, node j is an ancestor of node k if and only if node j has the smallest priority of all nodes between j and k.

   Thus, j is an ancestor of both i and k if and only if node j has the smallest priority of all nodes between node i and node k. The probability of this event is exactly $\mathbf{1/(k-i+1)}$. (In particular, nodes i and k have *exactly* one common ancestor with intermediate rank.) ■

   (b) What is the *exact* expected length of the unique path from node i to node k in T?

   Solution: Let $\ell(i,k)$ denote the length of the unique path from node i to node k. We can express this path length as $\ell(i,k) = \mathit{depth}(i) + \mathit{depth}(k) - 2 \cdot \mathit{depth}(\mathit{lca}(i,k))$, where $\mathit{lca}(i,k)$ is the lowest common ancestor of nodes i and k. Linearity of expectation implies that

$$\mathrm{E}[\ell(i,k)] = \mathrm{E}[\mathit{depth}(i)] + \mathrm{E}[\mathit{depth}(k)] - 2 \cdot \mathrm{E}[\mathit{depth}(\mathit{lca}(i,k))]$$

   Recall from the lecture notes that $\mathrm{E}[\mathit{depth}(i)] = H_i + H_{n-i+1} - 2$. Similarly, $\mathrm{E}[\mathit{depth}(k)] = H_k + H_{n-k+1} - 2$. It remains only to compute the expected depth of $\mathit{lca}(i,k)$.

   The depth of $\mathit{lca}(i,k)$ is exactly the number of common ancestors of i and k. For any index j, let $X_j = 1$ if node j is a common ancestor of nodes i and k, and $X_j = 0$ otherwise. Generalizing the argument in part (a), we have

$$\Pr[X_j = 1] = \begin{cases} 1/(k-j+1) & \text{if } j < i \\ 1/(k-i+1) & \text{if } i \le j \le k \\ 1/(j-i+1) & \text{if } j > i \end{cases}$$

   Thus, the expected number of common ancestors of i and k is exactly

$$\begin{aligned} \sum_{j=1}^{n} \mathrm{E}[X_j] &= \sum_{j=1}^{i-1} \frac{1}{k-j+1} + \sum_{j=i}^{k} \frac{1}{k-i+1} + \sum_{j=k+1}^{n} \frac{1}{j-i+1} \\ &= \sum_{\ell=k-i}^{k} \frac{1}{\ell} + 1 + \sum_{\ell=k-i+2}^{n-i+1} \frac{1}{\ell} \\ &= \left(H_k - H_{k-i+1}\right) + 1 + \left(H_{n-i+1} - H_{k-i+1}\right) \\ &= H_k + H_{n-i+1} - 2H_{k-i+1} + 1 \end{aligned}$$

   Putting all the pieces together, we conclude:

$$\begin{aligned} \mathrm{E}[\ell(i,k)] &= \mathrm{E}[\mathit{depth}(i)] + \mathrm{E}[\mathit{depth}(k)] - 2 \cdot \mathrm{E}[\mathit{depth}(\mathit{lca}(i,k))] \\ &= \left(H_i + H_{n-i+1} - 2\right) + \left(H_k + H_{n-k+1} - 2\right) - 2\left(H_k + H_{n-i+1} - 2H_{k-i+1} + 1\right) \\ &= \mathbf{4H_{k-i+1} + (H_i - H_k) + \left(H_{n-k+1} - H_{n-i+1}\right) - 6}. \end{aligned}$$

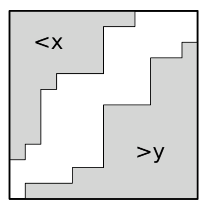
   Because $H_i < H_k$ and $H_{n-k+1} < H_{n-i+1}$, we also have a simple upper bound $\mathrm{E}[\ell(i,k)] < 4H_{k-i+1} = O(\log(k-i+1))$. ■

2. Let $M[1..n, 1..n]$ be an $n \times n$ matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of M 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 smaller than $M[i, j]$ and larger than $M[i', j']$.

   Solution: We describe and analyze an algorithm NumBetween(M, x, y) that returns the number of elements $M[i, j]$ such that $x \le M[i, j] \le y$. The number of elements of M smaller than $M[i, j]$ and larger than $M[i', j']$ is exactly NumBetween$(M, M[i', j'], M[i, j]) - 2$.

   Our algorithm uses a subroutine PrefixesAndSuffixes that computes two arrays $P[1..n]$ and $S[1..n]$, where $P[i]$ is the number of elements of row i that are less than x, and $S[i]$ is the number of elements of row i that are greater than y. These arrays will be useful in our later algorithms. Because M is totally monotone, the elements in any row or column of M that are less than x define a *prefix* of that row or column. Thus, the first $P[i]$ elements any row i are less than x, and $P[i] \ge P[i+1]$ for every index i. Similarly, the elements in any row or column of M that are greater than y define a *suffix* of that row or column. Thus, the last $S[i]$ elements any row i are less than x, and $S[i] \le S[i+1]$ for every index i. Intuitively, all elements smaller than x lie above a 'staircase' in the upper left corner of M, and all elements larger than y lie above a 'staircase' in the lower right corner of M.




```
ComputeFixes(M, x, y):
  P[0] ← n;  S[0] ← 1
  for i ← 1 to n
      P[i] ← P[i − 1]
      while (P[i] ≥ 1) and (M[i, P[i]] ≥ x)
          P[i] ← P[i] − 1
      S[i] ← S[i − 1]
      while (S[i] ≤ n) and (M[i, n + 1 − S[i]] ≤ y)
          S[i] ← S[i] + 1
  return P[1..n], S[1..n]
```

```
NumBetween(M, x, y):
  P, S ← PrefixesAndSuffixes(M, x, y)
  count ← 0
  for i ← 1 to n
      count ← count + n − P[i] − S[i]
  return count
```

   In ComputeFixes, the line $P[i] \leftarrow P[i] - 1$ is executed at most n times, and the line $S[i] \leftarrow S[i] + 1$ is executed at most n times. We conclude that ComputeFixes, and therefore NumBetween, runs in $O(n)$ time. ■

   (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: Again, we actually describe an algorithm to choose a random element between arbitrary numbers x and y. We start by computing the arrays P and S using the ComputeFixes algorithm from part (a). For any index i, *Total*$[i]$ denotes the number of elements between x and y in the first i rows of M. Our algorithm chooses a random integer r between 1 and *Total*$[n]$, and then finds the rth element of M between x and y in row-major order.

```
RandomBetween(M, x, y):
  P, S ← ComputeFixes(M, x, y)
  Total[0] ← 0
  for i ← 1 to n
      Total[i] ← Total[i − 1] + n − P[i] − S[i]
  r ← Random(Total[n])
  i ← 1
  while r > Total[i]
      i ← i + 1
  j ← r − Total[i − 1]
  return M[i][j]
```

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

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

Solution: The following recursive algorithm selects the kth smallest element of M between x and y. To find the median element of M, we would call Select$(M, 0, \infty, n^2/2)$.

```
Select(M, x, y, k):
  if NumBetween(M, x, y) < k
      return None
  pivot ← RandomBetween(M, x, y)
  rank ← NumBetween(M, x, pivot)
  if rank = k
      return pivot
  else if rank < k
      return Select(M, pivot, y, k − rank)
  else if rank > k
      return Select(M, x, pivot, k)
```

Let $T(n, B, k)$ denote the expected running time of this algorithm when B is the number of elements between x and y. Because the pivot element is equally likely to be any of the B elements between x and y, we have the following recurrence:

$$T(n, B, k) = O(n) + \frac{1}{B}\left(\sum_{i=1}^{k-1} T(n, i-1, B-k) + \sum_{i=k+1}^{B} T(n, B-i, k)\right)$$

Following the crude analysis of randomized quicksort (for nuts and bolts), let us call a trial of Select *good* if *rank* is between $B/4$ and $3B/4$, and *bad* otherwise; a trial is good with probability $1/2$. Let $T(n, B) = \max_k T(n, B, k)$. If the trial is good, then the expected time for the recursive call to Select is at most $T(n, 3B/4)$; if the trial is bad, the recursive call to Select is faster than starting over from scratch. Thus, we have

$$T(n, B) \le O(n) + \frac{1}{2} T(n, 3B/4) + \frac{1}{2} T(n, B)$$

which implies

$$T(n, B) \le O(n) + T(n, 3B/4)$$

The recursion tree method implies that $T(n, B) = O(n \log B)$. We conclude that the expected time to find the median of M is $T(n, n^2) = O(n \log n)$, as required. ■

3. Suppose we are given a complete undirected graph G, in which each edge is assigned a weight chosen independently and uniformly at random from the real interval $[0,1]$. Consider the following greedy algorithm to construct a Hamiltonian cycle in G. We start at an arbitrary vertex. While there is at least one unvisited vertex, we traverse the minimum-weight edge from the current vertex to an unvisited neighbor. After $n-1$ iterations, we have traversed a Hamiltonian path; to complete the Hamiltonian cycle, we traverse the edge from the last vertex back to the first vertex. What is the expected weight of the resulting Hamiltonian cycle? *[Hint: What is the expected weight of the first edge? Consider the case $n=3$.]*

Solution: We start with a useful lemma. Recall that the expectation of a continuous random variable X over the interval $[a,b]$ is defined as

$$\mathrm{E}[X] = \int_a^b \Pr[X \geq x]\, dx.$$

For any set S of k random variables over the interval $[0,1]$, we have

$$\mathrm{E}[\min S] = \int_0^1 \Pr\left[\min S \geq x\right]\, dx = \int_0^1 \Pr\left[\bigwedge_{X \in S} X \geq x\right]\, dx.$$

If the variables in S are independent and uniformly distributed over $[0,1]$, then

$$\Pr\left[\bigwedge_{X \in S} X \geq x\right] = \prod_{X \in S} \Pr[X \geq x] = (1-x)^k,$$

and therefore

$$\mathrm{E}[\min S] = \int_0^1 (1-x)^k dx = \left.\frac{-(1-x)^{k+1}}{k+1}\right|_0^1 = \frac{1}{k+1}.$$

Now let W_i denote the weight of the ith edge selected by the randomized algorithm, and let $W = \sum_i W_i$. For each index $i < n$, the weight W_i is the minimum of $n-i$ independent random variables distributed uniformly over the real interval $[0,1]$. Finally, W_n is uniformly distributed over the interval $[0,1]$. Thus, we conclude that

$$\mathrm{E}[W] \;=\; \sum_{i=1}^{n} \mathrm{E}[W_i] \;=\; \sum_{i=1}^{n-1} \frac{1}{n-i+1} + \frac{1}{2} \;=\; \sum_{j=2}^{n} \frac{1}{j} + \frac{1}{2} \;=\; H_n - \frac{1}{2} \;=\; \Theta(\log n)$$

■

4. (a) Prove that VERTEXCOVER can return a vertex cover that is $\Omega(n)$ times larger than the smallest vertex cover. You need to describe both an input graph with n vertices, for any integer n, and the sequence of edges and endpoints chosen by the algorithm.

Solution: Consider a tree with $n+1$ vertices $u, v_1, v_2, \ldots, v_n$, and edges uv_i for every i. Suppose that in every iteration, VERTEXCOVER considers some edge uv_i and adds v_i to the vertex cover. The resulting vertex over contains n vertices, but there is a vertex cover $\{u\}$ of size 1. ■

(b) Prove that the expected size of the vertex cover returned by RANDOMVERTEXCOVER is at most $2 \cdot \mathrm{OPT}$, where OPT is the size of the smallest vertex cover.

Solution: See part (c). ■

(c) Prove that the expected weight of the vertex cover returned by RANDOMWEIGHTEDVERTEXCOVER is at most $2 \cdot \mathrm{OPT}$, where OPT is the weight of the minimum-weight vertex cover.

Solution: Fix a vertex z, and let T_z be an arbitrary subset of edges incident to z. (These edges comprise a star tree with z at its center.) Let $C(T_z)$ denote the vertices that are added to C by examining edges in T_z. That is, when RANDOMWEIGHTEDVERTEXCOVER examines any edge uz in T_z, if neither u nor z is in C, then either u or z is added to both C and $C(T_z)$.

First we prove by induction that $\mathrm{E}[w(C(T_z))] \le 2w(z)$ for any vertex $z \in Z$. We prove this claim by induction. If T_z is empty, the claim is trivial because $w(C(T_z))) = w(\varnothing) = 0$. Otherwise, consider an arbitrary edge $uz \in T_z$ considered by RANDOMWEIGHTEDVERTEXCOVER. The inductive hypothesis implies that $\mathrm{E}[w(C(T_z \setminus uz))] \le 2w(z)$. There are two cases to consider.

- If either u or z is already marked when RANDOMWEIGHTEDVERTEXCOVER considers edge uv, then $C(T_z) = C(T_z \setminus uz)$, and thus $\mathrm{E}[w(C(T_z))] = \mathrm{E}[w(C(T_z \setminus uz))] \le 2w(z)$.
- Otherwise, we have

$$\begin{aligned}
\mathrm{E}[w(C(T_z))] &= \frac{w(u)}{w(u)+w(z)} w(z) + \frac{w(z)}{w(u)+w(z)} (w(u) + \mathrm{E}[w(C(T_z'))]) \\
&\le \frac{w(u)}{w(u)+w(z)} w(z) + \frac{w(z)}{w(u)+w(z)} (w(u) + 2w(z)) \\
&= \frac{2w(u)w(z) + 2w(z)^2}{w(u)+w(z)} \\
&= 2w(z).
\end{aligned}$$

Now let Z be an *arbitrary* vertex cover, and let $w(Z) = \sum_{z \in Z} w(z)$ denote its total weight. Partition the edges of G by assigning each edge to an *arbitrary* vertex in Z that covers it; let T_z denote the set of edges assigned to vertex $z \in Z$. We now have $C = \bigcup_{z \in Z} C(T_z)$, and therefore $w(C) = \sum_{z \in Z} w(C(T_z))$. The previous arugment implies that

$$\mathrm{E}[w(C)] \;\le\; \sum_{z \in Z} \mathrm{E}[w(C(T_z))] \;=\; \sum_{z \in Z} 2w(z) \;=\; 2w(Z) \;\le\; 2 \cdot \mathrm{OPT}.$$

■

5. (a) Suppose n balls are thrown uniformly and independently at random into m bins. For any integer k, what is the *exact* expected number of bins that contain exactly k balls?

Solution: For any index i, the ith bin contains exactly k balls with probability exactly $\binom{n}{k}(1/m)^k(1-1/m)^{n-k}$. Thus, by linearity of expectation, the expected number of bins containing exactly k balls is $m\binom{n}{k}(1/m)^k(1-1/m)^{n-k}$. ■

(b) Consider the following balls and bins experiment, where we repeatedly throw a fixed number of balls randomly into a shrinking set of bins. We start with n balls and n bins. In each round, we throw n balls into the remaining bins, and then discard any non-empty bins. Suppose that in every round, *precisely* the expected number of bins are empty. Prove that under these conditions, the experiment ends after $O(\log^* n)$ rounds.

Solution: Let $x \Uparrow k$ denote an exponential tower of k x's:

$$x \Uparrow k := \begin{cases} 1 & \text{if } k = 0 \\ x^{x \Uparrow (k-1)} & \text{otherwise} \end{cases}$$

We first prove by induction that after k rounds, the number of remaining bins is at most $n/(e \Uparrow k)$. The base case $k = 0$ is trivial. By part (a), after throwing n balls into n/α bins, the expected number of empty bins is

$$\frac{n}{\alpha}\left(1 - \frac{\alpha}{n}\right)^n < \frac{n}{\alpha e^\alpha} < \frac{n}{e^\alpha}.$$

The inductive hypothesis implies that there are at most $n/(e \Uparrow (k-1))$ bins available at the start of the kth round. Thus, after throwing n balls into these bins, the number of empty bins is at most $n/(e^{e \Uparrow (k-1)}) = n/(e \Uparrow k)$, as claimed.

By definition, we have $x < e \Uparrow (\log^* x) \le e^x$ for any real number x. Thus, after $\log^* n$ rounds, the number of remaining bins is less than 1, and therefore must be equal to 0. ■

*(c) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BallsDestroyBins(n) ends after $O(\log^* n)$ rounds.

(d) Now consider a variant of the previous experiment in which we discard balls instead of bins. Again, we start with n balls and n bins. In each round, we throw the remaining balls into n bins, and then discard any ball that lies in a bin by itself. Suppose that in every round, *precisely* the expected number of bins contain exactly one ball. Prove that under these conditions, the experiment ends after $O(\log \log n)$ rounds.

Solution: By part (a), after throwing k balls into $n > k$ bins, the expected number of bins containing exactly one ball is

$$k\left(1 - \frac{1}{n}\right)^{k-1} \ge k\left(1 - \frac{1}{n}\right)^{k} \ge k\left(1 - \frac{k}{2n}\right).$$

(The second step uses the inequality $(1-x)^k < 1 - kx/2$, which holds whenever $0 \le x \le 1/k$.) It follows immediately that the expected number of balls the survive to the next round is at most $k^2/2n$. In other words, if we start a round with n/α balls, for some $\alpha > 1$ we expect $n/2\alpha^2$ balls to survive into the next round.

Let $T(0) = 1$ and $T(r) = 2 \cdot T(r-1)^2$ for all $i > 0$. The previous argument implies inductively that after r rounds, the number of remaining balls is $n/T(r)$. The function $t(r) = \lg T(r)$ obeys the Tower of Hanoi recurrence $t(r) = 1 + 2t(r-1)$, which implies that $t(r) = 2^r - 1$, and therefore $T(r) = 2^{2^r - 1}$. Thus, if we set $r = \lceil \lg(\lg n + 1) \rceil + 1$ rounds, we have $T(r) > n$, which means less than 1 ball remains after r rounds.

We conclude that the experiment ends after at most $\lceil \lg(\lg n + 1) \rceil + 1$ rounds. ■

*(e) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BinsDestroySingleBalls(n) ends after $O(\log \log n)$ rounds.