1. Let $M[1..n, 1..n]$ be an $n \times n$ matrix in which every row and every column is sorted. 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 $\text{NumberBetween}(M, x, y)$ that returns the number of elements $M$ that are larger than $x$ and smaller than $y$, for arbitrary $x < y$. Our algorithm also computes two arrays $\text{LessEqX}[1..n]$ and $\text{LessY}[1..n]$ that will be useful in our later algorithms:

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

```plaintext
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 $jx ← jx − 1$ and $jy ← jy − 1$ are each executed at most $n$ times; we conclude that $\text{NumLess}$ runs in $O(n)$ time.

**Rubric:** This is not the only way to formulate this algorithm. Computing the arrays $\text{LessY}$ and $\text{LessEqX}$ 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 $\text{NumberBetween}(M, x, y)$, we choose a random number $r$ between 1 and the number of relevant elements, and then scan for the row containing the $r$th relevant element using the auxiliary arrays $\text{LessEqX}$ and $\text{LessY}$. 

1
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: This is not the only way to formulate this algorithm.

(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 ← NumberBetween(M, −∞, pivot)
        if rank = n/2
            return pivot
        else if rank < n/2
            lo ← pivot
        else if rank > n/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 = ∑_i X_i.

We can compute Pr[X_i = 1] by reformulating the algorithm as follows: Initialize lo ← −∞ and hi ← ∞, and consider the elements m ∈ 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. 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

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 $m/4 \leq rank < 3m/4$, 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

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

which implies $L(m) \leq 2 + L(3m/4)$ 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. ■

**Rubric:** These are not the only ways to either formulate or analyze the algorithm.
2. **Tabulated hashing** uses tables of random numbers to compute hash values. Suppose $|U| = 2^w \times 2^w$ and $m = 2^l$, so the items being hashed are pairs of $w$-bit strings (or $2w$-bit strings broken in half) and hash values are $l$-bit strings.

Let $A[0..2^w - 1]$ and $B[0..2^w - 1]$ be arrays of independent random $l$-bit strings, and define the hash function $h_{A,B} : U \rightarrow [m]$ by setting

$$h_{A,B}(x, y) := A[x] \oplus B[y]$$

where $\oplus$ denotes bit-wise exclusive-or. Let $\mathcal{H}$ denote the set of all possible functions $h_{A,B}$. Filling the arrays $A$ and $B$ with independent random bits is equivalent to choosing a hash function $h_{A,B} \in \mathcal{H}$ uniformly at random.

(a) Prove that $\mathcal{H}$ is 2-uniform.

**Solution:** Let $(x, y)$ and $(x', y')$ be arbitrary distinct elements of $U$, and let $i$ and $j$ be arbitrary (possibly equal) hash values. To simplify notation, we define

$$a = A[x], \quad b = B[y], \quad a' = A[x'], \quad \text{and} \quad b' = B[y'].$$

Say that $a, b, a', b'$ are **good** if $a \oplus b = i$ and $a' \oplus b' = j$. We need to prove that

$$\Pr[a, b, a', b' \text{ are good}] = \frac{1}{m^2}.$$

There are three cases to consider.

- Suppose $x \neq x'$ and $y \neq y'$. Then $a, b, a', b'$ are four different and therefore independent random $w$-bit strings. There are $m^4$ possible values for $a, b, a', b'$. If we fix $a$ and $a'$ arbitrarily, there is exactly one good value of $b$ and exactly one good value of $b'$, namely, $b = a \oplus i$ and $b' = a' \oplus i$. Thus, there are $m^2$ good values for $a, b, a', b'$. We conclude that the probability that $a, b, a', b'$ are good is $m^2/m^4 = 1/m^2$.

- Suppose $x = x'$ and $y \neq y'$. Then $a = a'$, so there are only $m^3$ possible values for $a, b, a', b'$. If we fix $a = a'$ arbitrarily, there is exactly one good value of $b$ and exactly one good value of $b'$, namely, $b = a \oplus i$ and $b' = a' \oplus i$. Thus, there are $m$ good values of $a, b, a', b'$. We conclude that the probability that $a, b, a', b'$ are good is $m/m^3 = 1/m^2$.

- The final case $x \neq x'$ and $y = y'$ is symmetric with the previous case. ■

**Rubric:** This is more detail than necessary for full credit. This is not the only correct solution. “See part (b)” is worth exactly the same number of points as your solution to part (b).

(b) Prove that $\mathcal{H}$ is 3-uniform. [Hint: Solve part (a) first.]

**Solution:** Let $(x, y)$, $(x', y')$, $(x'', y'')$ be arbitrary distinct elements of $U$, and let $i, j, k$ be arbitrary (possibly equal) hash values. To simplify notation, we define

$$a = A[x], \quad b = B[y], \quad a' = A[x'], \quad b' = B[y'], \quad a'' = A[x''], \quad b'' = B[y''].$$

Say that $a, b, a', b', a'', b''$ are **good** if $a \oplus b = i$ and $a' \oplus b' = j$ and $a'' \oplus b'' = k$. There are three cases to consider.
• Suppose $x, x', x''$ are all different. Arbitrarily fix $y, y', y''$. There are $m^3$ possible values for $x, x', x''$, but only one good value: $x = y \oplus i$ and $x' = y' \oplus j$ and $x'' = y'' \oplus k$.
• If $x = x' = x''$, then $y, y', y''$ must be all different, and we can argue exactly as in the previous case.
• The only remaining case (up to symmetry) is $x = x' = x''$ and $y \neq y' = y''$. Then there are $m^4$ possible values for $a, b, b', a''$. If we fix $a$ arbitrarily, the only good values of the remaining variables are $b = a \oplus i$ and $b' = a \oplus j$ and $a'' = b' \oplus k = a \oplus j \oplus k$. Thus, there are exactly $m$ good values for $a, b, b', a''$.

In all cases, we conclude that $\Pr[a, b, a', b', a'', b'' \text{ are good}] = 1/m^3$.

**Rubric:** This is not the only correct solution.

(c) Prove that $\mathcal{H}$ is not 4-uniform.

**Solution:** For any function $h \in \mathcal{H}$ and any $w$-bit strings $x, y, x', y'$, we have

\[
h(x, y) \oplus h(x', y) \oplus h(x, y') \oplus h(x', y')
\]
\[
= A[x] \oplus B[y] \oplus A[x'] \oplus B[y'] \oplus A[x] \oplus B[y] \oplus A[x'] \oplus B[y']
\]
\[
= A[x] \oplus A[x'] \oplus A[x'] \oplus A[x] \oplus B[y] \oplus B[y'] \oplus B[y] \oplus B[y']
\]
\[
= 0.
\]

It follows that for any hash values $i, j, k, l$, the probability

\[
\Pr[h(x, y) = i \land h(x, y') = j \land h(x', y) = k \land h(x', y') = l]
\]

is an integer multiple of $1/m^3$, and therefore cannot equal $1/m^4$.

**Rubric:** This is more detail than necessary for full credit. This is not the only correct solution.

**Solution:** Jeff proved this in class.

**Rubric:** Yes, this is worth full credit.