Q1. (a) Let X_j be a random variable representing the total number of bees that land at the j-th flower at round i. Since X_j models the number of successful bees that land in the j-th flower with probability $1/n$, then clearly $X_j \sim \text{Bin}(b, 1/n)$. Correspondingly, for a fixed flower j, the probability that such flower gets pollinated by exactly one bee at round i is:

$$\Pr[X_j = 1] = \binom{b}{1}\left(\frac{1}{n}\right)\left(1-\frac{1}{n}\right)^{b-1} = \frac{b}{n}\left(1-\frac{1}{n}\right)^{b-1}.$$

Now, if we let X model the total number of flowers which during round i got pollinated by exactly one bee, then clearly $X \sim \text{Bin}\left[n, \frac{b}{n}\left(1-\frac{1}{n}\right)^{b-1}\right]$. Recall that if X is binomial with probabily p then its expectation is np where n is the number of trials we run the experiment for. Thus, the expected number of flowers that get pollinated by exactly one bee during round i is:

$$\text{E}[X] = n \cdot \frac{b}{n}\left(1-\frac{1}{n}\right)^{b-1} = b\left(1-\frac{1}{n}\right)^{b-1}.$$

Notice, this result also captures the expected number of bees that successfully collect nectar during round i and fly out of round $i+1$. So if we define Y to be a random variable associated with the total number of bees remaining at the start of round $i+1$, then its expectation will be given as:

$$\text{E}[Y] = \text{E}[b - X] = \text{E}[b] - \text{E}[X] = b - b\left(1-\frac{1}{n}\right)^{b-1}.$$

(b) Notice from the exact formula in part (a) that a constant number of rounds, say k, suffices to reduce n bumblebees to $n/2$ bumblebees. Now, by recursively expanding the inequality $X_{t+1} \leq X_t^2/n$, we see that:

$$X_{k+t} \leq X_k^{2^t}/n^{2^t-1} \leq (n/2)^{2^t}/n^{2^t-1} \leq n/2^{2^t}.$$

This value is 1 precisely when $t = O(\log\log n)$. Since k is a constant, the total number of rounds is $k + t = O(\log\log n)$ as required!

Q2. (a) Let X be a random variable representing the total number of pairs (i, j) with $i < j$ such that guest i and the guest j are wearing matching masks. Then we define X_{ij} to be an indicator random variable associated with one of the single instances in which guest i and a guest j are wearing matching masks. By linearity of expectations, we have:

$$\text{E}[X] = \text{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\text{E}\left[X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\Pr\left[X_{ij}=1\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{m} = \frac{n(n-1)}{2m}.$$

When $\text{E}[X] = 1$, we have $n(n-1) = 2m$, or equivalently, $n^2 - n - 2m = 0$. Solving for n, we find that $n = \dfrac{1+\sqrt{1+8m}}{2}$.

(b) Let X be a random variable representing the total number of house pairs (i, j) with $i < j$ that received the same template design during the fundraising. Following the same strategic design as in part (a), we define an indicator random variable X_{ij} associated with the event in which houses i and j receive the same template design. Then, the $\text{E}[X] = \frac{200\times199}{2{,}000{,}000} = 0.0199$. Since $X \geq 0$, we use Markov's inequality to get the $\Pr[X \geq a] \leq \frac{1}{a}\text{E}[X]$. Setting $a = 1$, we get $\Pr[X \geq 1] \leq 0.0199$.

Q3. (a) *Algorithm Description*

We proceed with a "Divide and Conquer" strategy augmented with randomized quick select. Specifically, using quick select, we pick an element x from A with rank $k_{\lfloor m/2 \rfloor}$, and partition A

around that element. Since $k_{\lfloor m/2 \rfloor}$ is the rank of x, all the elements with ranks less than $k_{\lfloor m/2 \rfloor}$ will be to the left of x, and all the elements with ranks greater than $k_{\lfloor m/2 \rfloor}$ will be to the right of x. But this problem is structurally similar to the initial problem, so we invoke two additional recursive calls—one on the left and one on the right—to conquer these two subproblems with recursion. We stop when the set of ranks K becomes of size one, in which case we have the canonical selection problem which can be easily tackled with quick select. The pseudocode for this is given below.

(b) *Algorithm Pseudocode*

Algorithm 1 MULTI-QUICK-SELECT(A, K) // generalized selection problem

Input: An (unsorted) list of n distinct numbers, $A[1 \cdots n]$, together with a monotonically increasing sequence of m distinct sorted ranks $K[1 \cdots m]$.

Output: The k-th smallest number in A for all $k \in K$.

1: **if** $|K| = 1$ **then** // base case
2: $x \leftarrow$ QUICK-SELECT$(A, K[1])$ // randomized quick-select
3: PRINT(x)
4: **else**
5: $i \leftarrow \lfloor |K|/2 \rfloor$
6: $x \leftarrow$ QUICK-SELECT$(A, K[i])$
7: $A_L \leftarrow \{a \in A \mid a \leq x\}$ // aggregate all entries of $A \leq x$
8: $A_R \leftarrow \{a \in A \mid a > x\}$ // aggregate all entries of $A > x$
9: $r \leftarrow K[i]$ // define r to be the rank of x in A
10: $K_L \leftarrow \{k \in K \mid k \leq r\}$ // aggregate all entries of $K \leq r$
11: $K_R \leftarrow \{k - r \mid k \in K \wedge k > r\}$ // $\forall$ entries of $K > r$ aggregate their relative ranks $w.r.t.$ r
12: MULTI-QUICK-SELECT(A_L, K_L) // conquer the left (A_L, K_L) with recursion
13: MULTI-QUICK-SELECT(A_R, K_R) // conquer the right (A_R, K_R) with recursion
14: **end if**

(c) *Algorithm Correctness*

To demonstrate the correctness of Algorithm 1, we will assume a correct implementation of QUICK-SELECT and proceeed with (strong) induction on m—the number of elements in K—also denoted by $|K|$.

- *Base Case*: when $|K| = 1$ and K is a singleton containing only one element, Algorithm 1 degenerates to a canonical selection problem which is solved by the QUICK-SELECT call in Step 2. Since by hypothesis QUICK-SELECT is correct—in that it reports the k-th smallest element of A for the singular value of $k \in K$—it must follow that Algorithm 1 is also correct.

- *Induction Hypothesis*: assume Algorithm 1 correctly reports the k-th smallest number of A for every $k \in K$ and $2 \leq |K| < m$.

- *Inductive Step*: now consider when $|K| = m$. Steps 10 and 11 of Algorithm 1 partitions K into two (disjoint) sublists of nearly equal sizes. Let K_L and K_R be these lists. Clearly, $|K_L| < m$ and $|K_R| < m$. Thus applying the induction hypothesis, we are guaranteed that the two MULTI-QUICK-SELECT calls in Steps 12 and 13 of Algorithm 1 correctly report the k-th smallest number of A_L (for every $k \in K_L$) and the k-th smallest number of A_R (for every $k \in K_R$). Therefore, a total of $|K_L| + |K_R| = |K| = m$ elements of K are correctly reported, which is exactly what we wanted to show! □

(d) *Time Analysis*

Let $T_M(n, m)$ be the total running time incurred by Algorithm 1 on arbitrary inputs of size n and m. We define $T_Q(n)$ to be a random variable representing the expected running time of QUICK-

SELECT on an arbitrary, unsorted list of n distinct elements. Notice that due to the degeneracy of the problem at the base case, we have that $T_M(n, m) = T_Q(n)$ for $m = 1$. For the recursion, A_L is a list of size n_1 and A_R is a list of size n_2 where $n_1 + n_2 = n$. Furthermore, the QUICK-SELECT call in Step 6 induces on the fly the construction of A_L and A_R so there is not need to count the assignment calls of Steps 7 and 8 to the total running time. Additionally, per the problem definition, the entries of K are all distinct integers ranging from 1 to n so $|K| = m \leq n$. Thus constructing K_L and K_R in Steps 10 and 11 incurs at most $O(n)$ cost to the total running time. Stitching all these together, we have:

$$T_M(n, m) = \begin{cases} T_Q(n) & \text{if } m = 1 \\ T_M(n_1, \lfloor m/2 \rfloor) + T_M(n_2, \lceil m/2 \rceil) + T_Q(n) + n & \text{otherwise} \end{cases}$$

Following a similar probabilistic analysis to the randomized selection problem discussed in class, we get that $T_Q(n)$ solves to $O(n)$ on "average." Hence the recurrence becomes:

$$T_M(n, m) = \begin{cases} n & \text{if } m = 1 \\ T_M(n_1, \lfloor m/2 \rfloor) + T_M(n_2, \lceil m/2 \rceil) + n & \text{otherwise} \end{cases}$$

To solve this recurrence, we proceed with the recursion tree approach. For simplicity assume m is a power of 2. Then the recursion tree is a complete binary tree with m nodes at the leaves and depth $\log m$. The work at the root node is n. At the next level is $n_1 + n_2$ which is n. Each level incurs an $O(n)$ cost and there are $\log m$ levels hence the total work is $O(n \log m)$. □

Q4. (a) *Algorithm Description*

We employ a similar strategy to that of the selection problem discussed in class, but instead of looking at ranks we look at sums. Specifically, we partition A around a pivot, say x, and compute the sum of the entries that are less or equal to x. Then we have two possible scenarios at hand: (1) sum is greater than M, in which case we recurse on the left; and (2) sum is less or equal to M, in which case, we offset M by that sum and recurse on the right (see. Algorithm 2 for details).

(b) *Algorithm Pseudocode*

Algorithm 2 MAXIMAL-ELEMENT(A, M)

Input: An (unsorted) list of n distinct numbers, $A[1 \cdots n]$, together with a number M.
Output: The maximal element x of A such that the summation $\sum_{(a \in A) \wedge (a < x)} a \leq M$.

```
1: if |A| = 1 then
2:     return A[1]
3: end if
4:
5: x ← CENTRAL-PIVOT(A)
6: A_L ← {a ∈ A | a ≤ x}
7: A_R ← {a ∈ A | a > x}
8: S_L ← Σ_{a ∈ A_L} a
9:
10: if M < S_L then
11:     return MAXIMAL-ELEMENT(A_L, M)
12: else
13:     return MAXIMAL-ELEMENT(A_R, M − S_L)
14: end if
```

(c) *Algorithm Correctness*

To show the correctness of Algorithm 2 we will assume a correct implementation of CENTRAL-PIVOT and proceed with (strong) induction on n—the number of elements in A— also denoted by $|A|$.

- *Base Case*: when $|A| = 1$, we have the trivial case where the maximal element of A is none other than its singular entry $A[1]$. Furthermore, if A contains only one element (i.e. $|A| = 1$) then Step 2 of Algorithm 2 gets executed—reporting $A[1]$ as its output. It follows then that Algorithm 2 correctly executes as intended and produces the desired, expected output when $|A| = 1$.

- *Induction Hypothesis*: assume Algorithm 2 produces the correct maximal element of A whose sum of the elements less than it does not exceed M. For the purposes of induction, assume this hypothesis is true whenever $|A| \leq k$, for every $k = 2, 3, \ldots, n-1$.

- *Inductive Step*: now consider when $|A| = n$. Notice that the CENTRAL-PIVOT subroutine in Step 5 induces a partition on A (A_L and A_R) such that $n/4 \leq \min(|A_L|, |A_R|) \leq \max(|A_L|, |A_R|) \leq 3n/4$. Without loss of generality, let $|A_L| = n/4$ and $|A_R| = 3n/4$. Since A_L and A_R are disjoint, we have two execution paths: (1) the target, maximal element lies in A_L; (2) the target, maximal element lies in A_R.

  Let's first analyze execution path (1). When the target, maximal element lies in A_L, the sum of the elements of A_L is greater M, and we should recurse on A_L. (Notice Step 11 of Algorithm 2 does exactly that!) Now, since $|A_L| = n/4 < n$, we can invoke the induction hypothesis to claim that Algorithm 2 produces the correct maximal element of A on execution path (1).

  Similarly, when the target, maximal element lies in A_R, it must be the case that the sum of the elements of A_L is at most M. In this case, we should recurse on A_R and also offset M by the sum of all elements of A_L since they all are strictly less than the entries of A_R. (Notice Step 13 of Algorithm 2 does exactly that!) Now, since $|A_R| = 3n/4 < n$, we can invoke the induction hypothesis to claim that Algorithm 2 produces the desired, expected output on execution path (2).

  So irrespective of the execution path, Algorithm 2 always outputs the correct answer, when $|A| = n$. □

(d) *Time Analysis*

Let $T_M(n)$ represent the total running time of Algorithm 2 on an input of size n. We define $T_C(n)$ to be a random variable representing the expected running time of the CENTRAL-PIVOT subroutine on an arbitrary, unsorted list of n distinct elements. Then at the base case we have $T_M(n) = 1$ for $n = 1$. For the recurssion, we have one subproblem of size of $\max(|A_L|, |A_R|) \leq 3n/4$ followed by the cost of the CENTRAL-PIVOT subroutine ($T_C(n)$) and the cost of constructing A_L, A_R, and S in Steps 6, 7, and 8—which is $O(n)$. Then, the recurrence becomes:

$$T_M(n) = \begin{cases} 1 & \text{if } n = 1 \\ T_M(3n/4) + T_C(n) + n & \text{otherwise} \end{cases}$$

As discussed in class, $T_C(n) \sim \text{Geom}(1/2)$ with expectation $1/p = 2 = O(1)$. Substituting this result into the expression above, we get: $T_M(1) = 1$ and $T_M(n) = T_M(3n/4) + n$. With $a = 1, b = 4/3$, $f(n) = n$, and $c = 3/4$, this recurrence solves to $\Theta(n)$, according to case 3 of the Master Theorem discussed in class. □