

1. Consider again the hockey league we saw in Problem Set 3. Let's say that we have reached a point in the season where team  $T_1$  has no games left, each of the other  $n - 1$  teams has  $4n - 8$  games left, with exactly 4 games against each of the other teams. Furthermore, let's assume that team  $T_1$  has built up a lead such that it will have more wins at the end of the season than any other team, provided that no other team wins at least  $3n$  of its remaining games. We want to derive a bound on the probability that team  $T_1$  has the most wins at the end of the season. We shall assume that in each game, both teams have an equal probability of winning, and that the outcomes of all games are independent.
  - (a) Use Chebyshev's inequality to show that the probability that  $T_1$  has the most wins at the end of the season is at least  $1/2$ .
  - (b) Derive the largest bound you can on the same probability using a Chernoff bound. How do the two bounds compare (for large  $n$ )?
  
2. Consider a parallel computer that consists of  $n$  processors and  $n$  memory modules. During a step of computation, each processor chooses a memory module independently and uniformly at random, and sends a memory request to that module.
  - (a) During a step of computation, what is the expected number of memory modules that do not receive any requests?
  - (b) If a memory module that receives one request in a round services that request, but memory modules that receive more than one request discard all their requests, what is the expected number of processors whose requests are satisfied?
  - (c) If, on the other hand, in every round a memory module randomly chooses one of its requests to service and discards the other requests, what is the expected number of processors whose requests are satisfied?

On this question, use the approximation  $(1 - 1/k)^k \approx 1/e$ , if necessary.
  
3. Say we are given a binary matrix with 2 rows and  $N$  columns, and we want to determine  $\hat{N}$ , the number of columns that contain a 1 in at least one of the rows. We want to come up with an algorithm that is fast even when  $N$  is very large (and thus do not want to look at every column of the matrix). To do so, we shall use randomization, and instead of finding the exact answer, we shall find an estimate to the actual answer. We here assume that we can access any entry of the matrix in  $O(1)$  time, and also that for each row of the matrix, we can choose an entry in that row that contains a 1 uniformly at random from the set of those entries.
  - (a) Consider the following algorithm:

Let  $i = 0$

Repeat  $k$  times:

Pick a column uniformly at random.

If that column contains a 1,  $i = i + 1$ .

Return  $A = iN/k$ .

Show that for any  $k \geq 1$ ,  $E[A] = \hat{N}$ . Use a Chernoff bound to derive a bound on how large  $k$  must be so that for the worst case input matrix,  $\Pr[|A - \hat{N}| \geq \frac{\hat{N}}{2}] \leq \Delta$ ? Here,  $\Delta$  is any real number such that  $0 < \Delta < 1$ . Note that since we are looking at the worst case input, your answer should not be a function of  $\hat{N}$ . Compare the performance of this algorithm to simply looking at every column of the matrix.

(b) Say that we know values  $N_1$  and  $N_2$ , such that the first row contains  $N_1$  1's and the second row contains  $N_2$  1's. Now, consider the following algorithm:

Let  $i = 0$

Repeat  $k'$  times:

Flip a coin that is heads with probability  $\frac{N_2}{N_1 + N_2}$ .

If the coin came up heads,  $i = i + 1$ .

If the coin came up tails, pick an entry containing a 1 uniformly at random from the first row. If the column containing that entry does NOT have a 1 in the second row,  $i = i + 1$ .

Return  $A' = i(N_1 + N_2)/k'$ .

Show that for any  $k' \geq 1$ ,  $E[A'] = \hat{N}$ . Again, use a Chernoff bound to derive a bound on how large  $k'$  must be so that  $\Pr[|A' - \hat{N}| \geq \frac{\hat{N}}{2}] \geq \Delta$ ? Again, your answer should be for the worst case input of a matrix with  $N$  columns, and thus should not be a function of  $\hat{N}$ ,  $N_1$ , or  $N_2$ . Compare the performance of this algorithm to simply checking every column of the matrix.

(c) Say we are given a binary matrix with 3 rows and  $N$  columns, and we want to determine the number of columns that contain a 1 in at least one of the rows. Derive an efficient algorithm for this problem. The performance of this algorithm should be roughly the same as the faster of the two algorithms described in part (a) and part (b). If necessary, you can assume that we know the number of 1's in each of the rows.