Randomness and Computation 2018/19
Week 6 tutorial sheet (12-1pm, Tues 26th, Wed 27th February)

1. Recall the Coupon Collector problem where we repeatedly draw random cards uniformly at random from a sample pool of of n footballers, in the "with replacement" setting. We have previously used our knowledge of geometric r.v.s to show that the expected number of purchases to acquire all cards is $n \cdot H(n) \sim n \cdot \ln(n)$. Then we analysed the variance of the process and used Chebyshev's Inequality to show that it sufficed to buy $c \cdot n \cdot H(n)$ packets to have probability $(1 - \frac{\pi^2}{6 \cdot c^2 \ln(n)^2})$ of obtaining all cards.

   We now show how to obtain similar results without considering the geometric distribution. We will apply Chernoff bounds as part of this different approach.

   (a) Suppose we consider a specific footballer card of interest, and for $j = 1, \ldots,$ define the Bernoulli variable Z_j (with probability n^{-1}) to be 1 if we draw that footballer on the j-th purchase, 0 otherwise. Let $Z = \sum_{j=1}^{m} Z_j$ be the number of times we draw that footballer over m purchases. Clearly $E[Z] = m/n$.

   Use a one-sided Chernoff bound to show that if we have a number of samples slightly bigger than $3n \cdot \ln(n)$, more precisely $m \geq 3n(\ln(n) + \ln(k))$, this is enough to have $\Pr[Z < 1] \leq n^{-1}k^{-1}$ for sufficiently large n ($n \geq 8$ in this case).

   better: Can you show this for $m \geq n(\ln(n) + \ln(k))$?

   (b) Now apply the Union bound to show that for the same number m of random purchases, that the probability of collecting all footballers is at least $1 - k^{-1}$.

   (c) Compare and contrast your results with the analysis in the slides for lectures 4, 5 (using geometric random variables and Chebyshev).

2. Consider a specialised sorting problem where we know the items to be sorted are natural numbers from some bounded range $[0, 2^k)$, some large k.

   We are going to perform a "bucket sort", using a collection of initially-empty "buckets" (extendable arrays or lists). The buckets are defined wrt "short" binary numbers of length m, this being the "number of prefix bits" (substantially smaller than k). We have a bucket for each individual $\{0, 1\}^m$. The idea is to first do a linear scan of the inputs to be sorted, using their m-bit prefix to throw them into the correct bin. Later the individual bins are sorted using a standard sorting algorithm of (at most) quadratic running-time.

   Algorithm BucketSort$(a_1, \ldots, a_n)$

   (a) Do a linear scan of the inputs, adding a_i to the bucket matching its first m bits.

   (b) **for** every $b \in \{0, 1\}^m$ **do**

   (c) Sort bucket b with any $O(n^2)$ sorting algorithm.

Show that if we choose the prefix-length so that $m \geq \lg(n)$, then the expected running-time of BUCKETSORT is linear in n.

3. (a coursework 1 question) Consider a function $F : \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, m-1\}$ and suppose we know that for $0 \leq x, y \leq n-1$, $F((x+y) \bmod n) = (F(x) + F(y)) \bmod m$. The only way we know to evaluate $F(\cdot)$ is to examine the values in an array where the $F(\cdot)$ values have been stored (with entry i holding the value of $F(i)$). Unfortunately, a system failure has corrupted up to a $1/5$-fraction of the entries of the array, so we no longer have reliable values in all positions.

   Describe a simple randomized algorithm that, given an input $z \in \{0, \ldots, n-1\}$, outputs a value that equals $F(z)$ with probability at least $1/2$. Your algorithm should guarantee this $1/2$ probability of being correct for every value of z, regardless of which specific array entries were corrupted. Your algorithm should use as few lookups and as little computation as possible. Justify the $1/2$ correctness guarantee.

   Suppose you are allowed to repeat your initial algorithm three times before you return a result. What should you do in this case? Justify your answer.

4. (a coursework 1 question) Recall our analysis of the simple "Max-Cut" (or $\frac{|E|}{2}$-cut) algorithm in Lecture 6, and remember we chose to place each $v \in V$ into S or $V \setminus S$ with even (and independent) probabilities $1/2$; recall also that this generation of S could be considered as choosing a random subset of V (with every individual subset having the probability 2^{-n}, regardless of its size). We showed that when we generated S this way, the expected size of the cut between $(S, V \setminus S)$ was exactly $\frac{|E|}{2}$.

   Come up with a different algorithm to generate $(S, V \setminus S)$ in such a way that the expected size of the cut will be the slightly larger value $|E|\frac{|V|}{2|V|-1}$ (hence showing that there is at least one cut of this size).

   Note - there will be two slightly different cases, for odd n and even n, and the factors for these will be different (but at least $|E|\frac{|V|}{2|V|-1}$ in each case).

5. In Lecture 6 we saw how to "derandomize" our initial Max-Cut algorithm to get a deterministic algorithm/method which is *guaranteed* to return a solution at least as good as $\frac{|E|}{2}$.

   Show how to derandomize your improved algorithm of Question 4 above. Your algorithm should be low polynomial-time (something like $O(n^2)$ or $O(n^3)$). Justify the fact that your method will *definitely* return a cut with at least $\frac{n}{2n-1}m$ edges, using appropriate reference to conditional expectations.

   (This is hard! You will need to do something more interesting than with the algorithm from lecture 6)

Mary Cryan, 19th February