

**Homework 4**

Out: 8 Feb. Due: 20 Feb.

*Instructions: Put your solutions in the homework box on Soda level 2 by 5pm on **Tuesday** (note the non-standard day!). Take time to write clear and concise answers; confused and long-winded solutions will be penalized. You are encouraged to form small groups (two to four people) to work through the homework, but you **must** write up all your solutions on your own. Depending on grading resources, we reserve the right to grade a random subset of the problems and check off the rest; so you are advised to attempt all problems.*

1. Recall the randomized algorithm discussed in class for finding the median of a set  $S$  of  $n$  elements (see MU, Section 3.4). Explain how to generalize this algorithm so that it takes an additional parameter  $k$  as input and finds the element of rank  $k$  in  $S$  (i.e., the  $k$ th smallest element of  $S$ ). For convenience, you may assume that all elements of  $S$  are distinct, and that  $4n^{3/4} \leq k \leq n - 4n^{3/4}$ . You may also ignore rounding issues in your algorithm.

Your answer should be *fairly short* and you should *not* repeat unnecessary material from the text. The only things you need to include are the following:

- (i) a *specification* of the algorithm (in similar style to that in class and in the textbook); follow the same first step (namely, pick  $R$  of size  $n^{3/4}$ ); highlight those points where your algorithm differs from the median case;
- (ii) a *very brief outline* of the analysis of the algorithm, following the same path as in class and in the textbook; specifically, state the analogs of the events  $\mathcal{E}_1$ ,  $\mathcal{E}_2$ ,  $\mathcal{E}_3^1$  and  $\mathcal{E}_3^2$ , and express  $\Pr[\mathcal{E}_1]$  and  $\Pr[\mathcal{E}_3^1]$  as probability expressions involving the tail of a suitable binomial r.v. Do *not* repeat the explanation of the algorithm or the details of the calculations (which are essentially the same as in the median case); the above points are enough.

2. Let  $X_1, \dots, X_n$  be independent 0-1 valued random variables with  $\Pr[X_i = 1] = 1/i$ . Define the r.v.  $X$  to be their sum  $X_1 + \dots + X_n$ .

- (a) Write down expressions for  $E[X]$  and  $\text{Var}[X]$ .
- (b) Let  $p$  be the probability of the event  $X \geq 4 \ln n$ . Compare the upper bounds you can obtain on  $p$  using (i) Markov's inequality; (ii) Chebyshev's inequality; and (iii) Chernoff bounds. [NOTES: In your comparison, you should concentrate on the asymptotic behavior of the bounds as  $n \rightarrow \infty$ . Recall that the harmonic number  $H_n \sim \ln n$  as  $n \rightarrow \infty$ . For (iii) you will need the following form of the Chernoff bound which is an extension of the bound (4.2) in Theorem 4.4 of MU and is valid for all  $\delta > 0$ : with the notation of that theorem,  $\Pr[X \geq (1 + \delta)\mu] \leq e^{-\mu\delta^2/(2+\delta)}$ .]

3. [MU, Ex. 4.10] A casino is testing a new class of simple slot machines. In each game, the player puts in \$1 and the slot machine is supposed to return either \$3 to the player with probability  $4/25$ , \$100 with probability  $1/200$ , or nothing with the remaining probability. Each game is independent of all other games. The casino has been surprised to find in testing that the machines have lost \$10,000 over the first million games. Your task is to come up with an upper bound on the probability of this event, assuming that their machines are working as specified.

- (a) Let the r.v.  $X$  denote the *net loss* to the casino over the first million games. By writing  $X = X_1 + \dots + X_{10^6}$ , derive an expression for  $E[e^{tX}]$ , where  $t$  is an arbitrary real number.
- (b) Derive from first principles a Chernoff bound for the probability  $\Pr[X \geq 10,000]$ . [HINT: You should follow the proof of the Chernoff bound in class, by applying Markov's inequality to the r.v.  $e^{tX}$ . You should use the value  $t = 0.0006$  in your bound.]

**[Turn over for problem 4]**

4. A fundamental problem that arises in many applications is to compute the size of the *union* of a collection of sets. The setting is the following. We are given  $m$  sets  $S_1, \dots, S_m$  over a very large universe  $U$ . The operations we can perform on the sets are the following:

- $\text{size}(S_i)$ : returns the number of elements in  $S_i$ ;
- $\text{select}(S_i)$ : returns an element of  $S_i$  chosen u.a.r.;
- $\text{lowest}(x)$ : for  $x \in U$ , returns the smallest index  $i$  for which  $x \in S_i$ .

Let  $S = \bigcup_{i=1}^m S_i$  be the union of the sets  $S_i$ . In this problem we will develop a very efficient (polynomial in  $m$ ) algorithm for estimating the size  $|S|$ .

- (a) Let's first see a natural example where such a set system arises. Suppose  $\phi$  is a boolean formula in *disjunctive normal form* (DNF), i.e., it is the OR of ANDs of literals. Let  $U$  be the set of all possible assignments to the variables of  $\phi$  (i.e.,  $|U| = 2^n$  where  $n$  is the number of variables), and for each clause  $1 \leq i \leq m$  let  $S_i$  be the set of assignments that satisfy clause  $i$ . Then the union  $S = \bigcup_{i=1}^m S_i$  is exactly the set of satisfying assignments of  $\phi$ , and our problem is to count them.<sup>1</sup> Argue that all of the above operations can be efficiently implemented for this set system.
- (b) Now let's consider a naive random sampling algorithm. Assume that we are able to pick an element of  $U$  u.a.r., and that we know the size of  $U$ . Consider the algorithm that picks  $t$  elements of  $U$  independently and u.a.r. (with replacement), and outputs the value  $q|U|$ , where  $q$  is the proportion of the  $t$  sampled elements that belong to  $S$ . For the DNF example in part (a), explain as precisely as you can why this is not a good algorithm.
- (c) Consider now the following algorithm, which is again based on random sampling but in a more sophisticated way:

- choose a random set  $S_i$  with probability  $\frac{\text{size}(S_i)}{\sum_j \text{size}(S_j)}$
- $x = \text{select}(S_i)$
- if  $\text{lowest}(x) = i$  then output 1 else output 0

Show that this algorithm outputs 1 with probability exactly  $p = \frac{|S|}{\sum_j |S_j|}$ . [HINT: Show that the effect of the first two lines of the algorithm is to select a random element of the set of pairs  $\{(x, S_i) : x \in S_i\}$ .]

- (d) Show that  $p \geq \frac{1}{m}$ .
- (e) Now suppose that we run the above algorithm  $t$  times and obtain the sequence of outputs  $X_1, \dots, X_t$ . We define  $X = \sum_{i=1}^t X_i$ . Use a Chernoff bound to obtain a value for  $t$  (as a function of  $m$ ,  $\delta$  and  $\epsilon$ ) that ensures that

$$\Pr[|X - tp| \geq \epsilon tp] \leq \delta.$$

[HINT: You will need to use the bound from part (d) here.]

- (f) The final output for our algorithm will be  $Y = (\sum_j |S_j|) \cdot \frac{X}{t}$ , where  $X$  is as defined in part (e). Using part (e), show that this final algorithm has the following properties: it runs in time  $O(m\epsilon^{-2} \log(\delta^{-1}))$  (assuming that each of the set operations listed above can be performed in constant time), and outputs a value that is in the range  $[(1 - \epsilon)|S|, (1 + \epsilon)|S|]$  with probability at least  $1 - \delta$ .

---

<sup>1</sup>Note that deciding if  $\phi$  is satisfiable (i.e., has at least one satisfying assignment) is trivial for a DNF formula, unlike for a CNF formula where it is NP-complete (see CS170 or CS172). However, when it comes to *counting* satisfying assignments, it turns out that the problem is NP-hard even for DNF formulas! Thus we cannot hope to find a polynomial time algorithm that solves this problem exactly. Thus the approximation algorithm that we develop in this question is essentially the best one can hope for.