Graduate Algorithms Homework 2

Karan Sarkar
sarkak2@rpi.edu

September 2, 2020

Problem 1.

(a) The range of S is the set of integers i where $0 \leq i \leq n$. The probability of getting k ones in a specific order is $p^k(1-p)^{n-k}$. We must multiply this by the number of ways to pick k ones from n choices $\binom{n}{k}$. We have that $P(S=k) = \binom{n}{k}p^k(1-p)^{n-k}$. Therefore, $P(S=k) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k}$.

(b) By the linearity of expectation, we have $E[S] = p + p + \cdots + p$. Thus, the result is np.

(c) Because $X \leq Y$, We have $Y - X \geq 0$. Because $E[Z] \geq 0$ if $Z \geq 0$, we have $E[Y - X] \geq 0$. By linearity of expectation we have, $E[Y] - E[X] \geq 0$. Therefore, we have $E[X] \leq E[Y]$.

(d)

$$\begin{aligned} E[X] &= 0 \cdot P(X=0) + 1 \cdot P(X=1) + +2 \cdot P(X=2) + 3 \cdot P(X=3) + \cdots \\ E[X] &\geq P(X=1) + P(X=2) + P(X=3) + \cdots \\ E[X] &\geq P(X>0) \\ P(X>0) &\leq E[X] \end{aligned}$$

(e) Let $1_A(x)$ be the random variable such that $1_A(x) = 1$ if $x \in A$ and $1_A(x) = 0$ if $x \notin A$. Therefore, we have:

$$\begin{aligned} X &= X \cdot 1_{X>a} + X \cdot 1_{X \leq a} \\ E[X] &= E[X \cdot 1_{X>a} + X \cdot 1_{X \leq a}] \\ E[X] &= E[X \cdot 1_{X>a}] + E[X \cdot 1_{X \leq a}] \\ E[X] &\geq E[a1_{X>a}] + E[X \cdot 1_{X \leq a}] \\ E[X] &\geq E[a1_{X>a}] \\ E[X] &\geq aP(X>a) \\ P(x>a) &\leq \frac{E[X]}{a} \end{aligned}$$

Problem 2. First, break up the list into consecutive segments of length 100. Then, we sort each segment exactly. Let us call the sorted segments $s_0, s_1, \ldots s_m$. We now merge together sorted lists s_0 and and s_1 into sorted list t_1. In order to add s_2 we merge together t_1 and s_2 into t_2 except we ignore all but the last 100 elements of t_1. We repeat this process, merging together t_i and s_{i+1} into sorted list t_{i+1}. The last t list is the result.

Our method involves sorting segments of the total list and then merging them together. Had we not made the decision to ignore all but the last hundred elements of the t lists, we would have a clearly correct algorithm. Thus, we will explain why we made that decision. s_0 contains indices from 0 to 99. s_1 contains indices from 100 to 199. When we merge them together we have the 200-element list t_1. Because each index can only be displaced by 100, the elements that end up at indices 0-99 after sorting could only have come from s_0 and s_1. This means that we do not have to consider any other segments. Thus, we can safely only bother merging the last 100 elements of t_1 with s_2. This logic repeats for each merging.

Sorting each 100 element segment takes $O(1)$ time because 100 is constant. Thus sorting all segments is $O(n)$. Moreover, each merge is done between two 100-element lists i.e. the back 100 of t and the s. Because each merge is the same constant size, each merge takes $O(1)$ time. Thus, all merging takes $O(n)$ time. Therefore, the algorithm as a hole is $O(n)$.

Problem 3. Suppose A contains $a_1, a_2, \ldots a_n$ and B contains $b_1, b_2, \ldots, b_n$. We can create polynomial $A(x) = x^{a_1} + x^{a_2} + \cdots + x^{a_n}$ and $B(x) = x^{b_1} + x^{b_2} + \cdots x^{b_n}$. We can get the result by multiplying the polynomials. Term kx^c represents that the sum c is calculated k times. This works because every product term adds together the powers from the $A(x)$ and $B(x)$ polynomials. We input $A(x)$ and $B(x)$ into the FFT polynomial multiplication algorithm The run time is therefore by FFT, $O(10n \log(10n) = O(10n \log n) + O(10log(10)) = O(n \log n)$. Thus using FFT, we get an $O(n \log n)$ algorithm.