Randomized Algorithms
Tutorial Sheet 3

1. (a) For two independent random variables $X$ and $Y$, show that $\text{Var}[X - Y] = \text{Var}[X] + \text{Var}[Y]$.
   (b) For a geometric random variable $X$ with parameter $p$, calculate $\text{E}[X^3]$.

2. (This question is based on Section 5.2.2 in the book. Please attempt this question before reading that section, if you haven’t already done so.) Consider a specialised sorting problem where we know the items to be sorted are natural numbers from some bounded range $[0, 2^k)$, for some $k \geq 1$. These integers are represented by length $k$ binary strings. We are going to perform a “bucket sort”, using a collection of initially-empty “buckets” (extendable arrays or lists). The buckets are defined with respect to “short” binary numbers of length $m$ (substantially smaller than $k$), this being the “number of prefix bits”. We have a bucket for each individual bit string $b \in \{0, 1\}^m$. 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 bucket. Later the individual buckets are sorted using a standard sorting algorithm of (at most) quadratic running-time (such as Bubblesort or Insertion sort).

Algorithm $\text{BucketSort}(a_1, \ldots, a_n)$

(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 sorting algorithm that runs in quadratic time.

We want to analyze the running time of $\text{BucketSort}$ for random inputs. The first step takes time linear in the number of buckets, $2^m$, as well as in the number of inputs, $n$. Show that if the inputs are $n$ uniformly at random strings of length $k$, and we choose the prefix-length so that $m = \lceil \log(n) \rceil$, then the expected running-time of $\text{BucketSort}$ is linear in $n$.

3. Consider a function $F : \{0, 1, \ldots, n - 1\} \rightarrow \{0, 1, \ldots, m - 1\}$ and suppose we know that for $0 \leq x, y \leq n - 1$, $F((x + y) \mod n) = (F(x) + F(y)) \mod m$. The only way we have to evaluate $F(\cdot)$ is to examine the values in an array of length $n$ 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 do not 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 $z$, and 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.

Kousha Etessami