Alex Tsun
CSE 312: Foundations of Computing II PSet #3

PSet #3

With problems from several past UW CSE 312 instructors (Martin Tompa, Anna Karlin, Larry Ruzzo) and Stanford CS 109 instructors (Chris Piech, David Varodayan, Lisa Yan, Mehran Sahami)

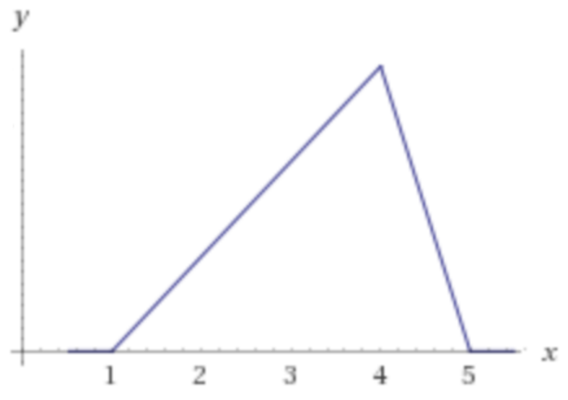
Groups: This pset may be done in groups of **up to 2 people**. This means that if you work with a partner, only one person will submit on Gradescope to “PSet 3 [Written]” and add their partner as a collaborator. For this pset, you may *also work on the [Coding] part in your pair*. The pair must be the same pair as the [Written]. Individuals and groups are encouraged to discuss problem-solving strategies with other classmates as well as the course staff, but each group must write up their own solutions.

Instructions: For each problem, remember you must briefly explain/justify how you obtained your answer, as correct answers without an explanation will receive **no credit**. Moreover, in the event of an incorrect answer, we can still try to give you partial credit based on the explanation you provide. It is fine for your answers to include summations, products, factorials, exponentials, or combinations; you dont need to calculate those all out to get a single numeric answer.

Submission: You must upload your written compiled LaTeX PDF to Gradescope under “PSet 3 [Written]” and your two code files `bloom.py` and `dist_elts.py` to “PSet3 [Coding]”. You must tag your written problems on Gradescope, or you will receive **no credit** as mentioned in the syllabus. Please cite any collaboration at the top of your submission (beyond your group members, which should already be listed).

1. Alex wants to be famous, so he came up with this new “Triangle Distribution”. For real numbers $a < b < c$, he says a (continuous) random variable $T \sim Triangle(a, b, c)$ if it has range $[a, c]$, with a peak at b. Suppose $X \sim Triangle(1, 4, 5)$ (see figure below).

Figure 1: The PDF of $X \sim Triangle(1, 4, 5)$



(a) Find the probability density function $f_X(x)$ (for $X \sim Triangle(1, 4, 5)$). To get full-credit, make sure you define it for all real numbers, by possibly using a piecewise function. (Hint: Start by figuring out what the peak value is at $x = 4$ using geometry.)

(b) Find the cumulative distribution function $F_X(x)$. To get full-credit, make sure you define it for all real numbers, by possibly using a piecewise function. (Hint: Instead of using integrals, use what you know about geometry.)

(c) Compute $E[X]$ and $Var(X)$. Carefully set up the integrals with clearly defined integrands and limits of integration. Then additionally give your answer to 4 decimal places either solving it yourself or using WolframAlpha.

(d) Compute the **median** of X: the median of a distribution is the point $m \in \Omega_X$ where $F_X(m) = 0.5$ (so half the area is to the left, and half the area is to the right). Show your work and give your answer rounded to 4 decimal places.

(e) Compute the **mode** of X. The mode of a random variable is the point $z \in \Omega_X$ with highest probability (if discrete), or the point with highest density (if continuous). Give your answer rounded to 4 decimal places. (Hint: You don't actually have to do any computation for this problem!).

2. A flea of negligible size is trapped in a large, spherical, inflated beach ball with radius r. At this moment, it is equally likely to be at any point within the ball. Let X be the distance of the flea from the center of the ball. Your answers for the following questions will be in terms of r.

   (a) Find the cumulative distribution function $F_X(x)$. To get full-credit, make sure you define it for all real numbers, by possibly using a piecewise function. (Hint: The volume of a sphere with radius t is $\frac{4}{3}\pi t^3$).

   (b) Find the probability density function $f_X(x)$. To get full-credit, make sure you define it for all real numbers, by possibly using a piecewise function.

   (c) Compute $E[X]$ and $Var(X)$. Carefully set up the integrals with clearly defined integrands and limits of integration. Then evaluate your integrals and give both quantities in terms of the radius r.

3. Alex computes final course grades in a large CSE 312 class as arbitrary real numbers, with most of them in the interval [0.0, 4.0]. He computes them so that the students' grades are normally distributed with some mean μ and variance σ^2, parameters that he keeps secret out of fear of a student revolution. If 6.68% of students get a grade of 3.8 or greater and 59.87% of students get a grade less than 2.8, what proportion of students got a grade less than 2.0? Show your work and give your answer rounded to 4 decimal places. (To help you avoid calculation errors, as a check of your calculation we will tell you that μ and σ are each integer multiples of 0.1.)

4. Suppose the time that Java takes to sort a 1,000,000 length array is approximately $J \sim \mathcal{N}(\mu = 46, \sigma^2 = 6^2)$ milliseconds (ms), since it uses the (randomized) QuickSort Algorithm.

   (a) Python initially implements a (deterministic) MergeSort Algorithm, and it always finishes in $P = 49$ ms. What is the probability that Java sorts a single 1,000,000 length array faster than Python does? Show your work and give your answer rounded to 4 decimal places.

(b) Python attempts to implement QuickSort as well, but did it less efficiently. Its runtime is approximately $P \sim \mathcal{N}(\mu = 55, \sigma^2 = 8^2)$. What is the probability that Java sorts a single 1,000,000 length array faster than Python does? Show your work and give your answer rounded to 4 decimal places.

(c) Keep the updated Python runtime assumptions from part (b). We compare Python and Java in a marathon race. We have 9 independent "games", and each game consists of sorting a 1,000,000 length array. The winner is the language who wins in at least 5 of 9 games. What is the probability that Java wins? Show your work and give your answer rounded to 4 decimal places.

(d) Keep the updated Python runtime assumptions from part (b). We compare Python and Java in a marathon race. We have 9 arrays, and we ask Java and Python to sort all 9 arrays, sequentially one at a time (independently). What is the probability that Java sorts 9 arrays before Python does? Show your work and give your answer rounded to 4 decimal places.

5. You have n batteries, each with a lifetime which is (independently) distributed as $Exp(\lambda)$. You have a choice of a weak flashlight, which requires one battery to operate, and a strong flashlight, which requires two batteries to operate. Assume that when a battery dies, you are lightning-quick and replace it with a new battery instantly.

   (a) If you choose to use the weak flashlight, what is the expected amount of time you can operate it for? (Hint: Cite the appropriate distribution, and your solution will be one-line.)

   (b) Recall the memoryless property in lecture 4.2. Suppose $W \sim Exp(\beta)$. Show that you understand what it means by computing $P(W > 17 | W > 10)$ explicitly using this property (do NOT reprove memorylessness).

   (c) For the strong flashlight, we need to compute the distribution of time until the first of the two batteries dies. If $X, Y \sim Exp(\lambda)$, show that the distribution of $Z = \min\{X, Y\}$ is $Exp(2\lambda)$. (Hint: Start by computing $P(Z > z)$, then use this to compute either the CDF or PDF).

   (d) If you choose to use the strong flashlight, what is the expected amount of time you can operate it for? Recall that if you currently have two batteries in the flashlight and one dies, you immediately replace the dead battery with a new one. (Hint: Use your answer to part (c) and what you learned in part (b)).

6. Aleks has written a Python function `gen_unif(a, b)` that gives a random sample from the *continuous* $Unif(a, b)$ distribution. However, what Aleks really wants is a function `gen_exp(lambduh)` that takes a parameter λ (lambduh as written in the code) and returns a sample from the $Exp(\lambda)$ distribution.

   Aleks is lazy, and thus thinks that he can code this new function in only one line and utilizing his function `gen_unif(a, b)`. To do this, he'll need to "transform" a sample of a uniform distribution into that of an exponential distribution.

(a) Let $X \sim Unif(0,1)$ be a continuous uniform random variable. Show that $Z = g(X) = -\frac{1}{\lambda}\ln(1-X) \sim Exp(\lambda)$. Aleks specifically wants you to show this using the CDF transformation method, by computing $F_Z(z)$.

(b) Aleks is only 90% sure his math was correct from earlier and still isn't convinced. Now he wants you to show the same thing (that $Z = g(X) = -\frac{1}{\lambda}\ln(1-X) \sim Exp(\lambda)$), this time using the explicit formula from section 4.4 to compute $f_Z(z)$. Make sure to explicitly verify the monotonicity and invertibility criteria.

(c) Now that we've verified that this transformation is "legit", help Aleks translate it into code! Write the Python function `gen_exp(lambduh)` using only `gen_unif(a, b)` to generate randomness. It should only be ONE LINE, and you may use `np.log` to compute the natural logarithm.

```
def gen_exp(lambduh:float) -> float:
    # Your one-line of code here.
```

(d) Note that the transformation we applied to X was the inverse of the CDF of the exponential distribution (if $Y \sim Exp(\lambda)$, then $F_Y(y) = 1 - e^{-\lambda y}$ for $y \geq 0$ and the inverse CDF is $g(y) = F_Y^{-1}(y) = -\frac{1}{\lambda}\ln(1-y)$.)
This process, in fact, works for any *arbitrary* continuous distribution! Explain how to generate a sample from a continuous distribution T, given only its CDF F_T and access to a single sample from $X \sim Unif(0,1)$. Explain why this process makes sense intuitively as well in AT MOST 2-3 sentences. (Hint: It's similar to the previous part!)

7. Choose a number X uniformly at random from the set of numbers $\{1,2,3,4,5\}$. Now choose a number uniformly at random from the subset no larger than X, that is from $\{1,\ldots,X\}$. Let Y denote the second number chosen.

(a) Fill out the joint probability mass function $p_{X,Y}(x,y)$ in Figure 2. Either put *fully simplified* fractions or numbers to *four* decimal places. Also, give the joint range $\Omega_{X,Y}$. (Recall an appropriate notation for the joint range would be something like $\Omega_{X,Y} = \{(x,y) \in \Omega_X \times \Omega_Y : \text{some condition is true}\}$.)

(b) Fill out the margins of the table in Figure 2, representing the marginal probability mass functions $p_X(x)$ and $p_Y(y)$. Either put *fully simplified* fractions or numbers to *four* decimal places. Your marginal distribution of X should be very simple (reread the first sentence of this question).

(c) Fill out the conditional probability mass function $p_{X|Y}(x|y) = P(X = x|Y = y)$ in Figure 3. Either put *fully simplified* fractions or numbers to *four* decimal places. Should each row sum to 1 or each column sum to 1?

(d) Are X and Y independent? Justify your answer with the **definition of independence**, and either prove it or give a counterexample. **An intuitive or English-only answer will not receive credit.**

Figure 2: Joint PMF $p_{X,Y}(x, y)$

X \Y	1	2	3	4	5	Row Totals
1	TODO	TODO	TODO	TODO	TODO	TODO
2	TODO	TODO	TODO	TODO	TODO	TODO
3	TODO	TODO	TODO	TODO	TODO	TODO
4	TODO	TODO	TODO	TODO	TODO	TODO
5	TODO	TODO	TODO	TODO	TODO	TODO
Col Totals	TODO	TODO	TODO	TODO	TODO	1

Figure 3: Conditional PMF $p_{X|Y}(x|y)$

X \Y	1	2	3	4	5
1	TODO	TODO	TODO	TODO	TODO
2	TODO	TODO	TODO	TODO	TODO
3	TODO	TODO	TODO	TODO	TODO
4	TODO	TODO	TODO	TODO	TODO
5	TODO	TODO	TODO	TODO	TODO

8. Consider a series of strings that independently get hashed into a hash table. Each such string can be sent to any one of $k + 1$ buckets (numbered from 0 to k). Let index i denote the i^{th} bucket. A string will independently get hashed to bucket i with probability p_i, where $\sum_{i=0}^{k} p_i = 1$ (so not necessarily uniformly!). Let N denote the number of strings that are hashed up to and including when one is hashed to any bucket other than bucket 0. Let X be the number of that bucket (i.e. the bucket not numbered 0 that receives a string).

   (a) Find Ω_N and a general formula for $p_N(n)$.

   (b) Find Ω_X and a general formula for $p_X(x)$.

   (c) Compute a general formula for $p_{N,X}(n, x)$ by determining what it means for $N = n$ and $X = x$ simultaneously. Are N and X independent? Justify your answer with the **definition of independence**, and either prove it or give a counterexample. **An intuitive or English-only answer will not receive credit.**

9. Suppose (X, Y) have the joint continuous distribution with joint density

$$f_{X,Y}(x, y) = \begin{cases} cx^6 e^{-y} & x^2 \leq y \leq 4 \\ 0 & \text{otherwise} \end{cases}.$$

   (a) Sketch the joint range $\Omega_{X,Y}$, and label the boundaries with the equations of the formulae. (You may attach a screenshot from WolframAlpha's RegionPlot - type in "RegionPlot[x>0 && y >0 && y<1-x]" for example).

(b) Give an expression for the value of c. Carefully set up the integrals with clearly defined integrands and limits of integration. Then additionally give your answer to 4 decimal places either solving it yourself or using WolframAlpha. Google search for "WolframAlpha double integral calculator".

(c) Compute the marginal densities $f_X(x)$ and $f_Y(y)$. Be sure to use piecewise functions for $f_X(x)$ and $f_Y(y)$ and for each identify the ranges Ω_X and Ω_Y.

(d) Are X and Y independent? Justify your answer with the **definition of independence**, and either prove it or give a counterexample. **An intuitive or English-only answer will not receive credit.**

(e) Give an expression for $E[\log|\tan^{-1}(X+Y)|]$. That is the log of the absolute value of the arctan of $X+Y$. This may not be a particularly relevant value, but you should apply what you know and set up the integral. Carefully set up the integrals with clearly defined integrands and limits of integration. You do **NOT** need to evaluate your answer for this part.

(f) Give an expression for $P(X > Y)$. Carefully set up the integrals with clearly defined integrands and limits of integration. Then additionally give your answer to 4 decimal places either solving it yourself or using WolframAlpha. (Hint: Draw/visualize this region on your joint range from part (a)!)

10. [**Coding+Written**] Google Chrome has a huge database of malicious URLs, but it takes a long time to do a database lookup (think of this as a typical `Set`). They want to have a quick check in the web browser itself, so a space-efficient data structure must be used. A **bloom filter** is a **probabilistic data structure** which only supports the following two operations:

    I. `add(x)`: Add an element x to the structure.

    II. `contains(x)`: Check if an element x is in the structure. If either returns "definitely not in the set" or "could be in the set".

    It does **not** support the following two operations:

    I. Delete an element from the structure.

    II. Give a collection of elements that are in the structure.

    The idea is that we can check our bloom filter if a URL is in the set. The bloom filter is always correct in saying a URL definitely isn't in the set, but may have false positives (it may say a URL is in the set when it isn't). Only in these rare cases does Chrome have to perform an expensive database lookup to know for sure.

    Suppose we have k **bit arrays** $t_1, \ldots, t_k$ each of length m (all entries are 0 or 1), so the total space required is only km bits or $km/8$ bytes (as a byte is 8 bits). Suppose the universe of URL's is the set $\mathcal{U}$ (think of this as all strings with less than 100 characters), and we have k ***independent and uniform*** hash functions $h_1, \ldots, h_k : \mathcal{U} \to \{0, 1, \ldots, m-1\}$. That is, for an element x and hash function h_i, pretend $h_i(x)$ is a **discrete** $Unif(0, m-1)$ random variable. Suppose we implement the `add` and `contains` function as follows:

Algorithm 1 Bloom Filter Operations

```
1: function INITIALIZE(k,m)
2:     for $i = 1, \ldots, k$: do
3:         $t_i$ = new bit array of m 0's
4: function ADD(x)
5:     for $i = 1, \ldots, k$: do
6:         $t_i[h_i(x)] = 1$
7: function CONTAINS(x)
       return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$
```

Refer to the notes from Section 9.4 of the textbook for more details on bloom filters.

(a) After inserting n distinct URLs to the bloom filter, suppose we had a *new URL* and wanted to check whether it was contained in the bloom filter. What is the probability of a false positive? That is, what is the probability the bloom filter returns True (incorrectly), in terms of k, m and n?

(b) Implement the functions `add` and `contains` in the `BloomFilter` class of `bloom.py`. What is the sample false positive rate? (This is printed out for you automatically).

(c) Let's compare this approach to using a typical `Set` data structure. Google wants to store 1 million URLs, with each URL taking (on average) 25 bytes. How much space (in MB, 1 MB = 1 million bytes) is required if we store all the elements in a set? How much space (in MB) is required if we store all the elements in a bloom filter with $k = 10$ hash functions and $m = 800,000$ buckets? Recall that 1 byte = 8 bits.

(d) Let's analyze the time improvement as well. Let's say an average Chrome user attempts to visit 51,000 URLs in a year, only 1,000 of which are actually malicious. Suppose it takes half a second for Chrome to make a call to the database (the `Set`), and only 1 millisecond for Chrome to check containment in the bloom filter. Suppose the false positive rate on the bloom filter is 4%; that is, if a website is not malicious, the bloom filter will will incorrectly report it as malicious with probability 0.04. What is the time (in seconds) taken if we only use the database, and what is the *expected* time taken (in seconds) to check all 51,000 strings if we used the bloom filter + database combination described earlier?

11. [**Coding+Written**] YouTube wants to count the number of ***distinct*** views for a video, but doesn't want to store all the user ID's. How can they get an accurate count of users without doing so? The problem is modelled as follows: a video receives a **stream** of 8-byte integers (user ID's), $x_1, x_2, \dots, x_N$, but there are only n *distinct* elements ($1 \le n \le N$), since some people rewatch the video. We don't know what N is, since people continuously view the video, but assume we cannot store all N elements; we can't even store the n distinct elements.

    (a) First, let's do this *seemingly* unrelated calculation. Let $U_1, \dots, U_m$ be m iid samples from the continuous $Unif(0,1)$ distribution, and let $X = \min\{U_1, \dots, U_m\}$. Show that $E[X] = \frac{1}{m+1}$. (Hint: Compute $F_X(x)$ similarly to 5(c), then compute $f_X(x)$).

    (b) If we were to solve this problem naively using a `Set`, what would the big-Oh space complexity be (in terms of N and/or n)? If a video had N =2 billion views, with only n = 900 million of them being distinct views, how much storage would we need for this one video to keep track of the distinct users? Give your answer with the closest unit (like 13.1 megabytes, 211.5 gigabytes, etc.).

    (c) Suppose the universe of user ID's is the set $\mathcal{U}$ (think of this as all 8-byte integers), and we have a single ***uniform*** hash function $h : \mathcal{U} \to [0,1]$. That is, for an element y, pretend $h(y)$ is a **continuous** $Unif(0,1)$ random variable. That is, $h(y_1), h(y_2), \dots, h(y_k)$ for any k **distinct** elements are iid continuous $Unif(0,1)$ random variables, but since the hash function always gives the same output for some given input, $h(y_1)$ and $h(y_1)$ are the "same" $Unif(0,1)$ random variable.

    I claim we can (approximately) solve this distinct elements problem using a single floating point variable (8 bytes), instead of the amount of memory the naive approach from part (b) requires. Pseudocode is provided which explains the two key functions:

    i. `update(x)`: How to update your variable when you see a new stream element.

    ii. `estimate()`: At any given time, how to estimate how many distinct elements you've seen so far.

    Argue why this randomized algorithm's estimate is a "good" one, with solid grounding in probability. Make sure to read how the "update" function is implemented as well, and use your answer from part (a).

Algorithm 2 Distinct Elements Operations

```
function INITIALIZE()
    val ← $\infty$
function UPDATE(x)
    val ← min {val, hash(x)}
function ESTIMATE()
    return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \dots, N$: do                          ▹ Loop through all stream elements
    update($x_i$)                                  ▹ Update our single float variable
return estimate()          ▹ An estimate for $n$, the number of distinct elements.
```

Refer to the notes from section 9.5 of the textbook for more details on the distinct elements algorithm.

(d) Implement the functions `update` and `estimate` in the `DistElts` class of `dist_elts.py`. What is the estimated and the true number of distinct elements in `stream_small.txt`? (This is printed out for you automatically).

(e) The estimator we came up with in (c) has high variance, so isn't great sometimes. To solve this problem, we will keep track of K DistElts classes, take the mean of our K mins, and then apply the same trick as earlier to give an estimate. This will reduce the variance of our estimate significantly. Implement the functions `update` and `estimate` in the `MultDistElts` class of `dist_elts.py`. What is our improved estimate of the number of distinct elements for $K = 50$? (This is also printed out for you automatically).

(f) How much space is saved from part (b) if YouTube wants to use $K = 10,000$ reps? Assume for simplicity each DistElt class only takes 8-bytes (since it only stores one float variable). Give your number as a multiplicative factor of savings (e.g., 10x, 2x, etc).

12. **(Extra Credit)**: If you worked with a partner that you were randomly paired with during a social event or through the partner survey, attach a screenshot here to get extra credit (you can get extra credit even if you use the same partner)! If it was a social event zoom call, your screenshot must include the zoom meeting information to prove it was one of our social zoom meetings.

13. **(Extra Credit)** [**Written**] The distinct elements algorithm is very theoretically interesting: it can be used to actually solve a similar problem! Suppose we have a stream $x_1, \ldots, x_N$, with only $n < N$ distinct elements. The **distinct sum** problem is: instead of returning n, return the *sum* of the n distinct elements. When asked for pseudocode, copy the LaTeX pseudocode from Q11, and modify it below.

    (a) Suppose the stream elements are all *positive integers*. Explain how we can solve this "positive integer *distinct sum*" problem. Then, explicitly give pseudocode for the `update` and `estimate` functions. (Hint: You can "reduce" this problem to the distinct elements problem. This means transforming the stream somehow and using instance(s) of the distinct elements class. You can assume you have a distinct elements implementation which handles any datatype (int, string, etc.).)

    (b) Suppose the stream elements are all *integers* (could be positive or negative). Explain how we can solve this "integer *distinct sum*" problem. Then, explicitly give pseudocode for the `update` and `estimate` functions. (Hint: Reduction.)