Remember that you must write up your solutions **independently**, and **list your collaborators** by name clearly at top of your submission (or "no collaborators" if none).

65 points total, with an additional 15 bonus points.

1. (14 pts total) Random bits of various shapes and sizes.

a. (5 pts) Suppose your quantum physicist friend told you that some extremely low-level subroutine `probefluxcapacitor()` in your computer was truly random, and each time called output an independent bit, equal to 1 with probability p and 0 with probability $1 - p$, for some probability p not necessarily equaly to $1/2$. The hitch is that you need uniformly random bits which are 1 with probability exactly $1/2$. Show how to use `probefluxcapacitor()` to implement `randbit()`, which outputs a uniformly random bit, but with no other sources of randomness. Give the expected runtime of your algorithm in terms of p.)

b. (3 pts) Now suppose you want to go the other way: you have access to a trusted `randbit()` function, but want to implement `randbit(`p`)`, which outputs an independent random bit with $\Pr[$`randbit(`p`)` $= 1] = p$. (Note that `randbit(1/2)` is the same as `randbit()`.) Assuming we can write $p = k/2^n$ for some integers k, n, give an algorithm to implement `randbit(`p`)` using n calls to `randbit()`.

c. (6 pts) What if p is not of the form $k/2^n$ for any k, n? Here is a very clever algorithm to implement `randbit(`p`)` for *any* probability p:

```
def randbit(p):
  for i = 1,2,3,...:
    let d = getdigit(p,i)
    if randbit() != d, return d
```

Here of course `getdigit(`p`,`i`)` returns the ith digit in the binary representation of $p = 0.b_0b_1b_2\ldots$. Show that this implementation of `randbit(`p`)` is correct, and give a bound on its expected runtime assuming `getdigit(`p`,`i`)` runs in time polynomial in i (i.e. $O(i^c)$ for some c).

2. (6 pts) Consider the problem of determining whether there is an element appearing strictly more than $n/2$ times in an array, and suppose your only access to the array is the function `arethesame(`i`,`j`)`, which tests whether or not elements i and j are equal. Design a linear time randomized algorithm which always correctly says NO if there is no such element, and if there is, says YES with probability $1 - p$ where p is the probability that a meteor strikes your computer in any given nanosecond.

3. (10 total pts, **plus 4 bonus**) The ever-despicable Gru is trying to distribute his minions among his k pods, but the giant tube dispensing the minions has lost stability, with the result that every second an additional minion is added to a random pod. (At time 0 all pods were empty.) Help Gru understand his situation by answering the following questions. (Hint: use notes from class, on the website, but **CITE** them.)

- **a.** (2 pts) What is the expected number of minions in the first pod after t seconds?
- **b.** (2 pts) In expectation, when will the first pod get its first minion?
- **c.** (2 pts) In expectation, when will all pods have at least one minion?
- **d.** (4 pts) Verify (**c**) numerically, by simulating this process many different times for many different values of k, and then plotting all of the data points together in one scatter plot. Then if your answer to (**c**) was $f(k)$, plot $c_1 f(k)$ and $c_2 f(k)$ for values of c_1, c_2 which show that in fact data points are sandwiched between these two. For your answer, show your plots and values of c_1, c_2, and turn in your code as a separate file.
- **e.** **(Optional: 4 bonus pts)** Gru can only start his nefarious mission when all pods have at least ℓ minions. Show that if ℓ is large enough, then with high probability (i.e. $1 - p$ for some given very small p), all pods have at least ℓ minions after $\Theta(\ell k)$ seconds. (Hint: use the central limit theorem, and **CITE** your source.)

4. (12 total pts, **plus 4 bonus pts**) We will show how to implement Karger's min-cut algorithm using Kruskal: we give uniformly random weights (in $[0, 1]$ say) to the edges, run Kruskal to produce an MST T, take the maximum-weight edge e of T, and return the cut formed by removing e from T.

- **a.** (3 pts) Explain how to view Karger's algorithm as constructing a spanning tree. (Note: not necessarily a *minimum* spanning tree.)
- **b.** (5 pts) Show that the distribution over spanning trees is the same in Karger and in our randomized Kruskal. That is, show that the probability of spanning tree T being selected is the same in both.
- **c.** (2 pts) Assuming T was chosen by both algorithms, explain why the max-weight edge of T in Kruskal defines the final cut output by Karger.
- **d.** (2 pts) Put the above three steps together to prove the equivalence of the two algorithms: the distribution over cuts they return is the same.
- **e.** (**Optional: 4 bonus pts**) What is the runtime? Can you implement it faster?

5. (10 pts total, **plus 4 bonus pts**) Implement Karger's min-cut algorithm (just the single-pass, not the multiple iterations) using an off-the-shelf union-find implementation and a graph library. Your code should look similar to the pseudocode of Karger, without needing to implement standard graph iterators through nodes and edges, but do not use more advanced subroutines like edge contraction – implement that yourself. (I would recommend NetworkX.)

- **a.** (6 pts) Take random graphs of $n = 5, 6, 7, \dots, 20$ and show numerically the probability of selecting the min cut on a single pass as a function of n. (Similar to 3.**d.**.) You may use e.g. NetworkX's `gnp_random_graph` to generate the graphs, but to be relatively confident of computing the real min cut size, you should run your algorithm about $10n^2$ times and then take the minimum. Show your plots and include your code in a separate file.
- **b.** (4 pts) In class there were several questions about which graphs give Karger higher or lower chances of finding a min cut. Give two graphs on $n = 20$ nodes with very different success probabilities for Karger.
- **c.** **(Optional: 4 bonus pts)** Implement Karger-Stein (using your Karger code as a subroutine) and perform similar experiments.

6. (8 pts) The minions are organizing a soccer game and have chosen two captains A and B, which will take turns selecting their players. The players are lined up in a long narrow corridor with a door in the front and the back, and thus captains may only select the player which is currently in the front or the back of the line. Assuming player i has some known ability $x_i > 0$, derive an optimal strategy for A to select players, maximizing the sum of player abilities, assuming A chooses first.

7. (5 pts, **plus 3 bonus pts**) Gru has given each of his n minions each a unique designated location to stand before his big launch. They enter one-by-one, but the first minion gets confused and instead stands in a random position. Thereafter, each minion either stands in their designated position, or if that position is taken, chooses a random position which has not already been taken. The only trouble is that Gru cannot launch unless the very last minion, who has the launch codes (bad idea Gru), is in the correct position. What is the probability that Gru can launch?

- **a.** (5 pts) Solve the problem using a recurrence which expresses the problem in terms of smaller versions of the problem.
- **b.** **(3 bonus pts)** Solve the problem *without* a recurrence, in a short paragraph and no formulas!