1. A Poisson Binomial random variable is the sum of n independent Bernoulli random variables whose probabilities of success are not constant, say $p_1, p_2, \ldots, p_n$. The Poisson Binomial probability mass function (pmf) is considerably more complex than the Binomial pmf. It is

$$\begin{aligned} p_n(k) &= \sum_{\substack{z_1=0 \\ z_1+\cdots+z_n=k}}^{1} \cdots \sum_{z_n=0}^{1} p_i^{\mathbb{1}\{z_i=1\}}(1-p_i)^{\mathbb{1}\{z_i=0\}} \\ &= \left(\prod_{i=1}^{n}(1-p_i)\right) \sum_{\substack{z_1=0 \\ z_1+\cdots+z_n=k}}^{1} \cdots \sum_{z_n=0}^{1} \prod_{i=1}^{n} w_i^{\mathbb{1}\{z_i=1\}}, \end{aligned}$$

where $w_i = p_i/(1-p_i)$. For large n, $p_n(k)$ is practically impossible to compute via this exact form.

To motivate your work, I will provide a dataset from my current consulting work. These are based on genome sequencing data from a relative of the peanut plant. Several regions were selected for sequencing, and each region was sequenced many, many times. These data are the attempts to sequence one particular site in the genome along with the sequencing machine's estimate of the probability that an error was made during each sequencing attempt. It is of interest to study the total number of errors observed in n sequencing attempts.

(a) (`C` code) Write a function to read in the data from `stdin` (to skip the first header line, you may loop `getchar()` until you read the first `'\n'`). Write a function to implement the naive algorithm for computing $p_n(k)$ given an array of error probabilities, k and n as input. Write a `main()` function such that the program can be called as

```
cat stat580_hw3.txt | ./poisbin -n 300 -k 10
```

to compute $p_n(k)$. For what n do the calculations become notably slow?

(b) (`C` code) Wikipedia suggests recursion

$$p_n(k) = \frac{1}{k}\sum_{i=1}^{k}(-1)^{i-1}p_n(k-i)T(i),$$

for $k > 0$ and

$$T(i) = \sum_{j=1}^{n}\left(\frac{p_j}{1-p_j}\right)^i.$$

The recursion is initialized with $p_n(0) = \prod_{i=1}^{n}(1-p_i)$. Write a function to implement this recursion for computing $p_n(k)$ given an array of error probabilities, k and n as input.

(c) Wikipedia mentions that the recursion from Part b may be unstable. Do you find evidence it is unstable? What if you use it to compute the distribution of the number of non-errors, the ith of which occurs with probability $1-p_i$?

(d) (`C` code) There is another more stable recursion that is a generalization of Pascal's triangle for binomial coefficients. Given $p_1(0) = 1-p_1$ and $p_1(1) = p_1$, it is not hard to see that

$$\begin{aligned} p_j(0) &= (1-p_j)p_{j-1}(0) \\ p_j(i) &= p_j p_{j-1}(i-1) + (1-p_j)p_{j-1}(i) \qquad \text{for } 1 \le i \le j-1 \\ p_j(j) &= p_j p_{j-1}(j-1). \end{aligned}$$

Indeed, this recursion can even be superior to computing the pmf of the Binomial(n,p). Implement this recursion and verify its stability relative to the recursion of Part c.

(e) Count the number of operations (addition, multiplication, subtraction, *etc.*) involved in each algorithm. Which should be the fastest? Can you think of ways to make any or all of them faster? (You do not need to implement these ideas.)