Biostatistics 615 Learning Exercise #4 (10 pts)

Due by September 24th 2024 (Tuesday) 11:59pm. Use Gradescope (via Canvas) to submit an R file.

- Your submission should only consist of a single `R` file named `fastSumOfBernoulli.R`, which contains a function named `fastSumOfBernoulli(x)`.
- Your code will be evaluated in Gradescope using 10 different test cases using an automated script. Full credit will be given if your code passes all test cases.
- You are allowed to submit multiple times before the deadline, but only the last submission will be graded. Automated feedback will be provided for each submission.
- You need to implement the function to work with arbitrary (valid) input values beyond the 10 cases tested. If you adjust your implementation to work only for the specific test cases, you will not receive any credit.
- Implement your function as efficient as you can. If your program does not finish after running for 1.0 seconds, you will lose the points for those test cases. Note that the official solution finishes within 0.6 seconds for any test case, so this should be a reasonable time limit.

Problem 1 - Fast Sum of Bernoulli Variables (10 pts)

Write an R script `fastSumOfBernoulli.R` that contains a function `fastSumOfBernoulli(p)`, which computes the probability mass function of sum of independent Bernoulli random variables using a divide-and-conquer algorithm within $O\left(n(\log n)^2\right)$ time complexity.

Consider $X_i \sim \text{Bernoulli}(p_i), i \in \{1, 2, \dots, n\}$ to be independent (but not identical) random variables in $\{0, 1\}$ with $\Pr(X_i = 1) = p_i, (p_i \in [0, 1])$.

The function `fastSumOfBernoulli(p)` takes `p` as an input vector representing $(p_1, \dots, p_n)$ and returns a vector of length $n + 1$, which contains the probability mass function of $Y = \sum_{i=1}^{n} X_i \in \{0, 1, \dots, n\}$, storing from $\Pr(Y = 0)$ to $\Pr(Y = n)$ in order.

There are specific requirements in the implementation:

- The algorithm must be a divide-and-conquer algorithm that uses recursion. In other words, `accurateSumOfBernoulli()` function should be implemented using recursion, meaning it must call itself within its implementation.
- The time complexity of the algorithm must be strictly faster than $O(n^2)$, and should be $O\left(n(\log n)^2\right)$ if implemented correctly. When the input size doubles (e.g., from 1,000 to 2,000), the computational time is expected to increase by approximately a factor of 2. Because the input sizes of 10 test cases are 4, 10, 10, 10, 10, 100, 100, 1000, 5000, and 10000, respectively, you will be able to see the empirical time complexity by observing the computational time of the test cases.
- NO statistical functions may be used. You may not use `dbinom`, `pbinom`, `pnorm`, or any other similar statistical functions or external libraries. In other words, you need to implement the p.m.f. from scratch without relying on existing ones.

- You are NOT allowed to use any functions outside the `base` package in your implementation, EXCEPT FOR the `convolve()` function available in `stats` package. Use `help(...)` to check whether a function you want to use belongs to the `base` package or not.

If you implementation violates any of the above requirements, you will receive 0 points for this problem even if Gradescope returns positive scores.

As stated above, the only function you are allowed to use outside the `base` package is `convolve()` function in the `stats` package, which allows efficient computation of convolution of two vectors by using Fast Fourier Transform in $O(n \log n)$ complexity. `convolve()` is useful to evaluate the multiplication of two polynomials. For example, given two polynomials $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{m} b_i x^i$, then we can evaluate the product $p(x)q(x)$ by using `convolve(a, rev(b), type="open")`. For more details on the `convolve()` function, refer to the R documentation by using `help("convolve")`. The following example code shows how to evaluate the product of two polynomials $f(x)$ and $g(x)$ to evaluate $h(x) = f(x)g(x)$.

$$\begin{aligned} f(x) &= 1 + 2x + 3x^2 + 4x^3 \\ g(x) &= 5 + 6x + 7x^2 + 8x^3 \\ h(x) &= f(x)g(x) = 5 + 16x^2 + 34x^3 + 60x^4 + 61x^5 + 52x^6 + 32x^7 \end{aligned}$$

```
> a = c(1,2,3,4)
> b = c(5,6,7,8)
> c = convolve(a, rev(b), type="open")
[1]  5 16 34 60 61 52 32
```

Note that evaluation of polynomial is not necessary in this homework, but you can expand the idea to evaluate the PMF of Y in a divide-and-conquer fashion.

Evaluating the product of two polynomials with degree n typically takes $O(n^2)$ time complexity. However, by leveraging the Fast Fourier Transform used in `convolve()`, we can evaluate the product in $O(n \log n)$ time complexity. This allows us to implement `fastSumOfBernoulli()` in $O\left(n(\log n)^2\right)$ time complexity based on the following formula:

$$T(n) = 2T(n/2) + O(n \log n)$$

Please refer to the Topic 2 lecture slides to show that the time complexity is $O\left(n(\log n)^2\right)$. FFT is a fascinating application of the divide-and-conquer algorithm, and there are excellent youtube videos produced by Veritasium (`https://youtu.be/h7apO7q16V0`), 3Blue1Brown (`https://youtu.be/spUNpyF58BY`), Reducible (`https://youtu.be/h7apO7q16V0`), and and MIT OpenCourseWare (`https://youtu.be/iTMn0Kt18tg`). While you do not have to fully understand the FFT algorithm to implement `fastSumOfBernoulli()`, watching some of these videos is strongly recommended.

For example, when `p = c(0.1, 0.2, 0.3, 0.4)`, the five (length of $p+1$) probability values should be as follows (rounded to 10 digits):

```
0.3024000000
0.4404000000
0.2144000000
0.0404000000
0.0024000000
```

The time complexity is very important. If your function does not finish in 1 second with a very large input (e.g. length of 10,000), it is a strong indicator that your algorithm does not have $O\left(n(\log n)^2\right)$ time complexity.

One of the caveats of using `convolve()` is that the absolute error in the output is limited by the machine's numerical precision. Therefore, the relative error can be very high when the true probability is very small. For example, when the true probability is 10^{-20}, your implementation may report is as -10^{-20} due to the numerical precision issue. So, the evaluation of correctness will be performed based on 10 digits under the decimal point, not based on the significant digits in the scientific notation. Be aware that `convolve()` may occasionally produce values outside the expected range of $[0, 1]$ due to numerical precision issues. Ensure that your function returns probabilities within the range of 0 to 1.

Although it is not required in this homework, in the Learning Exercise #5, you will be asked to implement the $O(n^2)$ algorithm, which does not use `convolve()` function, from scratch to understand the tradeoff between the accuracy and the computational time.

When calling the `convolve()` function, make sure to directly call `convolve()` without referring to any other library. For example, calling `stats::convolve()` or running `library(stats)` is NOT allowed and will result in errors.

You do not need to implement error handling for malformed arguments in this function.