CSCI 5454 Spring 2023: Assignment #2
Due Date: Thursday, February 9, 2023.

Topics: Knuth-Morris-Pratt Alg., Divide-And-Conquer Algorithms (FFT)

P1. (10 points) Suppose we are using KMP algorithm to match a pattern against a text and we do not find a pattern match. This problem asks you to modify the pseudo-code to find the longest prefix of the pattern that matches the text: output the ending position of the match in the text string and the *length* of the longest prefix match. If there are multiple matches of the same prefix length: you should find the *earliest* match in the test.

Modify the pseudocode below and write a very brief explanation of why your modification works. As always, you should implement this in your favorite language and test it thoroughly. But no need to submit code or report on that for this assignment.

Here is the pseudo-code for the original KMP algorithm as presented in class. Assume π is the table constructed as shown in class.

```
def kmp_matcher( txt , pat , $\pi$ ):
    (n, m) = len(txt), len(pat)
    i = -1 # i is the last index processed in text
    j = -1 # j is the last index processed in pattern
    while i < n-1 :
       if pat[j+1] == txt[i+1]:
             (i, j) = (i + 1, j + 1)
       else:
             if j == -1:
                 i = i + 1
             else:
                 j = $\pi[j]$
       if j == m -1 :
          print(f'Pattern matched at position {i}')
          j = pi[j]
```

Your answers should present pseudo-code (python code is fine). Add a few lines of explanations but if your logic is clear, comments in the code can be an explanation. For our convenience, it would be great if you can highlight your changes in the pseudocode.

P2. (15 points) We have sets S_1, S_2 and S_3 that are all subsets of $\{0, \ldots, n-1\}$ for a number $n > 0$. Each set is given to you as an array of Booleans. Eg., for $n = 5$ and $S_1 = \{1, 2, 4\}$, we have the array $[0, 1, 1, 0, 1]$ that denotes the set S_1.
(A, 7 points) Given a target number T, write an efficient algorithm to calculate the number of triples (i_1, i_2, i_3) such that $i_1 \in S_1$, $i_2 \in S_2$ and $i_3 \in S_3$ such that $i_1 + i_2 + i_3 = T$. Your algorithm must operate in running time $O(n \log(n))$.
(B, 8 points) Find the number of triples (i_1, i_2, i_3) such that $i_1 \in S_1$, $i_2 \in S_2$ and $i_3 \in S_3$ such that $i_1 + i_2 - i_3 = 0$. Your algorithm must operate in running time $O(n \log(n))$.

For each part, the solution can be written out in at most 5 sentences. No need for pseudocode.

P3 (10 points) In class, we will be studying a divide and conquer algorithm with the following recurrence relation for its running time complexity on input of size n:

$$T(n) = \begin{cases} 2T(n/2) + C_0(n \log n) & n > 1 \\ C_1 & n \leq 1 \end{cases}.$$

for some positive constants C_0, C_1. Prove that $T(n) = \Theta(n(\log n)^2)$.