LAB TASK 11

Q1)

A software team is designing a key-value store using a hash table. The hash table has a size of 7 (indices 0 to 6). The chosen hash function is the Division Method: $H(key)=key(\text{mod}7)$.

They need to insert the following sequence of keys: 76, 93, 40, 47, 10, 55.

1. Linear Probing:
   - Trace the insertion of all six keys into the hash table using the Linear Probing technique. Assume the probing step is $(i+1)(\text{mod}7)$.
   - Identify the key(s) that caused the first collision, and state the index where the colliding key was ultimately placed.
2. Separate Chaining:
   - If the team had chosen Separate Chaining (Open Hashing) instead, describe the structure of the hash table after all six keys have been inserted.
   - For the key 55, state its calculated hash index and describe its final placement in the table under this method.

Q2)

A database system uses a hash table with a current **size of 100** and stores **75 elements**. The hash table uses a collision resolution method that degrades performance significantly when the table becomes too full.

1. **Load Factor and Rehashing:**
   - Calculate the current **Load Factor** for this hash table. Show the formula and the calculated value.
   - The system is configured to trigger a **Rehashing** operation when the Load Factor reaches 0.75. Based on your calculation, determine if rehashing is immediately required. Explain your answer.

2. **Rehashing Implementation:**
   - If rehashing were to be performed, what is the generally suggested strategy for calculating the **New Table Size**? Propose a new size based on this strategy, starting with the current size of 100.
   - Outline the **three main steps** required to execute the rehashing process and replace the old table with the new one

Q3) String Searching Algorithms

A bioinformatics researcher is analyzing a long DNA sequence (Text, T) to find the locations of a specific gene fragment (Pattern, P).

- Text (T): GCAATGCCCTATGTGACC
- Pattern (P): TATGT

1. Brute Force Algorithm:

   - Explain the core steps the Brute Force Algorithm takes to find the first occurrence of P in T.
   - Trace the first two comparison windows for the pattern starting at Text index 0 and 1, explicitly showing the characters being compared and where the first mismatch occurs for each window.

2. Boyer-Moore's Bad Character Heuristic:

   - Assume the pattern is aligned starting at T[3]:

     T G C A A T G C C T A T G T G A C C

     P T A T G T

   - A mismatch occurs at the last character of the pattern (P[4]='T' vs. T[7]='C'). The Bad Character is 'C'.
   - The pattern P is 'TATGT'. Determine the amount of shift suggested by the Bad Character Heuristic (Case 2: Pattern moves past the mismatch character) in this scenario. Explain the reasoning for the shift distance.

Q4) KMP Prefix Function:

The Knuth-Morris-Pratt (KMP) Algorithm is known for its linear time complexity in string searching, which is achieved by utilizing a preprocessed structure called the LPS array (Longest Proper Prefix which is also a Suffix).

Consider the following pattern: P = “ABABA"

1. **LPS Array Construction:**
   - Construct the LPS array for the pattern P = “ABABA". Show the value for each index.
   - State what the LPS value at index 4 (corresponding to the last 'A') means in the context of the KMP algorithm.

2. **Skipping Comparisons on Mismatch**:
   - Suppose the KMP algorithm is searching for P in a text, and a mismatch occurs at the character P[4] (the last 'A'). The LPS value of the character *before* the mismatch (at P[3], the second 'B') is 3.
   - Describe the KMP's action following this mismatch. From which index of the pattern will the comparison restart with the current text position?