DAY 16:

Task 3: Naive Pattern Search
Implement the naive pattern searching algorithm to find all occurrences of a pattern within a given text string. Count the number of comparisons made during the search to evaluate the efficiency of the algorithm.
A)

Introduction:

The naive pattern searching algorithm scans through the text, checking if the pattern matches at each position. If a match is found, it records the index of the match. This process continues until the entire text is scanned. It's simple and straightforward but may not be the most efficient for large texts or complex patterns.

Java code:

```
package Day16;

import java.util.List;
import java.util.ArrayList;
public class NaivePatternSearch {
        public static void main(String[] args) {
    String text = "ABABABABA";
    String pattern = "ABA";
    int[] occurrences = naivePatternSearch(text, pattern);
    int comparisons = countComparisons(text, pattern);
    System.out.println("Occurrences: ");
    for (int occurrence : occurrences) {
      System.out.println(occurrence);
    }
    System.out.println("Number of comparisons: " + comparisons);
  }

  public static int[] naivePatternSearch(String text, String pattern) {
    int n = text.length();
    int m = pattern.length();
    List<Integer> occurrences = new ArrayList<>();

    for (int i = 0; i <= n - m; i++) {
      int j;
      for (j = 0; j < m; j++) {
        if (text.charAt(i + j) != pattern.charAt(j)) {
          break;
        }
      }
      if (j == m) {
        occurrences.add(i);
```

```
            }
        }

        return occurrences.stream().mapToInt(i -> i).toArray();
    }

    public static int countComparisons(String text, String pattern) {
        int n = text.length();
        int m = pattern.length();
        int comparisons = 0;

        for (int i = 0; i <= n - m; i++) {
            int j;
            for (j = 0; j < m; j++) {
                comparisons++;
                if (text.charAt(i + j) != pattern.charAt(j)) {
                    break;
                }
            }
        }

        return comparisons;
    }
}
```
Explanation:

1. naivePatternSearch():

   - It iterates through the text and compares substrings of length m (length of the pattern)

     with the pattern itself.

   - When a match is found, it adds the starting index of the match to the occurrences list.

2. countComparisons():

   - It counts the number of comparisons made during the search process.

The main() method demonstrates how to use these functions. It prints the occurrences of the pattern in the text and the number of comparisons made.