Project 2 Written Project Report

Names:
Anthony Le | antlecsuf@csu.fullerton.edu
Ryan Nishikawa | ryannishikawa48@csu.fullerton.edu
Dylan Tran | dylanht341@csu.fullerton.edu
Jasmine Youssef | JasmineYoussef@csu.fullerton.edu

Algorithm 1: Target Terms or Substrings

Pseudocode:

```
START
   INPUT array_A, array_B
   DEFINE result_indices as empty list
   DEFINE result_words as empty list

   FOR each char in array_A:
      FOR each word in array_B:
        IF first char of word = char A:
          Check if word = substrA
          IF check == true
            APPEND index to result_indices
            APPEND word to result_words

   RETURN result_indices, result_words
END
```

Mathematical Analysis:

Time Complexity:
Searching for a word in Array A using substring search std::string::find() in C++) takes O(N) time in the worst case. Iterating through Array B (with size m) means O(m * N) complexity. Sorting the results takes O(m log m) time. Thus, the overall worst-case time complexity is O(mN + m log m).

Efficiency Class:
The algorithm runs in O(mN + m log m) time, which is efficient for moderate sized inputs but can be improved using advanced string searching techniques like the KMP algorithm (O(N + M) complexity).

Implementation Notes:
The algorithm will be implemented in C++. The program will read input from a file (in2a.txt) rather than being hardcoded. The output will be formatted and stored in an output file.

Algorithm 2: Run Encoding Problem

The Run Encoding Problem is:

Input: a string S of n characters, where each character is a lower-case letter or space

Output: a string C where each run of k repetitions of the character x is replaced with the string "kx"

```
def runEncode(S):
    n = length of string
    answer = " "

    for i from 0 to n-1:
        k = 1

        while i is less than n-1 and S[i] is equal to S[i+1]:
            k = k + 1
            i = i + 1

        if k > 1:
            append k as a string to answer
        append S[i] to answer

    return answer
```

Mathematical Analysis:

"for i from 0 to n-1" runs at most n times.

The while loop would loop, in the worst case, an n amount of times if all the characters within string S are the same.
Since i only moves forward, the total number of loops happening would be at most n.

Appending to the string answer is a (amortized) constant time operation, O(1).

Therefore, the efficiency class of this algorithm is O(n).

Algorithm 3: Merging Sorted Arrays Problem

The Merging Sorted Arrays Problem is:
Input: A list of K sorted arrays, where each array contains integers in ascending order.

Output: A single sorted array containing all elements from the input arrays, merged in ascending order.

Pseudocode for Merging Sorted Arrays Using a Min-Heap

```
def mergeSortedArrays(arrays):
```

```
    merged_result = []  # Stores the final merged sorted list
    min_heap = MinHeap()  # Priority queue to track the smallest elements

    # Insert the first element of each array into the heap
    for i from 0 to length(arrays) - 1:
        if arrays[i] is not empty:
            min_heap.insert((arrays[i][0], i, 0))  # (value, array index, element index)

    # Process the heap until it is empty
    while min_heap is not empty:
        current_value, array_idx, element_idx = min_heap.extract_min()
        merged_result.append(current_value)

        # If there are more elements in the current array, insert the next one
        if element_idx + 1 < length(arrays[array_idx]):
            next_value = arrays[array_idx][element_idx + 1]
            min_heap.insert((next_value, array_idx, element_idx + 1))

    return merged_result
```

Mathematical Analysis:

Heap Initialization:
Inserting the first element of each of K arrays into the heap takes O(K log K) time (since each insertion is O(log K)).

Heap Operations During Merging:
There are N total elements across all arrays (N = sum of lengths of all arrays).
Each element is extracted once (O(log K) per extraction).
Each element (except the last in its array) is inserted once (O(log K) per insertion).
Total operations: O(N log K).

Space Complexity:
The heap stores at most K elements at any time → O(K).
The merged result stores N elements → O(N).
Total space: O(N + K).

Efficiency Class:
Time Complexity: O(N log K) (where N = total elements, K = number of arrays).
Space Complexity: O(N)