Name: Rudolph Laprade

Main Idea:

The algorithm will use a greedy approach. It will repeatedly take the two strings with the largest overlap and merge them until all strings have been combined into a single one. After merging two strings, we can compute the overlap between the new string with other strings based on previous calculations for the overlap between the constituent strings and the other strings.

Psuedocode:
procedure RECONSTRUCT(n dna strings)

1. pq = priority queue
2. for every pair (i, j) of strings (order matters, i and j unique): put (i,j) into pq with priority = OVERLAP(str1, str2)
3. while pq is not empty:
   (a) get pair i, j with greatest overlap from pq
   (b) if i or j is not valid: next iteration of while loop
   (c) newStr = COMBINE(i, j)
   (d) for every valid string k:
      i. put (k, newStr) into pq with priority OVERLAP(k, i)
      ii. put (newStr, k) into pq with priority OVERLAP(j, k)
      iii. if newStr contains k (determinable based on overlaps with i and j): put (newStr, k) into pq with priority = length of k
   (e) invalidate i and j
4. return newStr

subroutine OVERLAP(str1, str2) - Finds the maximum overlap between str1 and str2, starting with with str1 to left of str2 using Knuth-Morris-Pratt algorithm. Simulates using a finite state machine to determine how much of str2 is matched at each index of str1. Uses memoization to remember OVERLAP values and finite state machine models.

subroutine COMBINE(str1, str2) - concatenates 2 strings without duplicating overlap

Analysis of running time:

The running time of the OVERLAP subroutine will be on the order of the sum of the lengths due to the KMP algorithm. It does one run over the second string to generate the finite state machine model. Then, using the model it is able to find the OVERLAP with one run over the first string. Therefore, the worst case running time of OVERLAP for not yet computed values will be $O(k)$ where k is the length of the longest string in the input.

If n is the number of line (strings) in the input file, there will at first be $O(n^2)$ calls to OVERLAP

since we call the subroutine twice for every pair of strings. For each of these OVERLAP calls, there is also an insertion into pq, which should individually take $O(\log n)$ time. Therefore, the first for loop (in the psudocode) should take $O(n^2(\log n + k))$.

Since every time we merge two strings we decrease the number of valid strings by 1, there will be $O(n)$ merges in the while loop. Each time we do a merge (which is at worst a linear operation in the total length of the two strings), we also do $2m$ insertions into pq, where m is the number of remaining valid strings. There are also $O(m)$ calculations for overlaps, but because these are computed based on previous values, each calculation takes constant time. This gives a total running time for the while loop in $O(nm \log n) = O(n^2 \log n)$ since m is in $O(n)$.

Thus the total running time of the algorithm is in $O(n^2(\log n + k))$.