ALGORITHM

1. Get score for reverse complement pairs

```
    INPUT: Fragment matrix
    OUTPUT: score matrix

    1.1. For all possible pairs of fragment pairs in the fragment matrix

        1.1.1. Initialize similarity matrix simil[n, m] :
            All first row cells of simil[ ] = 0
            All first column cells of simil[ ] = 0

        1.1.2. Fill the rest of similarity matrix simil[ ] and keep track of maximum score
        maxval from pairwise alignment of two fragments (string1, string2)
    using given weights for (G,C), (A,U), and (G,U) pairs, with other pairs  scored as 0.0:
            for i=1 to length of string1 do
                for j=1 to length string2 do
                    If string1 [ i-1 ] and string2 [ j-1 ]
                        match any combination in table(A), set score
                        value accordingly.
                    else
                        simil[i, j] = max( simil[i-1, j]-1,
                        simil[I, j-1]-1 and simil[i-1][j-1]+(-1 · score) )
                    if simil[i, j ] > maxval then maxval = simil[i ,j]
            RETURN maxval

        1.1.3. Store returned maxval value in corresponding cell in score matrix.

    1.2. OUTPUT: score matrix
```

2. Get best candidate fragment pair for start of sequence

```
  INPUT: Fragment matrix, reverse complements matrix, score matrix;
  OUTPUT: Best candidate fragment pair
  INIT: Best pair = null, high score = -1
```

```
    2.1. For every pair and its score:
            If current pair score is greater than best pair high score AND pair is attached at
    either tip then best pair = current pair
            If Best pair still = null then
            If current pair score is greater than best pair high score AND pair is attached at
    either tip then best pair = current pair
            OUTPUT: best pair

3. Create modified de bruijn graph
  INPUT: fragments, total length, best pair
  OUTPUT: directed graph (kmers [ ], kmers entry vertices [ ], kmers exit vertices[ ])
    3.1. Initialize arrays kmers [ ], kmers entry vertices [ ], kmers exit vertices[ ]

    3.2. Get kmer length:
            If total length is greater than or equal to 1000 then
                kmer length = round (total length · 0.23)
            Else kmer length = 15

    3.3. Merge fragments in any order PROVIDING that best pair are first and last.

    3.4. Divide merged fragments into kmers that are less than or equal to kmer length

    3.5. Add new kmer to kmers[ ]

    3.6. If kmers entry vertices [ ] does NOT contain new kmer
            then add new kmer’s substring (0..length-1) to kmers entry vertices[ ].

    3.7. If kmers exit vertices [ ] does NOT contain new kmer
            then add new kmer’s substring (1..length) to kmers exit vertices[ ].

    3.8. RETURN graph ([ ]kmers, [ ] kmers Left vertices, [ ] kmers right vertices) )
        INPUT: Array of fragments Fragments[ ]

4. Create all permutations of contents in Fragments[ ]

5. For every permutation of Fragments[ ]
```

```
5.1. go to 3 in main algorithm with permutation as input

5.2. If length of sequenceResult = total of lengths of fragments in permutation AND
        sequenceResult contains all fragments then
                                        OUTPUT: sequenceResult
```

BASE 1	BASE 2	SCORE
G	C	-8.4
C	G	-8.4
A	U	-8.0
U	A	-8.0
G	U	-1.0
U	G	-1.0