9.1

```
Algorithm: FW()
  initialize an array
  shortestPath (i, j, k)
     If i,j,k is in array
        Return array[i][j][k]
      Length = min(3 combos of shortest paths )
  Arr[i][j][k] = length
  return length
```

The number of recursions calculated was 137760

9.2

The if statement within the innermost loop was executed 158 times, this is a significant decrease from the previous approach. The second approach is much faster because it stores all of the computations so there is no repetition that needs to be done, as opposed to the previous approach.

9.3

Routes aren't computed just once because changes are constantly happening so without routes recomputing we cannot properly reflect the network.

9.5

```
Algorithm: naiveAlg( double [] arr )
  initialize sum and result variables
  for i 0 to arr.length
     for j 0 to arr.length
        sum = 0
        for k = i to j
           sum += X[k]
        end-for
           if sum > result then result = sum
      end-for
  end-for
  return result
```

9.6

The faster algorithm took 0 ms, while the naive algorithm took 402 ms

9.7

The run time of algorithm 1 would be O(n^2) because we are looping through every edge in the list. The run time of algorithm 2 would be O(n!) because we are going down the list and eliminating visited edges

9.8
For this problem I believe the binary optimal list would work

9.9
If we put an if statement before line 7 and add an else before line 8 we would handle the case when a subrange is invalid.

9.10

1. D_{ij} = max(D_{ik} , A[j])
2. D_{jk} = {max(D_{ik}, D_{jk}), A[k] if i = k or j = k}
3. D_{ij} = { max (D_{ik}, D_{jk}, A[j] + A[j - 1])

   A[i] + A[k] if k = i + 1

   A[k] + A[j] if k + 1 = j }