Question 1

1. Optimal parenthesization: ((((([5, 10] [10, 3])[3, 12])[12, 5])[5, 50])[50, 6])

2.
```
Table:
[ [0,   150,    330,    405,    1655,   2010]
, [0,   0,      360,    330,    2430,   1950]
, [0,   0,      0,      180,    930,    1770]
, [0,   0,      0,      0,      3000,   1860]
, [0,   0,      0,      0,      0,      1500]
, [0,   0,      0,      0,      0,      0] ]
```

3. Let *paren* be the function that returns the number of parenthesis needed for successful multiplication.

```
paren(1) = 0
paren(2) = 1
paren(3) = 2
paren(n+1) = n-1

----
paren(n+1) = paren(n) + 1

let n = 2
paren(2+1) = paren(2) + 1
paren(3) = 1 + 1 = 2
paren(3) = 2
```

Question 2

1. Running RECURSIVE-MATRIX-CHAIN is better because enumerating all of the different ways to multiply would be $(n-1)^{n-1}$ different checks, and each check would have $n-1$ matrix multiplications. The RECURSIVE-MATRIX-CHAIN has $\frac{1}{2}n^2$ different checks with each check being 2 matrix multiplications.

```
l = [10, 3, 15, 16, 25, 9, 5, 14, 18, 6, 17, 22, 80, 59, 11, 7]
                                   /\
10, 3, 15, 16, 25, 9, 5, 14     18, 6, 17, 22, 80, 59, 11, 7
             /\                                 /\
10, 3, 15, 16   25, 9, 5, 14    18, 6, 17, 22    80, 59, 11, 7
     /\                /\                /\                /\
10, 3  15, 16    25, 9  5, 14     18, 6  17, 22   80, 59  11,7
  /\       /\        /\      /\       /\       /\       /\       /\
10  3   15  16   25  9   5  14  18  6   17  22   80  59  11 7
 \/       \/        \/       \/       \/       \/        \/       \/
3, 10  15, 16   9, 25    5, 14   6, 18  17, 22   59, 80  7, 11
      \/              \/                \/                \/
3, 10, 15, 16   5, 9, 14, 25     6, 17, 18, 22   7, 11, 59, 80
             \/                                 \/
3, 5, 9, 10, 14, 15, 16, 25      6, 7, 11, 17, 18, 22, 59, 80
                                   \/
l = [3, 5, 6, 7, 9, 10, 11, 14, 15, 16, 17, 18, 22, 25, 59, 80]
```

1. Memoization doesn't help with a good divide and conquer algorithm because the runtime is already $O(nlog(n))$, there is no way to avoid seeing every item in the array so the algorithm must be ¿= $O(n)$, and trying to add anything to another array would only increase the runtime by another factor of n.
2. Yes, it exhibits optimal substructure. The check for the best just needs to be switched around so that the best multiplication is the one that has a higher number, instead of a lower one.

Question 4

```
kn(I, W) = {
    0
        if I = [] or W = 0
    i.v + kn(I - i, W - i.w)|i = max(i.v/i.w, i::I)
        otherwise
}
```

Question 5

1.
```
e(S, M, i, j) = {
    M
        if S = [] or i > j
    e(S, M - S[l_i + 1], i+1, j)
        otherwise
}

bl(S, M, i, j) = {
    ∞
        if M < 0
    e(S, M, i, j)
        otherwise
}

mb(S, M) = {
    bl(S, M, 0, ∞)
        when e(S, M, 0, ∞) >= 0
    bl(S[:mn(S, M, 0)], M, 0, mn(S,M)) + mb(S[:mn(S, M, 0)])
        otherwise
}

mn(S, M, i) = {
    i - 1
        if i > |S| or e(S[:i], M, 0, ∞) < 0
    mn(S, M, i+1)
        otherwise
}
```

Question 6

u = z:

V:	S	T	X	Y	Z
π	Z	X	Y	S	NIL
d:	2	5	6	9	0

u = s

V:	S	T	X	Y	Z
π	NIL	X	Y	S	T
d:	0	2	4	7	-2