CSCI 665 ASSIGNMENT 5

1.a

Table for optimal scalar operations is-

	0	150	330	405	1655	2010
		0	360	330	2430	1950
			0	180	930	1770
				0	3000	1860
					0	1500
						0

Choice table:-

		1	2	2	4	2
			2	2	2	2
				3	4	4
					4	4
						5

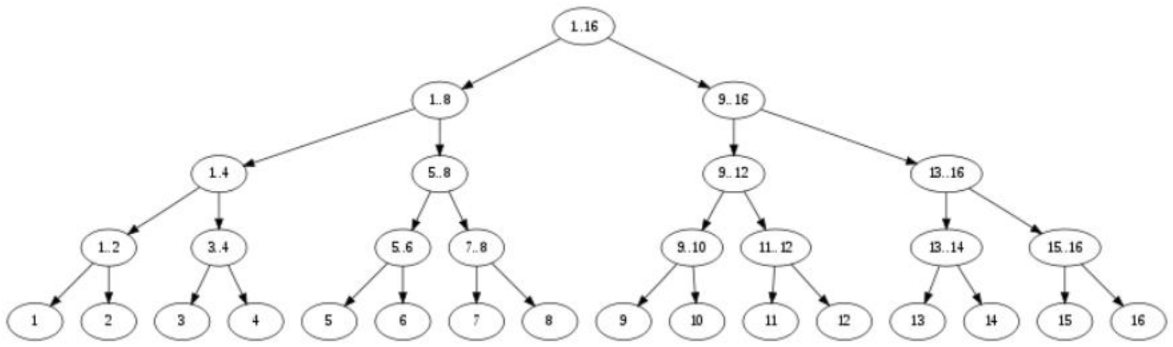
b.

We prove this by checking the hypothesis for a n+1-element expression.
Let k be a element at which this expression is split such that 1st part goes from 1 to k and 2nd part from k+1 to n+1.
Multiply them.
By induction,
For 1st part there are k-1 parentheses and n+1-(k+1) parenthesis for the 2nd part.
Therefore the total number of parentheses are- k-1+n+1-k-1+1=(n+1)-1.
Therefore there are (n+1)-1 parentheses for an n+1 element expression.

2.a

The runtime of of enumerating is just $n * P(n)$, while is we were running RECURSIVE-MATRIX-CHAIN, it would also have to run on all of the internal nodes of the subproblem tree. Also, the enumeration approach wouldn't have as much overhead.
The enumeration approach finds ways to parenthesize the left half and all ways to parenthesize the right half and finds all the combinations of left half with right half. The work taken to look at each combination is the product of the number of ways to parenthesize both the halves.

Recursive matrix finds the best way to parenthesize each half separately and then combines the two results. Thus the amount of work to combine is O(1).



b.

Memoization is a process where the result of an expensive functioned is stored to be used later. Merge-sort divides the problem into smaller non-overlapping problems.Since they are non-overlapping, they are called just once and there is no need of memoization. Hence memoization does not optimise the speed of merge sort.

c.
Yes, this problem does exhibit optimal substructure property.
After splitting the matrix chain into two sub-chains, the parenthesization within each sub-chain must be such that they maximize the number of scalar multiplications involved for each sub-chain. We can parenthesize the sub-chain such that it increases the total number of multiplications for the entire chain. The multiplication cost depends on the dimensions of the matrix so we should find a split point considering the cost of optimally parenthesizing any one sub-chain internally.

4.a
```
def knapsack(vwpair, weight):
   wt <- array(vwpair.keys())
   val <- array(vwpair.values())
   n <- length(weight)
   if n is 0 or weight is 0:
     return 0
   if wt[n-1] greater than weight:
      vwpair.remove(n-1)
      return knapsack(W, vwpair)
   else:
      vwpair.remove(n-1)
      return max(
         val[n - 1] + knapsack(
            W - wt[n - 1], vwpair),
         knapsack(W, vwpair))
```

5.a.
```
e(S,M,i,j):
space=M-j+1
for k in (i,j+1)
```

```
        Reduce space length of S
Return space
```

c.

```
bl(S,Mi,j):
space=e(S,M,i,j)
if space<0
        return infinity
return space
```

e.

```
mb(S, M):
   if bl(S, M, 0, length(S)-1) == M and bl(S, M, 0, length(S)-1) == -1:
      return 0
   else:
      for j in (length(S)-1, 0, -1):
         badness = bl(S,M,0,j)
         if badness == -1:
            nextLineBadness = mb(S[j+1 : length(S)], M)
            if nextLineBadness > badness:
                  return nextLineBadness
            else:
                  return badness
```

f.

```
mb`(S, M, i):
  if bl(S[i:],M,0,length(S[i:])-1) equals M and bl(S, M, 0, length(S[i:])-1) equals -1:
     return 0
  else:
     for j in (length(S[i:])-1, 0, -1):
        badness <- bl(S[i:],M,0,j)
        if badness equals -1:
           nextLineBadness <- mb`(S, M, j+1)
           if nextLineBadness > badness:
                 return nextLineBadness
           else:
                 return badness
```

6.

The d values are (with vertex z):-

S	T	X	Y	Z
∞	∞	∞	∞	0
2	∞	7	∞	0
2	5	7	9	0

2	5	6	9	0
2	4	6	9	0

The π values are :-

s	t	x	y	z
NIL	NIL	NIL	NIL	NIL
z	NIL	z	NIL	NIL
z	x	z	s	NIL
z	x	y	s	NIL
z	x	y	s	NIL

After changing the weight of edge (z,x) to 4 and rerunning with s as the source , we have the following table:-

s	t	x	y	z
0	∞	∞	∞	∞
0	6	∞	7	∞
0	6	4	7	2
0	2	4	7	2
0	2	4	7	-2

The π values are :-

s	t	x	y	z
NIL	NIL	NIL	NIL	NIL
NIL	s	NIL	s	NIL
NIL	s	y	s	t
NIL	x	y	s	t
NIL	x	y	s	t