Algorithms Homework 5

1. (a) $(((A_1 \times A_2)(A_3 \times A_4))(A_5 \times A_6))$

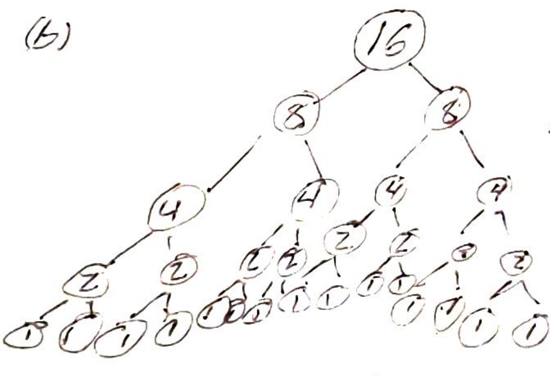
(b) Base Case: $(A_1 \times A_2)$: One set of Parentheses

Induction Step: Suppose for some $k \geq 1$ that each integer n with $1 \leq n \leq k$ $(A_1 \ldots A_k)$

Prove:

2 (a) The more efficient way of determining optimal number of multiplications is enumerating all possible multiplications. The recursive approach has to run its recursion on each subproblem in the recursion tree more than once.

(b)



Memoization fails to speed up a problem like merge sort because there is limited repeating of problems in the way memoization supports.

(C.) This problem does exhibit optimal substructure, If any of the subproblems had a better solution than a given solution that would have a larger final cost.

4. (a) 0-1 KnapSack(item :: items, capacity)

```
if capacity == 0 or item == null
    return 0
if item[0] > capacity
    return 0-1 KnapSack(items, capacity)
else
    return largest of (0-1 KnapSack(items, capacity),
        item[0] + 0-1 KnapSack(items, capacity - item[0]))
```

5. (a) e(S, M, i, j)

```
    return M - j + i - WordSum(S)
WordSum(S1 :: S2)
    return length(S) + WordSum(S2)
```

(C) bl(S, M, i, j)

```
    if e(S, M, i, j) ≥ 0
        return e(S, M, i, j)
    else
        return inf
```

(e) mb(S, M)

```
    mbHelper(S, M, 0, 0)

mbHelper(S, M, i, j)
    if j == length(S)  # if we get j to len(S) without "breaking" then thats the last line
        return 0
    else if bl(S, M, i, j) == inf
        return bl(S, M, i, j) + mbHelper(S, M, j-1, j-1)
    else
        return mbHelper(S, M, i, j+1)
```

(f) mb'(S, M, i)

return mbHelper(S, M, 1, i) mb helper unchanged from e