1. Take the cost of mult(A, B, l, m, n) to be

$$2lmn + ln$$

(i.e. lmn additions, lmn multiplications and ln zero-assignments). [Arguably, you might want to worry about the loop control, but that's a detail.]

Let $n = 3$ and suppose l_3 is even but $l_1, l_2 l_4$ are odd.

Cost of multiplication using bracketing $(A_1 \cdot A_2) \cdot A_3$ is.

$$c = (2l_1 l_2 l_3 + l_1 l_3) + (2l_1 l_3 l_4 + l_1 l_4)$$

Cost of multiplication using bracketing $A_1 (A_2 \cdot A_3)$ is

$$d = (2l_2 l_3 l_4 + l_2 l_4) + (2l_1 l_2 l_4 + l_1 l_4)$$

But c is odd and d is even.

2. We shall keep $n \times n$ upper-triangular matrices M, N: $M_{i,j}$ represents the cost of computing the product $A_i \cdot \cdots \cdot A_j$ under the best possible bracketing, and $N_{i,j}$ records the index of the last array in the left-hand major operand of this bracketing.

We start by assigning the diagonal of A:

$$M_{i,i} = 0$$

since no multiplications are required in this case.

Now suppose $M_{i,j}$ has been assigned for all i,j $(1 \le i \le n-l,\ j = i+l)$ where $0 \le l < n-1$. We proceed to assign $M_{i,j}$ for all i,j in the same range, but with l replaced by $l+1$.

Take any i $(1 \le i \le n-(l+1))$ and let $j = i+(l+1)$. The best bracketing of $A_i \cdots A_j$ is

$$(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$$

for some k $(i \le k < j)$. The cost of computing the left operand is M_{ik} and the cost of computing the right operand $M_{k+1,j}$. So the total cost is

$$c_{ijk} = M_{ik} + M_{k+1\,j} + 2l_i l_{k+1} l_j + l_i l_j$$

Therefore we may assign

$$M_{ij} = \min\{c_{ijk} \mid i \le k < j\},$$

and we may set $N_{i,j}$ to be an index k where this minimum is achieved.

So the main algorithm is

```
ComputePlan ( l_1 ... l_n )

    for i = 1 to n                          } initialize diagonal
        M_{i,i} = 0
    for l = 1 to n-1
        for i = 1 to n-l
            j = i + l
            c = ∞
            for k = i to j-1
                c' = ( M_{ik} + M_{k+1 j} + 2 l_i l_{k+1} l_j + l_i l_j )
                if c > c'                   } calculate min cost in
                    c = c'                  } c and index where
                    s = k                   } achieved in s
            M_{i,j} = c                     } record min cost
            N_{i,j} = s                     } record index where min achieved
    return (M, N)                           } return
```

This computes all the information required to read of the multiplication plan. The total best cost will be stored in $M_{1,n}$.

We probably want a read-out algorithm which takes N and prints out a nice tree, say, as a string. Here is one possibility

```
readOut (n, N)                          } If no multiplications, just
    if  n = 1                           } print A_1
        return  "A_1"
    else                                } top level: no surrounding
        k := N_{1,n}                    } parentheses.
        return readOut1(1, k, N) + "·" + readOut1(k+1, n, N)

readOut1 (i, j, N)                      } not at top level;
                                        } remember parentheses
    if  j = i+1
        return  "(A_i · A_{i+1})"
    else
        k := N_{i,j}
        return "(" + readOut1(i, k) + "·" + readOut1(k+1, j) + ")"
```

Thus, the whole algorithm is

```
multiplicationPlan (l_1, ..., l_n)
    (M, N) = computePlan(l_1, ..., l_n)
    return readOut (n, N)
```

3. From computePlan, it is obvious that the algorithm runs in time $O(n^3)$.