```
#include <stdio.h>
#include <stdlib.h>

#define MAX 80

/* ------------------------------------------------------------ */
/*   i n i t _ a r r a y ( )                                    */
/* ------------------------------------------------------------ */
int ** init_array( int n )
{
   int i, j   ;
   int n1     ;
   int ** t   ;

   n1 = n+1 ;  /* For use with origin 1 indexing.  */
   t = (int **) malloc( n1 * sizeof(int *) )  ;

   for ( i = 0 ; i < n1 ; i++ ) {
      *(t+i) = (int *) malloc( n1 * sizeof(int) ) ;
   }

   for ( i = 0 ; i < n1 ; i++ ) {
      for ( j = 0 ; j < n1 ; j++ ) {
          t[i][j] = 0 ;
      }
   }
   return(t) ;
}

/* ------------------------------------------------------------ */
/*   r e a d _ s i z e s ( )                                    */
/* ------------------------------------------------------------ */
int * read_sizes( int * np )
{
   int i, n, n1 ;
   int * t ;

   scanf( "%d", &n ) ;
   *np = n ;
   n1 = n + 1 ;
   t = (int * ) malloc( n1 * sizeof(int) ) ;
   for ( i = 0 ; i < n1 ; i++ ) *(t+i) = 0 ;
   for ( i = 0 ; i < n1 ; i++ ) scanf( "%d", (t+i) ) ;
   return(t) ;
}

/* ------------------------------------------------------------ */
/*   s h o w _ M ( )                                            */
/* ------------------------------------------------------------ */
void show_M( int n, int ** M )
{
   int ell, i, j ;

   /* We show the table M[i][j] in order of increasing      */
   /* difference between j and i.  I.e., ell = j - i        */

   for ( ell = 0 ; ell < n ; ell++ ) {
      for ( i = 1 ; i <= (n-ell) ; i++ ) {
         j = i + ell ;
         printf( "%12d", M[i][j] ) ;
      }
      printf( "\n" ) ;
   }
}
```

```
/* ------------------------------------------------------------ */
/*   o p t o ( )                                                */
/* ------------------------------------------------------------ */
void  opto( int n, int * r, int ** M, int ** K )
{
   int i, j, k, ell  ;
   int k_hat ;
   int v, v_hat ;

   for ( i = 1 ; i <= n ; i++ ) M[i][i] = 0 ;

   for ( ell = 1 ; ell < n ; ell++ ) {
       for ( i = 1 ; i <= (n - ell) ; i++ ) {
          j = i + ell ;
          /* Find minimum. */
          /* Get minimum started for the first value of k.      */
          k = i ; k_hat = k ;
          v = M[i][k] + M[k+1][j] + r[i-1]*r[k]*r[j] ;  v_hat = v ;
          /* Loop over remaining values of k to find  minimum.  */
          for ( k = (i+1) ; k < j ; k++ ) {
             v = M[i][k] + M[k+1][j] + r[i-1]*r[k]*r[j] ;
             if ( v_hat > v ) {
                v_hat = v ;
                k_hat = k ;
             }
          }
          M[i][j] = v_hat ;
          K[i][j] = k_hat ;
       }
   }
}

/* ------------------------------------------------------------ */
/*   p r e t t y _ p r i n t ( )                                */
/* ------------------------------------------------------------ */
void pretty_print( int i, int j, int ** M , int ** K, char buff[], int pflag )
{
   char L_buff[MAX] ;
   char R_buff[MAX] ;
   int k_hat  ;

   if ( i == j ) {
      sprintf( buff, "M_%d", i ) ;
   }
   else {
       k_hat = K[i][j] ;
       pretty_print( i, k_hat, M , K, L_buff, 1 ) ;
       pretty_print( k_hat+1, j, M , K, R_buff, 1 ) ;
       if (pflag) {
          sprintf( buff, "( %s * %s )", L_buff, R_buff ) ;
       }
       else {
          sprintf( buff, " %s * %s ", L_buff, R_buff ) ;
       }
   }
}
```

```
/* ------------------------------------------------------------ */
/*   M A I N ( )                                                */
/* ------------------------------------------------------------ */
int main()
{
    int * r ;
    int i, n ;
    int ** M ;
    int ** K ;
    char buffer[MAX] ;

    r = read_sizes( &n ) ;

    M = init_array( n ) ;
    K = init_array( n ) ;

    opto( n, r, M, K ) ;

    printf("\n\n") ;
    printf("M:\n\n" ) ;
    show_M( n, M ) ;

    printf("\n\n") ;

    pretty_print( 1, n, M, K, buffer, 0 ) ;

    printf( "Optimal ordering:   %s\n", buffer ) ;

   printf("\n\n") ;
}
```

```
hoku% optmm
4
10 20 50 1 100


M:

           0           0           0           0
       10000        1000        5000
        1200        3000
        2200


Optimal ordering:    ( M_1 * ( M_2 * M_3 ) ) * M_4
```