Question 1A

Write a recursive formula for computing the optimal value for the total bonus (i.e., define the variable that you wish to optimize and explain how a solution to computing it can be obtained from solutions to subproblems).
Submit: the recursive formula, along with definitions and explanations on what is computed.

Solution

Given the length of the garden (L), the locations of the tables(x_1 ... x_n), and the bonuses given per each table (b_1 ... b_n), we can use dynamic programming to calculate the optimal total bonus from all the possible floral arrangements. Using the recursive algorithm provided below, it is possible to calculate the optimal bonus in a top-down approach using memoization.

This problem is considered a dynamic programming one, both because it can first be broken down into smaller problems (overlapping subproblems) and then the subproblems optimized together to find the overall solution (optimal substructure). As we recurse through the table locations, we are attempting to calculate the maximum bonus possible from the left and only save the specific floral arrangements that maximize the bonus from that point backwards. I use a dictionary of max values called memo that stores this information during the recursion.

Algorithm

The first part of the algorithm provides base cases for returning from a recursive call. It can occur either because the provided current index is less than 0 and no longer valid, or because the subproblem bonus has already been calculated and returns the cached value.

The second part of the function `solve_recurse()` contains the recursive part of the algorithm. There are two variables representing the options of either placing flowers on the current table, or on one table before represented by the `place_first` and `place_second` integer variables, respectively. If placed on the current table, the algorithm would recurse only on the next table at least 5 meters away, which is accounted for by the `get_left()` function. This function returns the index of the next valid table to place flowers on.

Each one of these recursions gives a subproblem with a bonus value that can be memoized into a dictionary for quicker calculations in other subproblems. The final result returned from the algorithm is the max bonus calculated from all the subproblems.

```
memo = {}

// Recursively solves the problem by placing flowers on the first table
// or skipping the first table and placing on the second, using memoization
function solve_recurse(locations, values, x):
   if x < 0:
       return 0
   if x in memo:
       return memo[x]

   place_first = solve_recurse(locations, values, get_left(locations, x)) +
values[x]
   place_second = solve_recurse(locations, values, x - 1)

   // Memoize the maximum bonus for the sub-problem
   memo[x] = max(place_first, place_second)
   return memo[x]

// Iterates backwards from the currentIndex to find the table index
// that is > 5 units away by subtraction of x-positions
function get_left(locations, current):
   for i from (current - 1) to 0:
       if locations[current] - locations[i] > 5:
           return i
   return -1
```

Question 3

List all of the different orders in which we can multiply five matrices A, B, C, D, and E.

The different ways to multiply 5 matrices A B C D E can be expressed via encapsulating them with parenthesis, the following are the 14 different ways these matrices can be multiplied:

1. (A (B ((C D) E)))
2. (A ((B C) (D E)))
3. (A (B (C (D E))))
4. (A ((B (C D)) E))
5. ((A B) (C (D E)))
6. ((A (B C)) (D E))
7. ((A (B (C D))) E)
8. ((A ((B C) D)) E)

9. ((A (B C)) (D E))
10. ((A B) ((C D) E))
11. (((A B) C) (D E))
12. (((A (B C)) D) E)
13. ((((A B) C) D) E)
14. (((A B) (C D)) E)