Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, and if applicable, efficiency, as well as correctness. Out of 80 points.

**Backtracking and Dynamic Programming**

For the problem below, a correct backtracking recursive algorithm is given, with an example that illustrates how the algorithms works.

Consider the following problem (a version of which was on the midterm).

Smallville is the last city on Earth not saturated by Big Bucks coffee shops. Smallville has one business street with \( n \) blocks. The profit associated with putting a coffee shop on block \( i \) is given in an array \( \text{Profit}[i] \). However, we cannot put coffee shops within \( d \geq 1 \) blocks from each other, i.e., if we put a shop in block \( i \) then we cannot put one in block \( i-d, i-d+1, i-1 \) or \( i+1, i+2, ... i+d \).

An backtracking algorithm for computing the maximum total profit of BigBucks coffee shops is as follows:

\[
\text{BTBigBucks}(d, \text{Profit}[1..n])
\]

1. IF \( n = 0 \) return 0.

2. IF \( n \leq d + 1 \) return \( \max_{1 \leq i \leq n} \text{Profit}[i] \)

3. \( \text{Case 1} \leftarrow \text{Profit}[1] + \text{BTBigBucks}(d, \text{Profit}[d+2..n]) \) \{If we put a shop in block 1, we cannot put one in \( 2, 2+d \) \}

4. \( \text{Case 2} \leftarrow \text{BTBigBucks}(d, \text{Profit}[2..n]) \) \{If we don’t put a shop in block 1, there are no other restrictions\}

5. Return \( \max(\text{Case 1}, \text{Case 2}) \)

**Part 1: 10 points** Illustrate this algorithm on the following inputs: \( d = 2, n = 8, \text{Profit}[1..8] = 2, 4, 3, 7, 8, 4, 7, 5 \) (as a tree of recursive calls and answers).

**Part 2: 20 points** Give an upper bound on the number of recursive calls the above algorithm makes, in the worst-case. (Some points will be based on how tight the bound is. Be sure to expain your answer.)

**Part 3: 30 points** Give a dynamic programming version of the recurrence.

**Part 4: 10 points** Give a time analysis of this dynamic programming algorithm.

**Part 5: 10 points** Show the array that your dynamic programming algorithm produces on the above example.