1. Consider Problem 1 from Lecture 5 in Jeff Erickson’s notes.

(a) Solve Problem 1(a) from Jeff Erickson’s notes.

(b) Now we want to solve the problem of using the fewest number of bills to make \( k \) Dream Dollars. Let \( D[1..8] \) denote the size-8 array that holds the given denominations: so \( D[1] = 1, D[2] = 4, D[3] = 7, \) etc. For any \( k', 0 \leq k' \leq k \) and \( j, 1 \leq j \leq 8, \) let \( C(k', j) \) denote the fewest number of bills from denominations in \( D[1..j] \) that make change for \( k' \) Dream Dollars. Write down a recurrence for \( C(k', j), \) for \( 0 \leq k' \leq k, \) \( 1 \leq j \leq 8. \) Make sure that the base cases are all carefully specified. 

Hint: The trivial observation is that in the optimal change for \( k' \) using denominations in \( D[1..j], \) we either use a bill with denomination \( D[j] \) or we don’t.

(c) The recurrence from (b) can be implemented as a recursive function, though you don’t need to do this. Now think about the memoized version of this recursive function using a 2-dimensional \((k + 1) \times 8\) table in which the table-slot \( \text{Table}[k', j] \) is filled with \( C(k', j). \) Figure out the order in which this table is filled and then write pseudocode for a function that finds and returns the fewest number of bills needed to make change for \( k \) Dream Dollars, when the denominations come from \( D[1..8]. \) This function uses two nested loops to fill out the table.

(d) Write a function that takes as input \( k, D, \) and \( \text{Table} \) (filled out using the function in (c)) and returns the optimal set of bills of denominations \( D[1..8] \) used to make change for \( k. \)

2. You are given a an array \( A[1..n] \) of numbers (which can be positive, 0 or negative). You need to design an algorithm that finds a contiguous subsequence of \( A \) with largest sum. (This is just a restatement of Problem 2(a) in Jeff Erickson’s Lecture 5.) For example, given the array \([-6, 12, -7, 0, 14, -7, 5]\), the contiguous subsequence \([12, -7, 0, 14]\) has the largest sum, 19.

(a) For \( 0 \leq j \leq n, \) let \( S(1, j) \) denote the largest sum of a contiguous subsequence from \( A[1..j], \) such that the contiguous subsequence includes \( A[j]. \) For \( 0 \leq j \leq n, \) let \( S(2, j) \) denote the largest sum of a contiguous subsequence from \( A[1..j]. \) Write down recurrences for \( S(1, j) \) and \( S(2, j). \) Make sure that you take care of all the base cases. 

Hint: To figure out the recurrence for \( S(2, j), \) start with the trivial observation that either \( A[j] \) is included in the contiguous subsequence with largest sum or it is not. Note that \( S(2, j) \) may depend on \( S(1, \cdot) \).

(b) The recurrence from (a) can be implemented as a recursive function, though you don’t need to do this. Now think about the memoized version of this recursive function using a 2-dimensional \( 2 \times (n + 1) \) table in which the table-slot \( \text{Table}[i, j] \) is filled with \( S(i, j), \) where \( i \in \{1, 2\} \) \( 0 \leq j \leq n. \) Figure out the order in which this table is filled and then write pseudocode for a function that finds and returns the largest sum of a contiguous subsequence of \( A[1..n]. \)

(c) Write a function that takes as input \( A \) and \( \text{Table} \) (filled out using the function in (b)) and returns the optimal contiguous subsequence from \( A[1..n]. \)