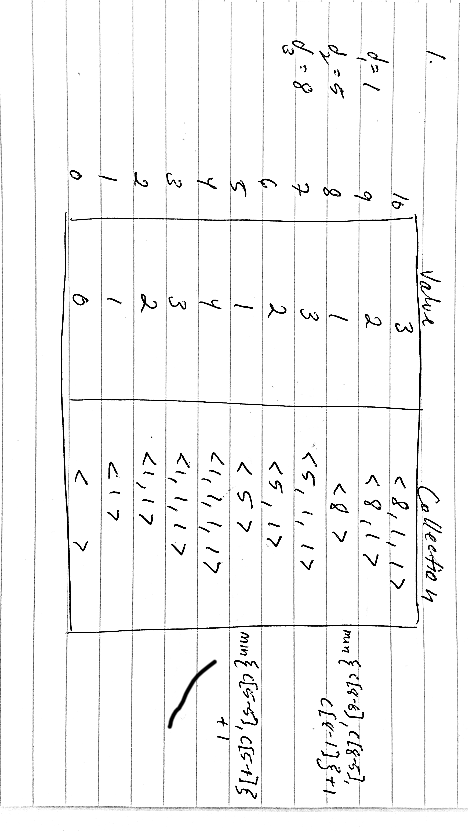
Brian Bauman

**CSC 421 - Assignment 3**

1.

2.

a.

C[i, j] = { 1 when i = 1,

1 when i = j,

C[i, j - 1] + C[i -1, j - 1] when i > 1 and i != j }

b.

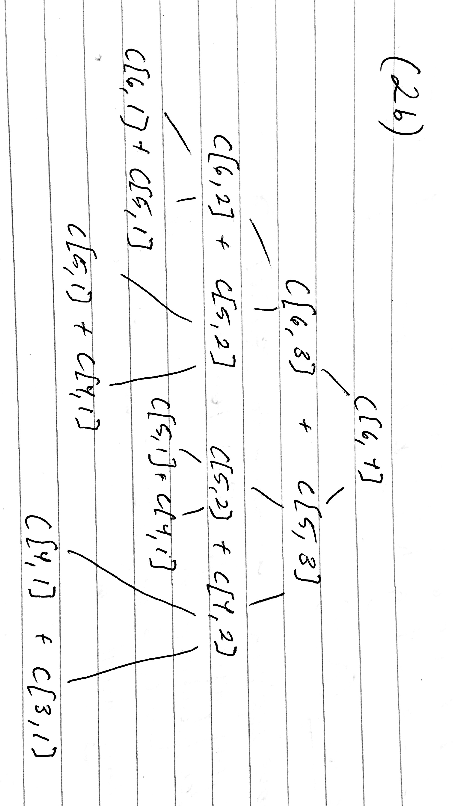
computePascal(i, j):

if (i = 1 or i = j) then

return 1

else

return computePascal(i, j - 1) + computePascal(i - 1, j - 1)



As you can see, a recursive algorithm would perform overlapping computations.

c.

pascalsTriangle(n):

declare array of arrays, P, of length n

for i = 1 to n do

P[i] = new array of length i, populated with 1s

for i = 1 to n do

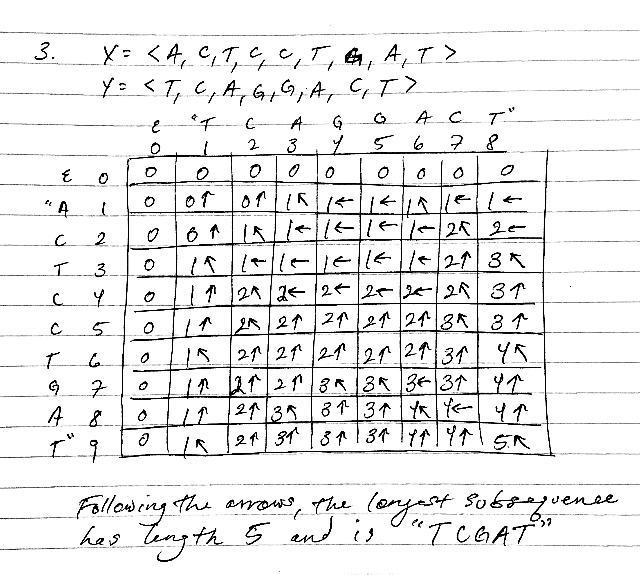
for j = 1 to i do

if (j > 1 and j != i) then

P[i][j] = P[i - 1][j] + P[i - 1][j - 1]

This algorithm runs in O(n^2) time as both for loops execute over 1/2 \* n^2 elements.

3.



4.

(using merge sort and longest common subsequence algorithms discussed in class)

longestIncreasingSubsequence(S):

T = mergeSort(S); (runs in O(n\*lgn) time)

return longestCommonSubsequence(S, T); (runs in O(n^2) time)

5.

a.

T[0, k] can easily be computed as “dagger” (or the empty set) for all values of k = 0, …, s. This is because no you cannot ever sum to s when starting with an empty set of integers.

b.

When T[i, s’] exists and the element s\_i does not belong to this set, we can express T[i, s’] as equal to the set contained in T[i - 1, s’]. When T[i, s’] does include s\_i, it can be expressed as the set contained in T[i, s’ - s\_i] unioned with the value s\_i.

c.

subsetSum(S, t):

s = sum of all n values in S

define T[] as a two dimensional array of size [n, s]

initialize T[0,k] = dagger for all values k = 0, …, s

for i = 1 to n do

for j = 0 to s do

if (T[i - 1, j] != dagger) then

T[i, j] = T[i - 1, j]

else if T[i - 1, j - s\_i] != dagger then

T[i, j] = T[i - 1, j - s\_i] U s\_i

As is clear from the nested for loop, this algorithm runs in O(n\*s) time.