1. Pascal's triangle looks as follows:

   1

   1 1

   1 2 1

   1 3 3 1

   1 4 6 4 1

   ...

   The first entry in a row is 1 and the last entry is 1 (except for the first row which contains only 1), and every other entry in Pascal's triangle is equal to the sum of the following two entries: the entry that is in the previous row and the same column, and the entry that is in the previous row and previous column.

   (a) Give a recursive definition (relation) for the entry C[i, j] at row i and column j of Pascal's triangle. Make sure that you distinguish the base case(s).

   Assuming C[1,1] is the first element in the triangle

   If j=1 or i=j, C[i,j]=1

   Else C[i,j] = C[i-1,j] + C[i-1,j-1]

   (b) Give a recursive algorithm to compute C[i, j], i ≥ j ≥ 1. Illustrate by drawing a diagram (tree) the steps that your algorithm performs to compute C[6, 4]. Does your algorithm perform overlapping computations?

```
Pascal(i,j)
    if (j==1 || j==i) then return 1
    else return Pascal(i-1,j) + Pascal(i-1,j-1);
    end
```

```
                     C[6,4]

                C[5,4]          C[5,3]

        C[4,4]    C[4,3]        C[4,3]   C[4,2]

1       C[3,3]   C[3,2]         C[3,3] C[3,2]         C[3,2]     C[3,1]         9+1=10

        1        C[2,2] C[2,1]  1      C[2,2] C[2,1]  C[2,2] C[2,1]    1        6+3=9

                 1       1             1       1      1       1        1        6
```

C[6,4]=10

(c) Use dynamic programming to design an O(n^2) time algorithm that computes the first n rows in Pascal's triangle.

```
for i=1 to n do
    for j=1 to n do
        if(j==1 || j==i) then C[i,j]=1;
        else C[i,j] = C[i-1,j] + C[i-1,j-1];
        end
    end
end
```

2. In a previous life, you worked as a cashier in the lost Antarctican colony, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource in Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change.

(a) Suppose that the currency of the colony was available in the following denominations: 1, 4, and 6. Consider an algorithm that repeatedly takes the largest bill that does not exceed the target amount. For example, to make 11 using this algorithm, we first take a 6 bill, then a 4 bill, and finally a 1 bill. Give an example where this greedy algorithm uses more bills than the minimum possible.

Assume the target is 21, according to this greedy algorithm, it goes 6,6,6,1,1,1, which makes 6 bills. It isn't the minimum possible, we can choose 6,6,4,4,1 i.e., 5 bills

(b) Describe and analyze a recursive algorithm that computes, given an integer n and an arbitrary system of k denominations <d1 = 1, . . . , dk>, the minimum number of bills needed to make the amount n.

1. *Sort the array of bills D*
2. *Initialize min=0; target=n;*
3. *Start from rear end of the array D for each d(i)*
4. *If target becomes 0, then return min.*
5. *Find the largest denomination in D that is smaller than target.*
6. *Add 1 to min. , as a bill is selected*
7. *Subtract value of found denomination from target and recursively call the alg with new target*

MinBills(D,n)

Sort array D of length k

min=0; target=n;

```
        for(int i=k-1;i<=0;i--) do
            if( target ==0 ) then return min;
            else if(target>=D[i]) then {
                min=1+MinBills(D, target-D[i]);


            }
            end
    end
```

(c) (10 points) Describe a dynamic programming algorithm that computes, given an integer n and an arbitrary system of k denominations hd1 = 1, . . . , dki, the minimum number of bills needed to make amount n

```
        MinBills(D,n)
        target=n;
        var B = [];//Hash table
        B[n]=0;
        for(int i=k-1;i<=0;i--) do
            if( target ==0 ) then return B[n];
            else if(target>=D[i]) then {
                if(B[target-D[i]] !=undefined){
                    B[n]=1+ B[target-D[i]]    ;
                }
                Else{
                    B[target-D[i]] = MinBills(D, target-D[i]);
                    B[n]=1+ B[target-D[i]];
                }
            }
            end
    end
```

3. Suppose that you are given an array A[1..n] of numbers, which may be positive, negative, or zero, and which are not necessarily integers. Give an O(n)-time algorithm that finds the largest sum of

elements in a contiguous subarray A[i..j] of A. For example, given the array [−6, 12, −7, 0, 14, −7, 5] as input, your algorithm should return 19, which is the content of A[2..5].

Algorithm(A)

```
int max=0; sum=0;
for(int i=0;i<n;i++) do
        sum= sum+A[i];
        if(sum > max) then max=sum; // Replacing with the new max
        if(sum < 0) then sum = 0  // Resetting on the occurrence of negative sums
end
return max;
```

We visit each element only once so it’s a O(n) algorithm.

In the case of all the numbers in the array are negative, max would be the largest element

Algorithm(A)

```
int max=A[0]; sum=A[0];
for(int i=1;i<n;i++) do
        sum= Math.max(A[i], sum+A[i]); // Setting to largest sum
        max = Math.max(max, sum); //Max of both
end
return max;
```

4. A subsequence of a sequence is anything obtained from a sequence by extracting a subset of elements, but keeping them in the same order; the elements of the subsequence need not be contiguous in the original sequence. For example, the strings C, DAMN, YAIOAI, and DYNAMICPROGRAMMING are all subsequences of the string DYNAMICPROGRAMMING.

(a) Let A[1..m] and B[1..n] be two arbitrary arrays. A common subsequence of A and B is another sequence that is a subsequence of both A and B. A longest common subsequence of A and B is a subsequence of A and B of maximum length. 2 For example, if A = hCT GCGT GT Ci and B = hGT CGT GGCi, then the length of the longest common subsequence of A and B is 6, and the sequence hT CGT GCi is such a longest common subsequence of A and B. Describe an O(nm)-time algorithm to compute the length of the longest common subsequence of two given sequences A and B. (Hint. You may modify the Edit Distance algorithm so that you consider only the two operations: Insert and Delete. What is the connection between this variant of Edit Distance and the problem of computing a longest common subsequence?)

To solve this problem using only Insert and Delete cases of Edit Distance problem

Let's consider two cases

- *If both arrays A and B end with the same element then we can divide into subproblems, so longest common subsequence (LCS) of A and B would be LCS(A[1…m], B[1…n]) = LCS(A[1…m-1], B[1…n-1]) + A[m] or LCS(A[1…m], B[1…n]) = LCS(A[1…m-1], B[1…n-1]) + B[n], here A[m]=B[n]*
- *If the last element of both arrays isn't the same, then it we compute*
    - *LCS(A[1…m-1], B[1…n])*
    - *LCS(A[1…m], B[1…n-1])*

  The longest sequence among the above two would be the solution

  So LCS(A[1…m], B[1…n]) = max (LCS(A[1…m-1], B[1…n]), LCS(A[1…m], B[1…n-1])

Algorithm LCS(A,B)

```
Int S[][]; // Stores the length of LCS of all elements of A and B

if (m == 0 || n == 0) then return 0;

For(int i=0;i<=m;i++) do

        For(int j=0;j<=n;j++) do

                If(i==0 || j==0) then S[i][j] =0;

                If(A[m]==B[n]) then S[i][j] = S[i-1][j-1]+1;

                Else S[i][j] = max(S[i-1][j], S[i][j-1])

                end

        end

end

return S[m][n];
```

(b) A palindrome is any sequence that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA. Use part (a) above to give an $O(n^2)$-time algorithm to find the length of the longest subsequence of a given sequence of length n that is also a palindrome

For a sequence A[1...n], the longest subsequence to check if it's a palindrome, we evaluate two cases

- *If the first and last characters are same, then LPS(i,j) = LPS(i+1,j-1) + 2*
- *Else we compute the max by eliminating the first element or last element, then*

  LPS(i,j) = max(LPS(i+1,j), LPS(i,j-1))

Using algorithm in (a)

Algorithm LPS(A, i, j)

```
if (i>j) then return 0; // Not a palindrome
if (i==j) then return 1; // Only one element
if(A[i]==A[j]) then
        return LPS(A, i+1, j-1) +2; // First and last match, add two chars to the length
else return max ( LPS(A, i+1,j), LPS(A, i,j-1))
```