ASSIGNMENT NO-3

NAME: NIHAR MUNIRAJU
EMAIL: nmuniraj@depaul.edu.

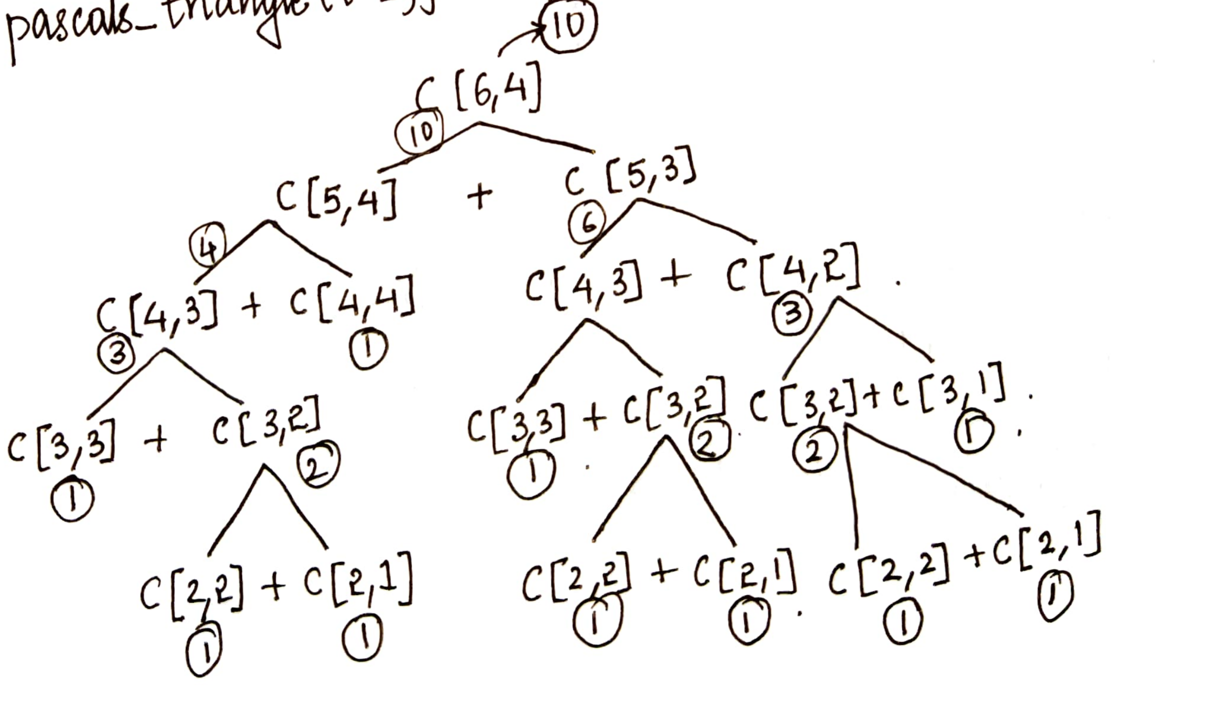
1 a) Recursive definition for Pascal's Triangle C[i,j] at row i and column j of Pascal's Triangle.

$$C[i,j] = \begin{cases} 1 & \text{if } j=1 \\ 1 & \text{if } i=j \\ C(i-1,j-1) + C(i-1,j) & \text{for } j \neq 1 \text{ and } i \neq j \end{cases}$$

⇒ Pascal's Triangle is a triangular array of the binomial co-effecients
⇒ It takes in an integer value n as input and prints first n lines of the Pascal's Triangle.

1 b) To compute C [i, j], $i \geq j \geq 1$ to perform an algorithm C [6,4]

```
Pascals_Triangle [i,j]
{
 if (j==1)
  return 1
elseif (i==j).
  return 1
else
 return pascals_triangle (i-1,j-1)+ pascals_triangle (i-1,j).
}
```



1c) def print_pascal~~___~~_triangle (rows):

```
left = [1];
right = [1, 1];
triangle = [left, right];
r = []
if rows == 1:
left [0] = str (left [0]).
print(' '.join(left))
elif rows == 2:
for o in triangle:
for a in range (len (o)):
  o[a] = str (o[a])
print((' ')*(2-(a+1)), (' '.join (o)))
else:
for i in range (2, rows):
triangle.append ( [1]*i).
for n in range (1,i):
triangle [i] [n]= (triangle [i-1][n-1] + triangle [i-1][n]).
triangle [i].append (1).
for x in range (len (triangle)):
for y in triangle [x]:
  s = str[y]
r.append(s).
print ((' ')*(rows-(x+1)), (' '.join(r))).
  r = [].
print_pascal_triangle (5).
```

2 a) Let us consider denominations are 1, 4 and 6. So in the lost Antarctican Colony the cashier has to hand the change with maximum possible bills of denomination 6, in order to minimize the uses of paper to the consumers.

Pseudo Code of the Algorithm :-

The Greedy Algorithm Uses :

```
def change (change_amount)
{
 //number of papers with denomination 6.
  paper_6 = change_amount/6.
 remaining_amount = change_amount %6.

 //number of papers with denomination 4.
  paper_4 = change_amount /4
  remaining_amount = change_amount %4.
 total papers = paper_6 + paper_4 + remaining amount.
}.
```

2b) Let us use an recursive algorithm that computes, given an integer n and an arbitrary system of k denominations.

⇒ The minimum number of denominations for a value a can be used for computation of an arbitrary system by using recursive formula.

```
def denom{
  V == 0
  else
  V>0
  }
  for minCoins (coins [0...d-1], a)
```

④

print (min { 1 + minCoins (a - coin [i]) }.
where i varies from 0 to d-1 & coin [i] <= a

⇒ Since the solution Using the recursive algorithm is to be exponential. We can see that the main problem is been sub-divided one after the other. By trying to avoid overlapping computations for this specific Algorithm.

2c)

```
def minBills (denominations, amount):
  if amount in dp:
    return dp[amount].
  #base condition.          // Base Case when amount=0, return 0;
  if amount ==0:
    dp [0]=0
    return 0
  # initalize to INFINITY
  minPossible = float("inf")    // Return the value to the already
                                   calculated.
  for bill in denominations:
# current denomination <= amount
   if bill <= amount:              // bill is greater than the given
                                      amount.
  current_minimum = min Bills (denominations, amount - bill)
  if current_minimum +1 < min Possible:  // subtract from the amount
                                          & store it in a memory
  min Possible = current_minimum+1           to return;
 #return minimum possible number of bills.
  dp [amount]= min Possible
   return dp[amount].
  denominations = [1, 4, 6].
   amount = 9
   dp = {}
 print (min Bills (denominations, amount)).
```

3) Algo PseudoCode :-

final_max = 0, curr_max = 0
Loop from starting to end of an array:
i) curr_max = curr_max + A[i]
ii) if (final_max < curr_max) final_max = curr_max.
iii) if (curr_max < 0) curr_max = 0

Algorithm

```
double largest_cont (double A[], int n) // find largest sum of elements.
{
double final_max = 0;
double curr_max = 0;
for (int i = 0; i<n; i++)
  {
  curr_max = curr_max + A[i]; // new current max.
  if (final_max < curr_max)  // if new max is greater than previous replace.
      final_max = curr_max;
  if (curr_max < 0)   // if subarray is -ve then start Once more.
      curr_max = 0;
  }
if (maximum < 0) // To check all the elements are -ve & push it to the final max.
 {
 final max = maximum;
 }
return final_max.
}
```

⇒ Time Complexity of the Algo passes through two arrays one to check & find the maximum subarray sum inside the array and the second passes contains all the negative numbers and checks $O(f(n)) = O(2n + C) = O(n)$.

4] A Subsequence of a Sequence is anything obtained from the Sequence by extracting a subset of elements, but keeping them on Same Order

→ To Compute the Length of the Longest Common Subsequence of two given Sequences A and B, we can use the Edit Distance Algorithm.

→ We Initialize a matrix $d[0..m, 0...n]$ where $d[i, j]$ is the longest common Sequence of $A[1..i]$ and $B[1..j]$

→ we then compute $d[i, j]$ in a reverse Bottom up fashion, using the following recursive relation.

→ $d[i, j] = \max(d[i-1, j], d[i, j-1], d[i-1, j-1]+1$

if $A[i] = B[j]$

→ The Length of the longest Common Sequence of A and B is then given by $d[m, n]$. The running time of this Algorithm is $O(nm)$.

```
def longcommon_seq (A, B)
{
double A [X+1]=M;
double B[Y+1]=N;
double m,n;
for A in [X+1]:
  if{(m=0 !! n==0).
else
  (A[X-1]== B[Y-1];
elseif
  max (A [X-1][Y], [Y-1][X]);
  }
return max(double m, double n);
}.
```

4b) (7)

⇒ A palindrome is any Sequence that is exactly the same as its reversal, like I, or DEED or RACECAR or PLANACATACANAL PANAMA.

⇒ To find the length of the longest seomence in a subsequence of given length n that is also a palindrome, we can use the Edit Distance Algorithm.

⇒ We Initialize a matrix $d[0 \ldots n, 0 \ldots n]$ where $d[i,j]$ is the length of the longest Palindrome subsequence of $A[i..j]$.

⇒ We then compute $d[i,j]$ in a bottom up approach using the same recursive relation function where.

⇒ $d[i,j] = \max(d[i-1,j], d[i,j-1]+1$

if only $A[i] = A[j]$.

⇒ Then the length of the longest palindrome subsequence of A is then given by $d[n,n]$.

⇒ The running time of this Algorithm is $O(n^2)$.