Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

CODE :

```python
# Write a program to print fibonacci series upto n terms in python
num = 10
n1, n2 = 0, 1
print("Fibonacci Series:", n1, n2, end=" ")
for i in range(2, num):
  n3 = n1 + n2
  n1 = n2
  n2 = n3
  print(n3, end=" ")
print()
```

```python
# Python program to print Fibonacci Series
def fibonacciSeries(i):
 if i <= 1:
  return i
 else:
  return (fibonacciSeries(i - 1) + fibonacciSeries(i - 2))
num=10
if num <=0:
 print("Please enter a Positive Number")
else:
 print("Fibonacci Series:", end=" ")
for i in range(num):
 print(fibonacciSeries(i), end=" ")
```

OUTPUT :

1.Write a program non-recursive and recursive program to calculate Fibonacci . numbers and analyze their time and space complexity.

```python
# Write a program to print fibonacci series upto n terms in python
num = 10
n1, n2 = 0, 1
print("Fibonacci Series:", n1, n2, end=" ")
for i in range(2, num):
    n3 = n1 + n2
    n1 = n2
    n2 = n3
    print(n3, end=" ")

print()
```

```
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
```

```python
# Python program to print Fibonacci Series
def fibonacciSeries(i):
  if i <= 1:
    return i
  else:
    return (fibonacciSeries(i - 1) + fibonacciSeries(i - 2))

num=10
if num <=0:
  print("Please enter a Positive Number")
else:
  print("Fibonacci Series:", end=" ")
for i in range(num):
  print(fibonacciSeries(i), end=" ")
```

```
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
```

Write a program to implement Huffman Encoding using a greedy strategy.

CODE:

```
# Huffman Coding in python

string = 'BCAADDDCCACACAC'



# Creating tree nodes
class NodeTree(object):

    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def children(self):
        return (self.left, self.right)

    def nodes(self):
        return (self.left, self.right)

    def __str__(self):
        return '%s_%s' % (self.left, self.right)


# Main function implementing huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d


# Calculating frequency
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
```

```python
        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
    print(' %-4r |%12s' % (char, huffmanCode[char]))
```

OUTPUT:

```python
nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
    print(' %-4r |%12s' % (char, huffmanCode[char]))
```

```
 Char | Huffman code
----------------------
 'C'  |           0
 'A'  |          11
 'D'  |         101
 'B'  |         100
```

Write a program to solve a fractional Knapsack problem using a greedy method.

CODE:

```
# Structure for an item which stores weight and corresponding value of Item
class Item:
 def __init__(self, value, weight):
   self.value = value
   self.weight = weight
# Main greedy function to solve problem
def fractionalKnapsack(W, arr):
# sorting Item on basis of ratio
 arr.sort(key=lambda x: (x.value/x.weight), reverse=True)
 finalvalue = 0.0 # Result(value in Knapsack)
 for item in arr: # Looping through all Items
# If adding Item won't overflow, add it completely
   if item.weight <= W:
     W -= item.weight
     finalvalue += item.value
   else:
     finalvalue += item.value * W / item.weight
     break
 return finalvalue # Returning final value
# Driver's Code
if __name__ == "__main__":
 W = 50  # Weight of Knapsack
 arr = [Item(60, 10), Item(100, 20), Item(120, 30)]
 max_val = fractionalKnapsack(W, arr) # Function call
 print ('Maximum value we can obtain = {}'.format(max_val))
```

OUTPUT:

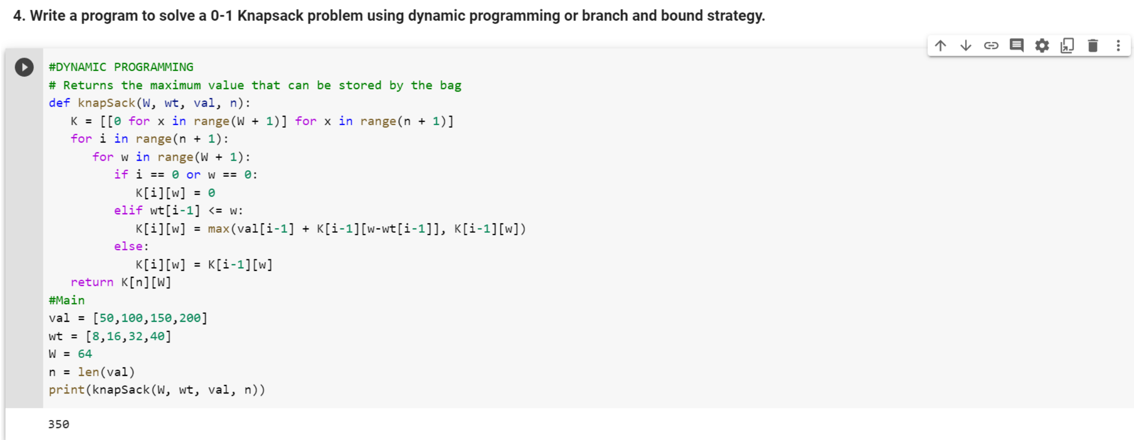
```
      W -= item.weight
      finalvalue += item.value

    # If we can't add current Item, add fractional part
    # of it
    else:
      finalvalue += item.value * W / item.weight
      break
  # Returning final value
  return finalvalue


# Driver's Code
if __name__ == "__main__":

  # Weight of Knapsack
  W = 50
  arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

  # Function call
  max_val = fractionalKnapsack(W, arr)
  print ('Maximum value we can obtain = {}'.format(max_val))

Maximum value we can obtain = 240.0
```

Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

CODE:

```
#DYNAMIC PROGRAMMING
# Returns the maximum value that can be stored by the bag
def knapSack(W, wt, val, n):
  K = [[0 for x in range(W + 1)] for x in range(n + 1)]
  #Table in bottom up manner
  for i in range(n + 1):
    for w in range(W + 1):
      if i == 0 or w == 0:
        K[i][w] = 0
      elif wt[i-1] <= w:
        K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
      else:
        K[i][w] = K[i-1][w]
  return K[n][W]
#Main
val = [50,100,150,200]
wt = [8,16,32,40]
W = 64
n = len(val)
print(knapSack(W, wt, val, n))
```

OUTPUT:



4. Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

```
#DYNAMIC PROGRAMMING
# Returns the maximum value that can be stored by the bag
def knapSack(W, wt, val, n):
   K = [[0 for x in range(W + 1)] for x in range(n + 1)]
   for i in range(n + 1):
      for w in range(W + 1):
         if i == 0 or w == 0:
            K[i][w] = 0
         elif wt[i-1] <= w:
            K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
         else:
            K[i][w] = K[i-1][w]
   return K[n][W]
#Main
val = [50,100,150,200]
wt = [8,16,32,40]
W = 64
n = len(val)
print(knapSack(W, wt, val, n))
```

```
350
```

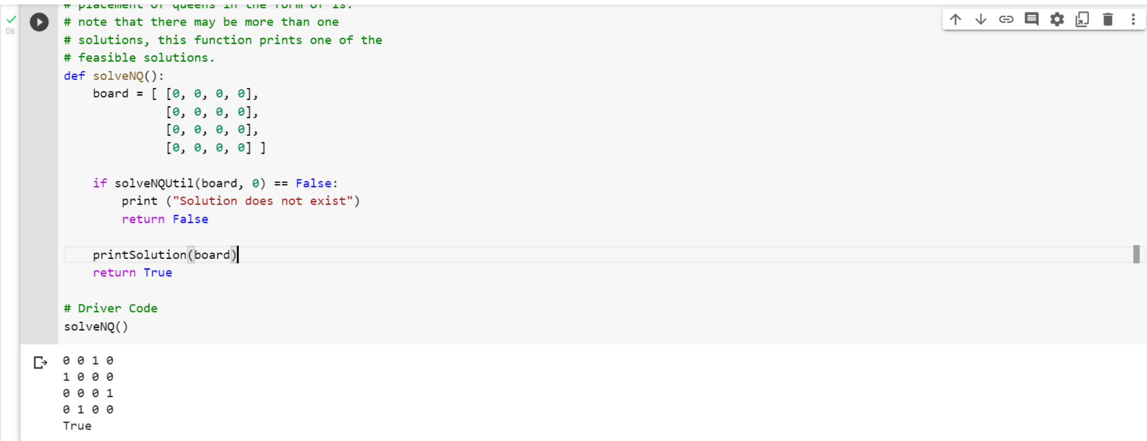
Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

CODE:

```
# Python3 program to solve N Queen
# Problem using backtracking
global N
N = 4
 def printSolution(board):
   for i in range(N):
       for j in range(N):
           print(board[i][j], end = " ")
       print(
def isSafe(board, row, col): # attacking queens
# Check this row on left side
   for i in range(col):
       if board[row][i] == 1:
           return False
# Check upper diagonal on left side
   for i, j in zip(range(row, -1, -1),
                   range(col, -1, -1)):
       if board[i][j] == 1:
           return False
# Check lower diagonal on left side
   for i, j in zip(range(row, N, 1),range(col, -1, -1)):
       if board[i][j] == 1:
           return False
   return True
 def solveNQUtil(board, col):
   # base case: If all queens are placed
   # then return true
   if col >= N:
       return True

   # Consider this column and try placing
   # this queen in all rows one by one
   for i in range(N):

       if isSafe(board, i, col):

           # Place this queen in board[i][col]
           board[i][col] = 1
```

```
            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True
                        # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0
# if the queen can not be placed in any row in
    # this column col then return false
    return False
  def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```

OUTPUT:



```
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
True
```

MINI PROJECT:
Implement the Naive String Matching Algorithm and Rabin-Karp Algorithm for string Matching. Observe difference in working of both algorithms for the same input.

CODE:
```
# Python program for Naive Pattern
# Searching algorithm
 def search(pat, txt):
    M = len(pat)
    N = len(txt)
    for i in range(N - M + 1): # A loop to slide pat[] one by one */
           j = 0
        # For current index i, check
        # for pattern match */
        while(j < M):
            if (txt[i + j] != pat[j]):
                break
            j += 1
           if (j == M):
            print("Pattern found at index ", i)
# Driver's Code
if __name__ == '__main__':
    txt = "GEEKS FOR GEEKS"
    pat = "GEEK"

    # Function call
    search(pat, txt)
```

OUTPUT :

CODE :

```python
# d is the number of characters in the input alphabet
d = 256

# pat  -> pattern
# txt  -> text
# q -> A prime number


def search(pat, txt, q):
    M = len(pat)
    N = len(txt)
    i = 0
    j = 0
    p = 0 # hash value for pattern
    t = 0 # hash value for txt
    h = 1

    # The value of h would be "pow(d, M-1)%q"
    for i in range(M-1):
        h = (h*d) % q

    # Calculate the hash value of pattern and first window
    # of text
    for i in range(M):
        p = (d*p + ord(pat[i])) % q
        t = (d*t + ord(txt[i])) % q

    # Slide the pattern over text one by one
    for i in range(N-M+1):
        # Check the hash values of current window of text and
        # pattern if the hash values match then only check
        # for characters one by one
        if p == t:
            # Check for characters one by one
            for j in range(M):
                if txt[i+j] != pat[j]:
                    break
                else:
                    j += 1

            # if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
            if j == M:
```

```
                print("Pattern found at index " + str(i))

        # Calculate hash value for next window of text: Remove
        # leading digit, add trailing digit
        if i < N-M:
            t = (d*(t-ord(txt[i])*h) + ord(txt[i+M])) % q

            # We might get negative values of t, converting it to
            # positive
            if t < 0:
                t = t+q


# Driver Code
if __name__ == '__main__':
    txt = "GEEKS FOR GEEKS"
    pat = "GEEK"

    # A prime number
    q = 101

    # Function Call
    search(pat, txt, q)
```

OUTPUT:

```
        # leading digit, add trailing digit
        if i < N-M:
            t = (d*(t-ord(txt[i])*h) + ord(txt[i+M])) % q

            # We might get negative values of t, converting it to
            # positive
            if t < 0:
                t = t+q


# Driver Code
if __name__ == '__main__':
    txt = "GEEKS FOR GEEKS"
    pat = "GEEK"

    # A prime number
    q = 101

    # Function Call
    search(pat, txt, q)

Pattern found at index 0
Pattern found at index 10
```