Q1.

```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
      print(n1)
      nth = n1 + n2
      # update values
      n1 = n2
      n2 = nth
      count += 1
```

#output:

```
How many terms? 8
Fibonacci sequence:
0
1
1
2
3
5
8
13
```

Q2

```
#Recursion  fibonacci using  python

def Fib(n):
  if n <= 1:
    return n
  else:
    return (Fib(n - 1) + Fib(n - 2))  # function calling itself(recursion)



n = int(input("Enter the Value of n: "))  # take input from the user
print("Fibonacci series :")
for i in range(n):
  print(Fib(i),end = " ")

 #permutation of string

def toString(List):
  return ''.join(List)
```

Output:

```
Enter the Value of n: 7
Fibonacci series :
0 1 1 2 3 5 8
```

Fibonacci number using iterative method time complixity O(n) but Recursive technique time complixity O(2^n), so more time recurred recursive method.

Q3.

```
def fibonacci(n, memo={}):
   if n in memo:
      return memo[n]

   if n <= 1:
      memo[n] = n
   else:
      memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)

   return memo[n]

# Example usage:
n = int(input("Enter the Value of n: "))  # Change this to the desired Fibonacci number you want to calculate
result = fibonacci(n)
print(f"The {n}-th Fibonacci number is: {result}")
```

output:
Enter the Value of n: 10
The 10-th Fibonacci number is: 55

Dynamic programming time complexity O(n) better than recursive method because in dynamic programming is memorization problem ,let f[0]=1,f[1]=1 f[2]=f[0]+f[1] ,f[2] store first time, next time does not calculate f[2].Already result store f[2] value.

Q4:

```
def toString(List):
  return ''.join(List)

# Function to print permutations
# of string
# This function takes three parameters:
# a. String
# l. Starting index of the string
# r. Ending index of the string.
def permute(a, l, r):
   if l == r:
      print (toString(a))
   else:
     for i in range(l, r + 1):
        a[l], a[i] = a[i], a[l]
        permute(a, l + 1, r)

        # backtrack
        a[l], a[i] = a[i], a[l]

# Driver code
string = "SUBRATA"
n = len(string)
a = list(string)
permute(a, 0, n-1)
```

output: Sample output

ABSARTU	ABSTURA	ARBUTAS	ARBTAUS	ARBSTUA	ARUTBAS	ARUSBAT	ARABTUS	ARATSBU	ARTUBAS
ABSARUT	ABSTUAR	ARBUTSA	ARBTASU	ARBSUTA	ARUTBSA	ARAUBTS	ARABTSU	ARASBTU	ARTUBSA
ABSATRU	ABSUATR	ARBUSTA	ARBTUAS	ARBSUAT	ARUTSBA	ARAUBST	ARABSTU	ARASBUT	ARTUSBA
ABSATUR	ABSUART	ARBUSAT	ARBTUSA	ARUBATS	ARUTSAB	ARAUTBS	ARABSUT	ARASTBU	ARTUSAB
ABSAUTR	ABSUTAR	ARBAUTS	ARBTSUA	ARUBAST	ARUSATB	ARAUTSB	ARATBUS	ARASTUB	ARTAUBS
ABSAURT	ABSUTRA	ARBAUST	ARBTSAU	ARUBTAS	ARUSABT	ARAUSTB	ARATBSU	ARASUTB	ARTAUSB
ABSTARU	ABSURTA	ARBATUS	ARBSATU	ARUBTSA	ARUSTAB	ARAUSBT	ARATUBS	ARASUBT	ARTABUS
ABSTAUR	ABSURAT	ARBATSU	ARBSAUT	ARUBSTA	ARUSTBA	ARABUTS	ARATUSB	ARTUABS	ARTABSU

Q5:

```
#Program to check if a number is prime or not
```

```
# To take input from the user
num = int(input("Enter a number: "))

# define a flag variable
flag = False

if num == 1:
   print(num, "is not a prime number")
elif num > 1:
   # check for factors
   for i in range(2, num):
      if (num % i) == 0:
         # if factor is found, set flag to True
         flag = True
         # break out of loop
         break

   # check if flag is True
   if flag:
      print(num, "is not a prime number")
   else:
      print(num, "is a prime number")
output:
Enter a number: 7
7 is a prime number
```