1) Python Program to Implement Pigeonhole Sort Algorithm

Program Code:

```
def pigeonhole_sort(a):
        # size of range of values in the list
        # (ie, number of pigeonholes we need)
        my_min = min(a)
        my_max = max(a)
        size = my_max - my_min + 1
        # our list of pigeonholes
        holes = [0] * size
        # Populate the pigeonholes.
        for x in a:
                assert type(x) is int, "integers only please"
                holes[x - my_min] += 1
        # Put the elements back into the array in order.
        i = 0
        for count in range(size):
                while holes[count] > 0:
                        holes[count] -= 1
                        a[i] = count + my_min
                        i += 1
a = [8, 3, 2, 7, 4, 6, 8]
print("Sorted order is : ", end =" ")
pigeonhole_sort(a)
for i in range(0, len(a)):
```

```
        print(a[i], end =" ")
```

Output:

Sorted order is : 2 3 4 6 7 8 8

2) Python Program to Implement Breath First Search(BFS)

Program code:

```
#BFS algorithm in Python
import collections
# BFS algorithm
def bfs(graph, root):
        visited, queue = set(), collections.deque([root])
        visited.add(root)
        while queue:
        # Dequeue a vertex from queue
        vertex=queue.popleft()
        print(str(vertex) + ” ”,end=” ”)
        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
                if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)
 if __name__ == '__main__':
graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
```

```
print("Following is Breadth First Traversal: ")
bfs(graph, 0)
```

Output:

```
Following is Breadth First Traversal:
0 1 2 3
```

Program 3:

```
import heapq
class Solution(object):
    #:type n: integer
    #:return type: integer
  def nth_Ugly_Number(self, n):
      ugly_num = 0
      heap = []
      heapq.heappush(heap, 1)
      for _ in range(n):
          ugly_num = heapq.heappop(heap)
          if ugly_num % 2 == 0:
              heapq.heappush(heap, ugly_num * 2)
          elif ugly_num % 3 == 0:
              heapq.heappush(heap, ugly_num * 3)
          else:
              heapq.heappush(heap, ugly_num * 2)
              heapq.heappush(heap, ugly_num * 3)
              heapq.heappush(heap, ugly_num * 5)
```

```
        return ugly_num
n = 7
S = Solution()
result = S.nth_Ugly_Number(n)
print("7th Ugly number:")
print(result)
n = 10
result = S.nth_Ugly_Number(n)
print("\n10th Ugly number:")
print(result)
```

Output:

```
7th Ugly number:
9
10th Ugly number:
16
```

4) Python Program to calculate sum of series $1^2+2^2+3^2+.....+N^2$ using recursion

```
def sum_of_square_series(number):
    if(number == 0):
        return 0
    else:
        return (number * number) + sum_of_square_series(number - 1)
```

```
num = int(input("Please Enter any Positive Number  : "))
total = sum_of_square_series(num)

print("The Sum of Series upto {0}  = {1}".format(num, total))
```

Output:

```
Please Enter any Positive Number  : 9
The Sum of Series upto 9  = 285
```

Program 5: Python program to find the k^{th} (1 <=k<=array's length) largest element in an unsorted array.

```
import heapq
class Solution(object):
    def find_Kth_Largest(self, nums, k):
        """
        :type nums: List[int]
        :type of k: int
        :return value type: int
        """
        h = []
        for e in nums:
            heapq.heappush(h, (-e, e))
        for i in range(k):
            w, e = heapq.heappop(h)
            if i == k - 1:
                return e
```

```
arr_nums = [12, 14, 9, 50, 61, 41]

s = Solution()

result = s.find_Kth_Largest(arr_nums, 3)

print("Third largest element:",result)

result = s.find_Kth_Largest(arr_nums, 2)

print("\nSecond largest element:",result)

result = s.find_Kth_Largest(arr_nums, 5)

print("\nFifth largest element:",result)
```

Output:

```
Third largest element: 41

Second largest element: 50

Fifth largest element: 12
```

Program 6: Python program to print a heap as a tree-like data structure.

```
import math

from io import StringIO

#source https://bit.ly/38HXSoU

def show_tree(tree, total_width=60, fill=' '):

    """Pretty-print a tree.

    total_width depends on your input size"""

    output = StringIO()

    last_row = -1

    for i, n in enumerate(tree):
```

```
        if i:
            row = int(math.floor(math.log(i+1, 2)))
        else:
            row = 0
        if row != last_row:
            output.write('\n')
        columns = 2**row
        col_width = int(math.floor((total_width * 1.0) / columns))
        output.write(str(n).center(col_width, fill))
        last_row = row
    print (output.getvalue())
    print ('-' * total_width)
    return

#test
import heapq
heap = []
heapq.heappush(heap, 1)
heapq.heappush(heap, 2)
heapq.heappush(heap, 3)
heapq.heappush(heap, 4)
heapq.heappush(heap, 7)
heapq.heappush(heap, 9)
heapq.heappush(heap, 10)
heapq.heappush(heap, 8)
```

```
heapq.heappush(heap, 16)
heapq.heappush(heap, 14)
show_tree(heap)
```

Output:

```
                1
        2               3
    4       7       9       10
  8   16  14
```

Program 7:

```
import turtle
def draw_bear():
    turtle.speed(2)
    turtle.bgcolor("lightblue")
    #Draw the bear’s head
    turtle.penup()
    turtle.goto(0, -100)
    turtle.pendown()
    turtle.color("brown")
    turtle.begin_fill()
    turtle.circle(80)
    turtle.end_fill()
    #Draw the bear’s ear
    turtle.penup()
```

```
  turtle.goto(-30, 50)
  turtle.pendown()
  turtle.color("brown")
  turtle.begin_fill()
  turtle.circle(20)
  turtle.end_fill()

  turtle.penup()
  turtle.goto(30, 50)
  turtle.pendown()
  turtle.color("brown")
  turtle.begin_fill()
  turtle.circle(20)
  turtle.end_fill()

# Draw the bear’s eyes
  turtle.penup()
  turtle.goto(-20, 10)
  turtle.pendown()
  turtle.color("black")
  turtle.begin_fill()
  turtle.circle(8)
  turtle.end_fill()

  turtle.penup()
```

```
turtle.goto(20, 10)
turtle.pendown()
turtle.color("black")
turtle.begin_fill()
turtle.circle(8)
turtle.end_fill()
#Draw the bear’s nose
turtle.penup()
turtle.goto(0, -5)
turtle.pendown()
turtle.color("black")
turtle.begin_fill()
turtle.circle(3)
turtle.end_fill()
#Draw the bear’s Smiling mouth
turtle.penup()
turtle.goto(-20, -20)
turtle.pendown()
turtle.color("black")
turtle.width(3)
turtle.right(90)
turtle.circle(20, 180)  #    #Draw the semicircle for a smile
# Hide the turtle
turtle.hideturtle()
# Keep the window open
```

```
    turtle.done()
#Draw the bear
draw_bear()
```

Output:

Program 8: Python program to find out common items from two dictionary

```
keys1 =[]
vals1 = []
num1 = int(input("Enter the Number of elements for Dictionary-1: "))
print("Enter Values for Keys")
for i in range(0, num1):
    x = str(input("Enter Key " + str(i + 1) + "="))
    keys1.append(x)
print("Enter Values for Values")
for i in range(0, num1):
    x = str(input("Enter Value "+ str(i + 1) +"="))
    vals1.append(x)
dict1 = dict(zip(keys1, vals1))
print("Dictionary_1 Items:", dict1)
```

```
set1 = set(dict1.keys())

keys2 = []

vals2 =[]

num2 = int(input("Enter the Number of elements for Dictionary-2: "))

print("Enter Values for Keys")

for i in range(0, num2):

    y = str(input("Enter Key " + str(i + 1) + "="))

    keys2.append(y)

print("Enter Values for Values")

for i in range(0, num2):

    y = str(input("Enter Value " + str(i + 1) + "="))

    vals2.append(y)

dict2= dict(zip(keys2, vals2))

print("Dictionary_2 Items:", dict2)

set2 = set(dict2.keys())

common_items = set1 & set2

print("Common items from dictionary 1 and dictionary 2:", common_items)
```

Output:

```
Enter the Number of elements for Dictionary-1: 4

Enter Values for Keys

Enter Key 1=Orange

Enter Key 2=Banana

Enter Key 3=Grape

Enter Key 4=Apple
```

Enter Values for Values

Enter Value 1=22

Enter Value 2=33

Enter Value 3=11

Enter Value 4=55

Dictionary_1 Items: {'Orange': '22', 'Banana': '33', 'Grape': '11', 'Apple': '55'}

Enter the Number of elements for Dictionary-2:

Program: 9: Python program to create a game under the concept of rock, paper and scissor

```
import random
def play_game():
    choices = ["rock", "paper", "scissors"]
    #Get user input for their choice
    user_choice = input("Enter your choice (rock, paper, or scissors): ").lower()
    # Validate user input
    if user_choice not in choices:
        print("Invalid choice. Please choose rock, paper, or scissors.")
        return
    # Computer randomly selects its choice
    computer_choice = random.choice(choices)
    # Display choices
    print(f"\nYou chose: {user_choice}")
    print(f"Computer chose: {computer_choice}\n")
    #Determine the winner
    if user_choice == computer_choice:
```

```
        print("It's a tie!")
    elif (
        (user_choice =="rock" and computer_choice == "scissors") or
        (user_choice=="paper" and computer_choice =="rock") or
        (user_choice =="scissors" and computer_choice== "paper")
    ):
        print("You win!")
    else:
        print("Computer wins!")
if __name__=="__main__":
    play_game()
```

Output:

```
Enter your choice (rock, paper, or scissors): rock
You chose: rock
Computer chose: rock
It's a tie!
```

Program 10: Python program to find k number of pairs (U, V) which consists of one element from the first array and one element from the second array using heap queue algorithm.