ASSIGNMENT – 2

Q1) Write a recursive function power(base, exponent) that, when called, returns *base*^{exponent}.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
def power(base, exponent):
    if exponent < 0:
        return power(1/base, -exponent)
    if exponent == 0:
        return 1
    return base * power(base, exponent-1)
print(power(4,3))
print(power(4,-2))
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------
64
0.0625
```

Q2) The greatest common divisor of integers x and y is the largest integer that evenly divides into both x and y. Write and test a recursive function gcd that returns the greatest common divisor of x and y. The gcd of x and y is defined recursively as follows: If y is equal to 0, then *gcd(x, y)* is x; otherwise, *gcd(x, y)* is *gcd(y, x%y)*.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
def gcd(x, y):
    if y == 0:
        return x
    return gcd(y, x % y)
print(gcd(125,625))
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------
125
```

Q3) Write a recursive function that takes a number n as an input parameter and prints n-digit strictly increasing numbers.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
def print_increasing(n, curr="", lastDig = 0):
    if n==0:
        if curr!="":
            print(curr, end = " ")
        else:
            print("Invalid Input!!")
        return
    for i in range(lastDig+1, 10):
        print_increasing(n-1, curr + str(i),i)

n = int(input("Enter no. of digits: "))
```

```
print(f"\n{n}-digit strictly increasing numbers:-")
print_increasing(n)
```

<u>OUTPUT</u>: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
Enter no. of digits: 2

2-digit strictly increasing numbers:-
12 13 14 15 16 17 18 19 23 24 25 26 27 28 29 34 35 36 37 38 39 45 46 47 48 49 56 57 58 59 67 68 69 78 79 89
```

Q4) Implement a recursive solution for computing the nth Fibonacci number. Then, analyze its time complexity. Propose a more efficient solution and compare the two approaches.

<u>RECURSIVE Solution</u>: -

<u>Code</u>: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------''')
def fibo(n):
    if n==0:
        return 0
    if n==1:
        return 1
    return fibo(n-1) + fibo(n-2)
print(fibo(5))
```

<u>OUTPUT</u>: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
5
```

<u>Time Comlpexity</u>: O(2^n)

- The time complexity of the recursive solution is exponential. This is because the function calls itself twice for each call, leading to a tree-like structure with 2^n nodes.

<u>MORE EFFICIENT(ITERATIVE) Solution</u>: -

<u>Code</u>: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------''')
def fibo_efficient(n):
    if n==0:
        return 0
    if n==1:
        return 1
    a = 0
    b = 1
    for i in range(2, n+1):
        c = a + b
        a = b
        b = c
    return c
print(fibo_efficient(5))
```

<u>OUTPUT</u>: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
5
```

Time Comlpexity: O(n)

- The time complexity of the efficient solution is linear. This is because the function iterates n times to calculate the nth Fibonacci number.

Q5) Given an array of N elements, not necessarily in ascending order, devised an algorithm to find the kth largest one. It should run in O(N) time on random inputs.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
import random
def quickSelect(ar, k):
    if ar:
        # random pivot(to avoid worst case. Rarely, worst case may occur)
        pivot = ar[random.randint(0, len(ar)-1)]
        left = [x for x in ar if x < pivot]
        right = [x for x in ar if x > pivot]
        if k == len(right) + 1:
            return pivot
        elif k <= len(right):
            return quickSelect(right, k)
        else:
            return quickSelect(left, k - len(right) - 1)
    return None
print(quickSelect([3, 2, 1, 5, 4], 2))
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------
4
```

time complexity: O(n) in avg case, O(n^2) in worst case

Q6) For each of the following code snippets, determine the time complexity in terms of Big O. Explain your answer.

```
(a)   def example1(n):
        for i in range(n):
             for j in range(n):
              print(i, j)
```

Ans: -
time complexity: O(n^2)
The time complexity of the nested loops is quadratic. This is because the inner loop runs n times for each iteration of the outer loop, leading to n^2 iterations in total.

```
(b)   for i in range(n):
        print(i)
```

Ans: -
time complexity: O(n)
The time complexity of the loop is linear. This is because the loop runs n times, where n is the input to the function.

```
(c)   def example1(n): def recursive_function(n):
       if n <= 1:
           return 1
       return recursive_function(n - 1) + recursive_function(n - 1)
```

Ans: -

time complexity: O(2^n)
The time complexity of the recursive function is exponential. This is because the function calls itself twice for each call, leading to a tree-like structure with 2^n nodes.

Q7) Given N points on a circle, centered at the origin, design an algorithm that determines whether there are two points that are antipodal, i.e., the line connecting the two points goes through the origin. Your algorithm should run in time proportional to *N logN*.

<u>Code</u>: -
```python
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
def quadrant(x, y):
    if x >= 0 and y >= 0:
        return 1
    if x < 0 and y >= 0:
        return 2
    if x < 0 and y < 0:
        return 3
    return 4

def custom_sort(point):
    x, y = point
    q = quadrant(x, y)
    if x != 0:
        slope = y * 1.0 / x
    else:
        slope = float('inf')
    return (q, slope)

def checkAntipodal(points):
    points.sort(key=custom_sort)
    n = len(points)
    for i in range(n):
        x1, y1 = points[i]
        for j in range(i + 1, n):
            x2, y2 = points[j]
            if x1 == -x2 and y1 == -y2:
                return True
    return False

points = [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (-1, -1)]
if checkAntipodal(points):
    print("Antipodal points exist.")
else:
    print("Antipodal points do not exist.")
```

<u>OUTPUT</u>: -
```
Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------
Antipodal points exist.
```

Q8) Write a Python function *quick_sort* that implements the quicksort algorithm. The function should include a helper function *quick_sort_helper* to handle recursion. The helper function must take a starting and ending index as arguments and sort the array in-place. Demonstrate the function by sorting the given array and printing the sorted output.

<u>Code</u>: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------''')
def quick_sort(ar, s, e):
    if s<e:
        p = partition(ar, s, e)
        quick_sort(ar, s, p-1)
        quick_sort(ar, p+1, e)

def partition(ar, s, e):
    pivot = ar[s]
    i = s
    for j in range(s+1, e+1):
        if ar[j] < pivot:
            i += 1
            ar[i], ar[j] = ar[j], ar[i]
    ar[i], ar[s] = ar[s], ar[i]
    return i

ar = [37, 2, 6, 4, 89, 8, 10, 12, 68, 45]
print("Before sorting:", ar)
s = 0
e = len(ar)-1
quick_sort(ar, s, e)
print("After sorting: ", ar)
```

<u>OUTPUT</u>: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
Before sorting: [37, 2, 6, 4, 89, 8, 10, 12, 68, 45]
After sorting:  [2, 4, 6, 8, 10, 12, 37, 45, 68, 89]
```

Q9) You are given the following list of famous personalities with their net worth:
• Elon Musk: $433.9 Billion
• Jeff Bezos: $239.4 Billion
• Mark Zuckerberg: $211.8 Billion
• Larry Ellison: $204.6 Billion
• Bernard Arnault & Family: $181.3 Billion
• Larry Page: $161.4 Billion
Develop a program to sort the aforementioned details on the basis of net worth using
a. Selection sort
b. Bubble sort
c. Insertion sort.
The final sorted data should be the same for all cases. After you obtain the sorted data, present the result in the form of the following dictionary:
{'name1':networth1, 'name2':networth2, ...}

<u>Code</u>: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------''')

class Billionaires:
    def __init__(self, name, net_worth):
        self.name = name
        self.net_worth = net_worth
    def __str__(self):
        return f'{self.name}: {self.net_worth} Billion'
    def __lt__(self, other):
```

```python
        return self.net_worth < other.net_worth
    def __gt__(self, other):
        return self.net_worth > other.net_worth

def insertionSort(l):
    n = len(l)
    if n<1:
        return
    for i in range(1, n):
        key = l[i]
        j = i-1
        while j>=0 and key < l[j]:
            l[j+1] = l[j]
            j -= 1
        l[j+1] = key
    return l

def selectionSort(l):
    n = len(l)
    for i in range(n):
        min = i
        for j in range(i+1, n):
            if l[j] < l[min]:
                min = j
        l[i], l[min] = l[min], l[i]
    return l

def bubbleSort(l):
    n = len(l)
    for i in range(n):
        for j in range(n-i-1):
            if l[j] > l[j+1]:
                l[j], l[j+1] = l[j+1], l[j]
    return l

bData = {
    'Elon Musk': 433.9,
    'Jeff Bezos': 239.4,
    'Mark Zuckerberg': 211.8,
    'Larry Ellison': 204.6,
    'Bernard Arnault & Family': 181.3,
    'Larry Page': 161.4
}
l = [Billionaires(k, v) for k, v in bData.items()]
isl = insertionSort(l[:])
ssl = selectionSort(l[:])
bsl = bubbleSort(l[:])

isd = {x.name: x.net_worth for x in isl}
ssd = {x.name: x.net_worth for x in ssl}
bsd = {x.name: x.net_worth for x in bsl}

print(f"Insertion Sort:-\n{isd}\n")
print(f"Selection Sort:-\n{ssd}\n")
print(f"Bubble Sort:-\n{bsd}")
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
Insertion Sort:-
{'Larry Page': 161.4, 'Bernard Arnault & Family': 181.3, 'Larry Ellison': 204.6, 'Mark Zuckerberg': 211.8, 'Jeff
Bezos': 239.4, 'Elon Musk': 433.9}

Selection Sort:-
{'Larry Page': 161.4, 'Bernard Arnault & Family': 181.3, 'Larry Ellison': 204.6, 'Mark Zuckerberg': 211.8, 'Jeff
Bezos': 239.4, 'Elon Musk': 433.9}

Bubble Sort:-
{'Larry Page': 161.4, 'Bernard Arnault & Family': 181.3, 'Larry Ellison': 204.6, 'Mark Zuckerberg': 211.8, 'Jeff
Bezos': 239.4, 'Elon Musk': 433.9}
```

Q10) Use Merge Sort to sort a list of strings alphabetically.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------''')
def mergeSort(ar):
    if len(ar) <= 1:
        return ar
    mid = len(ar) // 2
    left = ar[:mid]
    right = ar[mid:]
    left = mergeSort(left)
    right = mergeSort(right)
    return merge(left, right)

def merge(l, r):
    i = j = 0
    ml = []
    while i<len(l) and j<len(r):
        if l[i] < r[j]:
            ml.append(l[i])
            i += 1
        else:
            ml.append(r[j])
            j += 1
    ml.extend(l[i:])
    ml.extend(r[j:])
    return ml

ar = ['apple', 'orange', 'banana', 'grape']
print(f"Before sorting: {ar}")
l = 0
r = len(ar)-1
print(f"After sorting:  {mergeSort(ar)}")
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
Before sorting: ['apple', 'orange', 'banana', 'grape']
After sorting:  ['apple', 'banana', 'grape', 'orange']
```

Q11) Without using the built-in sorted() function, write a Python program to merge two pre-sorted lists into a single sorted list using the logic of Merge Sort.

Code: -

```
print('''Name: Sidhanta Barik, RegNo: 2241002049
-----------------------------------------''')
def merge(l, r):
    i = j = 0
    ml = []
    while i<len(l) and j<len(r):
        if l[i] < r[j]:
            ml.append(l[i])
            i += 1
        else:
            ml.append(r[j])
            j += 1
    ml.extend(l[i:])
    ml.extend(r[j:])
    return ml

sortedList1 = [1,3,5,7]
sortedList2 = [2,4,6,8]
mergedList = merge(sortedList1, sortedList2)
print(f"Sorted Merged List: {mergedList}")
```

OUTPUT: -

```
Name: Sidhanta Barik, RegNo: 2241002049
----------------------------------------
Sorted List 1: [1, 3, 5, 7]
Sorted List 1: [2, 4, 6, 8]
Sorted Merged List: [1, 2, 3, 4, 5, 6, 7, 8]
```