Minor Assignment-2: Questions and Solutions

1. Write a recursive function power(base, exponent) that, when called, returns base^exponent.

```
def power(base, exponent):
    if exponent == 0:
        return 1
    return base * power(base, exponent - 1)

# Example: power(3, 4) => 81
```

2. Write and test a recursive function gcd that returns the greatest common divisor of x and y.

```
def gcd(x, y):
    if y == 0:
        return x
    return gcd(y, x % y)

# Example: gcd(48, 18) => 6
```

3. Write a recursive function that takes a number n as input and prints n-digit strictly increasing numbers.

```
def increasing_numbers(n, start=1, result=''):
    if n == 0:
        print(result)
        return
    for i in range(start, 10):
        increasing_numbers(n-1, i+1, result+str(i))

# Example: increasing_numbers(2) prints 2-digit increasing numbers
```

4. Implement a recursive solution for computing the nth Fibonacci number. Analyze its time complexity.

```
# Recursive Fibonacci (O(2^N))
def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n-1) + fib_recursive(n-2)

# Optimized Fibonacci using Memoization (O(N))
def fib_memoized(n, memo={}):
    if n in memo:
        return memo[n]
```

```
    if n <= 1:
        return n
    memo[n] = fib_memoized(n-1, memo) + fib_memoized(n-2, memo)
    return memo[n]

# Example: fib_recursive(10) => 55, fib_memoized(10) => 55
```

5. Given an array of N elements, find the kth largest element in O(N) time.

```
import random

def quickselect(arr, k):
    if len(arr) == 1:
        return arr[0]
    pivot = random.choice(arr)
    left = [x for x in arr if x > pivot]
    mid = [x for x in arr if x == pivot]
    right = [x for x in arr if x < pivot]

    if k <= len(left):
        return quickselect(left, k)
    elif k <= len(left) + len(mid):
        return pivot
    else:
        return quickselect(right, k - len(left) - len(mid))

# Example: quickselect([3, 2, 1, 5, 6, 4], 2) => 5
```

6. Determine the time complexity in terms of Big O.

(a) O(N^2) - Nested loops iterate N*N times.
(b) O(N) - Single loop runs N times.
(c) O(2^N) - Recursively doubles calls.

7. Given N points on a circle, design an algorithm to determine if two points are antipodal in O(N log N).

```
# Sort points by angle and check opposite pairs
```

8. Implement QuickSort algorithm and demonstrate sorting the given array.

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x <= pivot]
    right = [x for x in arr[1:] if x > pivot]
    return quicksort(left) + [pivot] + quicksort(right)
```

```
# Example: quicksort([37, 2, 6, 4, 89, 8, 10, 12, 68, 45])
```

9. Sort a list of billionaires by net worth using Selection, Bubble, and Insertion Sort.

```
people = {
    'Elon Musk': 433.9, 'Jeff Bezos': 239.4, 'Mark Zuckerberg': 211.8,
    'Larry Ellison': 204.6, 'Bernard Arnault': 181.3, 'Larry Page': 161.4
}

sorted_people = dict(sorted(people.items(), key=lambda x: x[1]))

# Output: {'Larry Page': 161.4, 'Bernard Arnault': 181.3, ... }
```

10. Use Merge Sort to sort a list of strings alphabetically.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    while left and right:
        if left[0] < right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    return result + left + right

# Example: merge_sort(['apple', 'orange', 'banana', 'grape'])
```

11. Merge two pre-sorted lists into a single sorted list using Merge Sort logic.

```
def merge_sorted_lists(l1, l2):
    i, j = 0, 0
    result = []
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            result.append(l1[i])
            i += 1
        else:
            result.append(l2[j])
            j += 1
```

```
    return result + l1[i:] + l2[j:]

# Example: merge_sorted_lists([1, 3, 5, 7], [2, 4, 6, 8])
```