Question # 1 [3+1 marks]: [Recursive Scripts] Consider the following recursive scripts:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage:
result = factorial(5)
print("Factorial of 5 is", result)
```

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return n % 10 + sum_of_digits(n // 10)

# Example usage:
result = sum_of_digits(12345)
print("Sum of digits:", result)
```

```
def binary_search(arr, target, low, high):
    if low > high:
        return -1
    mid = (low + high) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] > target:
        return binary_search(arr, target, low, mid - 1)
    else:
        return binary_search(arr, target, mid + 1, high)

# Example usage:
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target = 6
index = binary_search(arr, target, 0, len(arr) - 1)
print(f"Element {target} found at index {index}")
```

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

# Example usage:
result = gcd(48, 18)
print("GCD of 48 and 18 is", result)
```

```
def reverse_list(arr):
    if len(arr) == 0:
        return []
    else:
        return [arr[-1]] + reverse_list(arr[:-1])

# Example usage:
original_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(original_list)
print("Original List:", original_list)
print("Reversed List:", reversed_list)
```

```
def is_palindrome(s):
    s = s.replace(" ", "").lower()
    if len(s) <= 1:
        return True
    elif s[0] == s[-1]:
        return is_palindrome(s[1:-1])
    else:
        return False

# Example usage:
result = is_palindrome("A man a plan a canal Panama")
print("Is it a palindrome?", result)
```

[PART A] Find the recurrence relation of each recursive script.
[PART B] Solve the recurrence relation of sum_of_digits(..) and binary_Search(…)

Question # 2: [1 mark] Consider Suppose you are choosing between the following 3 algorithms:

- **Algorithm A** solves the problem of size n by dividing it into 5 subproblems of size n/2, recursively solving each subproblem, and then combining the solutions in linear time.
- **Algorithm B** solves the problem of size n by recursively solving two subproblems of size n − 1 and then combining the solutions in constant time.
- **Algorithm C** solves the problem of size n by dividing it into nine subproblems of size n/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each algorithm and which would you choose and why?

Question # 3: [2.5+2.5 mark] Compute the asymptotic complexity of the following iterative code fragment to its closed form. **Solve to find asymptotically tighter bounds.** Clearly show your working throughout.

[PART A]

```
Func1 (n)
for (int t=2; t 5/2+2/5 <= n; t++)
       cout<< "Pakistan!";

Func2 (int n)
for (i=1; i <n ;i++){
    for (j=1; j<=i^3 ; j++){
         for (k=1; k<=2000; k++)
                  cout<<"Pakistan";
}}

Func3 (int n, int m )
for (i=n/7; i >=1 ; i-- ){
    for (j=1; j<=log 3/2 m; j++){
                 cout<<"Pakistan";   }}

Func4 (int n )
for (i=n/3; i < n*n ; i++){
    for (j=1; j<=n/4; j=j* √ n){          # √ n is under-root n.
         for (k=1; k<=n; k=k*3.5)
                  cout<<"Pakistan!";
}}

Main ()
int n,m     # assume they are initialized to some values.

Func2 (n);
Func4 (n);
Func1 (m);
Func3 (n, m );
```

Help Notes: $\sum_{i=1}^{n} i = \left(\frac{n(n+1)}{2}\right)$,

$\sum_{i=1}^{n} i^2 = \left(\frac{n(n+1)(2n+1)}{6}\right)$, $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$

[PART B]

```
ValueReturn DoingSomething ( n )
if (n==1)
    return n
return n* DoingSomething(n-2)


int i, j, k, n
for ( k=1; k<=n; k++ )
      for ( j=1; j<=k; j++ )
            print(  DoingSomething(n)  )        #Hint: Solve this first on R.H.S rough space!
```

See if any series given below helps:

$\sum_{i=1}^{n} i = (\frac{n(n+1)}{2})$, $\sum_{i=1}^{n} i^2 = (\frac{n(n+1)(2n+1)}{6})$, $\sum_{i=1}^{n} \frac{1}{n} =$ (**logn**)

All the Best!