# University of Colombo School of Computing
## SCS 1308 - Foundations of Algorithms
Take-home 01

**Instructions**

- Try the following questions and upload your answer script as a zip file to the given link in the UGVLE on/before 1st December at 6pm.
- Note: Rename your zip file with your index number and name. (i.e: indexNo_Name.zip).

### 1. Tail-Recursive Factorial

The algorithm modifies the standard recursive approach by introducing an accumulator to carry the computation, allowing for tail-recursive optimization.

```
int int fact_helper(int n, int accumulator) {
  if (n <= 1)
    return accumulator;  // Base case: Return the accumulated result
  else
        return fact_helper(n - 1, n * accumulator);  // Recursive call with updated accumulator
}
int fact(int n) {
  return fact_helper(n, 1);  // Initial call with accumulator set to 1
}
fact_helper(int n, int accumulator) {
  if (n <= 1)
    return accumulator;  // Base case: Return the accumulated result
  else
        return fact_helper(n - 1, n * accumulator);  // Recursive call with updated accumulator
}
int fact(int n) {
  return fact_helper(n, 1);  // Initial call with accumulator set to 1
}
```

1.1. Derive the recurrence relation for the fact(n) algorithm. Explain each term in the recurrence.

1.2. Using the recursion tree method, calculate the total operations for fact(n) and explain why it has $\Theta(n)$ complexity.

2. **Binary Search Recurrence:**
   2.1. Write the recurrence relation for the worst-case time complexity of the binary search.
   2.2. Using Master's Theorem, solve $T(n)=T(n/2)+\Theta(1)$ and explain its complexity.

3. **Master's Theorem Application:**
   3.1. For the recurrence $T(n)=8T(n/4)+5n$, identify the values of a, b, and k. Determine which case of Master's Theorem applies and solve for T(n).
   3.2. Explain a situation where the Master's Theorem cannot be applied and propose an alternative method.

4. **Constructing a Recursion Tree:**
   4.1. Build a recursion tree for the algorithm fact(n). For each level of the tree, write down the number of nodes and the non-recursive work.
   4.2. Calculate the total work performed by summing across all levels and verify it matches $\Theta(n)$.

5. **Binary Search Recursion Tree:**
   5.1. Construct the recursion tree for binary search on an array of size n=16. Indicate the depth of the tree and the total work done at each depth.

6. **Direct Solution Size and Count:**
   6.1. For a recursive algorithm where $T(n)=2T(n/2)+n$, determine the DirectSolutionSize and DirectSolutionCount.
   6.2. Solve T(n) using the substitution method.

7. **Exponential Complexity Example:**
   7.1. Analyze the time complexity of a recursive algorithm that splits a problem into three subproblems, each of size n/2, with a non-recursive cost of $\Theta(n^2)$ .
   7.2. Write and solve the recurrence relation for this algorithm.

8. **When Master's Theorem Fails:**
   8.1. Construct a recurrence relation that does not fit the Master's Theorem form.
   8.2. Propose a method to solve such a recurrence.