CE6029: Advanced Algorithm (Fall 2025)

Tutorial 2: Recursion

Due: Sept 18, 2025

Instructor: Dr. Hao-Tsung Yang | Noted By: Po-Cheng Chan

The following just some example sentences. Feel free to modify, eliminate, re-write, $\cdots$.

Instructions. Show your work, justify asymptotics, and clearly label answers. Unless stated otherwise, you may use standard results covered in class.

0.1 Part I: Recursion, Iteration, and Reduction

Overview. This part introduces three fundamental paradigms of algorithm design and reasoning:

- **Recursion (Top-down)**: A problem is solved by defining a function that calls itself on smaller subproblems until reaching a base case.
- **Iteration (Bottom-up)**: A procedure builds the solution step by step, starting from the simplest case and working upward, typically using loops.
- **Reduction**: Transforming a problem into another well-studied problem so that a known algorithm can be applied.

Recursion in Detail.

- **Top-down view**: Design starts from the whole problem and expresses it in terms of smaller subproblems.
- **Function re-use**: The same function is called repeatedly on smaller inputs (*self-indication*).
- **Key challenge – avoid infinite calling**: A base case must be well defined and guaranteed to be reached; otherwise the function will recurse forever and never terminate.
- **Typical examples**: Factorial computation, binary search, divide-and-conquer sorting.

Iteration in Detail.

- **Bottom-up view**: Start from the simplest instance and build the solution upward, e.g., dynamic programming table filling or standard for/while loops.
- No function self-calls are required; all state is maintained explicitly (counters, arrays, stacks if needed).

Relationship and Use Cases.

- Any recursion can be transformed into iteration by simulating the call stack explicitly.
- Iteration can sometimes be more memory-efficient because it avoids the overhead of recursive calls.

Illustrative Pseudo-code. Below is a simple recursive vs. iterative factorial function:

Listing 1: Factorial: recursive vs. iterative

```
def fact_recursive(n):
    if n <= 1: return 1
    return n * fact_recursive(n - 1)

def fact_iterative(n):
    result = 1
    for i in range(2, n+1):
        result *= i
    return result
```

Problems

Problem 1 (Master Theorem warmup). [10 pts] Use the Master Theorem to solve: $T(n) = 3T(n/2) + n$.

Problem 2 (Recursion tree). [15 pts] Give a tight bound for $T(n) = T(n/2)+T(n/4)+n$.

Hint. Compare total work per level; the series is dominated by early levels.

Problem 3 (Divide-and-conquer pseudocode). [15 pts] Write pseudocode for MAX-SUBARRAY(A) using divide-and-conquer and give its running time.

Problem 4 (Short proof). [10 pts] Prove that $1 + 2 + \cdots + n = \Theta(n^2)$.

(Optional) Code Snippet

You may include reference code:

```
def merge(a, b):
    i=j=0; out=[]
    while i<len(a) and j<len(b):
        if a[i] <= b[j]:
            out.append(a[i]); i+=1
        else:
            out.append(b[j]); j+=1
    return out + a[i:] + b[j:]
```