Quiz 2 Solutions – Design and Analysis of Algorithms (CS 412)

Instructor: Dr. Ayesha Enayet

February 18, 2025

A. Best-Case Complexity of Merge Sort (0.5 marks)

Correct Answer: $\Omega(n \log n)$

Explanation: Merge sort has a best-case time complexity of $\Omega(n \log n)$ because it always divides the array into halves and requires n comparisons to merge these sorted halves, regardless of input order.

B. Master Method – Solve the Recurrence (1 mark)

Given recurrence:

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$$

Step-by-Step Solution:

- $a = 2$: Number of subproblems
- $b = 4$: Factor by which problem size is reduced
- $f(n) = \sqrt{n}$: Additional cost per level

Apply Master Theorem for recurrence relations of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Calculate the critical exponent:

$$\log_b a = \log_4 2 = \frac{\log 2}{\log 4} = \frac{1}{2}$$

Now we compare $f(n)$ with n^c where:

$$c = \log_b a = \frac{1}{2}$$

Analyze the function $f(n)$:

$$f(n) = \sqrt{n} = n^{1/2}$$

Now compare with n^c:

$$n^c = n^{1/2}$$

So, $f(n)$ matches n^c. This corresponds to **Case 2** of the Master Theorem, where:

$$f(n) = \Theta(n^c \log^k n)$$

Here $k = 0$ because there's no extra logarithmic factor.
Final Solution:

$$T(n) = \Theta(n^{1/2} \log n)$$

C. Divide & Conquer Algorithm to Find Median of Two Unsorted Arrays (1.5 marks)

Problem: Find the median of two unsorted arrays:
A = {7, 1, 5, 3}, B = {2, 6, 4, 8}
Step-by-Step Algorithm:

1. **Merge the Arrays:** Combine A and B into a single array:

$$M = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

2. **Sort Using Divide & Conquer (Merge Sort):** - Recursively divide arrays into halves. - Sort each half. - Merge sorted halves.

3. **Calculate Median:** For an even-sized array:

$$Median = \frac{SumOfMiddleElemets}{2}$$

For an odd-sized array:

$$Median = middleElemet$$

Pseudocode:

```
function findMedian(A, B):
    # Merge arrays
    M = mergeArrays(A, B)

    # Sort merged array using merge sort
    sortedArray = mergeSort(M)

    n = length(sortedArray)

    # Calculate median
    if n % 2 == 0:
        median = (sortedArray[n//2 - 1] + sortedArray[n//2]) / 2
    else:
        median = sortedArray[n//2]

    return median

function mergeArrays(A, B):
    return A + B

function mergeSort(arr):
    if length(arr) <= 1:
        return arr

    mid = length(arr) // 2
    left = mergeSort(arr[:mid])
    right = mergeSort(arr[mid:])
    return merge(left, right)

function merge(left, right):
    result = []
    i = j = 0

    while i < length(left) and j < length(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result += left[i:]
    result += right[j:]

    return result
```

Time Complexity: $O(n \log n)$

D. Master Theorem for Recurrence Tree (2 marks)

Part 1: Solve using Master Theorem

- $a = 3$: Number of subproblems
- $b = 3$: Factor of subproblem size reduction
- $f(n) = O(n)$: Linear additional cost

Apply Master Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Calculate log base:

$$c = \log_b a = \log_3 3 = 1$$

Compare $f(n)$ with n^c
Since $f(n) = \Theta(n^1)$, this corresponds to Case 2 of the Master Theorem.
Case 2: $f(n)$ matches the critical function.
Solution:

$$T(n) = \Theta\left(n^1 \log n\right) = \Theta(n \log n)$$

Part 2: Total cost of all internal nodes

$$= \sum_{k=0}^{\log_3 n - 1} n = O(n \log_3 n)$$