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

Instructor: Dr. Ayesha Enayet

February 18, 2025

Multiple Choice Questions

1. The total number of nodes in a binary tree of height 4 (with the root as level 1)

Height = 4 means levels = 4 (from level 1 to level 4).
Maximum nodes in a binary tree of height h: $2^h - 1$.
For $h = 4$: $2^4 - 1 = 15$.
Answer: 15

2. For a recurrence of the form $aT(n/b) + f(n)$, which conditions must hold to apply the Master Theorem?

The Master Theorem applies if $a > 0$ and $b > 1$.
Answer: $a > 0, b > 1$

3. For the recurrence $7T(n/2) + n^3$, identify whether $f(n)$ is greater, smaller, or equal to the watershed function.

Given recurrence: $T(n) = 7T(n/2) + n^3$.
$a = 7, b = 2, f(n) = n^3$.
Compute $\log_b a$:

$$\log_2 7 \approx 2.81$$

The watershed function is $n^{\log_b a} = n^{2.81}$.
Compare with $f(n) = n^3$:

$$n^3 > n^{2.81}$$

Answer: f(n) is greater than the watershed function.

Short Answer Questions

4. Two applications of the Divide-and-Conquer approach:

- Sorting
- Parallel computing

7. Solve the recurrence $T(n) = 2T(n/2) + \sqrt{n}\log^2 n$ using the Master Method:
Given recurrence: $T(n) = 2T(n/2) + \sqrt{n}\log^2 n$.
Identify parameters:

$$a = 2, b = 2, f(n) = \sqrt{n}\log^2 n$$
$$\log_b a = \log_2 2 = 1$$

Now, compare $f(n)$ with $n^{\log_b a}$:

$$n^{\log_b a} = n^1 = n$$

Check the growth of $f(n)$:

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

Since $f(n) = o(n^1)$, it is polynomially smaller than n.
By Case 1 of the Master Theorem:
$T(n) = \Theta(n)$

6. Propose an $\Omega(n \log n)$ solution for counting inversions.
Algorithm: Modified Merge Sort
Idea: Count inversions during the merge step of Merge Sort. Whenever we take an element from the right half before the left half, it indicates an inversion.

```
def mergeSortCount(arr):
    if len(arr) <= 1:
        return arr, 0
    mid = len(arr) // 2
    left, leftCount = mergeSortCount(arr[:mid])
    right, rightCount = mergeSortCount(arr[mid:])
    merged, mergeCount = mergeAndCount(left, right)
    return merged, leftCount + rightCount + mergeCount

def mergeAndCount(left, right):
    i = j = count = 0
    merged = []
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
            count += len(left) - i
    merged += left[i:]
    merged += right[j:]
    return merged, count
A, inversion_count = mergeSortCount(arr)
```

Time Complexity: $\mathcal{O}(n \log n)$