FUNDAMENTAL OF ALGORITHM DESIGN AND MACHINE LEARNING

LECTURE SCRIBED NOTES

DATE: 16TH JAN 2025

Anushka

24BM6JP10

DIVIDE AND CONQUER PARADIGM

1. Introduction

The Divide and Conquer strategy is a key algorithm design approach that involves breaking a problem down into smaller, more manageable sub-problems, solving each one separately, and then merging their solutions to address the original issue.

2. Steps in Divide and Conquer

1. Base Condition:
   the simplest version of the problem that can be solved directly without any further breakdown.
2. Decomposition Process:
   The problem is split into smaller, independent sub-problems of the same nature.
3. Subroutine Call:
   Each sub-problem is solved recursively using the Divide and Conquer method.
4. Recomposition Process:
   The solutions of the sub-problems are combined to form the solution to the original problem.

3. Complexity Analysis

- *General Formula:*

$$T(n) = \sum_{i=1}^{k} T(i) + D(n) + R(k)$$

- *Search & Sort Complexities List:*

Linear Search	O(n)
Binary Search	O(nlogn)

Selection Sort	$O(n^2)$
Bubble Sort	$O(n^2)$
Insertion Sort	$O(n^2)/O(nlogn)$
Heap Sort	O(nlogn)
Merge Sort	O(nlogn)
Quick Sort	$O(n^2)/O(nlogn)$

---

4. Quick Sort Algorithm

Pseudo Code for Quick Sort:

Pseudo Code for Quick Sort:

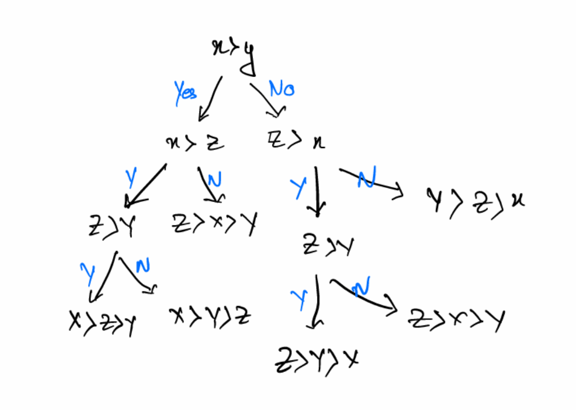
```
QS {
        If (|L| <= 1) return L
        Choose xᵢ ϵ L
        Create L₁ = { x | x < xᵢ}
        and L₂ = { x| x >= xᵢ }
         M₁  ← QS(L₁)
         M₂  ← QS(L₂)
        Return (M₁ || xᵢ||M2)
    }
```

Complexity: $T(n) = T(k) + T(n-k-1) + O(n)$

- ➔ Best Case: $k=n/2$ $T(n) = O(n\log n)$
- ➔ Worst Case: $k=1$ $T(n) = O(n^2)$
- ➔ Average Case: $T(n) = O(n\log n)$

---

5. Proving Lower Bound of Sorting based on comparison:



Leaves:

$\lceil \log(n!) \rceil$

$\Omega(\log n!) = \Omega(n\log n)$

---

6. Master Theorem for Divide and Conquer

The Master Theorem offers a way to solve recurrence relations that frequently occur in divide-and-conquer algorithms. These relations typically take the following form:

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{b}\right) + f(n) \geq f(n) \\
&= a^2 T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n) \\
&= a^3 T\left(\frac{n}{b^3}\right) + a^2 f\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n) \\
&= a^k T\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)
\end{aligned}
$$

Where:

a : Number of sub-problems.

b : Factor by which the problem size is divided.

f(n) : Additional work outside the recursive calls (e.g., combining the results).

Here $b^k=n$. Hence, K = $\log_b a$

Here, we can have 3 cases for f(n)

Case I : $O(n^{\log_b a - \epsilon})$

$$
\begin{aligned}
& f(n) = O(n^{\log_b a - \epsilon}) \quad \epsilon > 0 \quad \leq d \cdot n^{\log_b a - \epsilon} \\
& a^i \cdot d \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon} = a^i \cdot d \cdot n^{\log_b a} n^{-\epsilon} \underbrace{b^{\log_b a^{-i}}}_{a^{-i}} \cdot b^{i\epsilon} \\
& = d n^{\log_b a - \epsilon} \sum_{i=0}^{k-1} b^{i\epsilon} = d n^{\log_b a - \epsilon} \cdot \frac{b^{k\epsilon} - 1}{b^{\epsilon} - 1} \\
& = n^{\log_b a} \cdot d \cdot \left(\frac{b^{k\epsilon} - 1}{b^{\epsilon} - 1}\right) n^{-\epsilon} < n^{\log_b a} \\
& \therefore \Theta(n^{\log_b a})
\end{aligned}
$$

So, T(n) = $\boldsymbol{\Theta}$ *(*$n^{\log_b a}$*)*

Case II : **Θ** ($n^{\log_b a}$)

Here due to repetitive splitting, it will have log term

$$f(n) = \Theta(n^{\log_b a})$$

Per Level work :

$$a^i . f\left(\frac{n}{b^i}\right) = a^i \left(\frac{n}{b^i}\right)^{\log_b a} = n^{\log_b a}$$

$$T(n) = n^{\log_b a} . \log n = \Theta\left(n^{\log_b a} \log n\right)$$

So, T(n) = ***Θ*** *($n^{\log_b a}$ log n)*

Case III: Ω ($n^{\log_b a + \epsilon}$)

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

$$a f\left(\frac{n}{b}\right) < c f(n)$$

$$a^i f\left(\frac{n}{b^i}\right) < c^i f(n)$$

$$\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) < \sum_{i=0}^{\infty} c^i f(n)$$

$$\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) \le \frac{f(n)}{1-c} \qquad \boxed{c<1}$$

$$\therefore T(n) = \Theta(f(n))$$

So, T(n) = ($n^{\log_b a}$)

---

7. Multiplication of 2 n-bit Numbers

➔ *Traditional Complexity*: *O(n^2)*

X =

X_1	X_2

Y =

Y_1	Y_2

$X = 2^{n/2}X_1 + X_2$

$Y = 2^{n/2}Y_1 + Y_2$

$$X * Y = 2^n X_1Y_1 + 2^{n/2} (X_1Y_2 + X_2Y_1) + X_2Y_2$$

$$T(n) = 3\ T(n/2) + O(n)$$

---

8. Multiplication of 2 Matrix

➔ *Traditional Complexity*: $O(n^3)$

$$A \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \qquad B \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$$

$$A*B = \begin{bmatrix} a_1b_1 + a_2b_3 & a_1b_2 + a_2b_4 \\ a_3b_1 + a_3b_3 & a_3b_2 + a_4b_4 \end{bmatrix}$$

$$T(n) = 8 * T(n/2) + O(n^2)$$

$$O(n^{\log_2 8}) + O(n^2) = O(n^3)$$

Did you know?
The Strassen Algorithm reduces matrix multiplication complexity to $O(n^{2.31})$

Homework

Analys a^n from Divide & Conquer paradigm