week2_lec4

Main Ideas

- Master Theorem
- Algorithms
  - Merge Sort
  - Matrix Multiplication
  - Median

Divide and Conquer

Original problem is split into multiple subparts of the same type until they're simple enough. These solutions are then combined to form the final solution.

1. Master Theorem

This theorem is used to find the time complexity of an algorithm when we are aware of how the problem is split into subproblems of the same type and how these subproblems are combined to form the final solution, i.e the recurrence relation.

For a recurrence relation $T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$, then:

$O(n^d)$ being the complexity of other functions that join the subparts into a final part.

$T(n) = O(n^d) d > \log_b a$.

$$O(n^d \log_b n) d = \log_b a$$

$$O(n^{\log_b a}) d < \log_b a$$

Proof :

Here, the original problem is split into subparts recursively.

work done at kth level is $a^k \times O((\frac{n}{b^k})^d) = O(n^d) \times (\frac{a}{b^d})^k$

This geometric series has n^d as the first term and the common ratio is (a/b^d).

Total work is summation of the above series

For the **first case**, the ratio < 1. Series is always decreasing and is dominated by $O(n^d)$.

For the **second case**, the ratio is = 1. All the $log_b n$ terms all have work equal to $O(n^d)$ and hence totally the work will be $O(n^d log_b n)$

For the **third case**, the ratio > 1. The series is increasing, and its sum will be its last term:

$O(n^d \cdot (a/b^d)^{log_b n}) = O(n^{log_b a})$

where k is $log_b n$.

Algorithms

Merge Sort

Steps:

1. Split the array into two halves.
2. Recursively sort the two halves.
3. Merge the two sorted halves.

O(n) is the time taken for merging the two sorted halves.

Time Complexity → T (n) = 2T (n/2) + O(n).

By Master Theorem, we get the time complexity as O(n log n).

$\Omega(n \log n)$ is the most optimal complexity for sorting. Why?

For any n length array, there are $n!$ different permutations

When we make a comparison, we are deciding between two different permutations of the array.

To decide on an answer, i.e. to get to a leaf in the decision tree we must go through at least $\log n!$ comparisons. which is $\Omega(n \log n)$

Matrix Multiplication

Time Complexities of the methods:

- Naive:$O(n^3)$
- Strassen:$O(n^{log 7})$

- Fastest Known: $O(n^{2.37})$
- Optimum: Open!

Naive Method

Let X and Y be two n × n matrices.

Here, we divide each of them into four n2 × n2 sub-matrices and then multiply

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$T(n) = 8T(n/2) + O(n^2)$
$T(n) = O(n^3)$

Strassen's Algorithm

This algorithm needs only 7 multiplications and not 8.

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$$\begin{aligned} P_1 &= A(F-H) \\ P_2 &= (A+B)H \\ P_3 &= (C+D)E \\ P_4 &= D(G-E) \end{aligned} \qquad \begin{aligned} P_5 &= (A+D)(E+H) \\ P_6 &= (B-D)(G+H) \\ P_7 &= (A-C)(E+F) \end{aligned}$$

$$T(n) = 7T(n/2) + O(n^2)$$
$$O(n^{\log_2 7}) \approx O(n^{2.81})$$

Median

Problem : Find the mean of n numbers

Naive (Direct) Approach:

Here, we do merge sort and output the middle-ranked element.

Time complexity: $O(nlogn)$

Consider an operation: Divide and conquer:

For any number v, the list S is split into 3 categories:

- Smaller than v : S_L
- Equal to v : S_L
- Bigger than v : S_L

We can then recursively continue on the array as follows:

a. If $k \leq \mid S_L \mid$, *selection*(S_L, k)
b. If $\mid S_L \mid < k \leq \mid S_L \mid + \mid S_v \mid$, *v*

c. If $k > \mid S_v \mid + \mid S_L \mid$, *selection*$(S_R, k - \mid S_L \mid - \mid S_v \mid)$

Time Complexity : O(n)