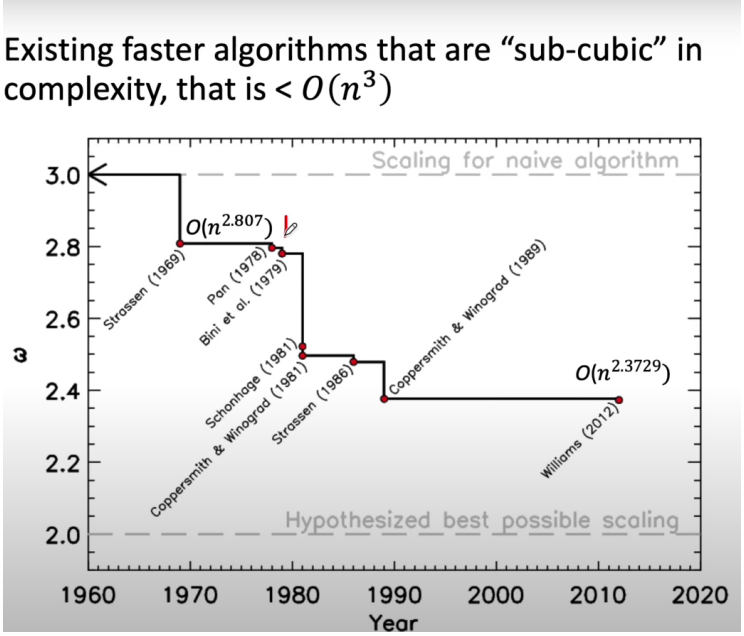
Frievald's Algorithm

History of Matrix Multiplication →



Talking about n x n Matrix Multiplication we have a faster algorithm having a time complexity of O(n^2.37) which can be seen from the above Picture which gives 100% correct result.

Is there an algorithm which takes time less than O(n^2.37) ?

We have Freivalds Algorithm, which is a randomized probabilistic algorithm that performs matrix product verification in lesser time with a probability of correctness.

What's special about it?

It takes time O(kn^2) which is less than the time taken by a simple algorithm for larger n i.e. larger size of matrix **But at what cost?**
There is a chance of failure less than 2^-k

Coming to the definition —>

Freivalds' algorithm is a probabilistic randomized algorithm used to verify matrix multiplication . Given three n x n matrices, Freivalds' algorithm determines in O(kn^2) whether the matrices are equal for a chosen k value with a probability of failure less than 2^-k.

Given three n×n matrices A,B, and C, the general problem is to verify whether A× B=C. A simple algorithm would compute the product A× B and compare term by term whether this product equals C.This will have 3 nested loops in code which might require O(n^3) time complexity, whereas Freivalds' algorithm has the time complexity O(kn^2), where k is a parameter. The Freivalds' algorithm verifies the correctness of the multiplication with probability of failure less than 2^(-k).

Freivalds' algorithm:

1. Input: n×n matrices A, B and C.

2. We would like to verify if A×B=C. Choose a n×1 column vector r→∈{0,1}n, uniformly and at random. In other words, each component of the vector is either 0 or 1 and chosen randomly and uniformly.

3. Compute A·(B·r) and C·r. This takes O(n^2) time.

4. Now comes the output

○ If A·(B·r)≠C·r, output **FALSE**. In other words, we say that the given matrix multiplication is incorrect. This has a probability of correctness = 1.

○ If A·(B·r)=C·r, output **TRUE**. In other words, we say that the given matrix multiplication is correct. This has a probability of correctness ≥1/2.

By iterating the algorithm *k* times and returning "Yes" only if all iterations yield "Yes", a runtime of O(kn^2) and error probability of <= 2^-k is achieved.

Lets see the Proof ?

Error analysis:

Let p equal the probability of error. We claim that if A × B = C, then p = 0, and if A × B ≠ C, then p ≤ 1/2.

#Case1: A × B = C:

Regardless of the value of vector r, since it uses only that A × B - C =0. Hence the probability for error in this case would be zero.

$$\begin{aligned}\vec{P} &= A \times (B\vec{r}) - C\vec{r} \\ &= (A \times B)\vec{r} - C\vec{r} \\ &= (A \times B - C)\vec{r} \\ &= \vec{0}\end{aligned}$$

$$\Pr[\vec{P} \neq 0] = 0$$

#Case2: A × B ≠ C:

Let

$$\vec{P} = D \times \vec{r} = (p_1, p_2, \ldots, p_n)^T$$

Where

$$D = A \times B - C = (d_{ij})$$

Since A × B ≠ C, we have that some element of D is nonzero. Suppose that the element d≠0.For some constant y. Using Bayes' Theorem, we can partition over y.

$$p_i = \sum_{k=1}^{n} d_{ik} r_k = d_{i1} r_1 + \cdots + d_{ij} r_j + \cdots + d_{in} r_n = d_{ij} r_j + y$$

$$\Pr[p_i = 0] = \Pr[p_i = 0 | y = 0] \cdot \Pr[y = 0] + \Pr[p_i = 0 | y \neq 0] \cdot \Pr[y \neq 0]$$

$$\Pr[p_i = 0 | y = 0] = \Pr[r_j = 0] = \frac{1}{2}$$

$$\Pr[p_i = 0 | y \neq 0] = \Pr[r_j = 1 \wedge d_{ij} = -y] \leq \Pr[r_j = 1] = \frac{1}{2}$$

$$\begin{aligned} \Pr[p_i = 0] &\leq \frac{1}{2} \cdot \Pr[y = 0] + \frac{1}{2} \cdot \Pr[y \neq 0] \\ &= \frac{1}{2} \cdot \Pr[y = 0] + \frac{1}{2} \cdot (1 - \Pr[y = 0]) \\ &= \frac{1}{2} \end{aligned}$$

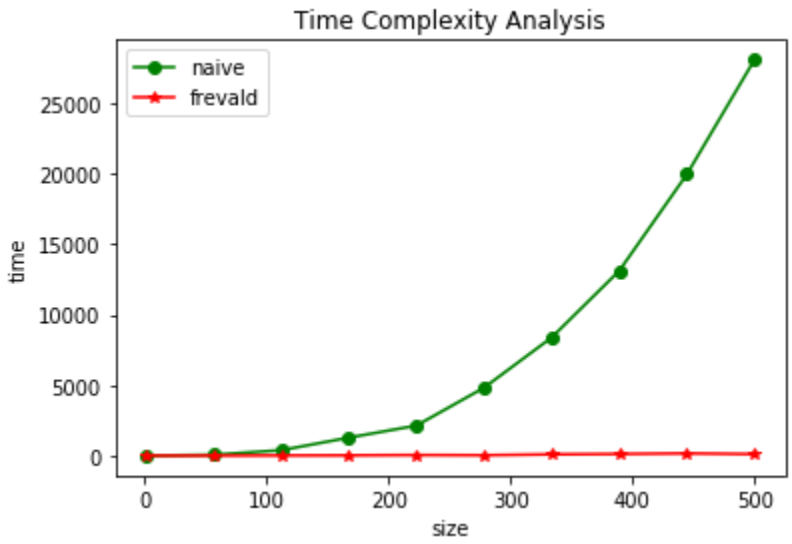
$$\Pr[\vec{P} = 0] = \Pr[p_1 = 0 \wedge \cdots \wedge p_i = 0 \wedge \cdots \wedge p_n = 0] \leq \Pr[p_i = 0] \leq \frac{1}{2}.$$

This completes the Proof.

Complexity

- **Worst case time complexity: Θ(kn^2)**
- **Space complexity: Θ(n^2)**

  k = number of times the algorithm iterates. We can choose the value of k.
- The simple matrix multiplication algorithm contains three nested loops. For each iteration of the outer loop, the total number of the runs in the inner loops would be equivalent to the length of the matrix. Here, integer operations take O(1) time. In general, if the length of the matrix is N, the total time complexity would be O(N ∗ N ∗ N) = O(N^3). ABr = A(Br),From freivald's algorithm we can see in code that we have 3 instances of an n×n matrix times an n-vector so These are O(n^ 2) time operations if done straightforwardly so,Total running time O(n^ 2)



Why does Freivalds Algorithm take less time for larger size?

For larger size n is larger and since we can choose k much larger than compared to when n is smaller so kn^2 < n^3 for simple algorithm, so it takes lesser time and also since k is larger then probability of failure which is 2^-k is smaller

This algorithm runs on O(kn^2) time complexity. This beats the classical deterministic algorithm's runtime O(n^3) or O(n^2.373) for fast matrix multiplication.

Report By —> Priyansh Sanjeev Shrivastav
210102118
ECE