WEEK 2, LECTURE 3 ON 25 AUGUST 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

So far we have dealt with both solvable and unsolvable problems.
Henceforth for a while we will deal with only *tractable* problems.

FIBONACCI NUMBERS

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Fibonacci number n is calculated using:

$$F_N = F_{N-1} + F_{N-2}$$

The problem: **WAP to take N and give F_N**

> We must ask ourselves some questions about every algorithm:
>
> - is the algorithm correct?
> - Will that take finite time? How much time?
> - Can we do better?

Recursion Algorithm

```
int fib1(int n)
{
    if(n == 0)
        return 0;
    if(n == 1)
        return 1;
    return fib1(n-1) + fib1(n-2);
}
```

The correctness of the algorithm is obvious since it follow directly from the definition of F_N

Since `fib1(n)` could be called multiple times for a single `n`, the number of recursions increases exponentially.

Can we do better? Yes.

Memoisation Algorithm

Recursion is done, but whenever a value for `fib2(n)` is calculated, store it in an array at index `n` and whenever `fib2(n)` is needed give the stored value.

```
1  int fib2(n)
2  {
3      if(n==0) return 0;
4      //create array f[0...n]
5      f[0]=0; f[1]=1;
6      for(int i =2; i<=n; i++)
7      {
8          f[i]=f[i-1]+f[i-2];
9      }
10     return f[n];
11 }
```

Correctness can be proved in the same way as the recursive algorithm.

Since the bits in $F_N \approx 0.694n$, the addition step in line 8 actually takes $O(n)$. So the overall complexity is $O(n^2)$.

Using Matrix multiplication

It is known that:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

This leads to a complexity of $O(M(n)\log n)$ where $M(n)$ is the complexity of multiplying matrices.

Using the recurrence relation formula

It is known that:

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

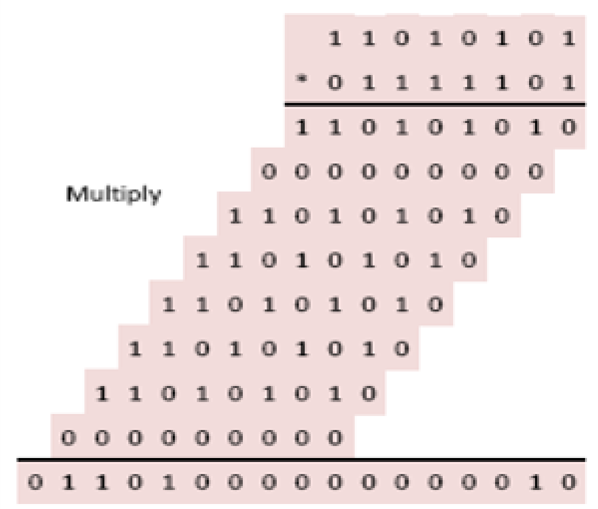
But accuracy issues arise with larger values since, memory limitations mean that irrational numbers can only be stored with limited accuracy.

Thus an $O(n \log^2 n)$ algorithm exists to compute Fibonacci numbers. Any further optimization is unknown.

MULTIPLYING LARGE INTEGERS

The Traditional Method

Two n-digit numbers a and b will need up to n^2 single digit multiplications, additions and shifts.



This algorithm will be $O(n^2)$

Karatsuba Algorithm

> link that helped

-Multiplying two complex numbers:

$$(a+ib)(c+id) = (ac-bd) + i(ad+bc)$$

This operation naively takes four multiplications namely: ac, bd, ad, bc

Instead we can:

- Compute ac
- Compute bd
- Compute $(a+b)(c+d)$

and then obtain $(ad+bc) = (a+b)(c+d) - ac - bd$

So two multiply two n-bit integers x and y, partition each of them into parts that contain $n/2$ of the bits each i.e.

$$x = 2^{n/2}a + b$$
$$y = 2^{n/2}c + d$$
$$\therefore\ x \cdot y = (2^{n/2}a + b) \cdot (2^{n/2}c + d)$$

Which is similar to the above method of computing complex products but instead of i we have $2^{n/2}$.

Using the Master Theorem (from the next lecture), we can glean the Karatsuba Algorithm's time complexity to be $O(n^{1.585})$

> The Fast Fourier transform and faster options exist with the best known yet to be $O(n \log n)$
