1. Pseudocode FFT

```
function FFT(x)
    n ← lengte(x)                ▷ Assumptie: n = 2^m voor een m, oftewel n is een tweemacht
    if n == 1 then
        X ← x
    else
        E ← FFT(x[0 :: 2])                                   ▷ Pak alle even indices
        O ← FFT(x[1 :: 2])                                   ▷ Pak alle oneven indices
        for i = 0 to n − 1 do
            if i < n/2 then
                X[i] ← E[i] + e^{−2iπk/n} · O[i]
            else
                X[i] ← E[i] − e^{−2iπk/n} · O[i]
            end if
        end for
    end if
    return X
end function
```

2. Bewijs van tijdscomplexiteit FFT

We willen bewijzen dat we de Fouriertransformatie van een of andere discrete functie kunnen vinden in $\mathcal{O}(n \log n)$ tijd, waar n de lengte van de invoer is.

Een college in complexiteitstheorie geeft ons de volgende twee definities.

$$f \in \theta(g) \Leftrightarrow \exists c \in \mathbb{R}_+ : \lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

Een *recurrente betrekking* $T(n)$ is volgt te omschrijven:

$$T(n) = \begin{cases} c & \text{als } n \leq d \\ aT(n/b) + f(n) & \text{anders,} \end{cases}$$

en geeft "het aantal stappen"van een bepaald type algoritme weer in formulevorm.

De (vereenvoudigde) stelling van Akra-Bazzi levert ons daarna het volgende resultaat:

$$\exists k \in \mathbb{N} \text{ zodat } f(n) \in \theta(n^{\log a/\log b} \log^k n) \text{ dan } T(n) \in \theta(n^{\log a/\log b} \log^{k+1} n).$$

Kijkende naar de opbouw van de algoritme, zien we dat $a = b = 2$ en $c = d = 1$. Verder is $f(n)$ te omschrijven als "het extra werk wat verricht moet worden om een tussenantwoord te krijgen", in andere woorden: het uitrekenen van de sommatie. Dit is in ons geval lineair in de lengte van de invoer, dus $k = 0$. Dit betekent dat $T(n) \in \theta(n \log n)$. Dus het aantal stappen (tijdscomplexiteit) van de FFT is $\theta(n \log n)$ dus zeker $\mathcal{O}(n \log n)$.