Individual Deep Dive: DFT vs. FFT
Jossian Garcia
10/28/25

Introduction

The goal of this project is to implement and benchmark the Fast Fourier Transform (FFT) algorithm in Kotlin and compare its performance against the Discrete Fourier Transform (DFT). The Fourier Transform is a foundational tool in mathematics, physics and engineering which allows the conversion of signals from the time domain to the frequency domain. This process gives the frequency components of a signal that are essential to look at in several applications such as signal processing, image compression, and audio filtering.

Even though the DFT can compute frequency signals, its computational cost increases quadratically with input size, at a time complexity of ($O(n^2)$). In 1965, Cooley and Turkey introduced the FFT, a divide and conquer algorithm that recursively splits a problem into smaller problems, reducing the computational cost to ($O(nlogn)$). This revolutionized computational signal processing and remains one of the most significant advances in numerical computing.

Results and Analysis

When looking at the difference of performance between the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT), I benchmarked both algorithms on input sizes ranging from 32 to 2048 points. Each input consisted of a sinusoidal sequence and the execution times were measured in milliseconds.

n (input)	DFT time (ms)	FFT time (ms)
32	1.37	0.75
64	0.75	0.34
128	2.32	0.42
256	5.76	0.37
512	9.06	0.88

n (input)	DFT time (ms)	FFT time (ms)
1028	25.70	0.63
2048	85.98	0.80

Table 1: Benchmark results for various input sizes using DFT and FFT

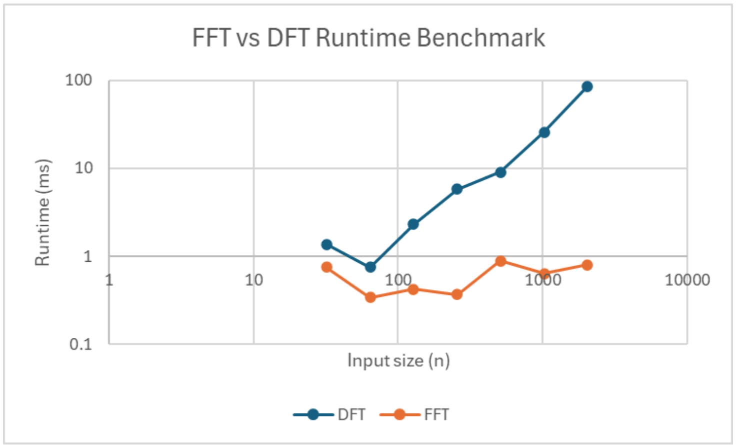

Figure 1: Log-log plot of FFT vs. DFT runtimes

The log-log plot shows a clear difference in scaling between the two algorithms. The DFT curve shows approximately a quadratic relationship between runtime and input size which lines up well with its theoretical time complexity of (*$O(n^2)$*). As n doubles, the runtime increases by about a factor of 4. The FFT curve however grows much slower at a more linear rate which corresponds to its theoretical time complexity of (*$O(nlogn)$*). The FFT does immensely improve the efficiency for large inputs and potential large datasets whereas the DFT requires tens to hundreds of milliseconds.

For small inputs (when n=32 or n=64) both algorithms perform relatively the same because the benefits of the FFT's efficiency only becomes noticeable as the data size grows. When the input size increases beyond n=256, the FFT's runtime remains nearly constant while the DFT's time grows rapidly. This difference demonstrates how FFT efficiency handles large datasets where DFT becomes computationally expensive.

Conclusion

This project implemented and benchmarked the DFT and FFT in Kotlin, confirming the expected performance between both of their time complexities. The results highlight how the FFT's divide and conquer framework enables massive gains in efficiency for large inputs, supporting its importance in world applications. Lastly, I enjoyed benchmarking and visualizing the data, as it allowed me to observe and analyze the computational efficiency of the algorithms using the code I had developed.