Complexity Analysis of Discrete Fourier Transform Algorithms

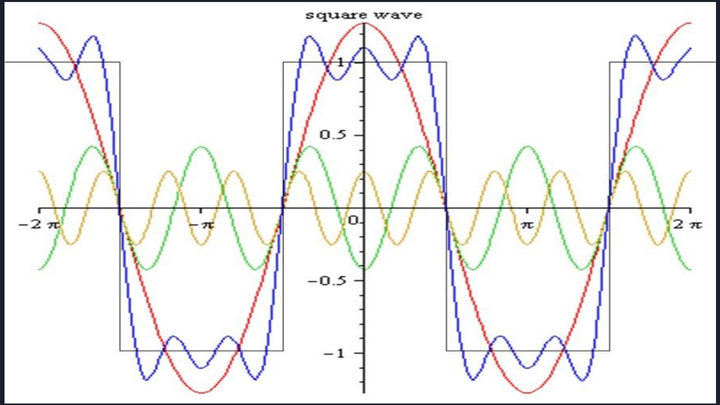
Final Project Presentation
Data Structures and Algorithms (ECE 573)

Yilin Yang
Peeyush Tambe
Siddharth Rupavatharam

Table of Contents

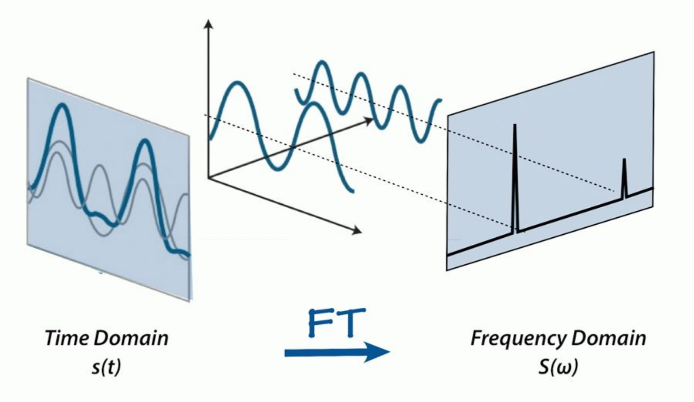
Introduction

- Fourier Transform developed by French mathematician Jean-Baptiste Joseph Fourier in late 1700's



- Any continuous function or signal can be decomposed as sum of infinite sines and cosines or complex exponentials

- Fourier Transform quantifies / measures the frequency content in a signal



Introduction

- Applications of Fourier Transform:
  - Spectrum Visualization
  - Simplifying complex operations like convolutions
  - Design and Implementation of Filters
  - Modern Communication Systems (OFDM)

- Discrete Fourier Transform was developed as the digital counterpart to the original Fourier Transform

- The Discrete Fourier Transform not feasible for real time implementations
  - Complexity $O(N^2)$

- The Fast Fourier Transform is a smarter implementation of the DFT
  - Complexity of $O(N \log(N))$ 100x faster than DFT

Algorithms: Discrete Fourier Transform

- The Discrete Fourier Transform is defined as follows:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

- Input $\rightarrow$ x(n), DFT $\rightarrow$ X(k) both length N

- N^2 number of complex multiplications and N(N-1) complex additions ($O(N^2)$ complexity)

Algorithms: Radix-2 Fast Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

- Input -> x(n), FFT ->X(k) both length N

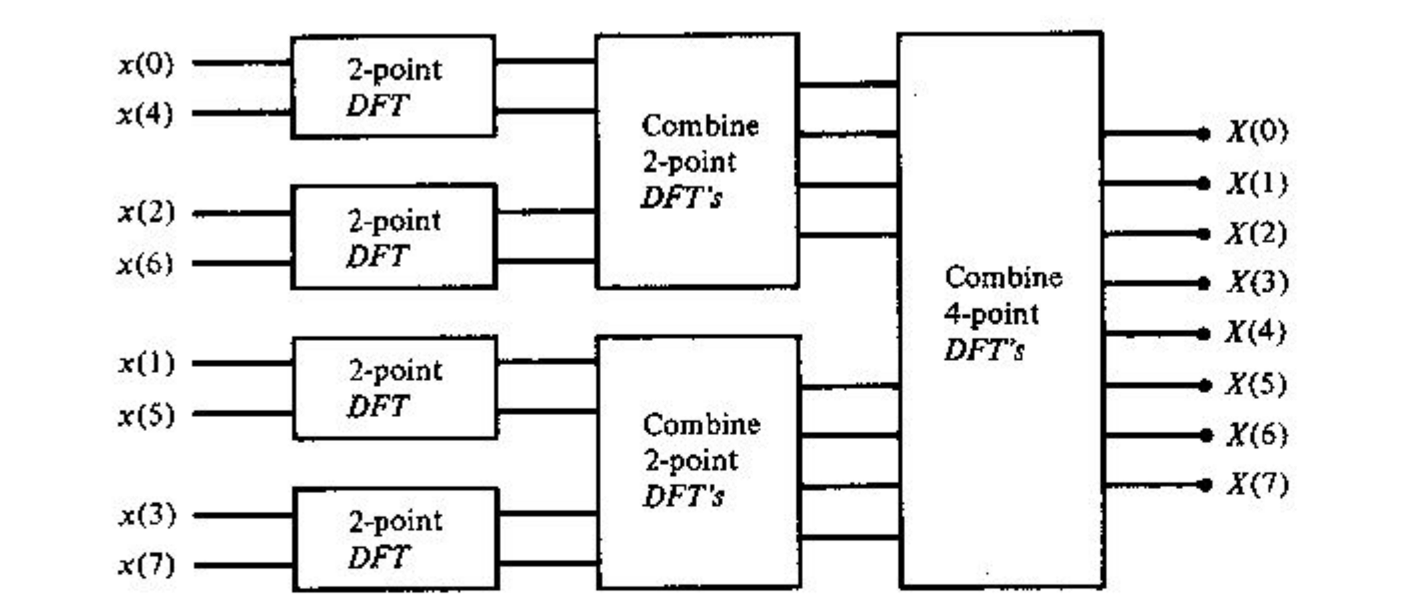
$$\mathrm{X_k} = \sum_{\mathrm{m}=0}^{\frac{\mathrm{N}}{2}-1} \mathrm{x(2.m).W_{\frac{N}{2}}^{m.k}} + \mathrm{W_N^k}.\sum_{\mathrm{m}=0}^{\frac{\mathrm{N}}{2}-1} \mathrm{x(2.m+1).W_{\frac{N}{2}}^{m.k}}$$

- Split input into even, odd halves and multiply twiddle factor with odd seq

$$\mathrm{X_k} = \mathrm{X_{k(even)}} + \mathrm{W_N^k}.\mathrm{X_{k(odd)}},\ \mathrm{k} = 0 \ldots \frac{\mathrm{N}}{2} - 1$$

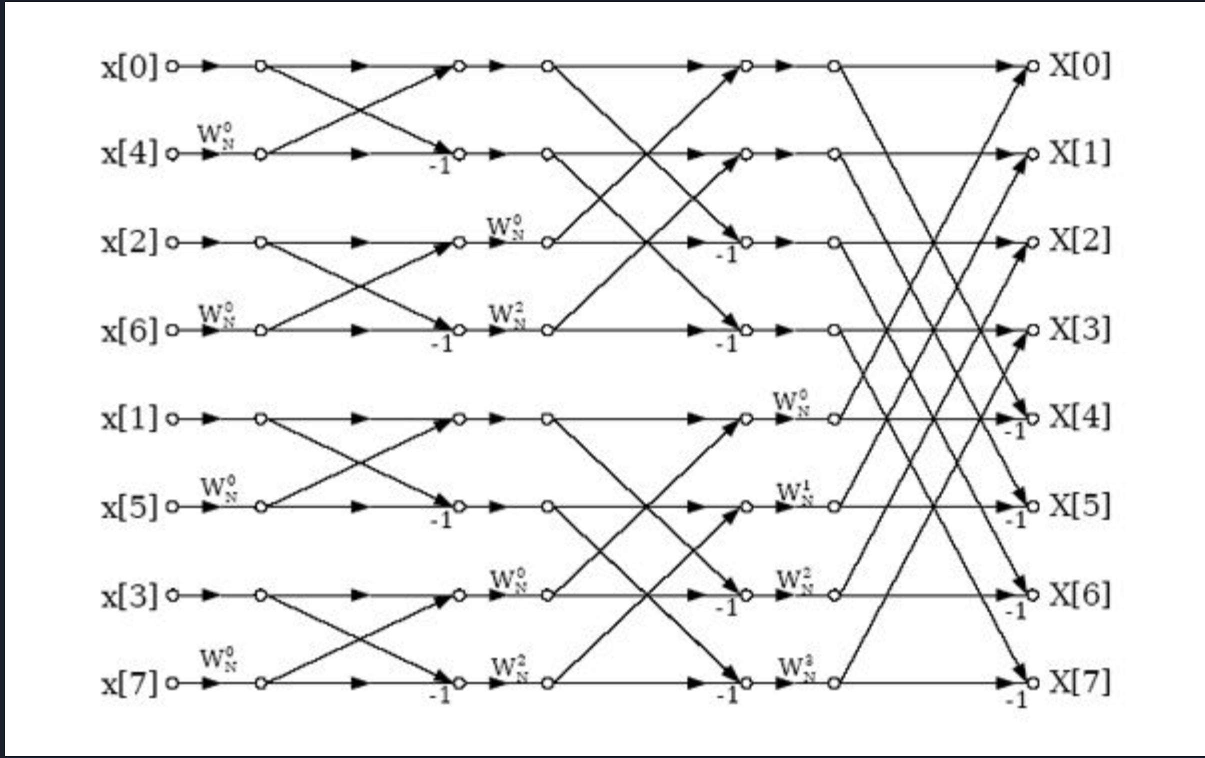
- $(N/2)^2$ number of complex multiplications reduces to (O(N log(N)) complexity

Algorithms: Radix-2 Fast Fourier Transform



Decimation in time

Algorithms: Radix-2 Fast Fourier Transform



Decimation in time

Experiment

Procedure

- DFT and FFT implemented using Python
- 4 Datasets generated for input
  - Zeros
  - Ones
  - -1 to 1
  - Cosine
- Problem sizes range from 256 (2^8) up to 32K (2^15)
- Performance evaluated by 2 criteria
  - Execution time
  - Number of operations

Hardware

- Experiments run on 2 machines
  - 64-bit Ubuntu platform running Linux OS
  - 64-bit Windows platform running virtual machine of Linux OS

Simulations

- 128 simulations performed
  - 2 Algorithms
  - 4 Datasets
  - 8 Problem Sizes
  - 2 Machines

Results

Dataset: Ones

Problem Size		DFT (Ubuntu VM)	DFT (Ubuntu)	FFT (Ubuntu VM)	FFT (Ubuntu)
256	2^8	0.11174	0.04728	0.00218	0.0063
512	2^9	0.29339	0.181	0.00849	0.001
1k	2^10	1.12745	0.72603	0.02346	0.016
2k	2^11	3.75467	2.85524	0.04848	0.0234
4k	2^12	14.83544	11.36227	0.09605	0.0379
8k	2^13	97.58259	52.01844	0.17482	0.0708
16k	2^14	359.97122	230.09249	0.2375	0.1477
32k	2^15	1160.80973	855.50801	0.4751	0.3079

Problem Size		DFT (Ubuntu VM)	DFT (Ubuntu)	FFT (Ubuntu VM)	FFT (Ubuntu)
256	2^8	0.12363	0.05373	0.0025	0.0067
512	2^9	0.29977	0.18792	0.0086	0.0125
1k	2^10	1.0552	0.70518	0.0204	0.013
2k	2^11	4.10548	2.81096	0.0503	0.0207
4k	2^12	15.2145	11.1803	0.1013	0.0371
8k	2^13	89.88514	50.29091	0.1536	0.0703
16k	2^14	395.60079	224.86547	0.33671	0.14658
32k	2^15	1265.86637	846.19161	0.7582	0.309

Dataset: Zeros

Results

Dataset: -1 to 1

Problem Size		DFT (Ubuntu VM)	DFT (Ubuntu)	FFT (Ubuntu VM)	FFT (Ubuntu)
256	2^8	0.1307	0.05571	0.00241	0.0054
512	2^9	0.29633	0.17978	0.00841	0.009
1k	2^10	0.96175	0.70308	0.026	0.017
2k	2^11	5.03086	2.77729	0.0589	0.0203
4k	2^12	22.94392	11.08887	0.09496	0.0407
8k	2^13	88.62576	52.68572	0.1955	0.0719
16k	2^14	356.47654	221.95255	0.2547	0.1522
32k	2^15	1204.3287	879.47403	0.4598	0.3054

Problem Size		DFT (Ubuntu VM)	DFT (Ubuntu)	FFT (Ubuntu VM)	FFT (Ubuntu)
256	2^8	0.10748	0.05634	0.0022	0.0062
512	2^9	0.28475	0.18265	0.0053	0.0038
1k	2^10	1.06186	0.70221	0.00949	0.0158
2k	2^11	4.9707	2.8	0.03841	0.0226
4k	2^12	16.20032	11.1536	0.09208	0.0397
8k	2^13	82.95861	50.31715	0.15306	0.0734
16k	2^14	353.26238	225.3965	0.2559	0.1484
32k	2^15	0.10748	0.05634	0.0022	0.0062

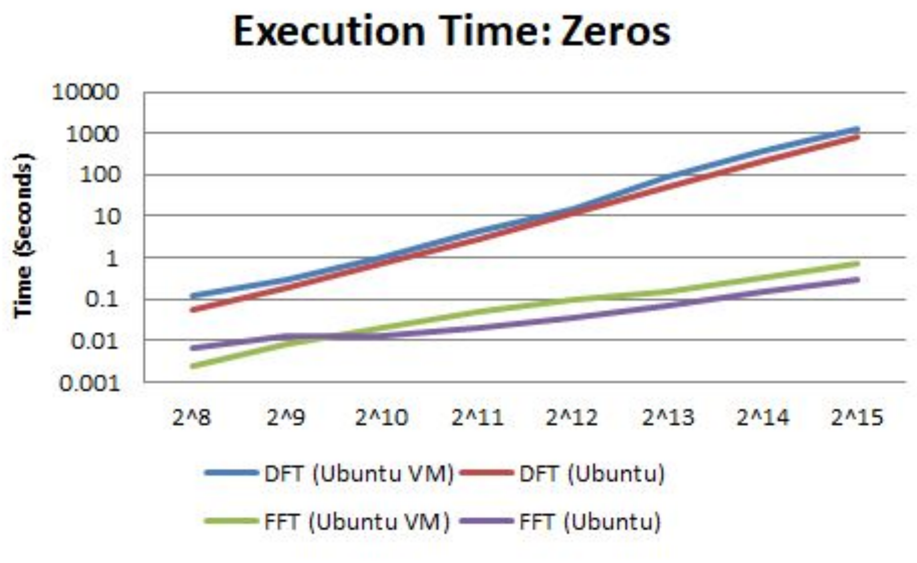
Dataset: Cosine

Results

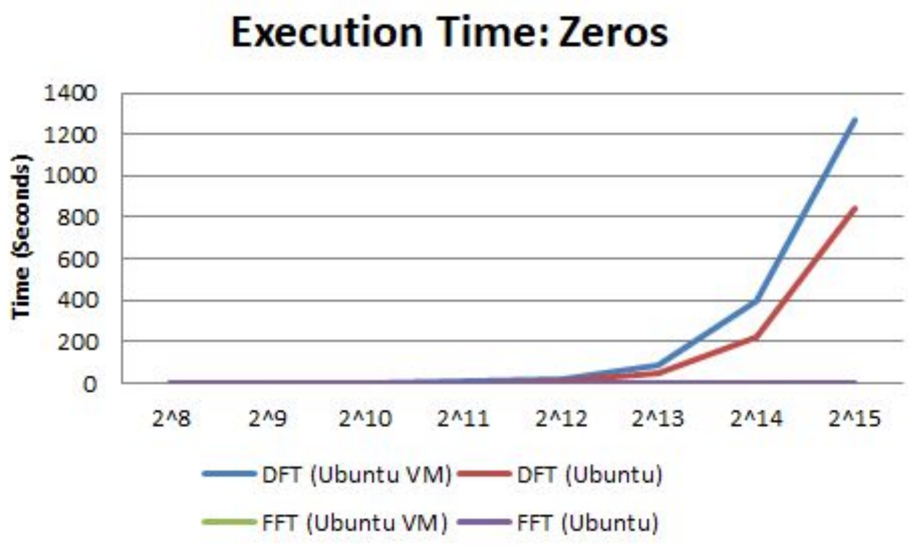
Number of Operations Performed

Problem Size		DFT	FFT
256	2^8	65536	2048
512	2^9	262144	4608
1k	2^10	1048576	10240
2k	2^11	4194304	22528
4k	2^12	16777216	49152
8k	2^13	67108864	106496
16k	2^14	268435456	229376
32k	2^15	1073741824	491520

Results

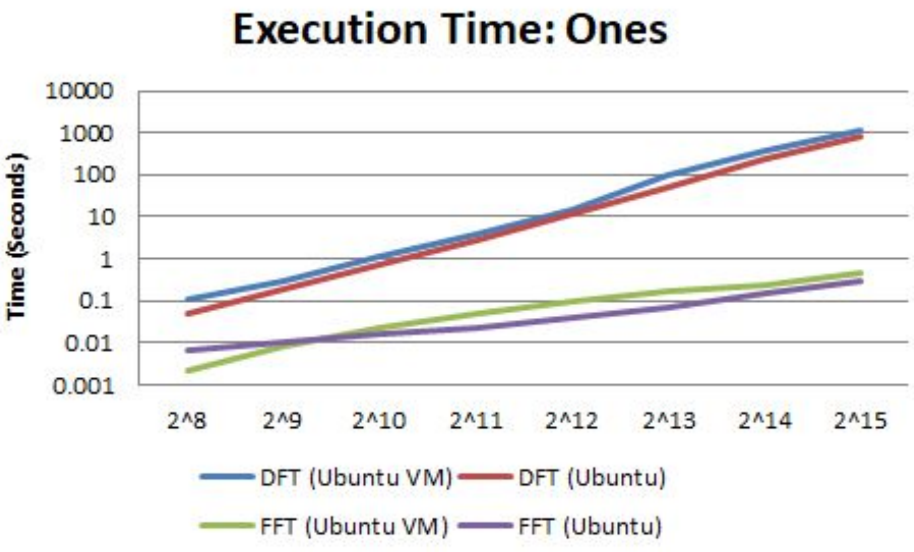


Logarithmic Scale

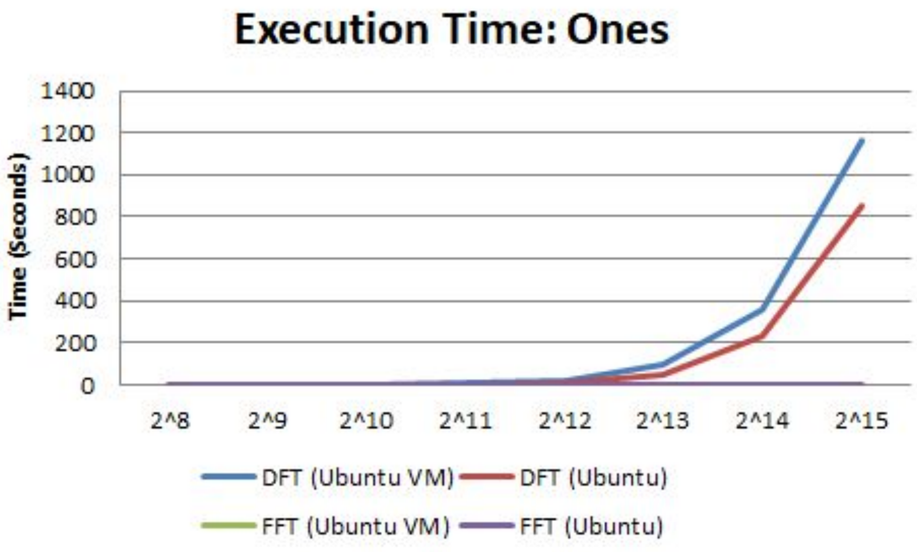


Linear Scale

Results

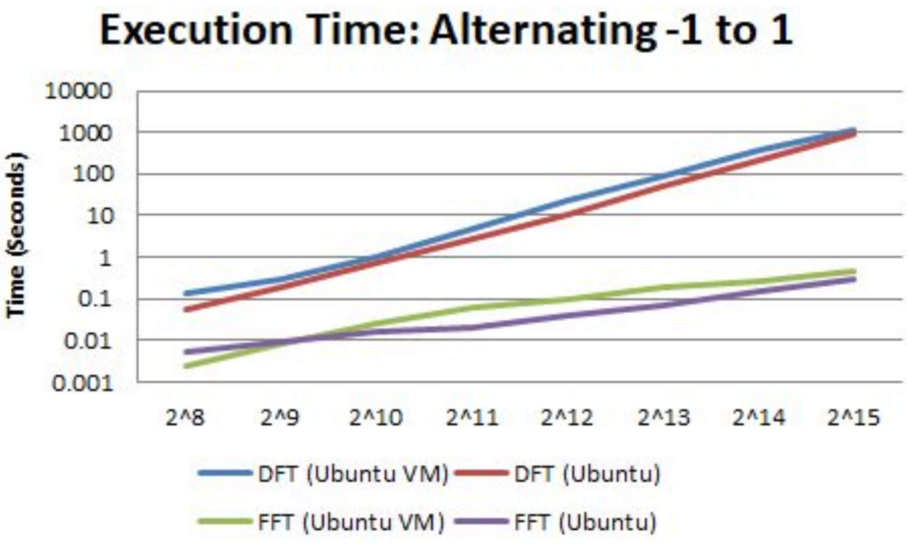


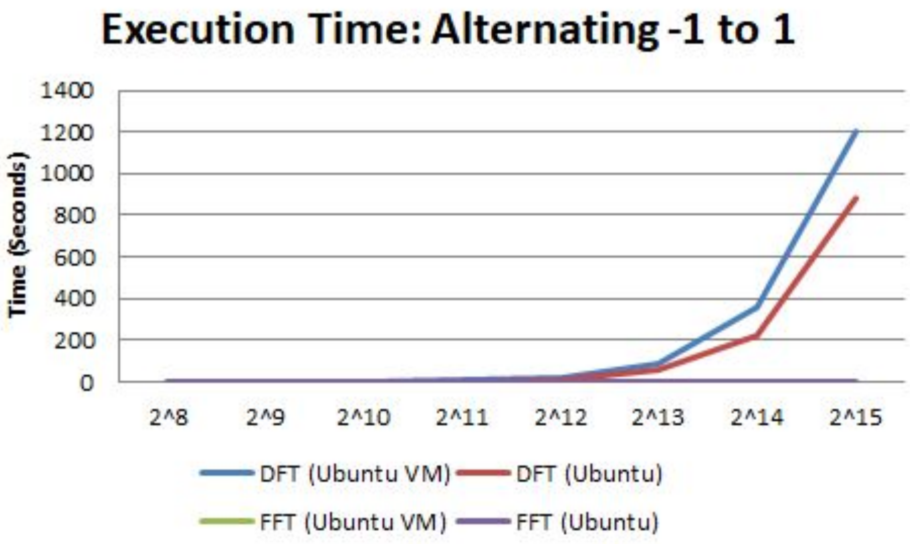
Logarithmic Scale



Linear Scale

Results

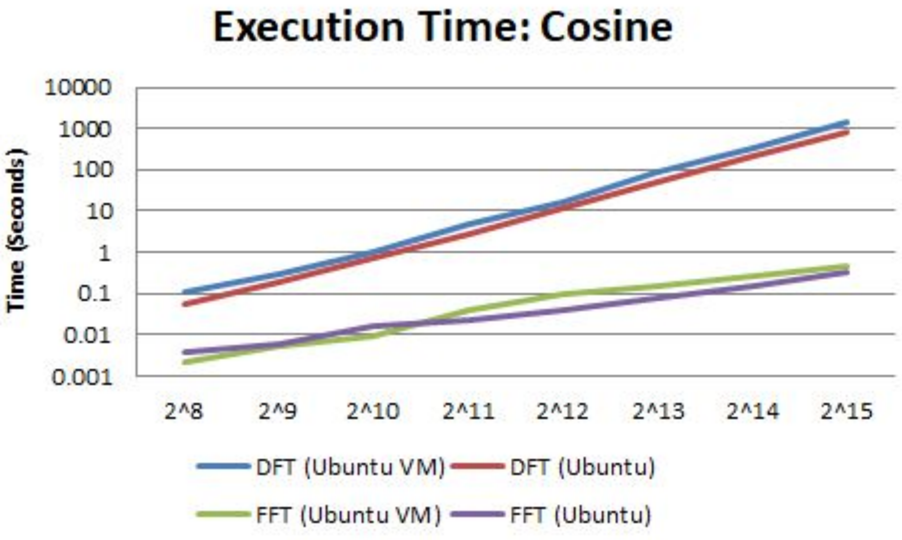


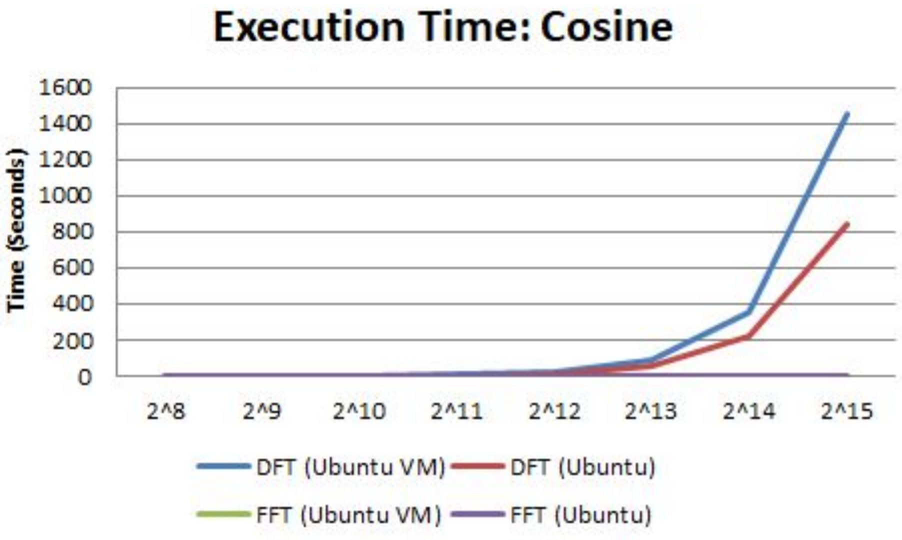


Logarithmic Scale

Linear Scale

Results

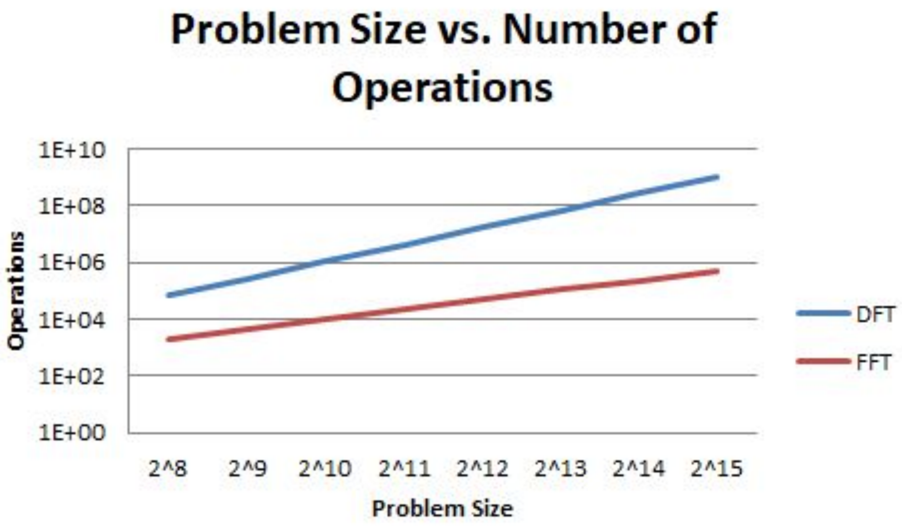


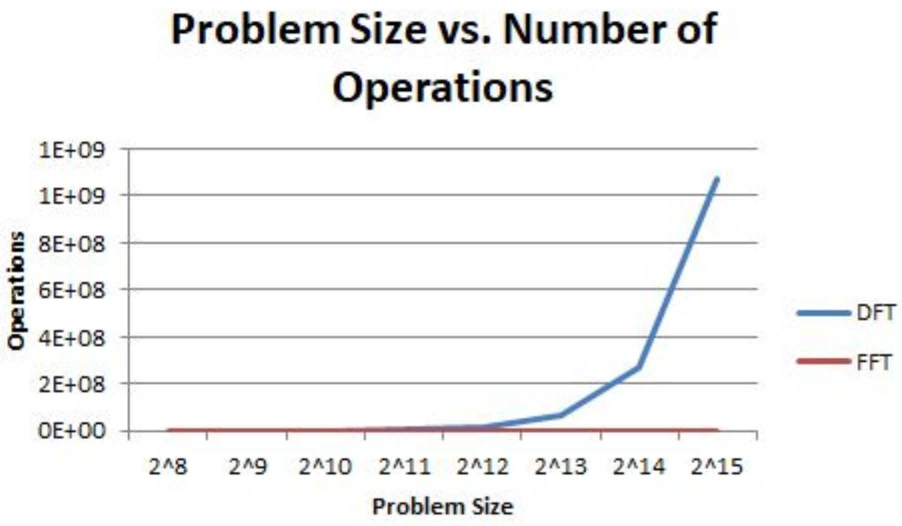


Logarithmic Scale

Linear Scale

Results





Logarithmic Scale **Linear Scale**

Conclusions

- 10x to <1000x reduction in number of operations
- 10x to >1000x increase in speed

- DFT (of length N) calculated by a summation of input sequence (length N) elementwise
  - $\rightarrow$ O(N^2) complexity

- Radix-2 FFT recursively halves input sequence(length N)to compute DFT on halved sequence
  - $\rightarrow$ O(N log(N)) complexity

QUESTIONS?