Article

Not peer-reviewed version

Computational Complexity of Radix-2, Radix-4 and Bluestein Algorithms Implementation of the Discrete Fourier Transform (DFT)

Georgios Giannakopoulos [*] , Khushbu Mehboob Shaikh , Maria Antonnette Perez

Posted Date: 23 January 2025

doi: 10.20944/preprints202501.1681.v1

Keywords: Discrete Fourier Transform; Fast Fourier Transform; Radix-2; Radix-4; Bluestein Algorithm; Computational Complexity; Signal Processing; MATLAB Implementation

Article

Computational Complexity of Radix-2, Radix-4 and Bluestein Algorithms Implementation of the Discrete Fourier Transform (DFT)

Georgios Giannakopoulos [1], Khushbu Mehboob Shaikh [2] and Maria Antonnette Perez [3]

[1] Independent Researcher, The Hague, The Netherlands
[2] Technical Lead, Staff Technical Account Manager, Twilio Inc., Irving, Texas, United States
[3] Independent Researcher, Manila, Philippines

Abstract: The computational complexity of Discrete Fourier Transform (DFT) algorithms plays a pivotal role in signal processing, influencing their applicability in various domains. This paper investigates three prominent Fast Fourier Transform (FFT) algorithms: Radix-2, Radix-4, and Bluestein, with a focus on their computational efficiency and suitability for different sequence lengths. MATLAB implementations were developed to optimize these algorithms, reducing the number of multiplications and additions required during runtime. A comparative analysis reveals that Radix-2 and Radix-4 algorithms are highly efficient for power-of-two and power-of-four data lengths, respectively, while the Bluestein algorithm provides unparalleled flexibility for arbitrary sequence lengths, including primes. The study demonstrates the trade-offs associated with each algorithm, highlighting their strengths and limitations. Radix-4 achieves greater efficiency over Radix-2 for longer sequences, while Bluestein eliminates the need for zero-padding at the cost of increased computational complexity. This research offers valuable insights into the selection of FFT algorithms based on application-specific requirements and data characteristics, laying the groundwork for further optimization and hybrid algorithm development. The findings underscore the enduring importance of FFTs in addressing the computational demands of modern signal processing tasks.

Keywords: Discrete Fourier Transform; Fast Fourier Transform; Radix-2; Radix-4; Bluestein Algorithm; Computational Complexity; Signal Processing; MATLAB Implementation

---

1. Introduction

The Discrete Fourier Transform (DFT) is a cornerstone of modern signal processing, enabling the transition between time-domain and frequency-domain representations of discrete signals. While the DFT is computationally intensive for large data sets, the development of Fast Fourier Transform (FFT) algorithms has revolutionized this field by significantly reducing computational complexity. Among the various FFT algorithms [1], Radix-2, Radix-4, and Bluestein [2] stand out for their unique advantages and applicability to different scenarios [3].

This paper investigates the computational complexity of these three prominent FFT algorithms, which have been implemented and analyzed using MATLAB [4]. Radix-2, known for its simplicity and efficiency for power-of-two data lengths, remains a widely adopted algorithm in many practical applications [3]. On the other hand, Radix-4 enhances this efficiency by processing four data points at a time, reducing the number of stages and multiplicative operations. However, both algorithms are constrained by their dependence on specific sequence lengths.

The Bluestein algorithm, often referred to as the chirp-z transform method, addresses this limitation by enabling FFT computations for arbitrary sequence lengths, including prime numbers. By reformulating the DFT as a convolution problem, the Bluestein algorithm leverages FFTs to compute convolutions efficiently. While this approach introduces additional computational overhead, it eliminates the need for zero-padding, which can distort frequency-domain results in certain applications [5].

doi:10.20944/preprints202501.1681.v1

To minimize computational complexity and assess the practical performance of these algorithms, MATLAB implementations were developed with optimization strategies to reduce the number of multiplications during runtime. These implementations facilitated a comparative analysis of the three algorithms, focusing on their multiplication and addition requirements across various data lengths.

This study aims to provide a comprehensive understanding of the trade-offs involved in selecting an FFT algorithm for specific applications. By evaluating the strengths and limitations of Radix-2, Radix-4, and Bluestein algorithms [2], this research offers valuable insights into their suitability for diverse signal processing tasks. Additionally, the findings contribute to the ongoing development and optimization of FFT algorithms, ensuring their continued relevance in addressing the computational demands of modern signal processing applications.

2. Complexity of Radix-2 DIT/DIF Algorithm

A MATLAB program has been written and is in Appendix 1, to compute the FFT of a set of discrete data: $\{x_n\}, n = 1, 2, 3, \ldots, N$ where $N = 2^n$ and n is an integer. A modification of the supplied Radix-2 program was made in order to reduce the number of multiplications involved and is highlighted in the attached Radix-2 program [3], [6].

The final loop in the program made use of the property $e^{-j\frac{2\pi(i-1)}{N}} = -e^{-j\frac{2\pi\left(\frac{N}{2}i+1\right)}{N}}$, which allowed the number of multiplications in the final loop to be halved. The program uses a recursive procedure which calls up the FFT of the odd and even set of data points [7].

In the special case when $N = 2$, there are no multiplications to evaluate and only 2 additions. In the general case when $N = 2^n$, the recursive procedure calls up the Fourier transform of two sets of data, each of length $N/2 = 2^{n-1}$. The final loop in the procedure contains $N/2$ multiplications and N additions. Using $N = 2^n$, $n = 1, 2, 3, \ldots,,$ let $M(n)$ represent the number of computed multiplications and $A(n)$ represent the number of computed additions. It can be deduced that $M(n)$ and $A(n)$ satisfy the difference equations:-

$$M(n+1) = M(n) + 2^n, n = 1, 2, 3, 4, \ldots \text{where } M(1) = 0 \tag{1}$$

$$A(n+1) = A(n) + 2^n, n = 1, 2, 3, 4, \ldots \text{where } A(1) = 2 \tag{2}$$

The solutions to these difference equations can be obtained by taking a z-transform of the equation. It can be shown that the formulas for $M(N)$ and $A(N)$ [7], [8] are given by:

$$M(n) = \frac{N}{2}[\log_2 N - 1] \tag{3}$$

$$A(n) = N \log_2 N \tag{4}$$

The values for $M(N)$ and $A(N)$ have been tabulated as shown in the Table 1 below:

Table 1. Radix-2 DIT and DIF Table, [1].

Radix-2 DIT and DIF			
n	**N**	**Number of Multiplications**	**Number of Additions**
2	2	0	2
4	4	2	8
8	8	6	24
16	16	24	64
32	32	64	160

This data is consistent with Cooley and Tukey algorithm [1] formula $M(n) = \frac{N}{2}\log_2(N)$ for large values of N. The above results were programmed with MATLAB and are listed in Appendix 1

3. Complexity of Radix-4 DIT/DIF Algorithm

A MATLAB program has been written and is in Appendix 2, to compute the FFT of a set of discrete data: $\{x_n\}, n = 1, 2, 3, \ldots, N$ where $N = 4^n$ and n is an integer.

The program uses a recursive call of a function that computes the FFT of the odd and even data points. The original program was modified to reduce the loop length by a factor of 4 by making use of the property $e^{-j\frac{2\pi(i-1)}{N}} = -e^{-j\frac{2\pi\left(\frac{N}{4}i+1\right)}{N}}$. Hence, the number of multiplications in the loop was halved [3].

In the special case when $N = 4$, there are no multiplications to evaluate and only 8 additions. In the general case when $N = 4^n$, let $M(n)$ represent the number of computed multiplications in the program.

The program has 4 function calls of the FFT of data which is half the original length and therefore contains $4 \times M(n-1)$ multiplications. The for loop in the algorithm contains one multiplication which is repeated $N/4$ times (i.e., 4^{n-1} times) [5], [9]. Hence, the number of multiplications in the program can be computed from the sequence:

$$M(n+1) = M(n) + 4^n, n = 1, 2, 3, 4, \ldots, \text{ where } M(1) = 0 \tag{5}$$

The solution to this difference equation can be shown to be given by [10], [11]:

$$M(n) = \frac{3N}{4}[\log_4 N - 1] \tag{6}$$

The number of additions in the program can be computed from the sequence:

$$A(n+1) = A(n) + 4^n, n = 1, 2, 3, 4, \ldots, \text{ where } A(1) = 0 \tag{7}$$

The solution to this difference equation can be shown to be given by:

$$A(n) = 2N \log_4 N = 2N \frac{\log_2 N}{\log_2 4} = N \log_2 N \tag{8}$$

This sequence is tabulated for various values of n and N as shown in Table 2 below:

Table 2. Radix-4 DIT and DIF Table [1,3]

Radix-4 DIT and DIF			
n	**N**	**Number of Multiplications**	**Number of Additions**
1	4	0	8
2	16	9	64
3	64	63	384
4	256	351	2048
5	1024	1791	10240

Compared to Radix-2, the Radix-4 algorithm, for large N, has reduced the number of multiplications by 62.5%, but the number of additions has remained unchanged [1], [3].

4. Complexity of Bluestein

The Bluestein algorithm [2] is a digital filtering approach to determine the DFT of a set of data and is valid for any value of N [7], [8].

Consider an input sequence $x(n)$, $0 \leq n \leq N-1$, which is input to a digital filter with an impulse response

$$h(n) = \begin{cases} 0 & \text{if} \quad n < 0 \\ e^{-j\pi n^2/N} & \text{if} \quad 0 \leq n \leq 2N-1 \\ 0 & \text{if} \quad n \geq 2N \end{cases}$$

The output of the filter in the range $N \leq n \leq 2\,N-1$ can be shown to be given by:

$$y(n) = \sum_{r=0}^{N-1} x_r h(n-r), \quad N \leq n \leq 2N-1 \tag{9}$$

By substituting $\mathrm{k} = \mathrm{n} - \mathrm{N}$ into (9), the output of the filter can be expressed in the form:

$$y(k) = e^{j\pi\left(k^2/N\right)} e^{j\pi N} \sum_{r=0}^{N-1} \left\{ x_r e^{j\pi\left(r^2/N\right)} \right\} e^{-j(2\pi/N)rk}, \quad 0 \leq k \leq N-1 \tag{10}$$

This equation implies that $\mathrm{y(n)}$ is the weighted output of the N point DFT of the sequence $x_r e^{j\pi\left(r^2/N\right)}$. This result suggests that the DFT of a sequence can be obtained by first pre-multiplying the input sequence by the factor of $\mathrm{e}^{-\mathrm{j}\pi\left(\mathrm{n}^2/N\right)}$ and inputting this sequence to a digital filter [10–12]. The output sequence can be obtained multiplying the filter output by a factor of $e^{-j\pi\left((n-N)^2/N\right)} e^{-j\pi N}$. This is called Bluestein's algorithm [2]. It should be noted that the filter output can be determined from the filter input by using a convolution of the impulse filter response and the input data [13]. A MATLAB program has been written to determine the DFT of a random set of data which uses the built-in MATLAB convolution function [7]. However, the complexity of the built-in MATLAB convolution function is not known. Therefore, a MATLAB program was written to evaluate the convolution using the result that the convolution of a function can be determined from the inverse transform of the transform products of the impulse response and the input data [6].

The convolution calculation in our MATLAB program requires three calculations of the FFT transform of a signal, which is of length $2N$ [5], [9]. However, we need not include the FFT transform of the impulse response in our complexity calculation, since this can be precalculated and is independent of the data [6].

As seen in the modified Bluestein MATLAB program, the number of multiplications can be summarized as follows:

$$\text{Bluestein Multiplications:-} \begin{cases} \text{N Scaling Inputs} \\ \text{2 FFTs each of length 2N} \\ \text{2 N FFT products} \\ \text{N Scaling Outputs} \end{cases}$$

Since the FFT calculations in the Bluestein Algorithm used the Radix-4 algorithm [14], [15], then the total number of multiplications computed, $M(N)$, is given by the formula:

$$M(N) = 4N + 2 \times \frac{3(2N)}{4}[\log_4(2N) - 1] \quad = 4N + \frac{3N}{2}[\log_4(2N) - 1] \tag{11}$$

For large N, this formula tends to $M(N) = \dfrac{3N}{2}\log_4(N) = \dfrac{3N}{4}\log_2(N)$. This is an increase of 50% over the Radix-2 calculation [16].

In the case of the addition complexity of the Bluestein algorithm in our program, this count is equal to the count of 2 FFT's of length $2N$, using the Radix-4 algorithm [5].

$$\text{i.e.,} \quad A(n) = 2 \times (2N)\log_2(2N) = 4N^2 \log_2(2N) \tag{12}$$

doi:10.20944/preprints202501.1681.v1

For large N , this value is four times that of the Radix-2 calculation.

It should be noted that the Bluestein algorithm is less efficient than the Radix-2 and Radix-4 algorithms [17], but has the advantage that the value of N does not need to be a power of 2 [1], [9]. The above results were programmed with MATLAB and are listed in Appendix 2.

5. Theoretical and Mathematical Model of Bluestein Algorithm Using Three FFTs for Convolution

The Bluestein algorithm [2], [3], also known as the chirp z-transform method, is a versatile approach for computing the Discrete Fourier Transform (DFT) of sequences where the length N is not restricted to powers of two [1]. This algorithm reformulates DFT as a convolution, enabling its computation via fast Fourier transforms (FFT), making it particularly efficient when N is not suitable for traditional radix-based FFT algorithms [6].

5.1. Formulation of the DFT

Let $x(n)$ be an input sequence of length N. The DFT of $x(n)$ is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad 0 \le k \le N-1 \tag{13}$$

5.2. Bluestein's Reformulation

Bluestein's [2] key insight was to rewrite the complex exponential term:

$$e^{-j2\pi kn/N} = e^{-j\pi kn/N} \cdot e^{j\pi k^2/N} \cdot e^{-j\pi n^2/N} \tag{14}$$

This allows us to express the DFT as:

$$X(k) = e^{j\pi k^2/N} \sum_{n=0}^{N-1} [x(n)e^{-j\pi n^2/N}] \cdot [e^{-j\pi(k-n)^2/N}] \tag{15}$$

5.3. Convolution Formulation

The definition of the two sequences is:

$$a(n) = x(n)e^{-j\pi n^2/N}, \quad n = 0, 1, \ldots, N-1 \tag{16}$$

$$b(n) = e^{-j\pi n^2/N}, \quad n = -(N-1), \ldots 0, \ldots N-1 \tag{17}$$

By using the (16) and (17), the DFT can now be expressed as a convolution:

$$X(k) = e^{j\pi k^2/N} \sum_{n=0}^{N-1} a(n)b(k-n) \tag{18}$$

5.4. Implementation by using 3 FFTs

To compute this convolution efficiently, we use the convolution theorem, which states that convolution in the time domain is equivalent to multiplication in the frequency domain [16], [13]. The $a(n)$ and $b(n)$ can be extended with zeros to length $M \ge 2N-1$. Typically, M is chosen as the next power of 2 greater than or equal to $2N-1$ for efficient FFT computation. This leads to the following steps:

The FFTs of the padded sequences can be computed as:

$$A(k) = \text{FFT}\{a(n)\} \tag{19}$$

$$B(k) = \text{FFT}\{b(n)\} \tag{20}$$

Multiply the FFTs pointwise by using the (19) and (20):

$$C(k) = A(k) \cdot B(k) \tag{21}$$

Then the (21) can compute the inverse FFT of the product:

$$c(n) = \text{IFFT}\{C(k)\} \tag{22}$$

By extracting the first N points of $c(n)$ from the (22) and multiply by the chirp:

$$X(k) = e^{j\pi k^2/N} c(k), \quad k = 0, 1, \dots, N-1 \tag{23}$$

5.5. Complexity Analysis

The complexity of this algorithm is dominated by the three FFT operations (two forward and one inverse) of length M [12]. Each FFT has a complexity of $O(M \log M)$. Since M is chosen to be the next power of 2 greater than $2N - 1$, we have $M < 4N$. Therefore, the overall complexity of the Bluestein algorithm is $O(N \log N)$ [8].

More precisely, if we use a radix-2 FFT algorithm, the number of complex multiplications is approximately:

$$3 \cdot \frac{M}{2} \log_2 M + 2N \tag{24}$$

The first term accounts for the three FFTs, and the additional $2N$ multiplications come from the pre- and post-processing steps [13].

5.6. Advantages and Considerations

The Bluestein algorithm [2] allows computation of DFTs for any length N, including prime lengths. It maintains the $O(N \log_2 N)$ complexity of the FFT for all N [5]. The algorithm introduces some overhead due to zero-padding and additional multiplications, making it less efficient than specialized FFT algorithms for highly composite lengths. The flexibility in choosing M allows for potential optimizations in specific hardware implementations [14].

In conclusion, the Bluestein algorithm provides a versatile approach to computing DFTs of arbitrary lengths while maintaining the efficiency of the FFT [7]. Its reformulation of the DFT as a convolution, computed via three FFTs, makes it a valuable tool in signal processing applications where the input length is not a power of two or a highly composite number [15].

6. Discussion and Conclusion

The computational complexity analysis of the Radix-2, Radix-4, and Bluestein algorithms, as presented in this study, provides key insight into the trade-offs [7] inherent in selecting an appropriate algorithm for Discrete Fourier Transform (DFT) computations [14]. Each algorithm has its advantages and limitations, which depend on the specific application requirements and sequence length N [5].

The Radix-2 algorithm, fundamental in FFT computations, achieves an efficient computational complexity of $O(N \log_2 N)$ for sequence lengths that are powers of two. Its simplicity and iterative nature make it suitable for general-purpose applications where the input size N aligns with its constraints. However, when N deviates from a power of two, zero-padding becomes necessary, which can introduce computational overhead [16], [18].

The Radix-4 algorithm extends the efficiency of Radix-2 by grouping data into quartets rather than pairs. This approach reduces the number of multiplicative operations required, achieving a complexity of $\frac{3}{4} N \log_4 N$, translating to significant computational savings for large N. However, the trade-off lies in the additional programming complexity and the restriction that N must be a power of four [17]. For applications where N meets this criterion, the Radix-4 algorithm provides a compelling choice with superior performance.

The Bluestein algorithm [2] emerges as a versatile solution for DFT computations with arbitrarily arbitrary N, including prime lengths or lengths that are not suitable for radix-based FFTs [17]. By reformulating the DFT as a convolution, Bluestein uses the FFT to compute this convolution efficiently [6]. However, the flexibility of the algorithm comes at the cost of increased computational complexity, particularly for large N. Using three FFTs of length $2N$, its complexity is approximately $9N \log_2(2N)$, which is higher than the Radix-2 and Radix-4 algorithms for highly composite N. However, the algorithm's ability to handle non-standard N without zero padding makes it indispensable in scenarios where input length restrictions cannot be relaxed [3].

A comparative analysis reveals that while the Radix-2 and Radix-4 algorithms excel in computational efficiency for their specific N constraints, they lack the flexibility of the Bluestein approach [17]. In contrast, Bluestein [2] provides a robust alternative for arbitrary N but at the expense of increased computational overhead and implementation complexity.

In practice, the choice of algorithm depends on the nature of the application. For general-purpose computations where N aligns with powers of two or four, Radix-2 and Radix-4 offer optimal performance [7]. For applications in digital signal processing and related fields, where N is dictated by the data acquisition process or hardware limitations, the flexibility of the Bluestein algorithm becomes essential despite its higher complexity.

Another consideration is the hardware implementation of these algorithms. Radix-based FFTs benefit from streamlined hardware designs because of their structured nature [17]. In contrast, reliance on the Bluestein algorithm on convolution computations introduces additional design complexity, which can impact real-time applications.

In conclusion, this study highlights the importance of understanding the computational and implementation trade-offs associated with each FFT algorithm. The Radix-2 and Radix-4 algorithms remain highly efficient for sequences with lengths that match their structural constraints, while the Bluestein algorithm provides unparalleled flexibility for arbitrary lengths [10]. Future work may explore hybrid approaches that combine the strengths of these algorithms or optimize hardware implementations for specific application domains, ensuring that the computational requirements of modern signal processing tasks are met effectively.

6.1. Future Work

Future research could focus on several promising directions to enhance the utility and efficiency of FFT algorithms. One potential avenue is the optimization of these algorithms for specific hardware architectures, such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs). Using parallel processing capabilities and hardware-specific designs could significantly reduce computational overhead and improve execution speeds.

Another area of exploration is the development of hybrid FFT algorithms that combine the strengths of the Radix-2, Radix-4, and Bluestein approaches [17]. Such hybrids could adapt dynamically to input characteristics, providing a balance between computational efficiency and flexibility.

Furthermore, the application of machine learning techniques to predict the most efficient FFT method for given data lengths and constraints could prove valuable. Adaptive systems that select algorithms based on real-time input characteristics and available computational resources could revolutionize the usage of FFT in dynamic environments.

Finally, extending the application of these algorithms to emerging domains, such as quantum computing and big data analytics, presents exciting opportunities. Quantum FFTs could leverage quantum parallelism, while big data applications may benefit from distributed and cloud-based FFT implementations that handle massive datasets effectively.

These directions underscore the enduring relevance and potential of FFT research, ensuring its continued impact on signal processing and related fields.

Acknowledgments

The authors acknowledge contributions from institutions and funding bodies supporting the study of Computational Complexity of Radix-2, Radix-4 and Bluestein Algorithms Implementation of the Discrete Fourier Transform (DFT).

Additional information

The authors declare that they have no conflict of interest.

Appendix A. Matlab Code For The Bluestein Algorithm Implementing Convolution

```

%Theory and application of digital signal processing Lawrence R,
%Rabiner-Bernard Gold pp392-396
%A linear Filetering approach to the computation of discrete fourier
%transform

clear variables;
N=64;
j=complex(0,1);

data = rand([1 N])+j*rand([1 N]); fftdata = fft(data); myfftdata=
ELEC503_fft_rad4(data);

for n=0:(N-1)
    x (n+1)=data (n+1)*exp(-j*pi*n^2/N);
end
for n=0:(2*N-1)
    h(n+1)=exp(j*pi*n^2/N);
end
y=conv (x,h);
for n=N:(2*N-1)
    X (n-N+1) = exp(-j*pi* (n-N) ^2/N)* exp(-j*pi*N)*y(n+1);
end
s=0;
disp('data= ');disp(data);
disp('FFT=');disp(myfftdata);
disp('Bluestein=');disp(X) ;
comperror=0;

for i=1:N
    if (floor(100*myfftdata(i)) ~= floor(100*fftdata(i)))
        disp(sprintf('error computation at %d : %f+j(%f) !=%f+j(%f)',i,real(
          ↪ myfftdata(i)),imag(myfftdata(i)),real(fftdata(i)),imag(fftdata(i)))
          ↪ ) ;comperror=1;
    end
end
if comperror == 0
    disp(sprintf('Passed for N=%i',N));
end
```

Appendix B. Matlab Code For The Bluestein Algorithm Using Three FFTs To Achieve Convolution

```

%Theory and application of digital signal processing Lawrence R,
%Rabiner-Bernard Gold pp392-396
%A linear Filetering approach to the computation of discrete fourier
```

```
%transform
clear variables;
N=64;
data = rand([1 N])+j*rand([1 N]);
alpha=log(2*N-1)/log(4);
L=4^ceil(alpha);
j=complex(0,1);
disp('N=');disp(N) ; disp('L=') ; disp(L);
fftdata = fft(data);
myfftdata=ELEC503_fft_rad4(data);
for n=0:(N-1)
    y (n+1)=data (n+1)*exp(-j*pi*n^2/N);
end
for n=N:(L-1)
    y (n+1)=0.0;
end
for n=0:(L-1)
    v (n+1) = 0.0;
end
for n=0:(N-1)
    v(n+1)=exp(j*pi*n^2/N);
end
for n=L-N+1:(L-1)
    v(n+1)=exp(j*pi*(L-n)^2/N);
    end
FFT_of_y= ELEC503_fft_rad4(y);FFT_of_v= ELEC503_fft_rad4(v);
for - n=\overline{0}:(L-1)
FFT_of_G (n+1)=FFT_of_y(n+1)*FFT_of_v(n+1);
end
G=1/L*ELEC503 fft rad4 inverse(FFT of G);
for n=0:(N-1)
    X (n+1)=G(n+1)*exp(-j*pi*n^2/N);

end
disp('Bluestein=');disp(X);
disp('fftdata=');disp(fftdata);
disp('myfftdata=');disp(myfftdata);
comperror=0;
for i=1:N
    if (floor(100*myfftdata(i)) ~= floor(100*fftdata(i)))
            disp(sprintf('error␣computation␣at␣%d␣:␣%f+j(%f)␣!=
%f+j(%f)',i,real(myfftdata(i)),imag(myfftdata(i)),real(fftdata(i)),imag(fft
data(i))));comperror=1;
    end
end
if comperror == 0
    disp(sprintf('Passed␣for␣N=%i',N));
end
```

References

1. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* **1965**, *19*, 297–301.
2. Bluestein, I.L. A linear filtering approach to the computation of discrete Fourier transform. *IEEE Transactions on Audio and Electroacoustics* **1970**, *18*, 451–455.
3. Winograd, S. On computing the Discrete Fourier Transform. *Mathematics of Computation* **1978**, *32*, 175–199.
4. MATLAB. MATLAB. https://www.mathworks.com/products/matlab.html, n.d. [Online].
5. Frigo, M.; Johnson, S. The design and implementation of fftw3. *Proceedings of the Ieee* **2005**, *93*, 216–231. https://doi.org/10.1109/jproc.2004.840301.
6. Rabiner, L.R.; Gold, B. *Theory and Application of Digital Signal Processing*; Prentice Hall, 1975.

7. Jayaram, K.; Arun, C. Survey Report for Radix 2, Radix 4, Radix 8 FFT Algorithms. *International Journal of Innovative Research in Science, Engineering and Technology* **2015**, *4*, 5149–5154. https://doi.org/10.15680/IJIRSET.2015.0407015.
8. Sinchana, G.S.; Padaki, S.; Ravi, V.; Varshini, V.S.; Raghavendra, C.G. Software Implementation of FFT Algorithms and Analysis of their Computational Complexity. In Proceedings of the 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Dec 2018, pp. 486–490. https://doi.org/10.1109/ICEECCOT43722.2018.9001665.
9. Moon, S.C.; Park, I.C. Area-efficient memory-based architecture for FFT processing. *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.* **2003**, *5*, V–V.
10. Rennels, D.A., Fault-tolerant computing. In *Encyclopedia of Computer Science*; John Wiley and Sons Ltd., 2003; p. 698–702. https://doi.org/10.5555/1074100.1074394.
11. Ahamed, S.F.; Laveti, G.; Goswami, R.; Rao, G.S. Fast Acquisition of GPS Signal Using Radix-2 and Radix-4 FFT Algorithms. In Proceedings of the 2016 IEEE 6th International Conference on Advanced Computing (IACC), Feb 2016, pp. 674–678. https://doi.org/10.1109/IACC.2016.130.
12. Jayakumar, D.; Logashanmugam, E. Design of Combined Radix-2, Radix- 4 and Radix-8 based Single Path Delay Feedback (SDF) FFT. *Indian journal of science and technology* **2016**, *9*.
13. Singh, P.K. Performance Analysis of Associate Radix-2, Radix- 4 and Radix-8 based FFT using Folding Technique. *International Journal for Research in Applied Science and Engineering Technology* **2021**.
14. Lee, J.; Lee, H.; in Cho, S.; Choi, S. A high-speed, low-complexity radix-2/sup 4/ FFT processor for MB-OFDM UWB systems. *2006 IEEE International Symposium on Circuits and Systems* **2006**, pp. 4 pp.–.
15. Johnson, S.G.; Frigo, M. A Modified Split-Radix FFT With Fewer Arithmetic Operations. *IEEE Transactions on Signal Processing* **2007**, *55*, 111–119. https://doi.org/10.1109/TSP.2006.882087.
16. Rader, C.M. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE* **1968**, *56*, 1107–1108.
17. Moon, Y.J.; Kim, Y.I. A mixed-radix 4-2 butterfly with simple bit reversing for ordering the output sequences. In Proceedings of the 2006 8th International Conference Advanced Communication Technology, Feb 2006, Vol. 3, pp. 4 pp.–1774. https://doi.org/10.1109/ICACT.2006.206332.
18. Sorensen, H.V.; Heideman, M.T.; Burrus, C.S. On computing the split-radix FFT. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **1986**, *34*, 152–156.