**Introduction**

Nuclear testing has been a significant concern for global security and the environment for decades. This report provides an overview of underground nuclear testing, its historical background, and its implications. The report also examines the importance of detecting underground nuclear explosions and proposes a detection strategy.

**Background**

**1 Historical Overview**

Underground nuclear testing refers to the detonation of nuclear devices below the Earth's surface. This type of testing was prevalent during the Cold War era when several countries, including the United States, the Soviet Union, and other nuclear-armed nations, conducted numerous nuclear tests in various geological settings to develop and improve their nuclear weapons capabilities.

**2 Process of Underground Nuclear Testing**

Underground nuclear testing involves drilling a borehole into the ground and placing a nuclear device, typically contained in a steel or concrete casing, at the bottom of the hole. The device is then detonated, creating a release of energy in the form of an explosion, which results in the vaporization of surrounding rock and formation of a cavity. The energy released in the explosion can cause seismic waves to propagate through the Earth, which can be detected and measured by seismometers located at various distances from the test site.

**The Problem’s Crucial Importance (purpose)**

**1 Global Security**

The detection of underground nuclear explosions is a critical issue due to significant implications for global security. Underground nuclear explosions may be associated with clandestine nuclear weapons testing by countries that are not compliant with international agreements such as the Comprehensive Nuclear-Test-Ban Treaty (CTBT), which aims to ban all nuclear explosions. Detecting such explosions can provide crucial information about potential violations of nuclear disarmament treaties and help prevent the proliferation of nuclear weapons.

**2 Environmental and Human Health**

Underground nuclear explosions can have significant environmental and human health impacts. These include the release of radioactive materials into the atmosphere and ground, contamination of soil and water, and potential harm to nearby populations. Detecting and monitoring underground nuclear explosions can provide early warning of such events, enabling timely response and mitigation measures to protect the environment and human health.

**3 Seismic Hazard Assessment**

Seismic signals generated by underground nuclear explosions can provide valuable data for seismic hazard assessment, which is important for understanding earthquake activity and improving earthquake prediction and mitigation strategies. Accurate detection and characterization of underground nuclear explosions can help differentiate them from natural earthquakes, ensuring that seismic hazard assessments are based on reliable data.

**Detection Strategies (scope)**

**1 Seismic Monitoring**

Seismic monitoring is the most effective and reliable method for detecting underground nuclear explosions. Seismometers located around the world can detect and measure the seismic waves generated by underground nuclear explosions. By analyzing the characteristics of the seismic waves, scientists can determine whether an explosion has occurred and estimate its yield.

**2 Radionuclide Monitoring**

Radionuclide monitoring involves the detection and measurement of radioactive particles that are released into the atmosphere during an underground nuclear explosion. These particles can be detected by monitoring stations located around the world, providing an additional means of detecting underground nuclear explosions.

**3 Infrasound Monitoring**

Infrasound monitoring involves the detection and measurement of low-frequency sound waves generated by underground nuclear explosions. Infrasound waves can travel long distances through the atmosphere, providing another means of detecting underground nuclear explosions.

**Takeway**

In conclusion, the detection of underground nuclear explosions is crucial for global security, environmental and human health, and seismic hazard assessment. Seismic monitoring is the most effective and reliable method for detecting underground nuclear explosions, while radionuclide and infrasound monitoring provide additional means of detection. The development and implementation of robust detection strategies are critical to preventing the proliferation of nuclear weapons and protecting the environment

**Proposed Solution**

To detect underground nuclear activity, we will employ the use of the Fourier Transform. The Fourier Transform is a mathematical tool that allows signals to be decomposed into their constituent frequencies. In the case of seismic data, this means that the vibrations of the Earth caused by underground nuclear explosions can be analyzed to determine their frequencies.

By analyzing the frequency content of seismic signals, it is possible to differentiate between natural earthquakes and man-made explosions, such as those caused by underground nuclear testing. This is because the frequency content of seismic signals generated by nuclear explosions is different from that of natural earthquakes.

To use the Fourier Transform to detect underground nuclear activity, seismic data must first be collected using a network of seismometers located around the world. The seismic data is then processed using the Fourier Transform to identify the frequency content of the signals.

The next step is to compare the frequency content of the seismic signals with known characteristics of underground nuclear explosions. This can be done by analyzing data from previous underground nuclear tests and using this information to create a database of known nuclear explosion signatures.

By comparing the frequency content of seismic signals with known nuclear explosion signatures, it is possible to determine whether an underground nuclear explosion has taken place. This information can then be used to detect potential violations of international nuclear disarmament treaties and to protect the environment and human health from the harmful effects of nuclear testing.

In addition to detecting underground nuclear activity, the Fourier Transform can also be used to improve seismic hazard assessments. By accurately identifying seismic signals generated by underground nuclear explosions, it is possible to differentiate them from natural earthquakes, ensuring that seismic hazard assessments are based on reliable data.

Overall, the use of the Fourier Transform to detect underground nuclear activity has the potential to be a powerful tool in the fight against nuclear proliferation and the protection of the environment and human health.

**Fourier Transform: A mathematical overview**

The Fourier Transform is a mathematical technique that transforms a function of time, x(t), to a function of frequency, X(ω).

The Fourier Transform of a function can be derived as a special case of the Fourier Series when the period, T→∞.

A picture containing text, watch

Description automatically generated

**The Discrete Fourier Transform**

In the real world, continuity is impractical as it is impossible to measure a range of values continuously over a dataset. Thus, it is much more practical and realistic to take finite values from a finite dataset. In this case, we modify the original definition of the Fourier Transform, from integrating over a continuous domain to summing over a discrete one.

Doing this results in what is known as the Discrete Fourier Transform (DFT). It is equivalent to the continuous Fourier Transform, but for singals known only at N instants separated by sample times T (i.e. a finite sequence of data).

Let *f(t)* be the continuos signal which is the source of the data. Let *N* samples be denoted *f[0], f[1], f[2], … , f[k], … , f[N-1].*

The Fourier Transform of the original signal, *f(t)*, would be

Diagram

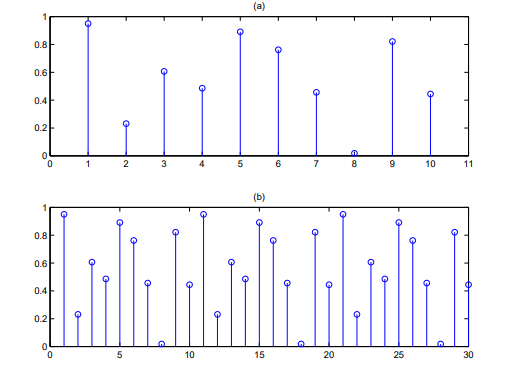
Description automatically generated with low confidence

We could regard each sample *f[k]* as an *impulse* having area *f[k].* Then, since the integrand exists only at the sample points:

Text, letter

Description automatically generated

In principle, this could be evaluated for any datapoint, but with only N data points to start with, only *N* final outputs will be significant. The DFT also assumes periodicity of the data samples i.e. data is repeated over a constant sampling rate. This is illustrated in the figure below:



Since the operation treats the data as if it were periodic, we evaluate the DFT equation only for the fundamental frequency:

Text

Description automatically generated with medium confidence

**DFT—An example**

Let the continuous signal be:

Chart, line chart

Description automatically generated

Let us sample *f(t)* at 4 Hz from *t=0 to t=3/4* . The values of the discrete samples are given by:



Text

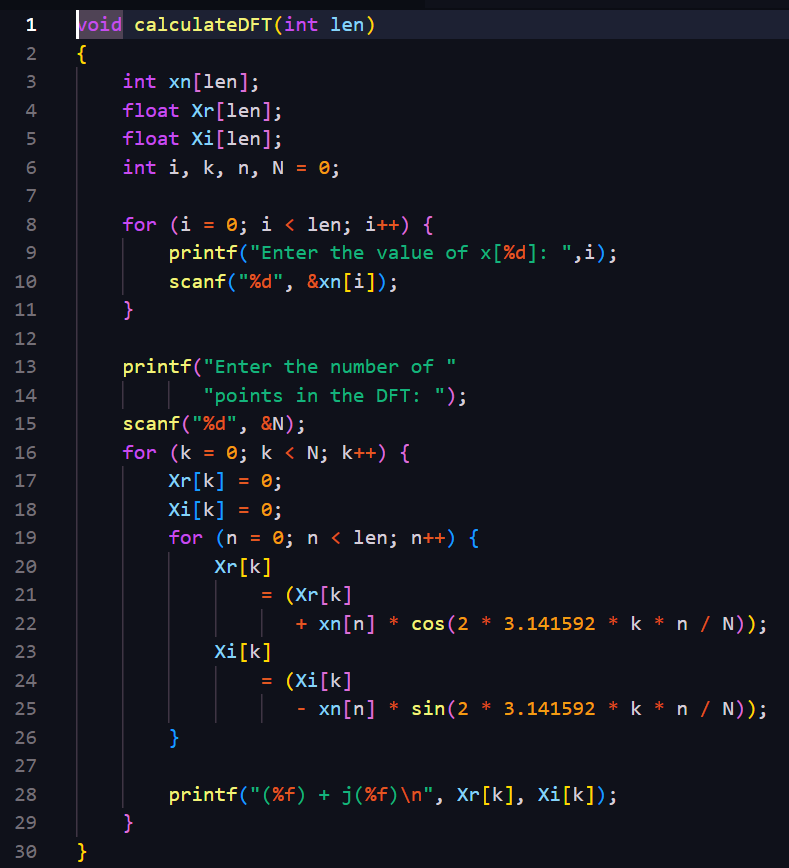
Description automatically generated

The magnitude of the DFT coefficients is shown below:

Chart

Description automatically generated

**Analysis of the Algorithm**



T(n) = O (n^2)

**The Problem with using a DFT Computation:**

While DFT is a widely used mathematical tool for analysis, there are several limitations and challenges that occur with its use in practical situations:

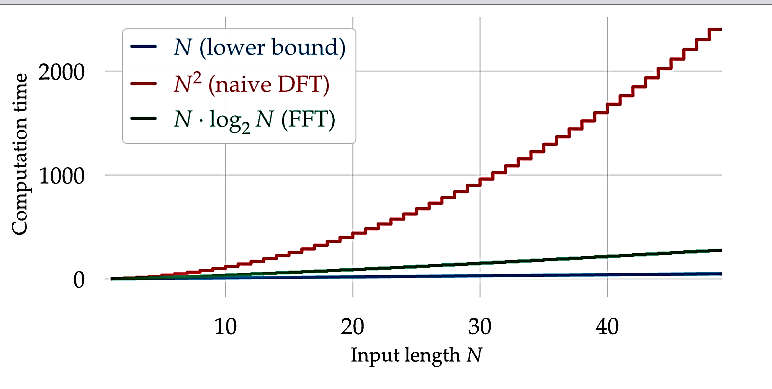
1. Computational Complexity: For large input sizes, a significant number of resources are required by this algorithm. The computation time complexity of the DFT is O(N^2), where N is the size of the input signal.
2. Memory Requirements: DFT requires large complex matrix multiplications and additions which demand storage of their intermediate results and complex-valued coefficients. This leads to high memory requirements, proving a limitation in resource-constrained environments or when dealing with large datasets.
3. Leakage and Aliasing Effects: The DFT assumes that the input signal is periodic, and this can result in leakage and aliasing effects. Leakage refers to the spreading of signal energy across multiple frequency bins, leading to reduced frequency resolution. Aliasing refers to the distortion of signal frequency components due to undersampling or inadequate representation of high-frequency components. These effects can affect the accuracy of the DFT analysis and can be challenging to mitigate.
4. Real-time Processing Challenges: DFT computations can be time-consuming, especially for large input signals, which can pose challenges for real-time processing applications that require low-latency processing. This can limit the use of DFT in certain time-critical applications, such as in the real-world problem chosen by us.

**The Solution:**

**The Fast Fourier Transform (FFT):**

The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT) and its inverse. It does so by exploiting the symmetry and periodic properties of the DFT to reduce the number of computations required to solve a single problem.

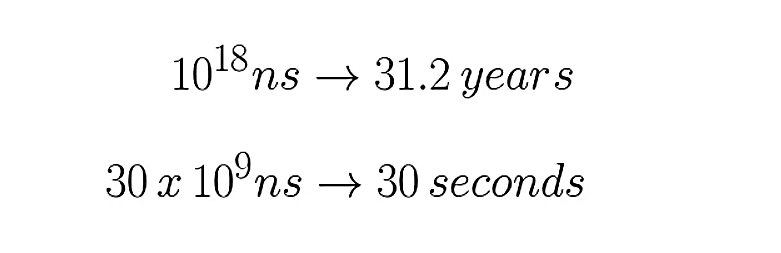
In contrast to the DFT’s O(N^2) time complexity, the FFT boasts a complexity of O(n log n), which is much faster for larger input sizes.



To demonstrate the marginal difference in computation, the table below displays the relevant information:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***N = 10³*** | ***N = 10⁶*** | ***N = 10⁹*** |
| **N²** | 10⁶ | 10^12 | 10^18 |
| **NlogN** | 10⁴ | 20 \* 10^6 | 30 \* 10^9 |

For an input size of 100 million, the FFT finishes its computation of what could potentially take years to under just a minute.



The FFT algorithm is typically implemented using a divide-and-conquer approach, where the input signal is recursively divided into smaller subproblems, and the DFT is computed for each subproblem separately. The computed DFTs are then combined to obtain the final DFT of the original input signal. This divide-and-conquer approach allows for efficient computation of the DFT by reusing intermediate results and avoiding redundant computations.

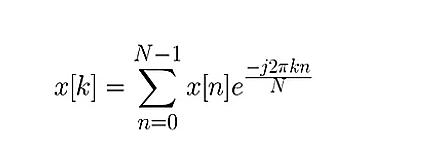
There are several popular FFT algorithms, including the Cooley-Tukey algorithm, the Radix-2 algorithm, and the Radix-4 algorithm, among others. These algorithms differ in their implementation details, but they all share the same basic principle of recursively dividing and conquering the input signal to compute the DFT efficiently.

The FFT has become a fundamental tool in many areas of science and engineering due to its computational efficiency and wide range of applications. It has revolutionized fields such as digital signal processing, telecommunications, and data compression, enabling advanced technologies and applications that rely on frequency domain analysis of signals and data.

**From DFT To FFT:**

Before we proceed to analyze the FFT algorithm, it is crucial to gain insight on how this miraculous algorithm actually came into being.

This is the formula for the Discrete Fourier Transform:



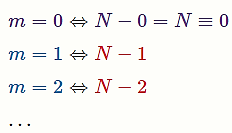
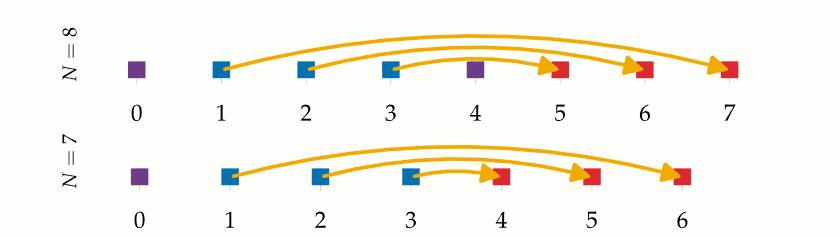
Fourier Transform makes use of two symmetric properties to reduce what are effectively redundant calculations:

1. Complex Conjugate Symmetry:

If x[n] is a real-valued signal with N samples. Then, the DFT series X[0],X[1],……, X[N-1] has conjugate symmetry:



So, if m = 0, 1, 2, …., N-1, we see that:

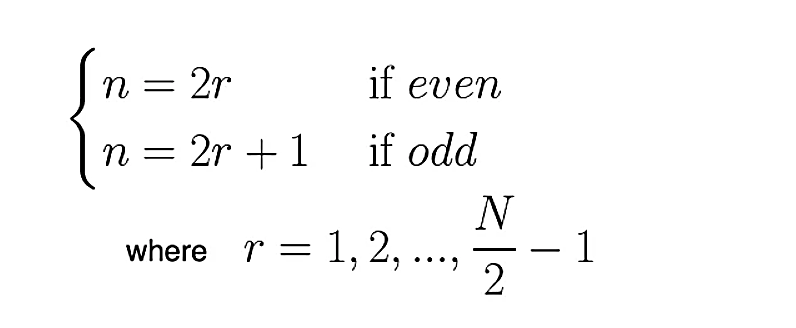
 

1. Periodicity:

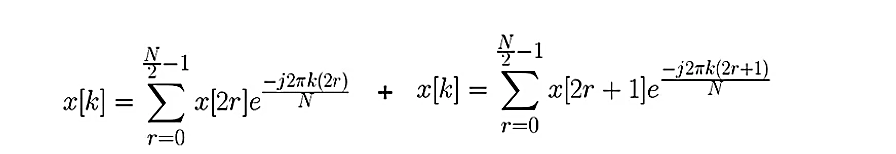
If x[n] and x[k] are a N point DFT pair, then:

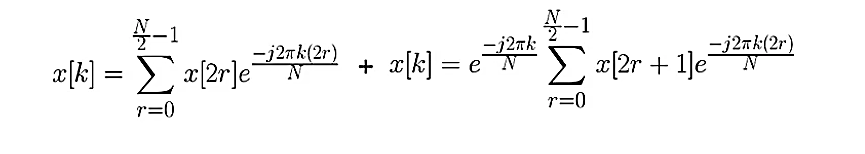


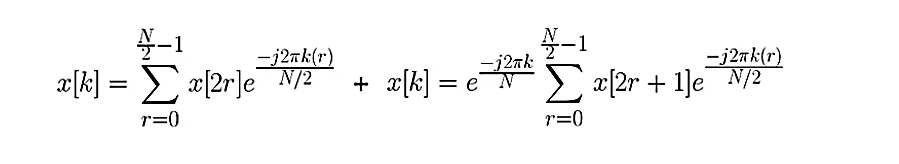
Following the properties above, we divide our N signals into even and odd subsequences of size N/2 by assuming the conditions below:



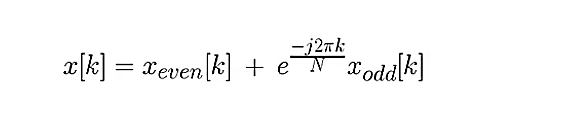
Substituting this into our original discrete function, we can split it into two summations of N/2 each. The advantage of this approach lies in the fact that the even and odd indexed sub-sequences can be computed concurrently.



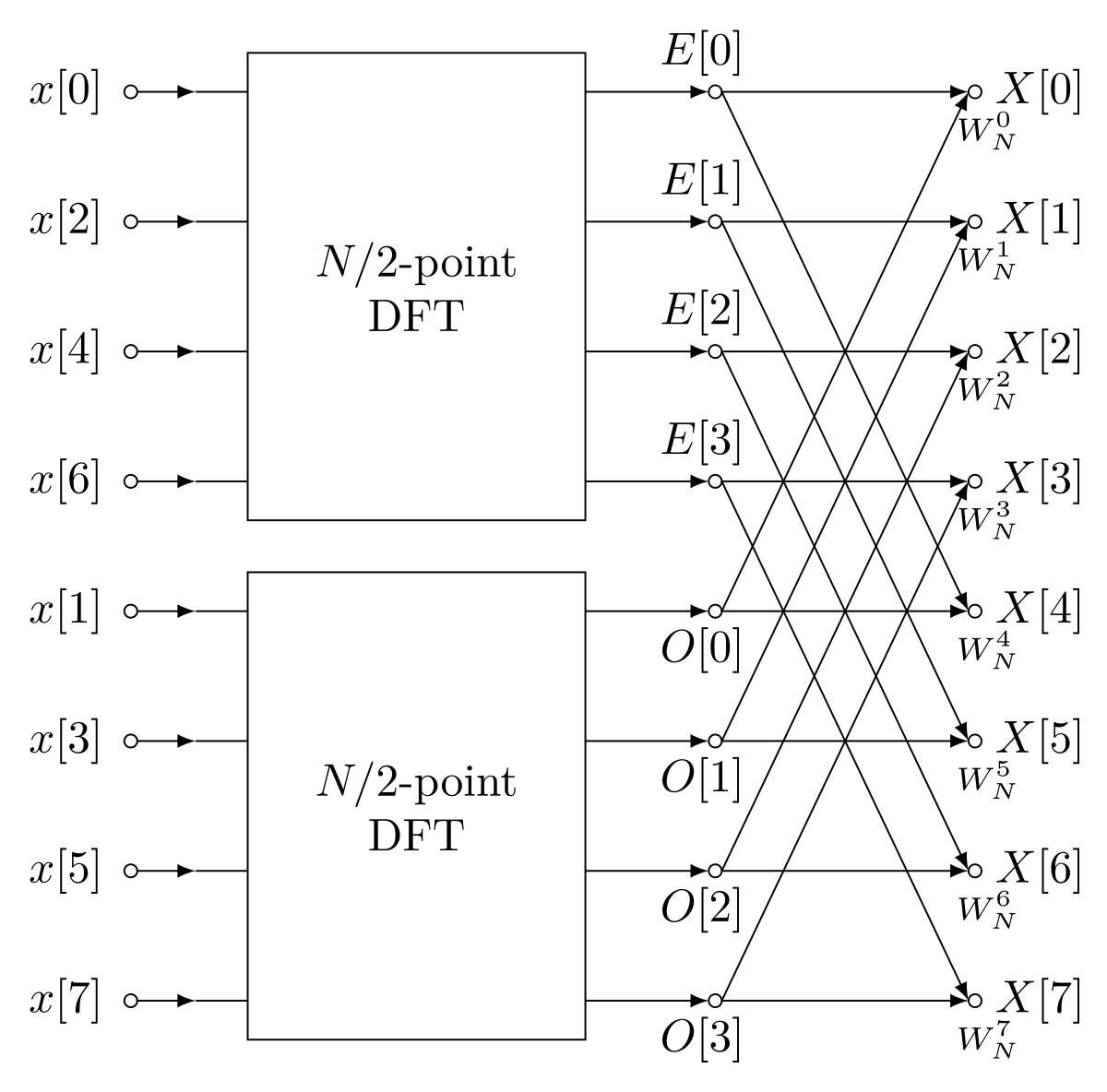




With simplification, we end up with our Fast Fourier Transform:



Suppose our dataset has N=8, we can visualize the working above through use of a butterfly diagram. The Discrete Fourier Transform for the even and odd terms are calculated simultaneously and those terms are then added together and multiplied to a ‘twiddle factor’ accordingly to find each frequency bin.



But there's no reason to stop there: as long as our smaller Fourier transforms have an even-valued split, we can reapply this divide-and-conquer approach, halving the computational cost each time, until our arrays are small enough that the strategy is no longer beneficial and there are N 1-point DFTs

With this recursion, we can reduce the complexity to a very feasible operation of O(nlogn). This method of conversion to the frequency domain has its very own algorithm known as the Cooley-Tukey algorithm.

**The Cooley-Tukey algorithm.**

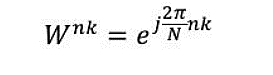
J. W. Cooley and John Tukey published a paper in 1965 describing what is now known as the most common form of the Fast Fourier Transform. This algorithm not only increases the efficiency by orders of magnitude but also scales much better. It’s time complexity of O (N log N) mean that a 256-byte sample takes only about 2000 calculations while a 1024 takes just over 10000

It is particularly suited for signal lengths that are a power of 2, and is also referred to as the Radix-2 algorithm

Here's a high-level explanation of the Cooley-Tukey algorithm for FFT:

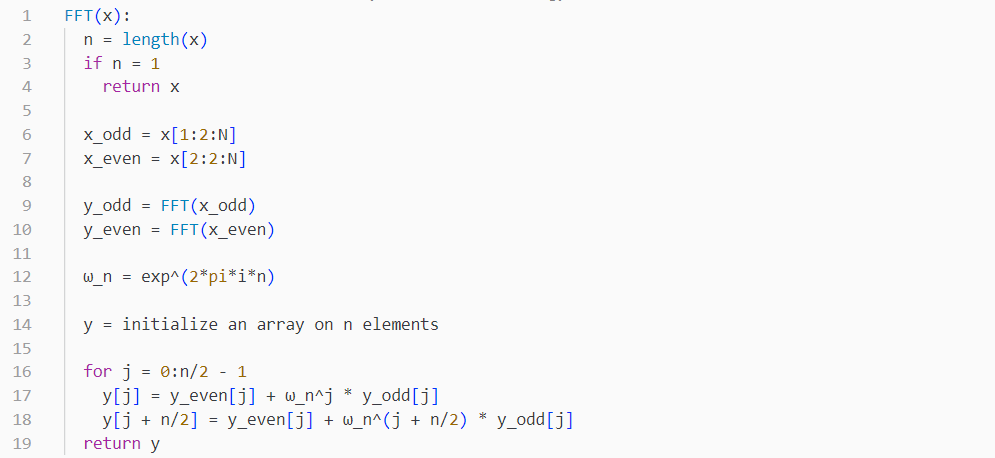
1. **Input**: The Cooley-Tukey algorithm takes as input a sequence of n complex numbers representing the discrete signal in the time domain that we want to transform into the frequency domain.
2. **Splitting**: The input signal is split into two smaller sequences by separating the even-indexed samples and odd-indexed samples. This results in two sequences, often referred to as "even" and "odd" halves of the original signal.
3. **Recursion**: The Cooley-Tukey algorithm is applied recursively to the even and odd halves obtained in the previous step. This means that the FFT is applied independently to the even half and the odd half of the input signal.
4. **Combining**: Once the FFT is applied recursively to the even and odd halves, the results are combined to obtain the final FFT of the original input signal. The combining step involves multiplying the odd half by a set of complex numbers known as "twiddle factors" and adding or subtracting them from the even half, depending on the index. The twiddle factors are carefully chosen complex numbers that depend on the signal length and the current index, and they play a key role in efficiently combining the results of the even and odd halves.
5. **Repeat**: Steps 2-4 are repeated recursively until the base case of the FFT is reached, which is when the input signal length is reduced to 1. At this point, the FFT is trivial to compute as it simply involves returning the input signal itself.
6. **Output**: The final output of the Cooley-Tukey algorithm is the FFT of the original input signal, represented as a sequence of complex numbers in the frequency domain.

Mathematically, a twiddle factor is represented as a complex number of the form:

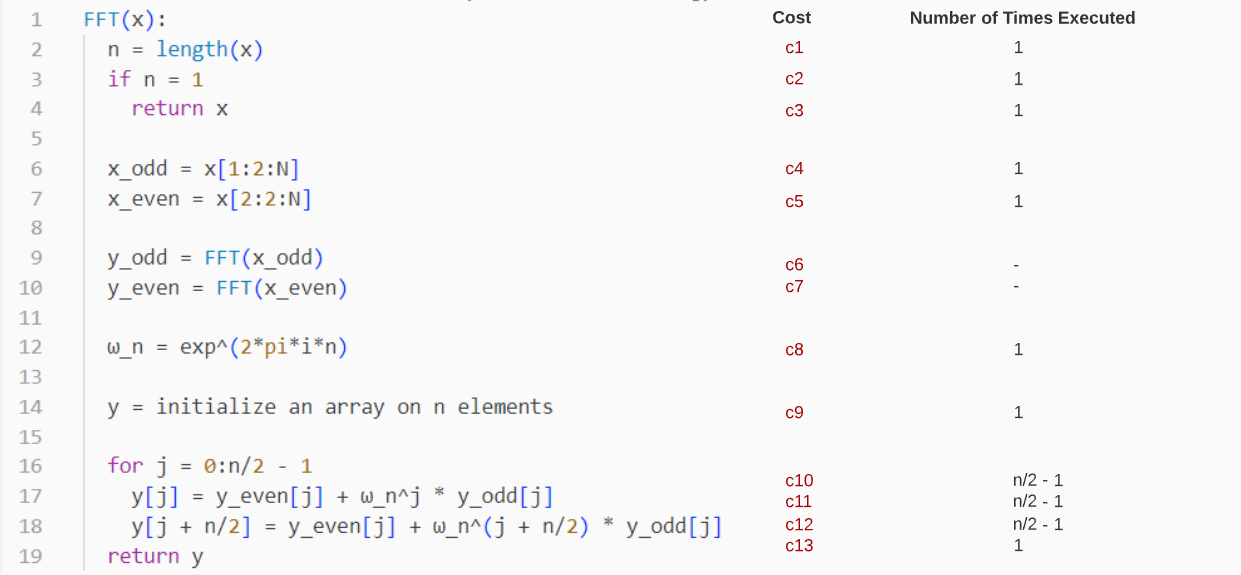


**Design of the Algorithm:**

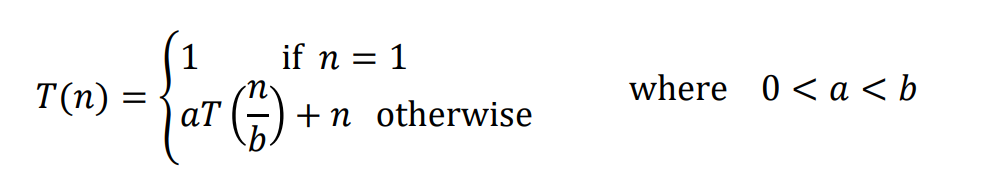
The pseudocode for this algorithm is shown as follows:



**Analysis of the Algorithm**



To verify the time complexity of this code, we first notice that this algorithm is a recursive algorithm. Hence, the following format will apply:



**a**: is the number of times the function calls itself

**b**: is the factor by which the input size is reduced.

**f(n):** is the complexity of the non-recursive part.

To find the non-recursive part:

T(n) = c1\*1 + c2\*1 + c3\*1 + c4\*1 + c5\*1 + c8\*1 +c9\*1 + c10\*() + c11\*() + c12\*() + c13\*1

= (c1 + c2 + c3 + c4 + c5 + c8 + c9 + c13 – c10 – c11 – c12) + ()\***n**

T(n) = O (n)

To find the recursive part:

We make use of the recursion tree concept.

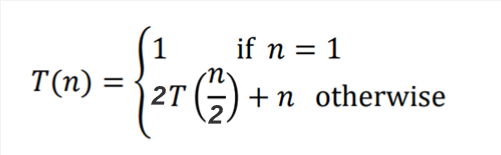
If a recursion tree is made of the Fast Fourier Transform, we know that the function is split into two recursive calls (one each for odd and even).

*Number of branches of the tree = a = 2*

With each recursive call, we pass half the size of our current sample.

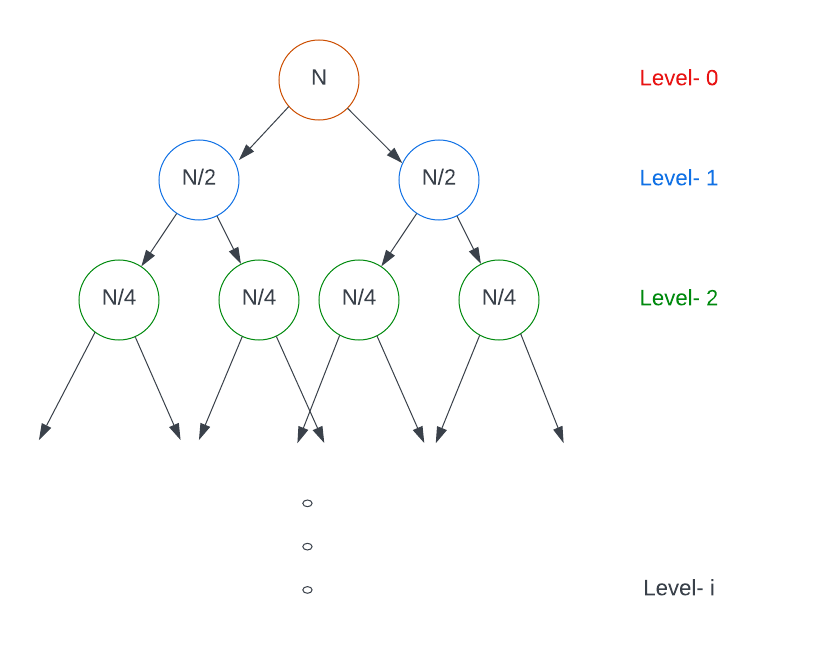
*Factor by which input size is reduced = b = 2*

Combining both our results, we get the recurrence relation as:



**Solving the Recurrence Relation**

We now draw the recursion tree of the given recurrence relation as follows:



We then determine the cost of each level as:

* Cost of level-0 = n
* Cost of level-1 = n/2 + n/2 = n
* Cost of level-2 = n/4 + n/4 + n/4 + n/4 = n

…and so on.

The total number of levels in the tree are calculated as:

* Size of sub-problem at level-0 = n/2⁰
* Size of sub-problem at level-1 = n/2
* Size of sub-problem at level-2 = n/2²

.

.

.

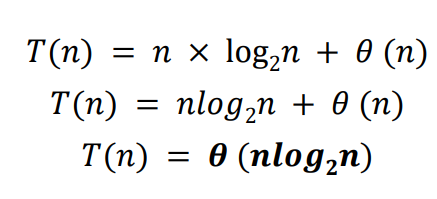
* Size of sub-problem at level-i = n/2ⁱ

Because at the last level the size of the sub-problem becomes 1, then n/2ⁱ = 1

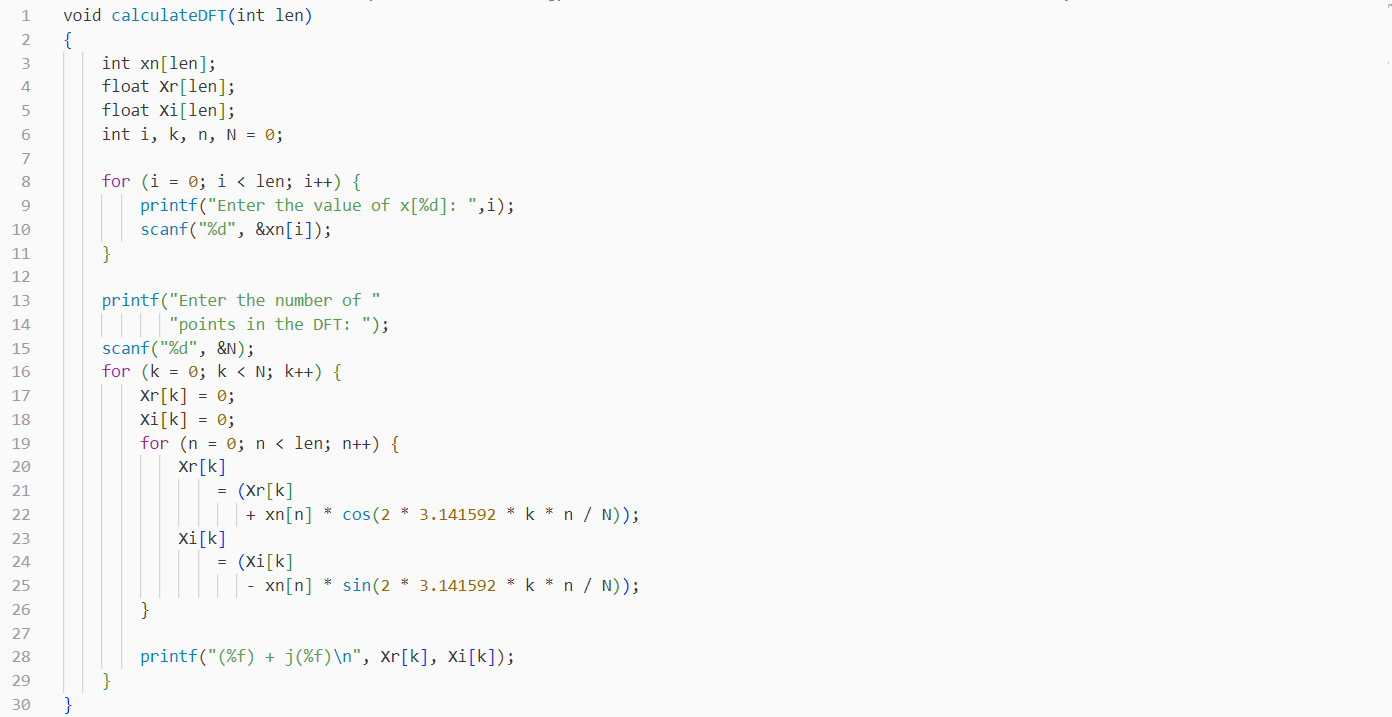
We now get i =

This means level-i or level- has nodes or n nodes

Adding the cost of all levels in the recursion tree and simplifying it, we get our complexity as:

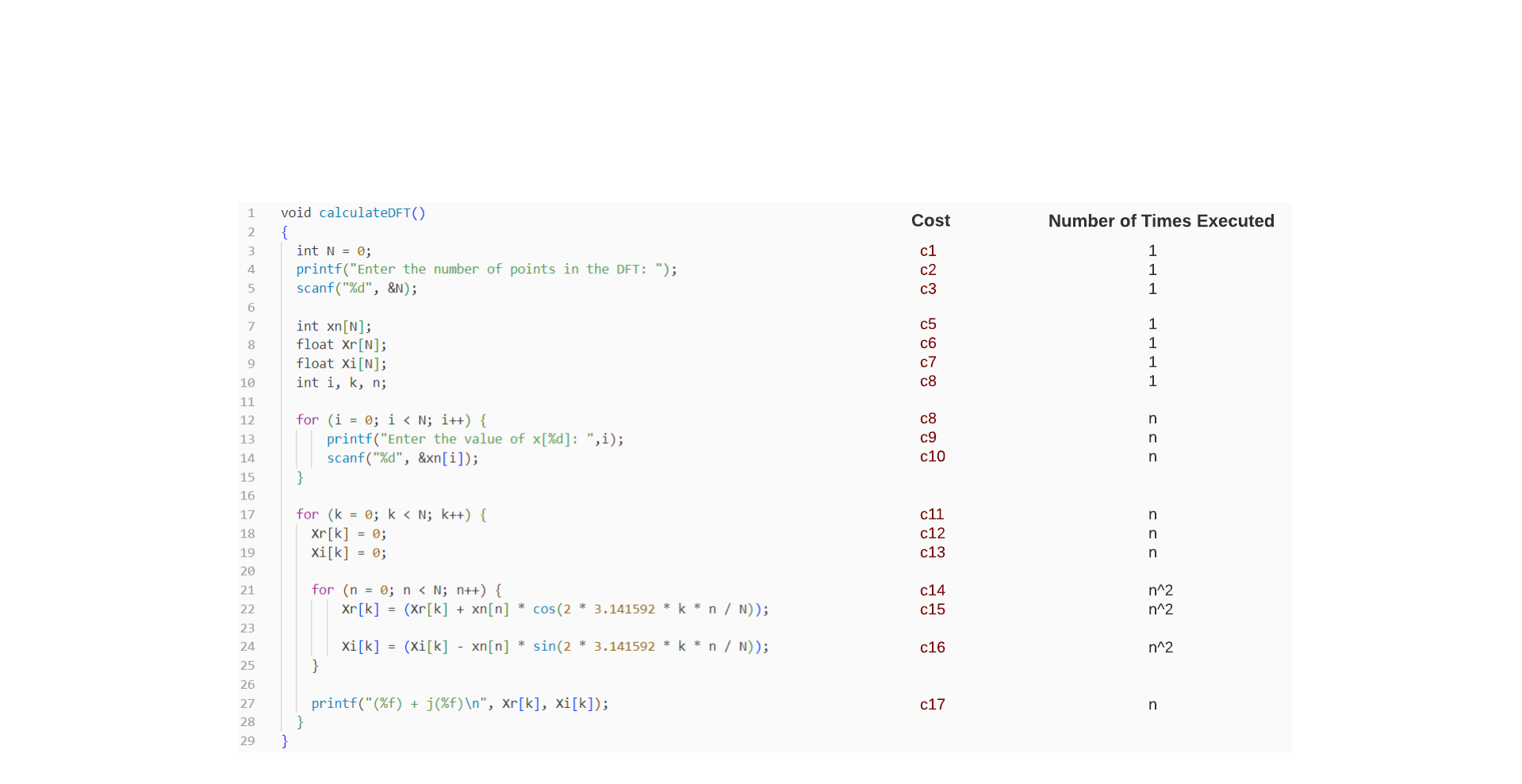


**Design of DFT algorithm:**



**Analysis of the Algorithm**

T(n) = O (n^2)



**The Problem with using a DFT Computation:**

While DFT is a widely used mathematical tool for analysis, there are several limitations and challenges that occur with its use in practical situations:

1. Computational Complexity: For large input sizes, a significant number of resources are required by this algorithm. The computation time complexity of the DFT is O(N^2), where N is the size of the input signal.
2. Memory Requirements: DFT requires large complex matrix multiplications and additions which demand storage of their intermediate results and complex-valued coefficients. This leads to high memory requirements, proving a limitation in resource-constrained environments or when dealing with large datasets.
3. Leakage and Aliasing Effects: The DFT assumes that the input signal is periodic, and this can result in leakage and aliasing effects. Leakage refers to the spreading of signal energy across multiple frequency bins, leading to reduced frequency resolution. Aliasing refers to the distortion of signal frequency components due to undersampling or inadequate representation of high-frequency components. These effects can affect the accuracy of the DFT analysis and can be challenging to mitigate.
4. Real-time Processing Challenges: DFT computations can be time-consuming, especially for large input signals, which can pose challenges for real-time processing applications that require low-latency processing. This can limit the use of DFT in certain time-critical applications, such as in the real-world problem chosen by us.

**The Solution:**

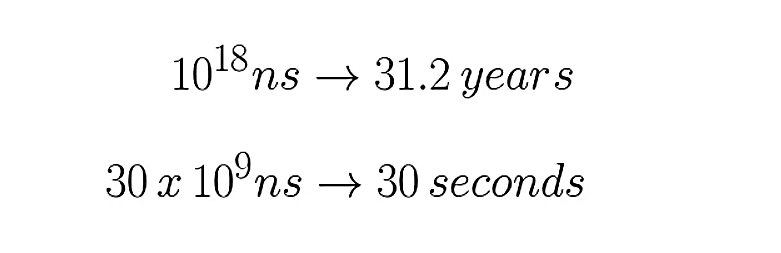
**The Fast Fourier Transform (FFT):**

The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT) and its inverse. It does so by exploiting the symmetry and periodic properties of the DFT to reduce the number of computations required to solve a single problem.

In contrast to the DFT’s O(N^2) time complexity, the FFT boasts a complexity of O(n log n), which is much faster for larger input sizes. To demonstrate the marginal difference in computation, the table below displays the relevant information:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***N = 10³*** | ***N = 10⁶*** | ***N = 10⁹*** |
| **N²** | 10⁶ | 10^12 | 10^18 |
| **NlogN** | 10⁴ | 20 \* 10^6 | 30 \* 10^9 |

For an input size of 100 million, the FFT finishes its computation of what could potentially take years to under just a minute.



The FFT algorithm is typically implemented using a divide-and-conquer approach, where the input signal is recursively divided into smaller subproblems, and the DFT is computed for each subproblem separately. The computed DFTs are then combined to obtain the final DFT of the original input signal. This divide-and-conquer approach allows for efficient computation of the DFT by reusing intermediate results and avoiding redundant computations.

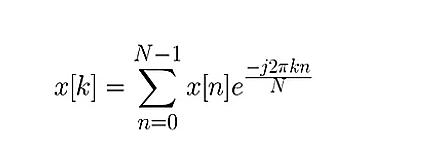
There are several popular FFT algorithms, including the Cooley-Tukey algorithm, the Radix-2 algorithm, and the Radix-4 algorithm, among others. These algorithms differ in their implementation details, but they all share the same basic principle of recursively dividing and conquering the input signal to compute the DFT efficiently.

The FFT has become a fundamental tool in many areas of science and engineering due to its computational efficiency and wide range of applications. It has revolutionized fields such as digital signal processing, telecommunications, and data compression, enabling advanced technologies and applications that rely on frequency domain analysis of signals and data.

**From DFT To FFT:**

Before we proceed to analyze the FFT algorithm, it is crucial to gain insight on how this miraculous algorithm actually came into being.

This is the formula for the Discrete Fourier Transform:



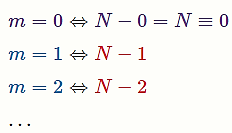
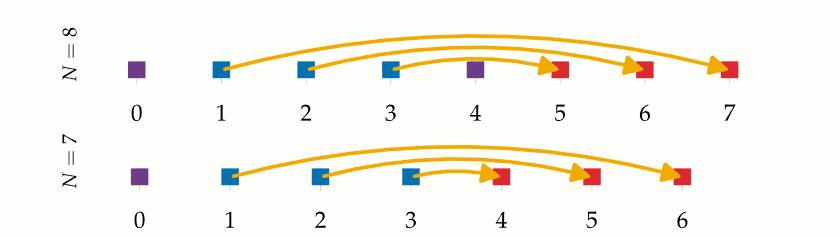
Fourier Transform makes use of two symmetric properties to reduce what are effectively redundant calculations:

1. Complex Conjugate Symmetry:

If x[n] is a real-valued signal with N samples. Then, the DFT series X[0],X[1],……, X[N-1] has conjugate symmetry:



So, if m = 0, 1, 2, …., N-1, we see that:

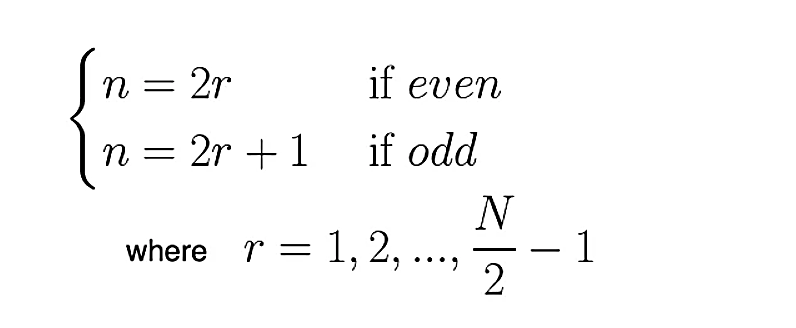
 

1. Periodicity:

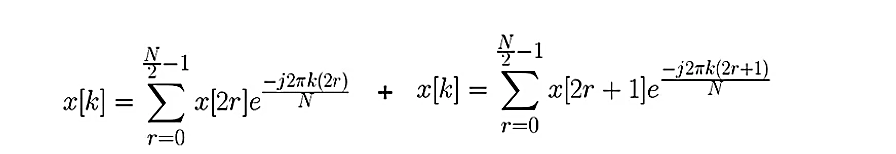
If x[n] and x[k] are a pair, then:

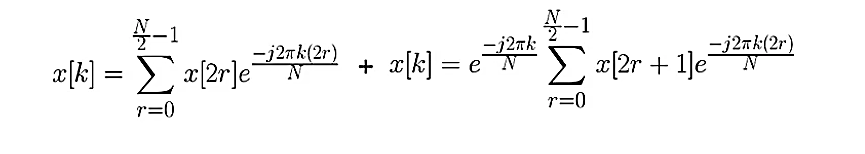


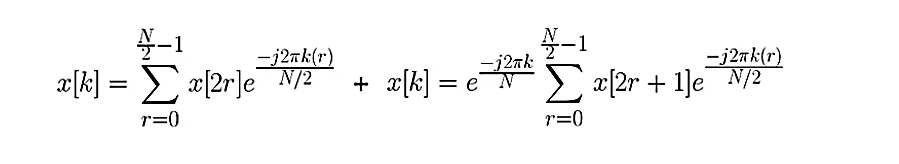
Following the properties above, we divide our N signals into even and odd subsequences of size N/2 by assuming the conditions below:



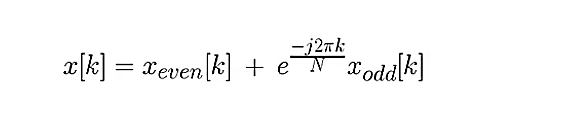
Substituting this into our original discrete function, we can split it into two summations of N/2 each. The advantage of this approach lies in the fact that the even and odd indexed sub-sequences can be computed concurrently.



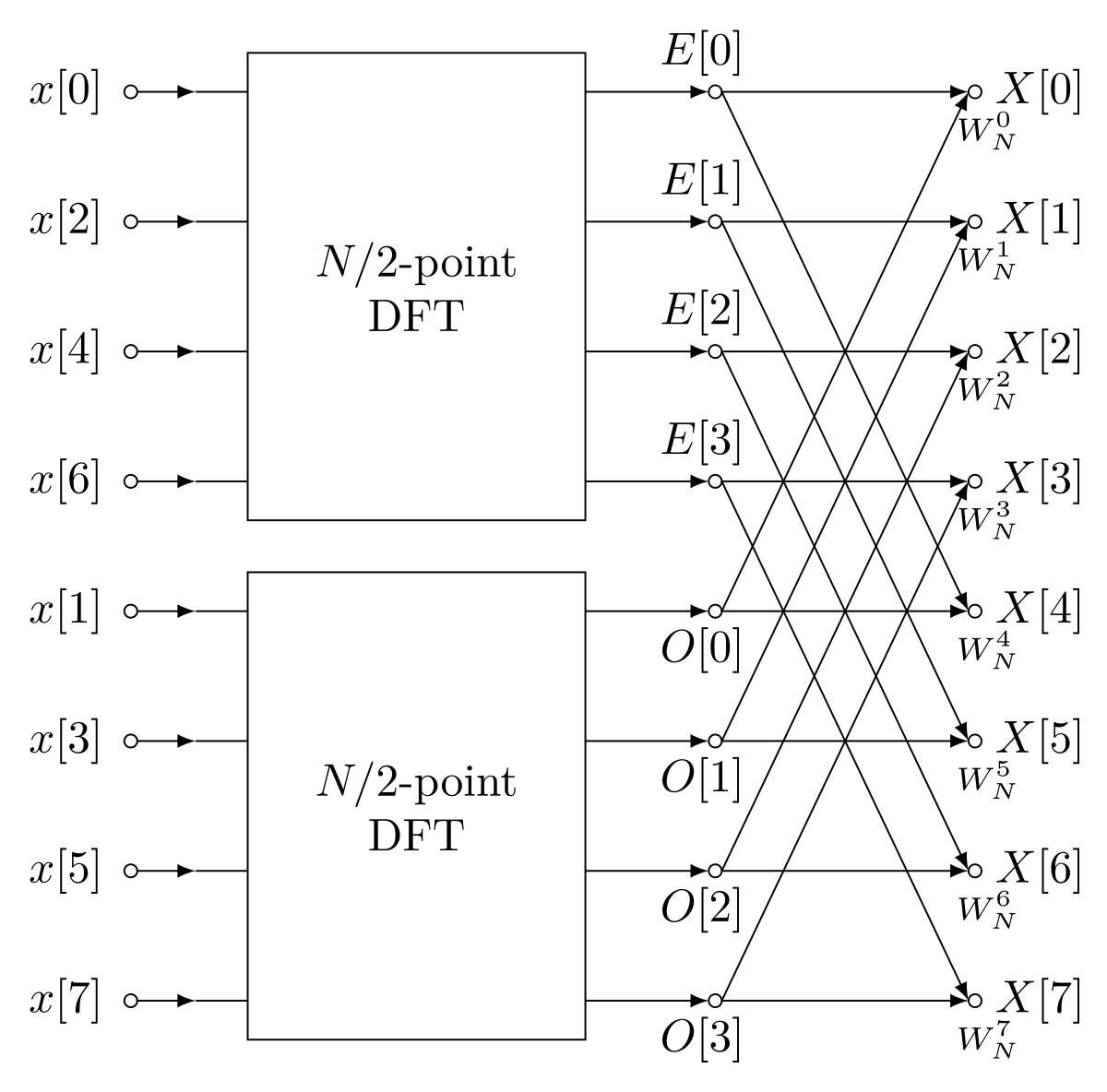




With simplification, we end up with our Fast Fourier Transform:



Suppose our dataset has N=8, we can visualize the working above through use of a butterfly diagram. The Discrete Fourier Transform for the even and odd terms are calculated simultaneously and those terms are then added together and multiplied to a ‘twiddle factor’ accordingly to find each frequency bin.



But there's no reason to stop there: as long as our smaller Fourier transforms have an even-valued split, we can reapply this divide-and-conquer approach, halving the computational cost each time, until our arrays are small enough that the strategy is no longer beneficial and there are N 1-point DFTs

With this recursion, we can reduce the complexity to a very feasible operation of O(nlogn). This method of conversion to the frequency domain has its very own algorithm known as the Cooley-Tukey algorithm.

**The Cooley-Tukey algorithm.**

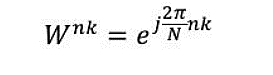
J. W. Cooley and John Tukey published a paper in 1965 describing what is now known as the most common form of the Fast Fourier Transform. This algorithm not only increases the efficiency by orders of magnitude but also scales much better. It’s time complexity of O (N log N) mean that a 256-byte sample takes only about 2000 calculations while a 1024 takes just over 10000

It is particularly suited for signal lengths that are a power of 2, and is also referred to as the Radix-2 algorithm

Here's a high-level explanation of the Cooley-Tukey algorithm for FFT:

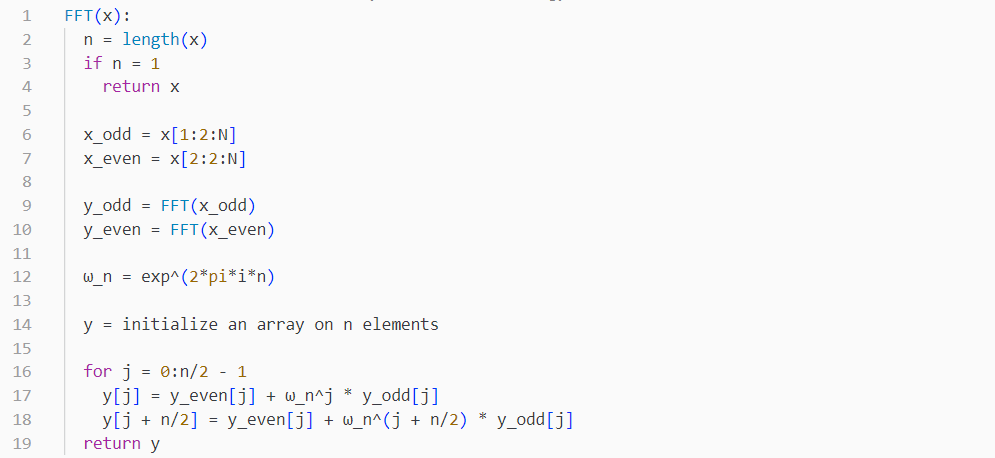
1. **Input**: The Cooley-Tukey algorithm takes as input a sequence of n complex numbers representing the discrete signal in the time domain that we want to transform into the frequency domain.
2. **Splitting**: The input signal is split into two smaller sequences by separating the even-indexed samples and odd-indexed samples. This results in two sequences, often referred to as "even" and "odd" halves of the original signal.
3. **Recursion**: The Cooley-Tukey algorithm is applied recursively to the even and odd halves obtained in the previous step. This means that the FFT is applied independently to the even half and the odd half of the input signal.
4. **Combining**: Once the FFT is applied recursively to the even and odd halves, the results are combined to obtain the final FFT of the original input signal. The combining step involves multiplying the odd half by a set of complex numbers known as "twiddle factors" and adding or subtracting them from the even half, depending on the index. The twiddle factors are carefully chosen complex numbers that depend on the signal length and the current index, and they play a key role in efficiently combining the results of the even and odd halves.
5. **Repeat**: Steps 2-4 are repeated recursively until the base case of the FFT is reached, which is when the input signal length is reduced to 1. At this point, the FFT is trivial to compute as it simply involves returning the input signal itself.
6. **Output**: The final output of the Cooley-Tukey algorithm is the FFT of the original input signal, represented as a sequence of complex numbers in the frequency domain.

Mathematically, a twiddle factor is represented as a complex number of the form:

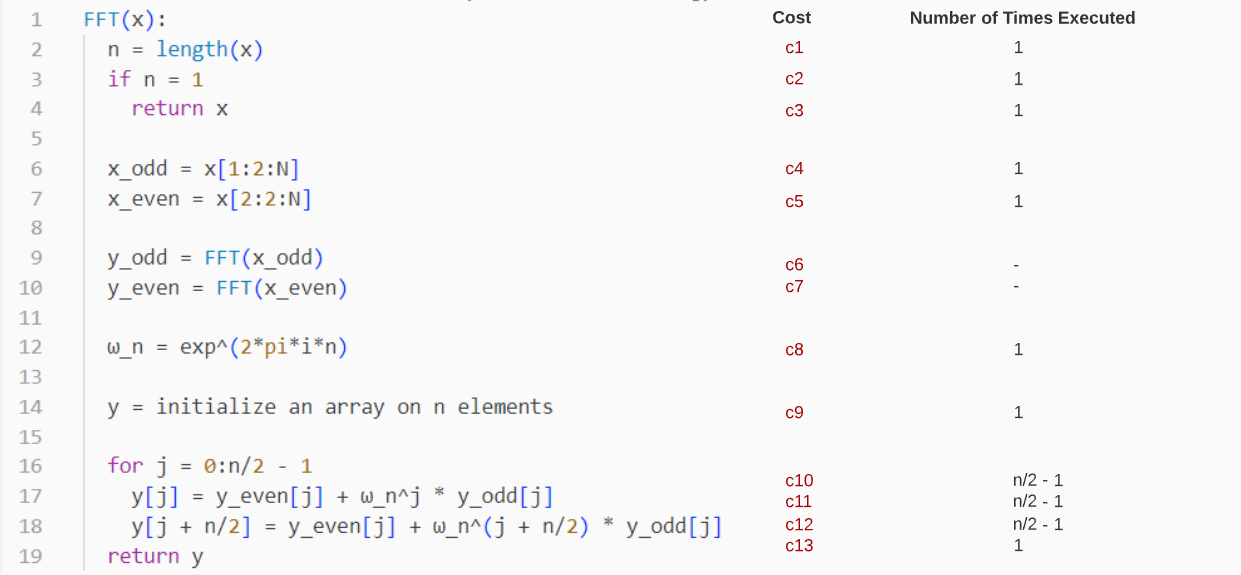


**Design of the Algorithm:**

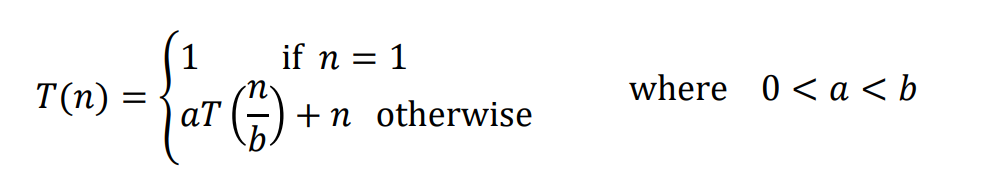
The pseudocode for this algorithm is shown as follows:



**Analysis of the Algorithm**



To verify the time complexity of this code, we first notice that this algorithm is a recursive algorithm. Hence, the following format will apply:



**a**: is the number of times the function calls itself

**b**: is the factor by which the input size is reduced.

**f(n):** is the complexity of the non-recursive part.

To find the non-recursive part:

T(n) = c1\*1 + c2\*1 + c3\*1 + c4\*1 + c5\*1 + c8\*1 +c9\*1 + c10\*() + c11\*() + c12\*() + c13\*1

= (c1 + c2 + c3 + c4 + c5 + c8 + c9 + c13 – c10 – c11 – c12) + ()\***n**

T(n) = O (n)

To find the recursive part:

We make use of the recursion tree concept.

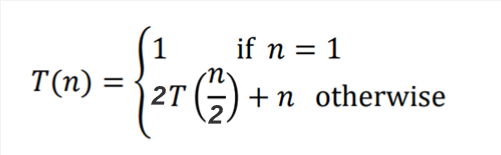
If a recursion tree is made of the Fast Fourier Transform, we know that the function is split into two recursive calls (one each for odd and even).

*Number of branches of the tree = a = 2*

With each recursive call, we pass half the size of our current sample.

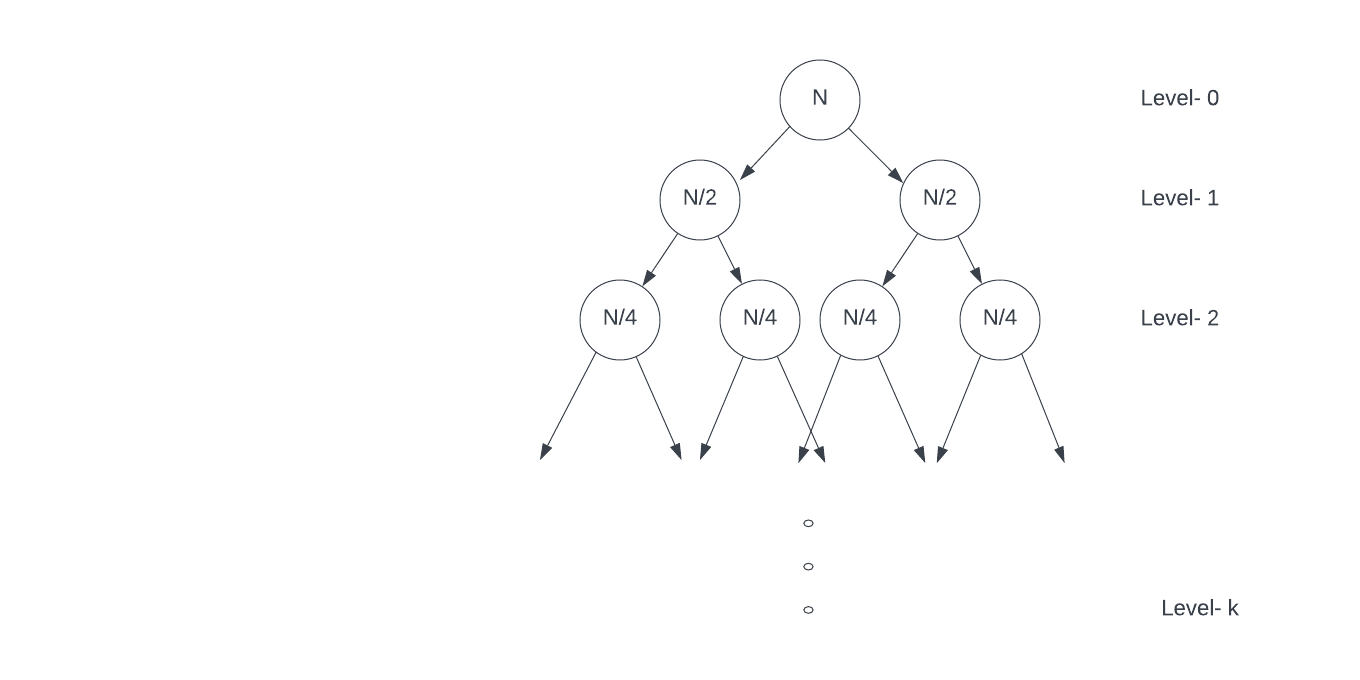
*Factor by which input size is reduced = b = 2*

Combining both our results, we get the recurrence relation as:



**Solving the Recurrence Relation**

We now draw the recursion tree of the given recurrence relation as follows:



We then determine the cost of each level as:

* Cost of level-0 = n
* Cost of level-1 = n/2 + n/2 = n
* Cost of level-2 = n/4 + n/4 + n/4 + n/4 = n

…and so on.

The total number of levels in the tree are calculated as:

* Size of sub-problem at level-0 = n/2⁰
* Size of sub-problem at level-1 = n/2
* Size of sub-problem at level-2 = n/2²

.

.

.

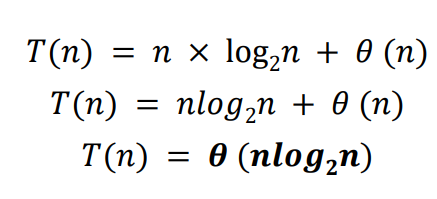
* Size of sub-problem at level-i = n/2ⁱ

Because at the last level the size of the sub-problem becomes 1, then n/2ⁱ = 1

We now get i =

This means level-i or level- has nodes

Adding the cost of all levels in the recursion tree and simplifying it, we get our complexity as:



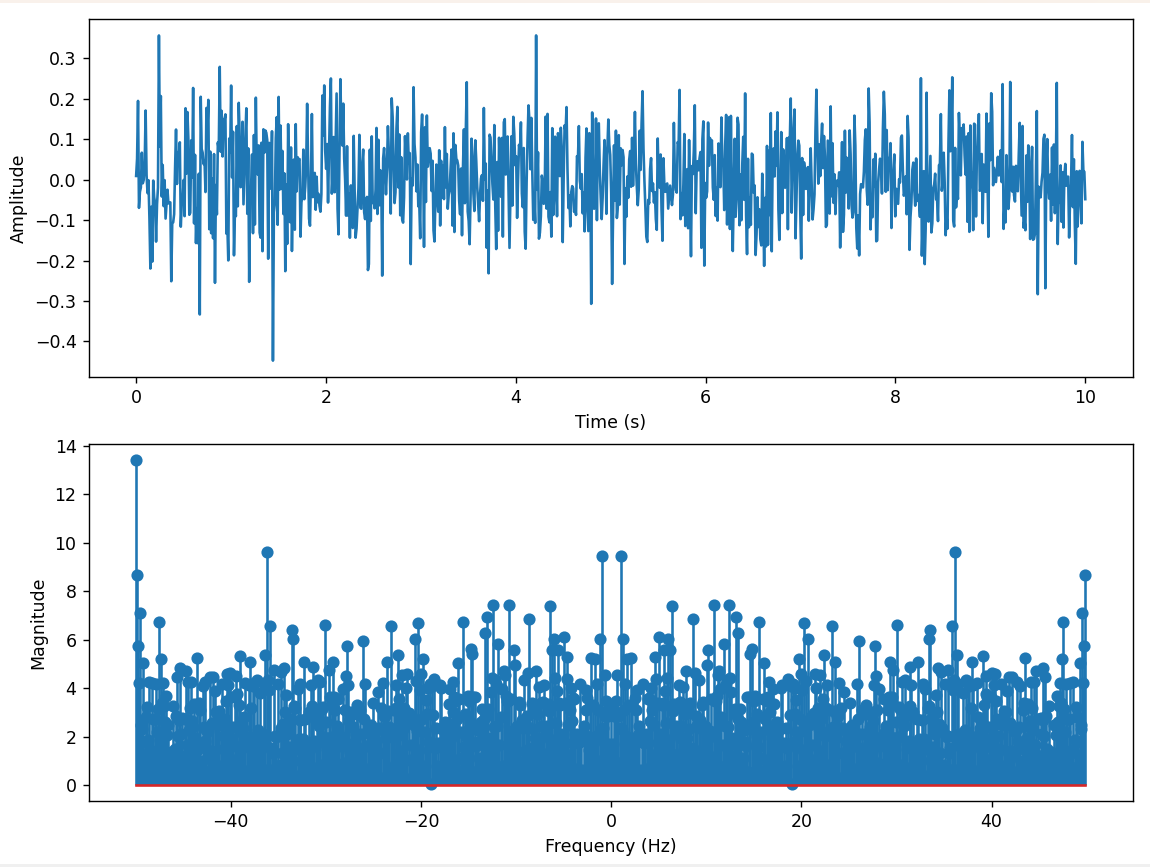
**Detecting signals through wave generation code**

To detect nuclear activity in seismic signals, we utilized the power of FFT in our Python code. First, we generated two distinct waveforms - one containing pure seismic signals and the other with seismic signals along with nuclear activity. These waveforms were created in the amplitude-time domain.

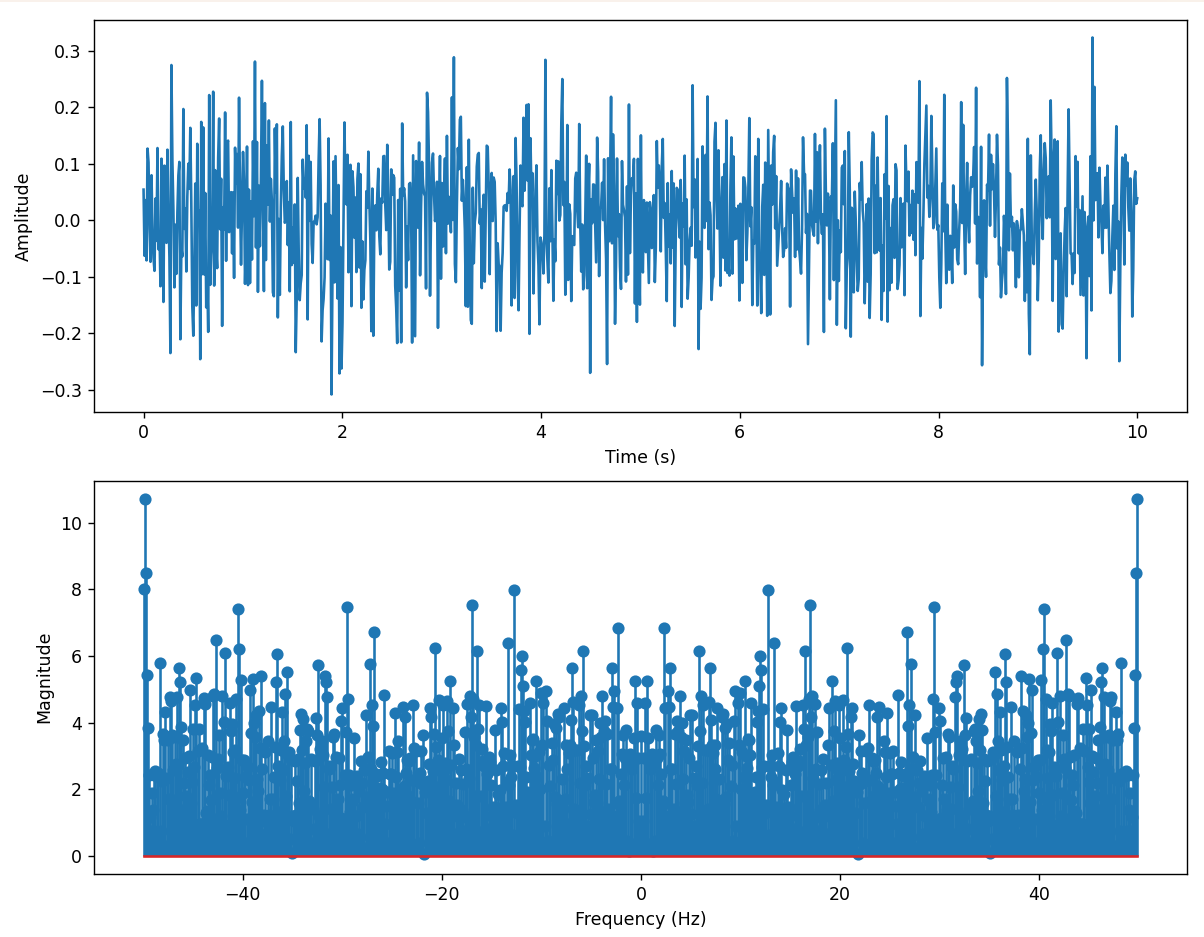
Next, we applied our FFT code to these waveforms to transform them into the amplitude-frequency domain. This enabled us to analyze the resulting graph and its peaks, which provided us with valuable insights regarding the presence of nuclear activity.

By examining the high-frequency components present in the graph, we were able to accurately identify the presence of nuclear activity. This technique not only allowed us to detect nuclear explosions, but it also helped us better understand the complex behavior of seismic signals in real-world scenarios.

No nuclear activity:



Nuclear activity:



**Achievement and Effort**

**The project aimed to develop an algorithm to convert the amplitude-time waveform of a real-life phenomenon, like an earthquake or nuclear explosion, into Fourier Transform. The team used advanced tools and techniques and faced challenges in optimizing the algorithm. They developed an O(NlogN) algorithm capable of producing accurate and reliable results in a timely manner. The project contributes significantly to signal processing and can be used to analyze a range of complex signals.**

**Problems encountered**

**The project aimed to develop an algorithm to convert the amplitude-time waveform of real-life phenomena like earthquakes and nuclear explosions, and faced several challenges such as data quality, algorithm optimization, signal processing complexity, and graphical visualization. Overcoming these challenges required careful planning and execution, but by leveraging the latest tools and techniques in computational science and signal processing, the team developed a high-quality algorithm that effectively achieved the project objectives and contributed to the field of signal processing.**

**Comments on result of Project**

**The project successfully achieved its goal of creating an algorithm that could transform the amplitude-time waveform of a real-world phenomenon with a high degree of accuracy and efficiency. However, there were some areas that could be improved upon. The algorithm's application could be broadened to cover a wider range of real-world occurrences, and the issue of data uncertainty was not addressed, which could affect the accuracy of the results.**

**Future Work**

**Future work could focus on expanding the algorithm's scope to cover a wider range of real-life phenomena such as volcanic eruptions, tsunamis, or weather events. Incorporating uncertainty quantification and optimizing computational efficiency could also improve the accuracy of results and reliability of the algorithm. Further validation using experimental data from real-life phenomena could also be performed to verify the algorithm's effectiveness in practical scenarios and identify areas for improvement.**

**References:**

<https://brianmcfee.net/dstbook-site/content/ch06-dft-properties/Conjugate-Symmetry.html#:~:text=DFT%20conjugate%20symmetry%20says%20that,be%20inferred%20once%20we%20have>