DFT and FFT Intro

Riyad 🔥 now this is the real beast — **Radix-2 Cooley-Tukey FFT**.

This is the algorithm that changed the world.

I'll explain this in 4 layers:

1. 💻 Code meaning
2. 🧠 Mathematical meaning
3. 🌳 Divide & conquer structure
4. ⚡ Why it becomes O(N log N)

Let's go step by step.

🔷 Class Definition

Python

```python
class FastFourierTransform(DFTAnalyzer):
```

💻 Code meaning:

This class **inherits** from `DFTAnalyzer`.

So:

- It can use `compute_dft()` from parent.
- But it overrides it with FFT version.

🧠 Concept meaning:

FFT computes the SAME DFT — just faster.

🔷 Power of 2 Check

Python

```python
def _is_power_of_two(self, n):
    return n > 0 and (n & (n - 1)) == 0
```

💻 What is `(n & (n - 1))`?

Binary trick.

Example:

8 = `1000`

7 = `0111`

Code

```
```

```
1000
0111
----
0000
```

Only powers of 2 behave like this.

So this checks if:

$$N = 2^m$$

🧠 Why power of 2?

Radix-2 FFT splits signal into halves repeatedly.

So N must be divisible by 2, then 4, then 8...

🔷 The Core: `_fft_recursive`

```Python
def _fft_recursive(self, x):
```

This is where magic happens.

Step 1: Base Case

```Python
if N <= 1:
    return x
```

If signal has 1 sample → DFT is itself.

That's the stopping condition of recursion.

Step 2: If Not Power of 2

```Python
if not self._is_power_of_two(N):
```

Fallback to naive DFT.

Because radix-2 method won't work.

🔷 Step 3: Split Even & Odd

Python

```python
even = self._fft_recursive(x[0::2])
odd = self._fft_recursive(x[1::2])
```

This is THE CORE IDEA of Cooley-Tukey.

💻 What does `x[0::2]` mean?

Take every 2nd element starting from index 0.

Example:

Code

```
x = [x0, x1, x2, x3, x4, x5]
```

Even → `[x0, x2, x4]`
Odd → `[x1, x3, x5]`

🧠 Why split?

Because DFT formula:

$$X[k] = \Sigma x[n]e^{-j2\pi kn/N}$$

We separate sum into:

Even indices
Odd indices

Then factor common exponential terms.

This reduces one N-point DFT into:

Two N/2-point DFTs.

That’s the breakthrough idea.

🔷 Twiddle Factors

Python

```python
T = np.exp(-1j * 2 * np.pi * np.arange(N // 2) / N) * odd
```

Let’s decode this slowly.

Step 1:

Python

```python
np.arange(N // 2)
```

Gives:

Code

```
[0, 1, 2, ..., N/2 - 1]
```

These are k values.

Step 2:

Python

```python
np.exp(-1j * 2 * np.pi * k / N)
```

This is:

$$W_N^k = e^{-j2\pi k/N}$$

Called **twiddle factors**.

They rotate the odd part properly.

Step 3:

Multiply by odd:

$$T[k] = W_N^k \cdot O[k]$$

🔷 Combine Step (Butterfly)

Python

```python
return np.concatenate([even + T, even - T])
```

This is the famous **butterfly operation**:

$$X[k] = E[k] + W_N^k O[k]$$

$$X[k + N/2] = E[k] - W_N^k O[k]$$

That’s it.

That’s the FFT formula.

🧠 Big Picture Structure

If N = 8

Code

```
8
↓
4 + 4
↓
2 + 2 + 2 + 2
↓
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
```

Each level splits into halves.

Number of levels:

$$\log_2 N$$

Each level processes N elements.

So total work:

$$N \log N$$

🔥 That's why FFT is fast.

🔷 IFFT Using Conjugate Trick

Python

```python
x = np.conj(self._fft_recursive(np.conj(X))) / N
```

This uses identity:

$$IFFT(X) = \frac{1}{N} \mathrm{conj}(FFT(\mathrm{conj}(X)))$$

Why does this work?

Because:

Forward transform has negative exponent.

Conjugating flips sign of exponent.

So FFT engine can be reused.

Very clever.

🔷 compute_dft()

Python

```python
if not self._is_power_of_two(N):
```

If not power of 2:

Find next power:

Python

```python
while next_pow2 < N:
    next_pow2 <<= 1
```

`<<= 1` means multiply by 2.

Example:

Code

```
1 → 2 → 4 → 8 → 16
```

Then zero-pad:

Python

```python
padded_signal = signal.pad(next_pow2)
```

Then compute FFT.

Then truncate back.

🔥 Important Subtle Point

When you pad:

You are changing frequency resolution slightly.

Returning only first N coefficients is approximation.

But acceptable in many applications.

⚡ Final Complexity

Naive DFT:

$$O(N^2)$$

FFT:

$$O(N \log N)$$

For N = 1,000,000:

DFT → 10^{12} operations 😵
FFT → 20,000,000 operations 😎

Huge difference.

🧠 Deep Understanding Question

Why does splitting into even and odd reduce complexity?

Because:

Original DFT:

$$\sum_{n=0}^{N-1}$$

FFT rewrites it as:

Two smaller sums of size N/2.

Repeated recursively.

🎯 Final Test for You

If N = 8,

How many butterfly stages are there?

Think:

$$\log_2(8) = ?$$

Answer that 😎

cant understand butterfly operation, why/ how

Riyad 🔥 PERFECT.

If you don’t understand butterfly, you don’t understand FFT.
So now we slow down completely.

No code.
No speed.
Just pure intuition.

🌊 Start From the Original DFT

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

This is one big sum.

Now the trick is:

👉 Split the sum into EVEN and ODD indices.

🧠 Step 1: Split the Signal

Instead of:

$$x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]$$

Group like this:

Even terms:

$$x[0], x[2], x[4], x[6]$$

Odd terms:

$$x[1], x[3], x[5], x[7]$$

🧠 Step 2: Rewrite the DFT

Original sum:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}$$

where

$$W_N = e^{-j2\pi/N}$$

Now separate:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m] W_N^{k(2m)} + \sum_{m=0}^{N/2-1} x[2m+1] W_N^{k(2m+1)}$$

🧠 Step 3: Simplify Exponents

Notice:

$$W_N^{k(2m)} = W_{N/2}^{km}$$

🔥 This is the KEY SIMPLIFICATION.

Because:

$$e^{-j2\pi k(2m)/N} = e^{-j2\pi km/(N/2)}$$

So the even part becomes:

$$E[k]$$

Which is just a smaller DFT of size N/2.

The odd part becomes:

$$W_N^k O[k]$$

Where:

$$O[k]$$

is another N/2 DFT.

🎯 So Final Formula Becomes

$$X[k] = E[k] + W_N^k O[k]$$

AND

$$X[k + N/2] = E[k] - W_N^k O[k]$$

THIS PAIR is called the butterfly.

🦋 Why Is It Called Butterfly?

Because the computation diagram looks like:

Code

```
 E[k]
   \    /
    +   -
   /    \
W*O[k]
```

Two inputs

Two outputs

Crossing structure → looks like butterfly wings.

💡 Intuition Without Math

Imagine:

Even part = some wave

Odd part = another wave

To build full spectrum:

You:

1. Rotate odd part (multiply by twiddle)
2. Add to even → gives one frequency

3. Subtract from even → gives another frequency

So from 2 smaller transforms,
you generate 2 outputs at once.

That’s the butterfly operation.

🔥 Very Small Numerical Example (N=2)

Suppose:

x[0] = a
x[1] = b

DFT of size 2:

$$X[0] = a + b$$

$$X[1] = a - b$$

THIS is already a butterfly.

Even = a
Odd = b
Twiddle = 1

Output:

a + b
a − b

That’s the simplest butterfly.

🧠 What FFT Really Does

At every stage:

Take pairs of values
Combine them as:

a + Wb
a − Wb

That’s it.

It keeps doing this recursively.

⚡ Why It Reduces Work

Instead of computing:

N^2 multiplications,

We compute:

Small DFTs
Then combine them using butterflies

Each stage costs N work
Number of stages = log N

So total = N log N.

🎯 Now I Ask You (Very Important)

In butterfly:

Why do we compute BOTH:

E[k] + W O[k]
and
E[k] − W O[k]

Why not just the first one?

Think carefully 😎

how do u separate

YESSSS 🔥 this is the exact point where most people get confused.

You're asking:

👉 How do we mathematically separate

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

into even and odd parts?

Let's do it VERY slowly.

🌊 Step 1: Start from original DFT

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

where

$$W_N = e^{-j2\pi/N}$$

So this is just:

$$x[0]W_N^{k0} + x[1]W_N^{k1} + x[2]W_N^{k2} + \cdots + x[N-1]W_N^{k(N-1)}$$

Nothing fancy.

🧠 Step 2: Split indices into EVEN and ODD

Every integer n is either:

- even → n = 2m
- odd → n = 2m + 1

Right?

So instead of summing over n,
we sum over m.

🔷 EVEN TERMS

Even indices:

$$n = 2m$$

If n goes from 0 to N−1,

then m goes from:

$$0 \text{ to } N/2 - 1$$

So even part becomes:

$$\sum_{m=0}^{N/2-1} x[2m]W_N^{k(2m)}$$

That's just replacing n by 2m.

Nothing magical.

🔷 ODD TERMS

Odd indices:

$$n = 2m + 1$$

So odd part becomes:

$$\sum_{m=0}^{N/2-1} x[2m+1]W_N^{k(2m+1)}$$

Again, just substitution.

🔥 That's It.

So we just rewrote:

$$\sum_{n=0}^{N-1}$$

as:

$$\sum_{\text{even } n} + \sum_{\text{odd } n}$$

That's how we separate.

No trick.

Just:

Every integer = either even or odd.

🧠 Why does m go to N/2 − 1?

Example:

Let N = 8.

Even indices:

0, 2, 4, 6

How many?

4 = N/2

So m runs:

0,1,2,3

So:

$$2m = 0, 2, 4, 6$$

Works perfectly.

🚀 Now the Magic Simplification

We look at:

$$W_N^{k(2m)}$$

Replace:

$$W_N = e^{-j2\pi/N}$$

So:

$$W_N^{2km} = e^{-j2\pi(2km)/N} = e^{-j2\pi km/(N/2)}$$

But that is exactly:

$$W_{N/2}^{km}$$

🔥🔥🔥

This is the key reason FFT works.

🎯 So Final Result

$$X[k] = \underbrace{\sum_{m=0}^{N/2-1} x[2m]W_{N/2}^{km}}_{E[k]} + \underbrace{W_N^k \sum_{m=0}^{N/2-1} x[2m+1]W_{N/2}^{km}}_{O[k]}$$

Where:

E[k] = DFT of even samples

O[k] = DFT of odd samples

That’s the full separation.

🧠 Super Simple Example (N=4)

Original sum:

$$x[0]W_4^{0k} + x[1]W_4^{1k} + x[2]W_4^{2k} + x[3]W_4^{3k}$$

Group:

Even:

$$x[0]W_4^{0k} + x[2]W_4^{2k}$$

Odd:

$$x[1]W_4^{1k} + x[3]W_4^{3k}$$

That’s all we did.

Just grouping terms.

🎯 Now I Ask You

Why does splitting into even and odd reduce N^2 work into N log N?

Where exactly is the saving happening?

Think deeply 😎

not understand

Perfect. Let’s forget abstraction.

Let’s do one **real numeric example** so you SEE it.

🌟 Take N = 4

Let the signal be:

$$x = [x_0, x_1, x_2, x_3]$$

The 4-point DFT is:

$$X[k] = \sum_{n=0}^{3} x[n] W_4^{kn}$$

where

$$W_4 = e^{-j2\pi/4}$$

Step 1 — Separate Even and Odd

Even samples:

$$x_0, x_2$$

Odd samples:

$$x_1, x_3$$

Now compute two **2-point DFTs**.

🔵 Even DFT (size 2)

For size 2:

$$W_2 = e^{-j2\pi/2} = -1$$

So:

$$E[0] = x_0 + x_2$$

$$E[1] = x_0 - x_2$$

That’s it.

🔴 Odd DFT (size 2)

$$O[0] = x_1 + x_3$$

$$O[1] = x_1 - x_3$$

Step 2 — Combine Them

Now comes the butterfly.

Formula:

$$X[k] = E[k] + W_4^k O[k]$$

$$X[k+2] = E[k] - W_4^k O[k]$$

Since N/2 = 2.

So we only compute for:

k = 0, 1

🔹 Case k = 0

$$W_4^0 = 1$$

So:

$$X[0] = E[0] + O[0]$$

$$X[2] = E[0] - O[0]$$

🔹 Case k = 1

$$W_4^1 = e^{-j\pi/2} = -j$$

So:

$$X[1] = E[1] + (-j)O[1]$$

$$X[3] = E[1] - (-j)O[1]$$

🔥 This pair:

$$X[k] = E[k] + W_4^k O[k]$$

$$X[k+2] = E[k] - W_4^k O[k]$$

is ONE butterfly.

It takes:

- one E value
- one O value
- multiplies O by twiddle
- does one add

- one subtract

That’s the butterfly structure.

🧠 Why This Saves Work

Without FFT:

4-point DFT → 16 complex multiplications.

With FFT:

- two 2-point DFTs (each 4 ops)
- plus 4 butterfly ops

Much smaller.

And this keeps splitting:

4 → two 2
8 → two 4
16 → two 8

That recursive splitting gives:

$$N \log_2 N$$

instead of

$$N^2$$

💡 The Real Intuition

Butterfly is just:

Take two partial DFT results
Rotate one (multiply by twiddle)
Add and subtract

That’s it.

No magic.

If you want, I can now draw the butterfly as a signal flow diagram so it clicks visually.

how

Ahhhhh 🔥 GOOD — this is the exact confusing line:

$$X[k + N/2] = E[k] - W_N^k O[k]$$

You're asking:

👉 **HOW does this appear?**

👉 Why does plugging in $k + N/2$ create a minus sign?

Let's derive it slowly. No jumps.

Step 1️⃣ Start from the FFT formula

After separating even and odd, we got:

$$X[k] = E[k] + W_N^k O[k]$$

Now we want to compute:

$$X[k + N/2]$$

So just replace **k** with **k + N/2** in the same formula.

Step 2️⃣ Substitute into definition

$$X[k + N/2] = E[k + N/2] + W_N^{k+N/2} O[k + N/2]$$

Now here's the key trick.

Step 3️⃣ Use periodicity of E and O

E[k] and O[k] are **DFTs of length N/2**.

DFTs are periodic with their length.

So:

$$E[k + N/2] = E[k]$$

$$O[k + N/2] = O[k]$$

Why?

Because their period is $N/2$.

So now equation becomes:

$$X[k + N/2] = E[k] + W_N^{k+N/2} O[k]$$

Step 4️⃣ Expand the Twiddle

$$W_N^{k+N/2} = W_N^k \cdot W_N^{N/2}$$

Now compute:

$$W_N^{N/2} = e^{-j2\pi(N/2)/N} = e^{-j\pi} = -1$$

🔥 THAT is the entire reason.

So:

$$W_N^{k+N/2} = W_N^k \cdot (-1) = -W_N^k$$

Step 5️⃣ Plug Back

$$X[k + N/2] = E[k] + (-W_N^k)O[k]$$

Which becomes:

$$X[k + N/2] = E[k] - W_N^k O[k]$$

Boom.

🎯 So Where Did the Minus Come From?

From this identity:

$$W_N^{N/2} = -1$$

That’s it.

Nothing else.

🧠 Big Insight

FFT works because:

$$W_N^{N/2} = -1$$

If that wasn’t true, butterfly wouldn’t form cleanly.

🔥 Final Butterfly Pair

$$X[k] = E[k] + W_N^k O[k]$$

$$X[k + N/2] = E[k] - W_N^k O[k]$$

One plus.
One minus.

That’s the butterfly.

If you want, next I can show why this gives symmetry in frequency bins — that’s another huge insight.

Why do we compute BOTH:

E[k] + W O[k]
and
E[k] − W O[k]

Why not just the first one?

YES 🔥
THIS is the right question.

You're no longer asking "how".
You're asking **why both outputs are necessary**.

Let's go slow.

🌊 Step 1 — What Are We Trying To Compute?

We are computing an **N-point DFT**.

That means we must compute:

$$X[0], X[1], X[2], \ldots, X[N-1]$$

All N frequency bins.

Not half.

All of them.

🌊 Step 2 — After Splitting

When we split into even and odd, we compute:

- E[k] → N/2 values
- O[k] → N/2 values

So far we only have **N/2 frequency indices** handled (k = 0 to N/2−1).

But the original DFT has N frequency indices.

So where do the other N/2 outputs come from?

They come from evaluating the formula at:

$$k + N/2$$

🌊 Step 3 — What Happens When We Plug That In?

We discovered:

$$X[k] = E[k] + W_N^k O[k]$$

$$X[k + N/2] = E[k] - W_N^k O[k]$$

So for each k (0 to N/2−1):

One butterfly produces **two outputs**.

That’s why we compute both.

🧠 Big Intuition

Think of it like this:

Each pair (E[k], O[k]) contains enough information to produce **two frequency bins**.

If you only computed:

$$E[k] + WO[k]$$

You would only get half the spectrum.

You would be missing:

$$X[k + N/2]$$

🔥 Concrete Example (N=4)

Signal:

$$x_0, x_1, x_2, x_3$$

DFT has 4 outputs:

$$X[0], X[1], X[2], X[3]$$

After splitting:

- E gives 2 values
- O gives 2 values

Each pair produces:

$$X[0] \text{ and } X[2]$$

and

$$X[1] \text{ and } X[3]$$

If you only used:

$$E[k] + WO[k]$$

You’d only get:

$$X[0], X[1]$$

Half the spectrum.

Incomplete DFT.

💡 Deeper Insight

The DFT matrix is full rank.

You need N independent equations.

The butterfly gives:

- one “sum mode”
- one “difference mode”

These correspond to two different frequency components.

Without the minus branch, you lose half the frequency information.

🎯 Physical Interpretation

Think of even and odd signals like:

Even part = base structure
Odd part = detail

The twiddle rotates the odd part differently for different frequencies.

The + branch and − branch separate how they combine at:

- lower half of spectrum
- upper half of spectrum

⚡ One-Line Answer

We compute both because:

Each butterfly must generate **two frequency bins** to fully reconstruct the N-point DFT.

Now I’ll ask you something that will deepen your understanding:

Why do you think the second half of the spectrum differs only by a minus sign and a phase factor?

That symmetry is not random 👀

understand, but why does it work? why?

YES.
Now you’re asking the **real** question.

Not “how”.
Not “formula”.

But:

> Why does this decomposition even work at all?

Good. Let’s go to the core.

🌊 Step 1 — What Is the DFT REALLY Doing?

The DFT computes:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

Interpretation:

It measures **how much of a complex sinusoid of frequency k** exists in the signal.

That’s it.

It’s a correlation with rotating exponentials.

🌊 Step 2 — Now Split the Signal

Every signal can be written as:

$$x[n] = \begin{cases} \text{even samples} \\ \text{odd samples} \end{cases}$$

So we are not changing the signal.

We are just reorganizing it.

Like separating a list into:

- elements at even positions
- elements at odd positions

Nothing is lost.

🌊 Step 3 — The Key Identity

The magic works because of this identity:

$$e^{-j2\pi k(2m)/N} = e^{-j2\pi km/(N/2)}$$

This means:

When you look at only even samples,
they behave like a **shorter DFT (length N/2)**.

That is the entire reason FFT exists.

The exponential structure allows this clean halving.

🌊 Step 4 — Why Do Two Outputs Appear?

Now here is the real reason.

Look at the exponential for two frequencies:

k
and
k + N/2

Compute:

$$e^{-j2\pi(k+N/2)n/N}$$

Expand:

$$= e^{-j2\pi kn/N} \cdot e^{-j2\pi(N/2)n/N}$$

The second term becomes:

$$e^{-j\pi n} = (-1)^n$$

So:

$$e^{-j2\pi(k+N/2)n/N} = e^{-j2\pi kn/N} \cdot (-1)^n$$

🌊 Step 5 — Why This Is Huge

Notice:

(-1)^n is:

+1 for even n
-1 for odd n

So the frequency k+N/2 behaves like:

Same as frequency k
BUT it flips sign on odd samples.

That's why:

One branch becomes plus
One branch becomes minus

It’s not arbitrary.

It comes from:

$$(-1)^n$$

🔥 The Deep Truth

FFT works because:

The complex exponentials have special symmetry:

$$W_N^{N/2} = -1$$

This creates a natural separation between:

- even samples
- odd samples

And between:

- lower half frequencies
- upper half frequencies

🎯 So Why Does It Work?

Because the exponential basis functions themselves have recursive symmetry.

The math of complex rotations allows the N-point basis functions to be built from N/2-point basis functions.

If we used random basis functions?

FFT would not exist.

🧠 Big Picture Insight

FFT works because:

Complex exponentials are eigenfunctions of time shifts.

That gives them deep symmetry.

That symmetry allows divide-and-conquer.

Let me ask you something important:

If instead of complex exponentials we used cosine-only basis (like real Fourier series), do you think this clean halving would still work?

Think carefully.