Mathématiques pour l'Informatique : Compte Rendu

Valentin VERSTRACTE & Evan PETIT

L3 — Jean-Luc BARIL — November 25, 2021

Table des matières

1 Introduction

Ce rapport concerne le rendu du projet de Mathématiques pour l'informatique. Le projet a été intégralement réalisé en collaboration sur GitHub, en langage C++ pour l'implémentation des algorithmes, et en LaTeXpour le compte-rendu.

Sommaire des fonctionnalités implémentées :

Transformée de Fourier discrète 1D & 2D Implémentation de la transformée discrète directe et inverse depuis la formule mathématique

Transformée de Fourier rapide 1D & 2D Implémentation de la transformée rapide avec l'algorithme de Cooley-Tukey

Séquence de tests Vérification des résultats obtenus

2 Transformée de Fourier discrète

2.1 Transformée de Fourier 1D directe et inverse

2.1.1 Formule

On rappelle la formule, où **g(x)** est le signal original, et $\widehat{\mathbf{g}}$**(x)** est son calcul par la TFD. La transformée de Fourier directe permet d'obtenir $\widehat{g}$(x) depuis g(x).

$$\widehat{g}(k) = \sum_{x=0}^{N-1} g(x) exp(\frac{-2i\pi kx}{N}) \qquad pour \qquad 0 \leq k < N \tag{1}$$

L'inverse permet... L'inverse. On remarque l'introduction d'une multiplication d'un facteur -1 dans l'exponentielle

$$g(k) = \sum_{x=0}^{N-1} \widehat{g}(x) exp(\frac{2i\pi kx}{N}) \qquad pour \qquad 0 \leq k < N \tag{2}$$

L'algorithme équivalent peut être implémenté de façon brutale. On a besoin de deux boucles : Une pour calculer le produit à l'intérieur de la somme pour les valeurs de k allant de 0 à N-1, et une pour sommer tous les résultats obtenus pour x allant de 0 à N-1. Modulo un facteur -1 pour obtenir la transformée inverse.

Algorithm 1: Transformée discrète 1D directe

Data: $g[N]$: Vecteur 1D complexe
Output: $G[N]$: Vecteur 1D complexe
begin
  for $k \leftarrow 0..N-1$ **do**
    for $x \leftarrow 0..N-1$ **do**
      $G[k] \leftarrow G[k] + g[x]*$(exponentielle complexe en fonction de k et x)
    end
  end
end

Comme dit précédemment, la transformée inverse ne demande que l'introduction d'un facteur -1 dans le contenu de l'exponentielle, ainsi que de diviser chaque indice de vecteur par la taille totale du vecteur)

2.1.2 Complexité

La complexité de l'algorithme se déduit assez facilement. Soit **g** le vecteur passé en entrée, de taille **n**, et **k** un indice quelconque de g.

- Pour calculer **g[k]** il faut calculer le produit de g[0] avec une exponentielle complexe, de même pour g[1], g[2], . . . , g[n-1] et faire la somme de tous ces produits. Le tableau est de taille n, on fait donc n produits ainsi qu'une addition (négligeable). On peut dire qu'on effectue **n** opérations élémentaires.
- Il faut répéter cette étape autant de fois qu'il y a d'indices dans le tableau. C'est à dire **n** fois.
- Au total on compte (à quelques constantes près) **n*n** opérations. La complexité finale est donc de l'ordre de O(n * n) = **O(n²)**

2.1.3 Diagramme

On peut réaliser un simple diagramme pour représenter cet algorithme. Ici g[x] est le signal d'entrée (représenté par un vecteur 1D), et G[x] le vecteur en sortie. Ils sont de taille N.

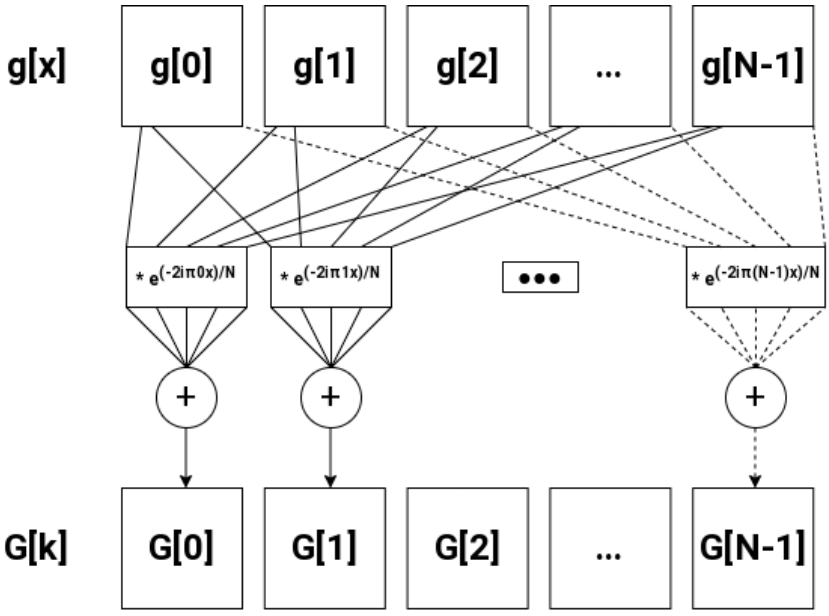

Figure 1: TFD 1D

2.2 Transformée de Fourier 2D discrète

La formule de la transformée de Fourier 2D discrète directe pour une matrice de taille N,M est la suivante

$$\widehat{g}(k,j) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(x,y) exp(-2i\pi(\frac{jx}{N} + \frac{ky}{M})) \quad pour \quad 0 \leq j < N \quad et \quad 0 \leq k < M \tag{3}$$

Cette fois-ci on boucle sur 2 éléments de taille N (x et j), et 2 éléments de taille M (y et k). La complexité ne s'exprime plus en O(N²) mais en **O(N²M²)**. Si on introduit une matrice carrée de taille N,N cela signifie que la complexité est en $\boldsymbol{O(N^4)}$. C'est gigantesque.

On admettra la propriété énonçant que la transformée de Fourier discrète directe (et indirecte) d'une matrice 2D consiste à effectuer la transformée de Fourier 1D des lignes, suivie de la transformée de Fourier 1D des colonnes.

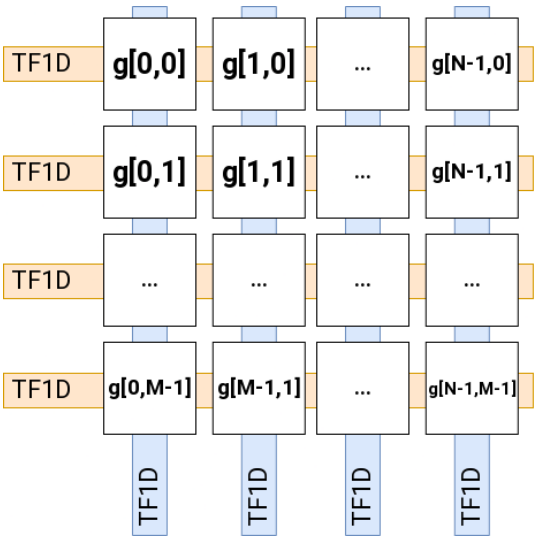

Figure 2: TFD 2D

3 Transformée de Fourier rapide

La transformée de Fourier directe présente un défaut majeur : Son absurde lenteur. On a vu que la transformée de Fourier discrète 1D possédait une complexité polynomiale (en temps O(N^2)).
Si l'on considère un vecteur de taille N avec $N = 2^{12} = 4096$, et la durée d'une opération élémentaire de $10ns$, alors le calcul de la transformée de Fourier de cette matrice prendrait $4096^2 * 10 = 167772160ns = 0.17s =$ **18 minutes**.
Pour un algorithme que l'on retrouve dans énormément de domaines - tant qu'il s'agit d'un pilier de l'informatique moderne - C'est beaucoup trop long.
On chercherait plutôt à se retrouver avec une complexité presque linéaire qu'on retrouve souvent en algorithmique, c'est à dire en temps **O(n log(n))**. Avec un tel algorithme, le temps de calcul pour une matrice avec $N = 2^{12}$, le temps de calcul chuterait à **491 520 ns**, une différence d'ordre de grandeur de 10^3!

Un algorithme permet cela, il s'agit de la transformée de Fourier rapide. Dans la suite sera présentée la version récursive de cet algorithme par les informaticiens **Cooley & Tukey** - Elle a pour avantage d'être assez simple, élégante, et il s'agit de la première version de cet algorithme à être vastement employée. Ses désavantages principaux sont de ne pouvoir travailler qu'avec des vecteurs ayant une taille en puissances de deux - et de devoir se servir de la pile de récursion qui est assez limitée en fonction du langage de programmation. Mais il s'agit d'une très bonne porte d'entrée pour la compréhension de la Transformée de Fourier rapide (la **FFT** pour **F**ast **F**ourier **T**ransform).

Nous ne présenterons que la transformée rapide 1D - En effet on a vu précédemment que la transformée 2D était la composée de deux transformée 1D.

3.1 Principe de l'algorithme de Cooley-Tukey

A l'instar du tri fusion ou du tri rapide, cet algorithme se base sur le principe *Divide and Conquer*.
En **divisant** (divide) le vecteur de base en deux sous-vecteurs de taille inférieure N/2, diviser ces sous-vecteurs en sous-sous-vecteurs, etc... On peut traiter des vecteurs de taille 1. La transformée de Fourier discrète d'un vecteur de taille 1 étant excessivement simple à calculer, et alors il suffit juste de **recombiner** (conquer) ces vecteurs entre eux pour obtenir le résultat : Ce type de schéma se prête très fortement à la récursion - *même si les algorithmes récents cherchent à l'éviter à cause de soucis de performance* - car on procède en problèmes de plus en plus petits.

Cependant, comment choisir de quelle manière diviser ces vecteurs? Il est possible de manipuler légèrement la formule de la transformée de Fourier directe pour obtenir ce que l'on souhaite.

3.2 Décomposition

Soit la formule de la transformée de Fourier 1D. **g** est le vecteur donné en entrée **G** est le calcul de la FFT de ce même vecteur, de taille $\mathbf{N} = 2^n$.

$$G_k = \sum_{x=0}^{N-1} g_x exp(\frac{-2i\pi kx}{N}) \qquad pour \qquad 0 \leq k < N \tag{4}$$

3.2.1 Indices pairs et impairs

La manière la plus courante pour décomposer cette équation en somme de deux termes est de séparer les indices pairs et impairs : x devient alors 2p ou 2p+1 en fonction de sa **parité.** Puisque N est une puissance de 2, il est pair, donc les deux sous-sommes obtenues sont de même taille : **N/2**

$$G_k = \underbrace{\sum_{p=0}^{N/2-1} g_{2p} exp(\frac{-2i\pi k(2p)}{N})}_{\text{Indices pairs}} + \underbrace{\sum_{p=0}^{N/2-1} g_{2p+1} exp(\frac{-2i\pi k(2p+1)}{N})}_{\text{Indices impairs}} \qquad pour \qquad 0 \leq k < N \tag{5}$$

Un des premiers éléments clés de cet algorithme apparaît alors. Même si pour le moment, le contenu des deux sommes n'est pas le même, il est possible que les exponentielles complexes le soient! Dans la somme des indices impairs, il suffit de factoriser par une **constante obtenue depuis le (2p+1)** :

$$G_k = \underbrace{\sum_{p=0}^{N/2-1} g_{2p} exp(\frac{-2i\pi k(2p)}{N})}_{\text{Indices pairs}} + \underbrace{\sum_{p=0}^{N/2-1} g_{2p+1} exp(\frac{-2i\pi k(2p)}{N} + \overbrace{\frac{-2i\pi k * 1}{N}}^{\text{Constante}})}_{\text{Indices impairs}} \quad pour \quad 0 \leq k < N \tag{6}$$

$$G_k = \underbrace{\sum_{p=0}^{N/2-1} g_{2p} exp(\frac{-2i\pi kp}{N})}_{\text{Indices pairs}} + \underbrace{exp(\frac{-2i\pi k}{N/2}) \sum_{p=0}^{N/2-1} g_{2p+1} exp(\frac{-2i\pi kp}{N/2})}_{\text{Indices impairs}} \quad pour \quad 0 \leq k < N \tag{7}$$

On peut déjà remarquer que les deux sommes correspondent à la formule de deux transformées de Fourier. Si l'on appelle **P** la transformée de Fourier des indices **P**airs, et **I** la transformée de fourier des indices **I**mpairs, alors on a :

$$G_k = \underbrace{\sum_{p=0}^{N/2-1} g_{2p} exp(\frac{-2i\pi kp}{N/2})}_{\text{Transformée de Fourier aux indices pairs : } P_k} + exp(\frac{-2i\pi k}{N}) \underbrace{\sum_{p=0}^{N/2-1} g_{2p+1} exp(\frac{-2i\pi kp}{N/2})}_{\text{Transformée de Fourier aux indices impairs : } I_k} \quad pour \quad 0 \leq k < N \tag{8}$$

$$G_k = P_k + \underbrace{exp(\frac{-2i\pi k}{N})}_{\text{Constante}} I_k \quad pour \quad 0 \leq k < N \tag{9}$$

3.2.2 Plage de valeurs de k

Il reste une dernière épine dans le pied : La plage de valeurs de k. En effet, on calcule G pour k de 0 à N-1, mais il est possible de vérifier que cette équation reste la même pour **k de 0 à N/2 - 1** et pour **k + N/2 avec k de 0 à N/2 - 1** (l'union de ces deux intervalles étant bien k de 0 à N-1)

Cas de l'exponentielle contenue dans I_k et P_k , avec k de 0 à N/2-1

Remplaçons k par k+N/2 et essayons de supprimer le terme N/2. Pour cela servons nous de la forme trigonométrique pour jouer avec la périodicité des fonctions sin et cos :

$$exp(\frac{-2i\pi(k+N/2)p}{N/2}) = cos(\frac{-2\pi(k+N/2)p}{N/2}) + i\, sin(\frac{-2\pi(k+N/2)p}{N/2}) \tag{10}$$

On peut distribuer (k+N/2) et simplifier :

$$exp(\frac{-2i\pi(k+N/2)p}{N/2}) = cos(\frac{-2\pi(N/2)p - 2\pi k}{N/2}) + i\, sin(\frac{-2\pi(N/2)p - 2\pi k}{N/2}) \tag{11}$$

$$exp(\frac{-2i\pi(k+N/2)p}{N/2}) = cos(-2\pi p + \frac{2\pi k}{N/2}) + i\, sin(-2\pi p + \frac{2\pi k}{N/2}) \tag{12}$$

Or, les fonctions cosinus et sinus sont de période 2π, de plus p est un entier. On peut donc se débarrasser des termes fraîchement obtenus, et regarder ce que l'on obtient sous forme exponentielle :

$$exp(\frac{-2i\pi(k+N/2)p}{N/2}) = cos(\frac{2\pi k}{N/2}) + i\, sin(\frac{2\pi k}{N/2}) \tag{13}$$

$$exp(\frac{-2i\pi(k+N/2)p}{N/2}) = exp(\frac{-2i\pi kp}{N/2}) \qquad pour \qquad 0 \leq k < N/2 \tag{14}$$

Cas de la constante devant la somme des index impairs, avec k de 0 à N/2-1

Au début il suffit de procéder de la même manière, cette fois ci avec seulement N au numérateur et sans le facteur p :

$$exp(\frac{-2i\pi(k+N/2)}{N}) = cos(\frac{-2\pi(k+N/2)}{N}) + i\ sin(\frac{-2\pi(k+N/2)}{N}) \tag{15}$$

$$exp(\frac{-2i\pi(k+N/2)}{N}) = cos(\frac{-2\pi(N/2)-2\pi k}{N}) + i\ sin(\frac{-2\pi(N/2)-2\pi k}{N}) \tag{16}$$

Cette fois ci on perd le facteur 2 en sortant $-2\pi(N/2)$ de la fraction car 2 n'est plus au dénominateur :

$$exp(\frac{-2i\pi(k+N/2)}{N}) = cos(-\pi + \frac{2\pi k}{N}) + i\ sin(-\pi + \frac{2\pi k}{N}) \tag{17}$$

Soustraire ou additionner π au sein d'un sinus ou d'un cosinus ne fait que changer son signe.

$$exp(\frac{-2i\pi(k+N/2)}{N}) = -cos(-\pi + \frac{2\pi k}{N}) - i\ sin(-\pi + \frac{2\pi k}{N}) \tag{18}$$

$$exp(\frac{-2i\pi(k+N/2)}{N}) = -(cos(-\pi + \frac{2\pi k}{N}) + i\ sin(-\pi + \frac{2\pi k}{N})) \tag{19}$$

$$exp(\frac{-2i\pi(k+N/2)}{N/2}) = -exp(\frac{-2i\pi k}{N/2}) \qquad pour \qquad 0 \leq k < N/2 \tag{20}$$

Ainsi, en divisant par deux la taille de l'intervalle de k, on divise par deux le nombre d'opérations à effectuer.

3.2.3 Synthèse de la décomposition

On peut définir notre transformée de Fourier en fonction de 2 cas : Quand l'indice est inférieur à N/2 (k), ou quand il est supérieur (k+N/2). Seul le signe de la constante devant la somme des indices impairs change.

$$\begin{cases} G_k = P_k + exp(\frac{-2i\pi k}{N})I_k \quad pour \quad 0 \leq k < N/2 \\ G_{k+\frac{N}{2}} = P_k - exp(\frac{-2i\pi k}{N})I_k \quad pour \quad 0 \leq k < N/2 \end{cases} \tag{21}$$

3.3 Cas de la Transformée de Fourier 1D d'un signal à 1 échantillon

Cette étape est triviale, mais permet de définir le cas de base de la récursion. On a donc pour un signal **g** à **N = 1** échantillon :

$$G_0 = \sum_{x=0}^{0} g_0 \underbrace{exp(\frac{-2i\pi 0x}{1})}_{=0} \qquad pour \qquad 0 \leq k < 1 \tag{22}$$

$$G_0 = g_0 \tag{23}$$

Donc dans le cas d'un vecteur contenant un seul élément, **sa transformée de Fourier équivaut à lui-même.**

3.4 Principe de récursion

3.4.1 Pseudo-code

Munis du cas de base et du cas de propagation, il est donc possible de définir un algorithme calculant la transformée de Fourier de manière récursive :

Algorithm 2: FFT1D(g[N]) - Transformée Rapide (algorithme de Cooley-Tukey)

Data: $g[N]$: Vecteur 1D complexe de taille N tel que N est une puissance de 2
Output: $G[N]$: Vecteur 1D complexe de taille N tel que N est une puissance de 2
begin
  if $N = 1$ **then**
    $G[0] \leftarrow g[0]$ ▷ Cas de base
  else
    $P[N/2] \leftarrow indices\ pairs\ de\ g$ ▷ Cas de propagation
    $I[N/2] \leftarrow indices\ impairs\ de\ g$
    $FFT1D(P)$
    $FFT1D(I)$
    for $k \leftarrow 0..N/2-1$ **do**
      $G[k] \leftarrow P[k] + exp(\frac{-2\pi k}{N})I(k)$
      $G[k+N/2] \leftarrow P[k] - exp(\frac{-2\pi k}{N})I(k)$
    end
  end
  return G
end

Il s'agit de cet algorithme que nous avons implémenté en C++.

3.4.2 Complexité

- Puisque les entrées sont des tableaux possédant une taille n telle que n est une puissance de 2, et que l'algorithme effectue des divisions par 2 sur un tableau de taille n jusqu'à des tableaux de taille 1 : Il faut compter $\boldsymbol{\log_2 n}$ étapes de division. On compte le même nombre d'étapes pour recombiner les tableaux entre eux, mais $O(2 * \log_2 n) = \boldsymbol{O(\log_2 n)}$
- A chaque étape, on effectue une addition ou une soustraction avec un complexe pour **chaque** tableau. C'est à dire qu'on effectue ces opérations n fois. Donc en $\boldsymbol{O(n)}$.
- Au total cela correspond à une complexité en $\boldsymbol{O(nlog_2 n)}$

3.5 Symmétrie de la transformée de Fourier

Le thème récurrent de la transformée de Fourier rapide est la symétrie : En particulier celles de nombres complexes opposés sur le plan complexe. Pour visualiser cela de manière élégante, il est très utile de présenter la notion de **racines n-ièmes de l'unité** - même si ce n'est pas nécessaire.

3.5.1 Racines n-ièmes de l'unité

On appelle **racine n-ième de l'unité** un nombre complexe z tel que $z^n = 1$. Il existe n racines n-ièmes de l'unité différentes et elles sont de la forme $\boldsymbol{exp(\frac{2i\pi k}{n})\ pour\ 0 \leq k < n}$ et cela se démontre facilement :

$$exp(\frac{2i\pi k}{n})^n = exp(\frac{2i\pi k}{n} n) = \boldsymbol{exp(2i\pi k) = 1} \qquad pour\ k\ entier \tag{24}$$

Montrons qu'il suffit que k soit compris entre 0 et n pour obtenir les racines n-ième de l'unité :

$$exp(\frac{2i\pi(k+n)}{n})^n = exp(\frac{2i\pi kn}{n} + \frac{2i\pi n}{n}) = exp(2i\pi k + 2i\pi) = 1 \quad pour \quad 0 \leq k < n \tag{25}$$

Par transitivité avec (23) on a alors :

$$exp(\frac{2i\pi(k+n)}{n})^n = exp(\frac{2i\pi k}{n})^n \quad pour \quad 0 \leq k < n \tag{26}$$

Ce nombre $exp(\frac{2i\pi k}{n})^n$ s'appelle **k-ième racine n-ième de l'unité**, et il est commun de la noter $\boldsymbol{W_n^k}$. Par définition, il s'agit de nombres complexes de module 1 et d'argument $\frac{2i\pi k}{n}{}^n$ d'après la notation exponentielle d'un nombre complexe : $mod.exp(arg)$.
On note aussi l'ensemble des racines n-ièmes de l'unité $\boldsymbol{W_n}$.

3.5.2 Multiplication par les racines n-ième de l'unité

On peut alors faire une remarque qui simplifie énormément la définition des sommes précédentes! En effet, tous les nombres complexes introduits dans les sommes des transformée de Fourier sont de la forme $exp(\frac{2i\pi(k)}{n})$ - Donc des racines n-ième de l'unité! Depuis les formules de la transformée de Fourier rapide on peut introduire la nouvelle notation :

$$\begin{cases} G_k = P_k + W_N^{-k} I_k \quad pour \quad 0 \leq k < N/2 \\ G_{k+\frac{N}{2}} = P_k - W_N^{-(k+n/2)} I_k \quad pour \quad 0 \leq k < N/2 \end{cases} \tag{27}$$

Pour un signal à 8 échantillons la première décomposition se visualise ainsi sur le cercle unité - En représentant le nombre complexe par lequel on multiplie I_k (dessins réalisés avec GeoGebra) :

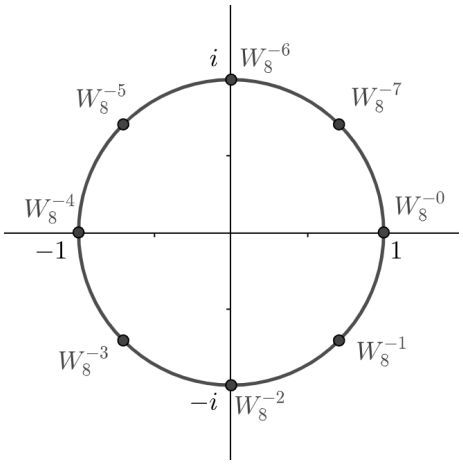

Figure 3: Une seule somme

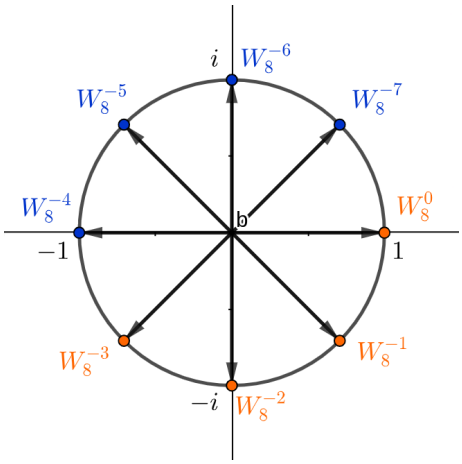

Figure 4: Après la division en 2 sommes

En orange, les termes G_k, et en bleu les termes $G_{k+N/2}$. Une symétrie centrale apparaît, il est alors clair que l'on a besoin de calculer seulement les termes oranges pour trouver les termes bleus. L'un étant l'opposé de l'autre. On passe du calcul de 8 à 4 termes! Nous l'avions déjà montré par le calcul, de l'équation (15) à (20). Cela nous permet de déduire une nouvelle propriété des racines n-ième de l'unité (seulement si n est pair) :

$$-W_n^k = W_n^{k+n/2} \tag{28}$$

3.5.3 Papillon

Il est très commun de sereprésenter la transformée de Fourier par un diagramme nommé papillon (*butterfly*) - Il montre comment se combinent les termes avec les racines n-ième de l'unité. Profitons en pour montrer comment se décompose un signal.
Exemple avec un vecteur de taille N = 8 :

On commence par décomposer le vecteur original en vecteurs de taille 1 à l'aide des divisions successives par 2. Ainsi on peut calculer la transformée de Fourier de chacun d'entre eux très simplement!

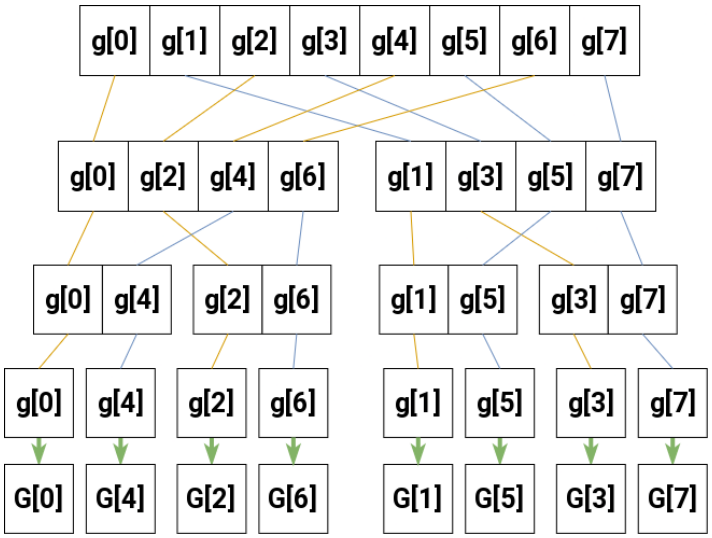

Figure 5: Divide . . .

Puis on recompose les 16 vecteurs de taille 1 en 8 vecteurs de taille 2, puis 4 vecteurs de taille 4, puis 2 vecteurs de taille 8, puis enfin notre vecteur résultat de taille 16! Le fameux diagramme papillon - On peut d'ailleurs voir qu'on utilise seulement les **k/2 racines n-ièmes de l'unité** à chaque étape.

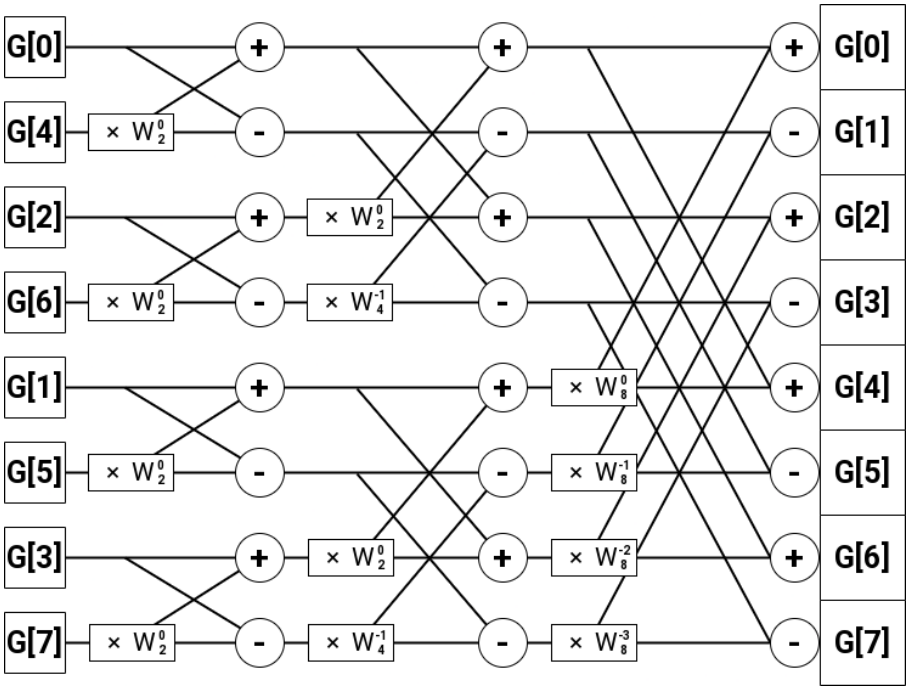

Figure 6: . . . and Conquer!

Le vecteur obtenu en sortie est le signal transformé. *De la même manière que pour la transformée brute, il suffira de changer un signe - par un signe + dans nos exponentielles pour obtenir la transformée inverse : Cela correspond à passer de* $\boldsymbol{W_n^{-k}}$ *à* $\boldsymbol{W_n^k}$