Le sujet de cette séance est les nombres complexes, qui est un thème assez difficile avec pas mal de techniques différentes à retenir pour résoudre les exercices.

Je recommande fortement d'apprendre par coeur les formules de trigonométrie (qui vous seront d'ailleurs très utiles pendant ce td), la plus utile étant $cos(x)cos(y) = \frac{1}{2}(cos(x+y) + cos(x-y))$ et qui permet de retrouver toutes les autres en cas de doute.

Enfin, les planches de td sont désormais disponibles sur `nathan-boyer.vercel.app/tutorat`

Exercice 1. Produit des racines de l'unité Cacluler $\Pi_{\omega \in \mathbb{U}_n} \omega$

Exercice 2. Un air familier Soit $z \in \mathbb{C}$, montrer que $|\frac{1-z^n}{1-z}| \leq \frac{1-|z|^n}{1-|z|}$

Exercice 3. Distances euclidiennes Soit $S = \{m^2 + n^2 | m, n \in \mathbb{N}\}$.

Montrer que S est stable par produit (ie $a, b \in S \Rightarrow ab \in S$).

Exercice 4. Somme de cosinus épisode 431 Soit $n \in \mathbb{N}$, calculer $\sum_{k=1}^{n} cos^2(k\theta)$

Exercice 5. Cosinus rationnels On cherche les rationnels $r = \frac{p}{q}$, $p \wedge q = 1$ tels que $cos(\pi r) \in \mathbb{Q}$. On écrit $2cos(\pi r) = \frac{a}{b}$ avec $a \wedge b = 1$. Pour $k \in \mathbb{N}$, on pose $u_k = 2cos(2^k \pi r)$.

1. Montrer que $u_{k+1} = u_k^2 - 2$. En déduire $u_k \in \mathbb{Q}$.
2. On pose $u_k = \frac{a_k}{b_k}$ avec $a_k \wedge b_k = 1$. Montrer que $b_{k+1} = b_k^2$.
3. Montrer que $\forall k, exp(i2^k \pi r)$ est une racine 2-qeme de l'unité.

   En déduire que l'ensemble $\{u_k \ k \in \mathbb{N}\}$ est fini.
4. Montrer que $\forall k, b_k = 1$. Conclure.

Exercice 6. Faites un dessin Soit $z \in \mathbb{U}$, montrer que $|1 + z| \geq 1$ ou $|1 + z^2| \geq 1$.

Exercice 7. Calcul de cosinus On veut connaître $cos(\frac{2\pi}{5})$.

1. Montrer que $1 + 2cos(\frac{2\pi}{5}) + 2cos(\frac{4\pi}{5}) = 0$
2. En déduire une équation du second degré vérifiée par $cos(\frac{2\pi}{5})$.
3. En déduire $cos(\frac{2\pi}{5})$.

Exercice 8. Transformée de Fourier rapide Dans tout cet exo on admettra ce théorème:

Interpolation de Lagrange: Soit P et Q de degré $\leq n$. Si P et Q coïncident sur n+1 points, alors ils sont égaux.

On va chercher à trouver un algorithme efficace pour multiplier 2 tels polynômes P et Q (de degré $\leq n$ mais pas forcément égaux).

1. Proposer un algorithme naïf pour multiplier 2 polynômes stockés sous la forme $\sum a_k X^k$.

   Combien d'opérations demande-t-il ?

2. Imaginez qu'on connaisse l'évaluation de P et de Q en $n+1$ mêmes points, combien d'opérations seraient alors nécessaires pour connaître l'évaluation de PQ en ces points.

3. On va donc chercher un algorithme qui convertit un polynôme sous forme classique à un polynôme sous forme "évaluée", pour mieux pouvoir les multiplier, puis les remettre sous forme classique.

   Soit $w = e^{\frac{2i\pi}{n}}$. Supposons qu'on ait un algorithme $A(\omega)$ qui a $\lambda_0, \dots, \lambda_{n-1}$ associe $P(\omega^0), \dots, P(w^{n-1})$, avec $P = \sum_{k=0}^{n-1} \lambda_k X^k$.

   Montrer que appliquer $A(\omega^{-1})$ à $P(\omega^0), \dots, P(\omega^{n-1})$ renvoie $n\lambda_0, \dots, n\lambda_{n-1}$.

4. On définit l'algorithme suivant :

   Entrée : Un polynôme $P = \sum_{k=0}^{n-1} \lambda_k X^k$ et ω, une racine primitive n-ième de l'unité.

   Sortie : $P(\omega^0), P(\omega^1), \dots, P(\omega^{n-1})$

Algorithme 1 : FFT(P, ω)

```
if P est constant then
    return P(1)
end
else
    Calculer P_pair et P_impair
    Calculer P_pair(ω^0), P_pair(ω^2), ..., P_pair(ω^{n-2}) et
     P_impair(ω^0), P_impair(ω^2), ..., P_impair(ω^{n-2})
    via FFT(P_pair, ω^2) et FFT(P_impair, ω^2), respectivement.
    for k ← 0 à n − 1 do
        P(ω^k) = P_pair(ω^{2k}) + ω^k · P_impair(ω^{2k})
    end
    return P(ω^0), ..., P(ω^{n-1})
end
```

   Prouver qu'il retourne bien le bon résultat.

   (Je précise que par exemple P_{pair} pour $P = 3X^3 + 2X^2 + X - 1$ est $2X - 1$, et P_{impair} est $3X + 1$).

5. On admettra que le nombre d'opérations faites est en $O(n \log n)$. (càd $= n \log n$ à une constante multiplicative près quand n est grand).

   Conclure sur l'algorithme entier pour la multiplication de 2 polynomes et comparer sa complexité (càd le nombre d'opérations réalisées) à celle de l'algorithme naïf.