Diviser pour régner

Activité – Terminale NSI

① Le principe : couper le problème en deux

Exercice 1 *Trouver le maximum, deux façons*

On souhaite trouver le maximum d'une liste de n nombres.

Approche directe (« force brute »). On parcourt la liste en gardant en mémoire le plus grand vu jusque-là.

Approche diviser pour régner. On coupe la liste en deux moitiés, on cherche récursivement le maximum de chaque moitié, puis on prend le plus grand des deux.

1. Décrire la première approche en pseudo-code.
2. Décrire la seconde approche en pseudo-code, en identifiant clairement :
   - le **cas de base** (liste de longueur 1) ;
   - l'étape **diviser** ;
   - l'étape **régner** (résoudre les sous-problèmes récursivement) ;
   - l'étape **combiner**.
3. Pour une liste de 8 éléments, dessiner l'« arbre de récursion » de l'approche diviser pour régner : à chaque niveau, le problème se divise en deux.

Remarque 1. Le paradigme **diviser pour régner** se résume en trois étapes :

1. **Diviser** le problème en sous-problèmes plus petits (souvent deux moitiés).
2. **Régner** sur chaque sous-problème, généralement par appel récursif.
3. **Combiner** les solutions partielles pour obtenir la solution globale.

② La recherche dichotomique vue comme « diviser pour régner »

Exercice 2 *Récrire la dichotomie en récursif*

La recherche dichotomique dans une liste triée est l'archétype du « diviser pour régner » : on regarde l'élément central, et on continue la recherche dans la moitié pertinente.

1. Compléter la fonction récursive ci-dessous :

```
def dicho(L, x, debut, fin):
    """Cherche x dans L[debut:fin+1]. Renvoie l'indice ou -1."""
    if debut > fin:
        return .....          # cas de base : intervalle vide
    centre = .....
    if L[centre] == x:
```

```
        return ............
    elif L[centre] < x:
        return dicho(L, x, ............, fin)
    else:
        return dicho(L, x, debut, ............)
```

2. Identifier dans cette fonction : le cas de base, l'étape *diviser*, l'étape *régner*, l'étape *combiner* (cette dernière est ici triviale : on renvoie directement le résultat du sous-problème).
3. Pour une liste de $n = 1\,000\,000$ d'éléments, combien d'appels récursifs au maximum ? Justifier en termes de $\log_2(n)$.

Remarque 2. La recherche dichotomique est un cas particulier où l'on n'explore qu'**une seule** des deux moitiés. C'est pour cela qu'elle est aussi rapide ($O(\log n)$). Le tri fusion, en revanche, doit explorer les *deux* moitiés : on verra qu'il est en $O(n \log n)$.

③ Le tri fusion (*merge sort*)

Propriété 1 **(Principe du tri fusion)**

1. **Diviser** la liste en deux moitiés.
2. **Régner** en triant récursivement chaque moitié.
3. **Combiner** les deux moitiés triées en une seule liste triée. C'est l'étape « fusion ».

Exercice 3 *Fusionner deux listes triées*

L'idée de la fusion. Considérer deux mains de cartes triées. Pour les rassembler dans une troisième main triée, on regarde la carte du dessus de chaque pile et on prend la plus petite. On répète jusqu'à épuisement.

1. Compléter la fonction `fusion(A, B)` qui prend deux listes *déjà triées* et renvoie leur fusion triée :

```
def fusion(A, B):
    resultat = []
    i = ............
    j = ............
    while i < len(A) and j < len(B):
        if A[i] <= B[j]:
            resultat.append(A[i])
            i = ............
        else:
            resultat.append(B[j])
            j = ............
    # Ajouter les éléments restants de A (ou de B) non encore copiés
    while i < len(A):
        resultat.append(A[i])
        i = i + 1
    while j < len(B):
        resultat.append(B[j])
        j = j + 1
    return resultat
```

2. Tester avec `fusion([1, 4, 7], [2, 3, 6, 8])`. Résultat attendu : `[1, 2, 3, 4, 6, 7, 8]`.

3. Compléter la fonction `tri_fusion(L)` qui trie `L` récursivement :

```
def tri_fusion(L):
    if len(L) <= 1:
        return ............
    m = ............
    gauche = tri_fusion(............)
    droite = tri_fusion(............)
    return fusion(............, ............)
```

④ L'exponentiation rapide

Exercice 4 *Calculer x^n rapidement*

Calculer x^{20} par la méthode naïve demande 19 multiplications. La méthode « diviser pour régner » en demande beaucoup moins.

L'idée : pour calculer x^n, on remarque que :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ (x^{n/2})^2 & \text{si } n \text{ est pair} \\ x \times (x^{(n-1)/2})^2 & \text{si } n \text{ est impair} \end{cases}$$

1. Compléter la fonction :

```
def puissance(x, n):
    if n == 0:
        return ............
    moitie = puissance(x, n // 2)
    if n % 2 == 0:
        return ............
    else:
        return ............
```

2. Détailler le calcul de `puissance(2, 10)` en suivant les appels récursifs.
3. Combien de multiplications effectue-t-on environ pour calculer x^n ? Comparer avec la méthode naïve qui en effectue $n - 1$.

⑤ Tous les algorithmes récursifs ne sont pas « diviser pour régner »

Exercice 5 *Diviser pour régner ou simple récursion ?*

Pour chaque fonction, indiquer si elle relève du paradigme « diviser pour régner » ou s'il s'agit d'une simple récursion linéaire. Justifier en une phrase.

1. La factorielle récursive `fact(n)`.
2. La somme récursive d'une liste `somme(L)`.
3. Le tri fusion.
4. La recherche dichotomique récursive.
5. Fibonacci récursif naïf.

6. L'exponentiation rapide.

⑥ Ce qu'il faut retenir

Propriété 2 **(Le paradigme)** Diviser pour régner : **(1)** diviser le problème en sous-problèmes plus petits ; **(2)** les résoudre récursivement ; **(3)** combiner les solutions.

Propriété 3 **(Trois algorithmes types)**

— **Recherche dichotomique** : $O(\log n)$ — on n'explore qu'une moitié.

— **Tri fusion** : $O(n \log n)$ — on explore les deux moitiés et on fusionne.

— **Exponentiation rapide** : $O(\log n)$ multiplications — on divise l'exposant par deux.

Remarque 3. La force du paradigme est de transformer un coût $O(n)$ ou $O(n^2)$ en $O(\log n)$ ou $O(n \log n)$: le gain devient énorme dès que n grandit. C'est aussi pour cela que le tri fusion est utilisé en pratique sur de très grandes listes, là où les tris naïfs deviennent inutilisables.