Programmation 2 Spring, 2026

TD 11: Divide-and-Conquer Algorithms 2

1. (**TP Question.**) Considérez le problème des personnes honnêtes/malhonnêtes sur une liste `A` de taille n.

   Pour simplifier, vous pouvez supposer que n est une puissance de 2.

   Voici un algorithme :

   > **Cas de base :** Si $|A| = 1$, retourner l'unique personne dans A.
   >
   > **Étape récursive :**
   >
   > - Coupez A en deux listes de taille égale A_1 et A_2.
   > - Calculez récursivement une personne honnête p_1 dans A_1.
   > - Calculez récursivement une personne honnête p_2 dans A_2.
   > - Avec une boucle, vérifiez si p_1 est honnête dans A (en posant $n-1$ questions). Si 'oui', retournez p_1, sinon retournez p_2.

   Montrez que si l'algorithme ci-dessus est appelé sur A (avec majorité stricte d'honnêtes), il retourne une personne honnête.

   Montrez également que le nombre de questions posées est au plus $n \log_2 n$.

2. (**TP Question**) Considérez le problème des personnes honnêtes et malhonnêtes sur une liste `A` de taille n. On suppose qu'il y a strictement plus de personnes honnêtes que de malhonnêtes dans `A`.

   Considérez les idées suivantes :

   (a) Interrogez la personne `A[1]` à propos de la personne `A[0]`.
   (b) Si `A[1]` répond que `A[0]` est malhonnête, que pouvez-vous en déduire sur `A[0]` ? Sur `A[1]` ? Sur les deux ?
   (c) Si `A[1]` répond que `A[0]` est honnête, comment pouvez-vous alors passer à la personne suivante `A[2]` ?

   Concevez un algorithme **itératif** basé sur les indices ci-dessus.

   Montrez que le nombre total de questions posées par l'algorithme est au plus $n-1$.

3. (**TP Question. Difficile**) Considérez le problème des personnes honnêtes et malhonnêtes sur une liste `A` de taille n.

   (a) Concevez un algorithme **récursif** qui fonctionne de la manière suivante :
      - Divisez `A` en $n/2$ paires de personnes.
      - Pour chaque paire, interrogez la première personne à propos de la seconde.
      - En utilisant les réponses obtenues, construisez un nouvel ensemble `B` de taille $n/2$ sur lequel l'algorithme s'appelle récursivement.

   (b) Montrez que lorsque l'algorithme est exécuté sur `A`, il retourne toujours une personne honnête (en supposant que la majorité des personnes dans `A` est honnête).

   (c) Montrez que le nombre total de questions posées par l'algorithme est au plus $n-1$.

4. Écrivez une fonction `merge` qui prend deux listes `A` et `B`, *les deux listes étant déjà triées par ordre croissant*. Elle renvoie ensuite une seule liste combinée, triée par ordre croissant.

   Votre algorithme doit prendre $O(\text{len(A)} + \text{len(B)})$ temps.

Example: pour `A = [1,4,30], B = [0,3,10,20,50]`, la fonction renvoie `[0,1,3,4,10,20,30,50]`.

5. Écrivez une fonction **récursive** `mergeSort_recursive` qui prend une liste `A` de nombres et les renvoie dans une nouvelle liste triée par ordre croissant.

> Voici l'algorithme du tri fusion (mergesort) :
>
> (a) Si la liste `A` a une taille inférieure ou égale à 1, retournez `A` (cas de base).
> (b) Divisez `A` en deux sous-listes `A1` et `A2` de taille égale (ou presque égale).
> (c) Triez récursivement `A1` et `A2` par appels à `mergeSort_recursive`.
> (d) Fusionnez (merge) les deux sous-listes triées en une seule liste triée.
> (e) Retournez la liste fusionnée.

Indice: vous pouvez utiliser la fonction `merge` de l'exercice précédent.

6. Écrivez une fonction **itératif** `mergeSort` qui prend une liste `A` de nombres et les trie. On peut supposer que `len(A)` est une puissance de 2.

> L'algorithme effectue les opérations suivantes :
>
> (a) Fusionner A[0] et A[1], fusionner A[2] et A[3], fusionner A[4] et A[5], et ainsi de suite jusqu'à la fin de la liste.
> (b) Fusionner ensuite A[0:2] et A[2:4], fusionner A[4:6] et A[6:8], et ainsi de suite jusqu'à la fin de la liste.
> (c) Fusionner ensuite A[0:4] et A[4:8], et ainsi de suite.
> ⋮

Quel est le temps d'exécution asymptotique de cet algorithme ?

Indice: vous pouvez utiliser la fonction `merge` de l'exercice précédent.

7. Écrivez une fonction récursive `quickSort_recursive` qui prend une liste `A` de nombres et les renvoie dans une nouvelle liste triée par ordre croissant.

> Algorithme (avec pivot = premier élément) :
>
> (a) Si $|A| \leq 1$, retourner A.
> (b) Soit $p = A[0]$ le pivot.
> (c) Soit $G = [x \in A[1:] \mid x \leq p]$ (éléments inférieurs ou égaux au pivot).
> (d) Soit $D = [x \in A[1:] \mid x > p]$ (éléments supérieurs au pivot).
> (e) Retourner quickSort_recursive$(G) + [p] +$ quickSort_recursive(D).

8. **(difficile)** Nous jouons au jeu des devinettes. Le principe est le suivant :

   (a) Je choisis un nombre entre 1 et n. Tu dois deviner lequel.
   (b) Chaque fois que tu te trompes, je te dirai si le nombre que j'ai choisi est supérieur ou inférieur.

   Cependant, si tu devines un nombre x et que tu te trompes, tu paies un prix de euro x.

   Tu gagnes la partie lorsque tu devines le nombre que j'ai choisi.

   Écrivez une fonction récursive `findBestStrategy_slow` pour calculer, étant donné n, le montant minimum nécessaire pour gagner.