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 $n \log_2 n$.

   Solution. Hypothèse de récurrence : Pour toute liste de taille $m < n$ contenant strictement plus d'honnêtes que de malhonnêtes, l'algorithme retourne une personne honnête.

   Lemme : Si A contient une majorité stricte d'honnêtes, alors au moins l'un des deux candidats p_1 ou p_2 est honnête.

   Preuve : D'abord, notez que au moins l'une des deux A_1 (avec M_1 malhonnêtes et H_1 honnêtes) ou A_2 (avec M_2 malhonnêtes et H_2 honnêtes) contient également une majorité stricte d'honnêtes:

   Supposons que **ni** A_1 **ni** A_2 n'ait une majorité stricte d'honnêtes. Cela signifie que dans chaque moitié, le nombre de malhonnêtes est **au moins égal** au nombre d'honnêtes :

   $$|M_1| \geq |H_1| \quad \text{et} \quad |M_2| \geq |H_2| \qquad \Longrightarrow \qquad |M_1| + |M_2| \geq |H_1| + |H_2|.$$

   Mais $|M_1| + |M_2| = |M|$ (le nombre total de malhonnêtes dans A), et $|H_1| + |H_2| = |H|$ (le nombre total d'honnêtes dans A). On a donc $|M| \geq |H|$, qui contredit l'hypothèse de départ, qui était $|H| > |M|$.

   Dans au moins l'une des deux listes A_1 ou A_2, le nombre d'honnêtes est supérieur au nombre de malhonnêtes. Par l'hypothèse de récurrence, l'appel récursif sur cette moitié retourne une personne honnête. Donc au moins p_1 ou p_2 est honnête.

   La boucle vérifie p_1 est honnete en posant $|A| - 1$ questions. Comme au moins l'un des deux est honnête, l'algorithme en trouvera un. Ainsi, l'algorithme appliqué sur A retourne toujours une personne honnête.

   L'arbre de récursion a une profondeur de $\log_2 n$ niveaux, car à chaque appel récursif la taille de la liste est divisée par deux (on coupe A en deux listes de taille égale A_1 et A_2), jusqu'à atteindre des listes de taille constante.

   Par conséquent, le nombre total de questions posées est :

   $$(\text{questions par niveau}) \times (\text{nombre de niveaux}) \leq n \cdot \log_2 n.$$

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 $n - 1$.

Solution. Voici l'algorithme :

(a) Ajoutez la personne `A[0]` à une nouvelle liste `B`.

(b) Pour chaque personne suivante `A[i]` avec $i = 1$ à $n - 1$:

- Si `B` est vide, ajoutez `A[i]` à `B` et passez à l'itération suivante.
- Sinon, interrogez `A[i]` à propos de `B[-1]` (la dernière personne de `B`).
- Si `A[i]` répond que `B[-1]` est honnête, ajoutez `A[i]` à la fin de `B`.
- Si `A[i]` répond que `B[-1]` est malhonnête, retirez `B[-1]` de `B` (et n'ajoutez pas `A[i]`).

(c) Après avoir traité `A[n-1]`, retournez `B[0]` (le premier élément de `B`).

Invariant : Dans `B`, il ne peut jamais arriver que `B[i]` soit malhonnête alors que `B[i+1]` est honnête.

Preuve : Considérons une paire adjacente quelconque (B[i], B[i+1]) dans `B`. L'élément `B[i+1]` a été ajouté à `B` lors du traitement de `A[k]`, et il a été ajouté précisément parce que `A[k]` (qui est devenu `B[i+1]`) a déclaré que `B[i]` était honnête. Si `B[i+1]` est honnête, alors sa déclaration est véridique, donc `B[i]` doit être honnête également. Si `B[i+1]` est malhonnête, l'implication "si `B[i+1]` est honnête alors `B[i]` est honnête" est vacuement vraie. L'invariant est donc préservé.

De cet invariant, la séquence `B` doit avoir la structure suivante : zéro ou plusieurs personnes honnêtes, suivies de zéro ou plusieurs personnes malhonnêtes.

Deuxième affirmation : L'algorithme préserve la propriété selon laquelle l'ensemble des personnes encore en considération (celles dans `B` plus celles pas encore traitées) contient strictement plus de personnes honnêtes que de malhonnêtes.

Preuve : Considérons une étape où nous traitons `A[i]` et `B[-1]` est le dernier élément de `B` (si `B` non vide).

- Si `A[i]` déclare que `B[-1]` est malhonnête, nous retirons `B[-1]` de `B` et nous écartons `A[i]`. Dans ce cas, au moins l'une des deux personnes `A[i]` ou `B[-1]` doit être malhonnête (car si les deux étaient honnêtes, `A[i]` aurait dit la vérité). Ainsi, nous retirons au plus une personne honnête, et au moins une personne malhonnête. La propriété est préservée.
- Si `A[i]` déclare que `B[-1]` est honnête, nous ajoutons `A[i]` à `B`. La propriété est trivialement préservée.
- Si `B` est vide, nous ajoutons `A[i]` à `B` sans poser de question. La propriété est également préservée.

Par la deuxième affirmation, à la fin de l'algorithme (après avoir traité tous les éléments), l'ensemble `B` contient strictement plus de personnes honnêtes que de malhonnêtes. Donc `B` est non vide et contient au moins une personne honnête. Puisque `B` a la structure (honnêtes d'abord, puis malhonnêtes), le premier élément `B[0]` est nécessairement honnête.

L'algorithme pose au plus une question par personne `A[1]` à `A[n-1]` (aucune question n'est posée lorsque `B` est vide). Le nombre total de questions est donc 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 $n - 1$.

Solution (English). If n is odd, remove any one person p from A and put them aside, so our input can be considered as

$$\mathsf{A} \quad + \quad \underbrace{\{p\}}_{\text{exists only if } n \text{ is odd}}$$

As the number of honest persons in the input is more than the number of dishonest persons, we have three cases:

- if n is even, then there is no person p, and the number of honest persons in A is strictly more than the number of dishonest persons in A.
- If n is odd and p is honest, then the number of honest persons in A is at least the number of dishonest persons in A.
- If n is odd and p is dishonest, then the number of honest persons in A is least 2 more than the number if dishonest persons in A.

Consider the following procedure on A:

> Pair up persons in A into exactly $n/2$ groups of two persons each.
>
> For each group, ask the first person about the second person.
>
> If the answer is 'Dishonest', then throw away *both* persons of this group.
>
> **Let B be the persons remaining**

We have the following.

Claim 1: If a person p_1 says that a person p_2 is dishonest, at least one of p_1 or p_2 *must* be dishonest.

This means that if A has strictly more honest persons than dishonest persons, the same is true of B. Similarly, if A has at least as many honest persons as dishonest persons, the same is true of B.

B is divided into groups, where in each group, the first person said that the second person was honest. The groups in B can be of three types:

(a) groups in which both persons are honest,

(b) the groups in which both persons are dishonest, and

(c) the groups in which the first person is dishonest, and the second person is honest.

We have the following.

Claim 2: If B has strictly more honest persons than dishonest persons, then the number of groups in which both persons are honest is strictly greater than the number of groups in which both persons are dishonest.

Claim 3: If B has at least as many honest persons than dishonest persons, then the number of groups in which both persons are honest is at least the number of groups in which both persons are dishonest.

> Now, from each group in B, throw away the first person (i.e., the one who asked the question), and only keep the second person.
>
> **Let C be the persons still remaining.**

Using Claim 2 and Claim 3, we have the following property.

Claim 4: If there were at least as many honest persons as dishonest persons in B, then in C, there are at least as many honest persons as dishonest persons.

If there were at least 1 more honest persons as dishonest persons in B, then in C, there are at least 1 more honest persons as dishonest persons.

If there were at least 3 more honest persons as dishonest persons in B, then in C, there are at least 2 more honest persons as dishonest persons.

If n is even, then by Claim 4, the set C has more honest persons than dishonest persons. So now we can recursively find one honest person in C, where len(C) $\leq$ len(A)$/2$.

It remains to consider the case when n is odd, and so person p exists.

Case: there are least 2 more honest persons as dishonest persons in $A + \{p\}$. Then if p was honest, then B has at least 1 more honest persons as dishonest persons, and so by Claim 4, C contains more honest persons than dishonest persons. Whether we recurse on C or $C + \{p\}$, in either case we will recurse with more honest persons.

Similarly, if p was dishonest, then B has at least 3 more honest persons as dishonest persons, and so by Claim 4, C contains at least 2 more honest persons as dishonest persons. Whether we recurse on C or $C + \{p\}$, in either case we will recurse with more honest persons.

Case: there are exactly 1 more honest persons as dishonest persons in $A + \{p\}$, and p is honest. If C has strictly more honest persons, then the previous argument applies and we can recurse on either C or $C + \{p\}$.

The bad case is when C has an equal number of honest and dishonest persons. But this can only happen if in B, each group was made up of either honest-honest or dishonest-dishonest persons, and had an equal number, say t, of such groups. In this case, $|B| = 4 * t$.

> If we find that B is divisible by 4, we recurse on $C + \{p\}$.

Case: there are exactly 1 more honest persons as dishonest persons in $A + \{p\}$, and p is dishonest. If B has least 3 more honest persons as dishonest persons, then C has least 2 more honest as dishonest persons, and we can recurse on either C or $C + \{p\}$.

The bad case is when B has exactly 2 more honest persons as dishonest persons, and C has only 1 more honest persons as dishonest persons. But this can only happen if in B, each group was made up of honest-honest and dishonest-dishonest persons, and there were t groups of dishonest-dishonest persons, and $t + 1$ groups of honest-honest persons. In this case, $|B| = 4t + 2$.

> If we find that B is not divisible by 4, we recurse on C.

For the total number of questions, observe that each time we ask a question, we throw away one of the persons. So at the end, we ask at most $n - 1$ questions.

Solution (French). Si n est impair, on retire une personne quelconque p de A et on la met de côté, de sorte que notre entrée peut être considérée comme

$$\mathsf{A} \quad + \quad \underbrace{\{p\}}_{\text{existe seulement si } n \text{ est impair}}$$

Comme le nombre de personnes honnêtes dans l'entrée est supérieur au nombre de personnes malhonnêtes, nous avons trois cas :

- si n est pair, alors il n'y a pas de personne p, et le nombre de personnes honnêtes dans A est strictement supérieur au nombre de personnes malhonnêtes dans A.
- Si n est impair et p est honnête, alors le nombre de personnes honnêtes dans A est au moins égal au nombre de personnes malhonnêtes dans A.
- Si n est impair et p est malhonnête, alors le nombre de personnes honnêtes dans A est au moins 2 de plus que le nombre de personnes malhonnêtes dans A.

Considérons la procédure suivante sur A :

> Pairer les personnes dans A en exactement $n/2$ groupes de deux personnes chacun.
>
> Pour chaque groupe, demander à la première personne son avis sur la seconde.
>
> Si la réponse est « Malhonnête », alors éliminer *les deux* personnes de ce groupe.
>
> **Soit B l'ensemble des personnes restantes**

Nous avons ce qui suit.

Assertion 1 : Si une personne p_1 dit qu'une personne p_2 est malhonnête, alors au moins l'une de p_1 ou p_2 *doit* être malhonnête.

Cela signifie que si A a strictement plus de personnes honnêtes que de malhonnêtes, il en va de même pour B. De même, si A a au moins autant de personnes honnêtes que de malhonnêtes, il en va de même pour B.

B est divisé en groupes où, dans chaque groupe, la première personne a déclaré que la seconde était honnête. Les groupes dans B peuvent être de trois types :

(a) les groupes où les deux personnes sont honnêtes,
(b) les groupes où les deux personnes sont malhonnêtes, et
(c) les groupes où la première personne est malhonnête et la seconde est honnête.

Nous avons ce qui suit.

Assertion 2 : Si B a strictement plus de personnes honnêtes que de malhonnêtes, alors le nombre de groupes où les deux personnes sont honnêtes est strictement supérieur au nombre de groupes où les deux personnes sont malhonnêtes.

Assertion 3 : Si B a au moins autant de personnes honnêtes que de malhonnêtes, alors le nombre de groupes où les deux personnes sont honnêtes est au moins égal au nombre de groupes où les deux personnes sont malhonnêtes.

Maintenant, pour chaque groupe dans B, éliminer la première personne (celle qui a posé la question), et ne garder que la seconde.

Soit C l'ensemble des personnes restantes.

En utilisant les Assertions 2 et 3, nous obtenons la propriété suivante.

Assertion 4 : S'il y avait au moins autant de personnes honnêtes que de malhonnêtes dans B, alors dans C, il y a au moins autant de personnes honnêtes que de malhonnêtes.

S'il y avait au moins 1 personne honnête de plus que de malhonnêtes dans B, alors dans C, il y a au moins 1 personne honnête de plus que de malhonnêtes.

S'il y avait au moins 3 personnes honnêtes de plus que de malhonnêtes dans B, alors dans C, il y a au moins 2 personnes honnêtes de plus que de malhonnêtes.

Si n est pair, alors par l'Assertion 4, l'ensemble C a plus de personnes honnêtes que de malhonnêtes. On peut donc trouver récursivement une personne honnête dans C, où len(C) $\leq$ len(A)$/2$.

Il reste à considérer le cas où n est impair, et donc la personne p existe.

Cas : il y a au moins 2 personnes honnêtes de plus que de malhonnêtes dans $A + \{p\}$. Alors si p était honnête, B a au moins 1 personne honnête de plus que de malhonnêtes, et donc par l'Assertion 4, C contient plus de personnes honnêtes que de malhonnêtes. Que l'on récurse sur C ou sur $C + \{p\}$, dans les deux cas on récurse avec plus de personnes honnêtes.
De même, si p était malhonnête, alors B a au moins 3 personnes honnêtes de plus que de malhonnêtes, et donc par l'Assertion 4, C contient au moins 2 personnes honnêtes de plus que de malhonnêtes. Que l'on récurse sur C ou sur $C + \{p\}$, dans les deux cas on récurse avec plus de personnes honnêtes.

Cas : il y a exactement 1 personne honnête de plus que de malhonnêtes dans $A + \{p\}$, et p est honnête. Si C a strictement plus de personnes honnêtes, alors l'argument précédent s'applique et on peut récurser sur C ou sur $C + \{p\}$.
Le mauvais cas est celui où C a un nombre égal de personnes honnêtes et de malhonnêtes. Mais cela ne peut se produire que si dans B, chaque groupe était composé soit de deux personnes honnêtes, soit de deux personnes malhonnêtes, et qu'il y avait un nombre égal, disons t, de tels groupes. Dans ce cas, $|B| = 4t$.

Si nous constatons que B est divisible par 4, nous récursons sur $C + \{p\}$.

Cas : il y a exactement 1 personne honnête de plus que de malhonnêtes dans $A + \{p\}$, et p est malhonnête. Si B a au moins 3 personnes honnêtes de plus que de malhonnêtes, alors C a au moins 2 personnes honnêtes de plus que de malhonnêtes, et nous pouvons récurser sur C ou sur $C + \{p\}$.
Le mauvais cas est celui où B a exactement 2 personnes honnêtes de plus que de malhonnêtes, et C n'a que 1 personne honnête de plus que de malhonnêtes. Mais cela ne peut se produire que si dans B, chaque groupe était composé de personnes honnêtes ou malhonnêtes, et qu'il y avait t groupes de deux personnes malhonnêtes, et $t + 1$ groupes de deux personnes honnêtes. Dans ce cas, $|B| = 4t + 2$.

Si nous constatons que B n'est pas divisible par 4, nous récursons sur C.

Pour le nombre total de questions, observons qu'à chaque question posée, nous éliminons au moins une personne (soit la première seulement quand la réponse est "honnête", soit les deux quand la réponse est "malhonnête"). Comme nous commençons avec n personnes et terminons avec 1 personne (le résultat), nous posons au plus $n - 1$ questions.

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]`.

Solution.

```
def merge(A, B):
    C = []
    a, b = 0, 0
    for i in range(0, len(A)+len(B)):
        if (b == len(B)) or (a < len(A) and A[a] <= B[b]):
            C.append(A[a])
            a += 1
        elif (a == len(A)) or (b < len(B) and B[b] <= A[a]):
            C.append(B[b])
            b += 1
    return C

print( merge( [1,4,30], [0,3,10,20,50] ) )
```

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.

Solution.

```
def merge(A, B):
    C = []
    a, b = 0, 0
    for i in range(0, len(A)+len(B)):
        if (b == len(B)) or (a < len(A) and A[a] <= B[b]):
            C.append(A[a])
            a += 1
        elif (a == len(A)) or (b < len(B) and B[b] <= A[a]):
            C.append(B[b])
            b += 1
    return C

def mergeSort_recursive(A):
    if len(A) == 1: return list(A)

    m = len(A) // 2
    G = mergeSort_recursive(A[:m])
    D = mergeSort_recursive(A[m:])
    return merge(G, D)

print( mergeSort_recursive([4,1,66,23,131,6]) )
```

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.

Solution.

```
def merge(A, B):
    C = []
    a, b = 0, 0
    for i in range(0, len(A)+len(B)):
        if (b == len(B)) or (a < len(A) and A[a] <= B[b]):
            C.append(A[a])
            a += 1
        elif (a == len(A)) or (b < len(B) and B[b] <= A[a]):
            C.append(B[b])
            b += 1
    return C

def mergeSort(A):
    s = 1
    while s < len(A):
        i = 0
        while (i < len(A)):
            A[i:i+2*s] = merge(A[i:i+s], A[i+s:i+2*s])
            i += 2*s
        s = 2*s

B = [9, 1, 53, 81, 16, 51, 12, 9]
mergeSort(B)
print(B)
```

L'algorithme effectue $\log_2 n$ passes. À chaque passe, il parcourt toute la liste et fusionne des sous-listes adjacentes. Chaque élément est copié une seule fois par passe. Le travail total par passe est donc $O(n)$. Comme il y a $\log_2 n$ passes, le temps d'exécution total est $O(n \log n)$.

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).

Solution.

```
def quickSort_recursive(A):
    if len(A) <= 1:
        return list(A)

    pivot = A[0]

    M = [ v for v in A if v == pivot ]
    G_input = [ v for v in A if v < pivot ]
    D_input = [ v for v in A if v > pivot ]

    G_output = quickSort_recursive(G_input)
    D_output = quickSort_recursive(D_input)

    return G_output + M + D_output
```

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.

   Solution.

```
def findBestStrategy_slow(i, j): #i, i+1, ..., j
    if i == j:
        return (i, 0)

    c1 = i + findBestStrategy_slow(i+1, j)[1]
    c2 = j + findBestStrategy_slow(i, j-1)[1]

    if c1 < c2:
        best_choice, best_choice_cost = i, c1
    else:
        best_choice, best_choice_cost = j, c2

    for k in range(i+1, j):
        l = findBestStrategy_slow(i, k-1)[1]
        r = findBestStrategy_slow(k+1, j)[1]

        c = k + max(l, r)
        if c < best_choice_cost:
            best_choice_cost = c
            best_choice = k

    return (best_choice, best_choice_cost)

print(findBestStrategy_slow(1,3))
```