Programmation 2 Spring, 2026

TD 10: Divide-and-Conquer Algorithms 1

1. (**TP question**) Pour le problème des personnes honnêtes/malhonnêtes sur une liste `A`, supposons que vous avez 4 personnes, parmi lesquelles une seule est malhonnête.

   Pouvez-vous trouver cette personne avec une seule question ?

2. (**Revision**) Donnez la définition formelle de $O(\cdot)$.

   Donnez une définition de $\Theta(\cdot)$ qui n'utilise que la notation $O(\cdot)$.

3. Pour chacune des paires suivantes, dites si $f(n) = O(g(n))$, $g(n) = O(f(n))$, ou les deux (Θ) :

   (a) $f(n) = n^2$, $g(n) = n^2 + n$

   (b) $f(n) = \log n$, $g(n) = \sqrt{n}$

   (c) $f(n) = 2^n$, $g(n) = 3^n$

4. Montrez que $f(n) = n \log n$ n'est pas $O(n)$.

5. Ordonnez les fonctions suivantes par ordre de croissance asymptotique (du plus petit au plus grand) :

$$\log n, \quad n, \quad n \log n, \quad n^2, \quad 2^n, \quad n!, \quad \sqrt{n}, \quad \frac{n}{\log n}$$

6. Déterminez si les affirmations suivantes sont vraies ou fausses. Justifiez brièvement.

   (a) $n^2 = O(n^3)$

   (b) $n^3 = O(n^2)$

   (c) $n \log n = \Theta(n \log(n^2))$

   (d) $2^{n+1} = O(2^n)$

   (e) $2^{2n} = O(2^n)$

7. Trouvez une fonction $f(n)$ telle que $f(n) = O(n)$ mais $f(n) \neq \Theta(n)$.

8. Soient $f(n)$ et $g(n)$ deux fonctions strictement positives. Est-il vrai que $f(n) + g(n) = \Theta(\max(f(n), g(n)))$? Justifiez.

9. Je pense à un nombre compris entre 1 et 100. Vous devez le deviner. Si vous vous trompez, je vous dirai si le nombre est supérieur ou inférieur à votre estimation.

   Voici les règles:

   - si vous devinez le nombre auquel je pense en un essai, vous gagnerez 5 euros.
   - si vous le devinez en deux essais, vous gagnerez 4 euros.
   - si vous gagnez en trois essais, vous gagnerez 3 euros.
   - si vous gagnez en quatre essais, vous gagnerez 2 euros.
   - si vous gagnez en cinq essais, vous gagnerez 1 euros.
   - si vous gagnez en six essais, vous gagnerez 0 euro.

- si vous gagnez en sept essais, **vous me donnez** 1 euros.

Devez-vous jouer à ce jeu ?

10. (**Revision**) On vous donne une liste `A` contenant n nombres distincts, déjà triés par ordre croissant.

    Vous devez écrire deux fonctions :

    — Écrivez une fonction `binarySearch_ITERATIVE` qui prend `A` et un nombre x. Elle renvoie l'indice de x dans `A` si x est présent, et `None` sinon.

      Votre algorithme doit utiliser des boucles.

    — Écrivez une fonction `binarySearch_RECURSIVE` qui prend `A`, un nombre x, ainsi que deux indices `start` et `end` délimitant la portion de la liste à explorer. Elle renvoie l'indice de x dans `A` si x est présent, et `None` sinon.

      Votre algorithme doit utiliser la récursion.

    Les deux algorithmes doivent s'exécuter en temps $O(\log n)$, où $n = \text{len(A)}$.

    Exemples :

    - `binarySearch_ITERATIVE([2, 5, 7, 9, 12], 7)` renvoie 2.
    - `binarySearch_RECURSIVE([2, 5, 7, 9, 12], 7, 0, 4)` renvoie 2.

11. On vous donne une liste `A` de valeurs **True**/**False**, telle que toutes les valeurs **False** apparaissent *avant* toutes les valeurs **True** (c'est-à-dire, la liste est de la forme `[False, ..., False, True, ..., True]`).

    Vous devez écrire deux fonctions qui trouvent le *premier* index de **True** dans un intervalle donné :

    — Écrivez une fonction `findFirstTrue_ITERATIVE` qui prend `A`, ainsi que deux indices `start` et `end`. Elle renvoie le premier index entre `start` et `end` (inclus) ayant la valeur **True**, ou `None` s'il n'existe pas.

      Votre algorithme doit utiliser des boucles.

    — Écrivez une fonction `findFirstTrue_RECURSIVE` qui prend `A`, ainsi que deux indices `start` et `end`. Elle renvoie le premier index entre `start` et `end` (inclus) ayant la valeur **True**, ou `None` s'il n'existe pas.

      Votre algorithme doit utiliser la récursion.

    Les deux algorithmes doivent s'exécuter en temps $O(\log n)$, où $n = \text{len(A)}$.

    Exemples :

    - `findFirstTrue_ITERATIVE([False, False, False, True, True], 0, 4)` renvoie 3.
    - `findFirstTrue_RECURSIVE([False, False, False], 0, 2)` renvoie `None`.

12. On vous donne une liste `A` de nombres, tels que `A[0] < A[len(A)-1]`. Notez que `A` *n'est pas trié*.

    Écrivez une fonction itérative `findIncreasing_ITERATIVE` qui prend `A` et renvoie un indice `i` tel que `A[i] < A[i+1]`.

    Votre algorithme doit prendre le temps $O(\log n)$, où `n = len(A)`.

Exemples :

- `findIncreasing_ITERATIVE([1, 3, 2, 4, 5])` pourrait renvoyer 0 (car `A[0] = 1 < 3 = A[1]`), ou 2 (car `A[2] = 2 < 4 = A[3]`), ou 3 (car `A[3] = 4 < 5 = A[4]`). N'importe lequel est accepté.
- `findIncreasing_ITERATIVE([5, 4, 3, 2, 1, 6])` renvoie 4 (car `A[4] = 1 < 6 = A[5]`). Notez que `A[0] = 5 < 6 = A[-1]` est bien vérifié.
- `findIncreasing_ITERATIVE([10, 8, 6, 4, 2, 12])` renvoie 4 (car `A[4] = 2 < 12 = A[5]`).

Indice : Utilisez une recherche dichotomique (binaire). Observez que :

- Si `A[mid] < A[mid+1]`, alors `mid` est une solution.
- Sinon, une paire croissante doit exister soit à gauche, soit à droite. Utilisez cette propriété pour réduire l'intervalle de recherche.

13. On vous donne une liste `A` de nombres, tels que `A[0] < A[len(A)-1]`. Notez que `A` *n'est pas triée*.

    Écrivez une fonction récursive `findIncreasing_RECURSIVE` qui prend `A`, ainsi que deux indices `start` et `end` (inclus), et renvoie un indice `i` (avec $\texttt{start} \leq \texttt{i} < \texttt{end}$) tel que `A[i] < A[i+1]`.

    Votre algorithme doit s'exécuter en temps $O(\log n)$, où $n = \text{len(A)}$.

    Exemples :

    - `findIncreasing_RECURSIVE([10, 8, 6, 4, 2, 12], 0, 5)` renvoie 4 (car `A[4] = 2 < 12 = A[5]`).
    - `findIncreasing_RECURSIVE([3, 1, 2, 5, 4], 0, 4)` pourrait renvoyer 1 (car `A[1] = 1 < 2 = A[2]`) ou 2 (car `A[2] = 2 < 5 = A[3]`).
    - `findIncreasing_RECURSIVE([60, 50, 40, 23, 28, 70], 0, 5)` renvoie 3.

14. Écrivez une fonction itérative `findMax_ITERATIVE` qui prend une liste `A` des entiers distinctes et renvoie le plus grand entier dans `A`.

    La liste `A` a une forme particulière :

    > dans l'ordre des indices, les nombres de `A` **augmentent strictement** jusqu'à un certain point, atteignent un **maximum unique**, puis **diminuent strictement**.

    Autrement dit, il existe un indice `p` (le pic) tel que :

    - `A[0] < A[1] < ... < A[p]` (partie croissante),
    - `A[p] > A[p+1] > ... > A[n-1]` (partie décroissante).

    Votre algorithme doit s'exécuter en temps $O(\log n)$, où `n = len(A)`.

    Exemples :

    - `findMax_ITERATIVE([1, 6, 9, 8, 2])` renvoie 9.
    - `findMax_ITERATIVE([3, 5, 7, 10, 6, 4])` renvoie 10.
    - `findMax_ITERATIVE([2, 4, 6, 8, 10])` renvoie 10 (liste strictement croissante, le maximum est à la fin).
    - `findMax_ITERATIVE([10, 8, 6, 4, 2])` renvoie 10 (liste strictement décroissante, le maximum est au début).
    - `findMax_ITERATIVE([5])` renvoie 5 (liste d'un seul élément).

★ Puzzle ★

Le Défi des Échançons Royaux

Un roi maléfique possède n bouteilles de vin.
Un espion insaisissable en a empoisonné ***exactement une****.*

†

Le poison est d'une efficacité redoutable : une seule goutte diluée au milliardième suffit à tuer. Mais il lui faut un mois entier pour faire effet.

†

Le roi a à son service des testeurs (des goûteurs) chargés de vérifier si sa nourriture ou son vin est empoisonné.

♦ VOTRE MISSION ♦

Le roi doit décider, avant le début du test, quel testeur boit quelles bouteilles. Une fois le plan établi, les testeurs boivent. Le roi attend un mois. Au bout de ce mois, il observe quels testeurs sont morts et lesquels sont encore vivants. À partir de ce seul résultat—un motif de morts et de survivants—le roi doit pouvoir calculer avec certitude quelle bouteille était empoisonnée.

Concevez un algorithme permettant d'identifier avec certitude la bouteille empoisonnée en un seul mois, en utilisant seulement $O(\log n)$ testeurs de goût. Chaque testeur peut boire un mélange de plusieurs bouteilles. *Vous pouvez supposer que n est une puissance de 2.*

⋆ EXEMPLE ⋆

Une stratégie naive utilise n testeurs :

> le i-ième testeur goûte uniquement la i-ième bouteille.

Un mois plus tard, le testeur décédé désigne la bouteille empoisonnée.

Votre défi : réduire le nombre de testeurs à $O(\log n)$.