Matthieu Boyer
**Planche A**



## Algorithmes classiques

On s'attachera plus à écrire du code correct qu'à l'efficacité du code. Avant d'implémenter une fonction, on donnera toujours le type attendu.

On s'intéresse à l'implémentation d'algorithmes classiques.

1. Implémentez une fonction qui, étant donnée une liste d'entiers, renvoie le minimum et le maximum de cette liste. Que renvoyer si la liste est vide ?
2. Implémentez une fonction qui, étant donnée une liste, renvoie une liste contenant les mêmes éléments triés par ordre croissant. On utilisera l'algorithme du tri par insertion.
3. Implémentez une fonction qui, étant donnée une liste triée et un élément, insère cet élément à la bonne position dans la liste triée.
4. Implémentez le tri par fusion (merge sort) : étant donnée une liste, la trier en la divisant récursivement en deux parties, en triant chaque partie, puis en fusionnant les résultats.
5. Implémentez une fonction qui, étant donnée une liste, calcule toutes ses sous-listes (ensemble des parties). Par exemple, pour `[1; 2]`, on obtient `[[];[1];[2];[1;2]]`.
6. Implémentez une fonction qui, étant donnée une liste, génère toutes ses permutations.
7. Implémentez une fonction qui, étant donnés une liste et un entier $k$, génère toutes les combinaisons de $k$ éléments parmi cette liste.
8. Bonus : Implémentez une fonction qui, étant donnée une liste d'entiers et une somme cible, détermine s'il existe un sous-ensemble de la liste dont la somme vaut exactement la cible.

## Manipulation de chaînes de caractères

On s'attachera plus à écrire du code correct qu'à l'efficacité du code. Avant d'implémenter une fonction, on donnera toujours le type attendu.

On travaille avec des chaînes de caractères représentées comme des listes de caractères : `type chaine = char list;` . On essaiera au maximum de réutiliser les fonctions précédentes, pour avoir un code le plus propre possible.

1. Implémentez une fonction qui, étant donnée une chaîne, renvoie sa longueur.
2. Implémentez une fonction qui, étant données deux chaînes, les concatène.
3. Implémentez une fonction qui, étant donnée une chaîne, renvoie la chaîne inversée.
4. Implémentez une fonction qui, étant données deux chaînes, vérifie si la première est un préfixe de la seconde.
5. Implémentez une fonction qui, étant donnés une chaîne et un caractère, compte le nombre d'occurrences de ce caractère dans la chaîne.
6. Implémentez une fonction qui, étant données une chaîne et un caractère séparateur, découpe la chaîne en une liste de chaînes selon le séparateur.
7. Implémentez une fonction qui, étant données une liste de chaînes et un caractère séparateur, les joint en une seule chaîne avec le séparateur entre chaque élément.
8. Bonus : Implémentez une fonction qui, étant données deux chaînes, vérifie si la première apparaît comme sous-chaîne dans la seconde.
9. Très Bonus : Implémentez efficacement une fonction qui, étant données deux chaînes $s$ et $t$, calcule leur distance de Levenshtein (nombre minimal d'opérations d'insertion, suppression ou substitution pour transformer $s$ en $t$).

## Le type option et gestion d'absence

On s'attachera plus à écrire du code correct qu'à l'efficacité du code. Avant d'implémenter une fonction, on donnera toujours le type attendu.

Le type `type 'a option = None | Some of 'a;` permet de représenter une valeur potentiellement absente.

1. Implémentez une fonction qui, étant donnée une `int list` et un indice, renvoie l'élément à cet indice sous forme d'`int option` (`None` si l'indice est invalide).
2. Implémentez une fonction qui, étant donné un `'a option` et une valeur par défaut, renvoie la valeur contenue dans le `Some` ou la valeur par défaut si c'est `None`.
3. Implémentez une fonction `map_option` qui, étant donnés une fonction `'a -> 'b` et un `'a option`, applique la fonction au contenu s'il existe. L'existence d'une telle fonction dit que le constructeur `option` est fonctoriel.
4. Implémentez une fonction qui, étant donnée une liste d'`'a option`, filtre et extrait toutes les valeurs `Some`, renvoyant une `'a list`.
5. Implémentez une fonction qui, étant donnés deux `'a option`, renvoie le premier qui contient une valeur, ou `None` si les deux sont `None`.
6. Implémentez une fonction qui, étant donnée une liste et un prédicat, renvoie le premier élément satisfaisant le prédicat sous forme d'`'a option`.
7. Implémentez une fonction qui, étant donnée une fonction `'a -> 'b option` et une `'a list`, applique la fonction à chaque élément et renvoie la liste des résultats réussis.
8. Bonus : Implémentez une fonction `bind_option` qui, étant donnés un `'a option` et une fonction `'a -> 'b option`, applique la fonction si la valeur existe. Quelle différence avec `map_option` ? On dit que le constructeur `option` est monadique.
9. Implémentez une fonction qui, étant donnée une liste d'opérations pouvant échouer (fonctions `'a -> 'a option`), les applique séquentiellement à une valeur initiale, s'arrêtant à la première qui échoue.

## Matrices et tableaux bidimensionnels

On s'attachera plus à écrire du code correct qu'à l'efficacité du code. Ainsi, on n'essaiera pas d'être *le plus efficace possible* mais on prendra tout de même garde aux opérations bien trop répétées. Avant d'implémenter une fonction, on donnera toujours le type attendu.

On représente une matrice comme une liste de listes : `type 'a matrice = 'a list list;` . Dans la suite on notera $M_{i,j}$ la valeur sur la $i$-ème ligne et la $j$-ème colonne de la matrice. On suppose que toutes les lignes ont la même longueur sans se préoccuper de le vérifier.

1. Implémentez une fonction qui, étant donnée une matrice, renvoie ses dimensions (nombre de lignes et nombre de colonnes).
2. Implémentez une fonction qui, étant données des dimensions $n$ et $m$ et une valeur par défaut, crée une matrice $n \times m$ remplie de cette valeur (c'est à dire tel que $M_{i,j} = x$ pour tous $i, j$). On pourra utiliser une fonction auxiliaire qui crée une liste de $n$ fois l'élément $x$.
3. Implémentez une fonction qui, étant donnés des indices $i$ et $j$ et une matrice, renvoie l'élément en position $(i, j)$, ou `None` si les indices sont hors limites. On prendra bien garde à la définition du type option.
4. Implémentez une fonction qui, étant donnée une matrice carrée d'entiers, calcule sa trace (somme des éléments diagonaux).
5. Implémentez une fonction qui, étant donnée une matrice, renvoie sa transposée, c'est à dire la matrice ${}^{\mathrm{t}}M$ telle que ${}^{\mathrm{t}}M_{i,j} = M_{j,i}$. On prendra bien garde à ne parcourir la matrice de départ qu'un nombre de fois indépendant de ses dimensions.
6. Implémentez une fonction qui, étant donnée une matrice, vérifie si elle est symétrique.
7. Implémentez une fonction qui, étant données deux matrices de flottants de dimensions compatibles, effectue leur addition. L'addition s'effectue point à point, et on s'attachera à vérifier les dimensions des deux matrices. On prendra bien garde à ne parcourir les matrices de départ qu'un nombre de fois indépendant de leurs dimensions.
8. Bonus : Implémentez une fonction qui, étant données deux matrices de dimensions compatibles, effectue leur multiplication matricielle, si vous savez faire.
9. Très Bonus : Implémentez une fonction qui, étant donnée une matrice carrée, calcule son déterminant par développement récursif selon la première ligne.