Algorithmique avancée
TD n° 1

1. Montrer que l'on peut implémenter une file avec deux piles : écrire les opérations spécifiées par la définition de la file en utilisant uniquement les opérations abstraites sur les deux piles. Quel est le coût d'une opération élémentaire sur la file (c'est à dire le nombre d'opérations élémentaires sur la pile utilisées pour faire une opération sur la file) ?
2. Montrer que l'on peut implémenter une pile avec deux files : écrire les opérations spécifiées par la définition de la pile en utilisant uniquement les opérations abstraites sur les deux files. Quel est le coût d'une opération élémentaire sur la pile (c'est à dire le nombre d'opérations élémentaires sur la file utilisées pour faire une opération sur la pile) ?
3. Le tri *par insertion* traite les éléments à trier un par un dans l'ordre où ils se présentent, sans savoir ce qui reste : il conserve les éléments déjà reçus dans une structure de données où ils sont dans l'ordre, et chaque élément traité est inséré à sa place dans cette structure de donnée. Montrer que si lon a un bouchon (un élément qui ne peut pas faire partie des éléments à trier) l'on peut utiliser une file pour faire un tri par insertion.
4. Le tri par sélection du minimum consulte à chaque étape tous les éléments qui restent à trier : il conserve les éléments déjà choisis dans une structure de données où ils sont dans l'ordre, et à chaque étape il choisit le plus petit de ceux qui restent et le place à la fin de la structure de données. Montrer que si les éléments à trier sont sur une pile, on peut les trier en utilisant deux autres piles.
5. Écrire un algorithme *fusion(liste1,liste2*) de deux listes triées (fusion sans répétition : un élément ne figure pas plus d'une fois dans la liste résultat). Les listes en entrée ne sont pas modifiées par l'algorithme, qui renvoie une nouvelle liste.
6. Écrire un algorithme *purger(liste1)* qui supprime toutes les répétitions dans une liste. La liste en entrée n'est pas modifiée par l'algorithme, qui renvoie une nouvelle liste.
7. Montrer que l'on peut implémenter une file ou une pile à l'aide d'une liste chaînée (donner les opérations abstraites en fonction de celles de la SDA *liste chaînée*. On peut utiliser des variables auxiliaires pour être plus efficace)
8. Écrire un algorithme *retourne(@liste1)* qui renverse (physiquement) une liste chaînée. *liste1* est l'adresse de la tête de liste. L'algorithme renvoie la nouvelle tête et laisse la liste modifiée
9. Donner les algorithmes des opérations *insérerAprès()* et *supprimer()* dans une liste doublement chaînée.