TP Algorithmique Avancée
Evaluation de performance

1. Ecrire un programme C dont la fonction main permet de générer aléatoirement les éléments d'un tableau. Le programme :
    a. Demande dans un premier temps la taille du tableau
    b. Réserve la zone mémoire correspondant à cette taille (fonction ***malloc***)
    c. Il génèrera de manière aléatoire les éléments du tableau
        i. Utiliser les fonctions ***srand*** et ***rand*** de la bibliothèque stdlib
            1. Initialiser le générateur aléatoire par l'instruction : ***srand(time(NULL))***
            2. Pour générer un entier aléatoirement, utiliser l'instruction ***rand()***
2. Ecrire une fonction **afficher** qui prend en paramètre ce tableau ainsi que sa taille pour afficher son contenu
    a. Signature : **void afficher(int * tab, int taille)**
3. Tester le programme en générant un tableau avec une taille saisie par l'utilisateur et en affichant son contenu
4. Ecrire une fonction triParSelection qui prend en paramètre un tableau d'entiers ainsi que sa taille et retourne un tableau de la même taille avec ses éléments triés du plus petit au plus grand
    a. Signature : **int * triParSelection(int * tab, int taille)**
5. Tester le programme en générant un tableau puis en l'affichant avant tri puis en affichant le tableau résultant de l'appel de la fonction triParSelection
6. On souhaite actuellement comptabiliser le temps nécessaire pour effectuer le tri. Nous allons pour cela utiliser la fonction **clock()** de la librairie **time.h**. Placer un appel avant et après l'appel à la fonction triParSelection et puis afficher le temps calculé.
    a. La fonction **clock** renvoie le temps CPU en millisecondes.
    b. Utiliser le type double pour mesurer le temps d'exécution
7. Introduire maintenant les fonctions relatives au tri par fusion. La fonction triParFusion prend en paramètre le tableau à trier ainsi que sa taille et renvoie un tableau contenant les éléments de ce tableau dans l'ordre croissant.
    a. void fusionner(int * tab, int p, int q, int r)
    b. void tri-fusion(int * tab, int p, int r)
    c. int * triParFusion(int * tab, int taille)

8. En séquence de l'appel à la fonction triParSelection, introduire un appel à la fonction triParFusion.
    a. Vérifier que les deux algorithmes renvoient le même résultat (utiliser des tableaux de petites tailles)
9. Afficher les temps d'exécution pour les deux algorithmes. Utiliser des tailles assez grandes (de l'ordre de 100 000) et conclure.
10. Le tri à bulles fonctionne en se basant sur une propriété nécessaire et suffisante P des tableaux triés : dans un tableau trié, chaque élément est plus petit que l'élément qui le suit. L'approche de l'algorithme est que tant que la propriété P est non vérifiée
    a. Parcourir le tableau du début jusqu'à la fin,
        i. Pour chaque i tel que tab[i] > tab[i+1], échanger les deux valeurs
    b. Dès que P est vérifiée sur un parcours complet, on peut déduire que le tableau est trié.
11. Introduire la fonction int* Tri_Bulles(int * tab, int taille)
12. En séquence de l'appel aux fonctions Tri_Selection et triParFusion, introduire un appel à la fonction Tri_Bulles.
    a. Vérifier que les trois algorithmes renvoient le même résultat (utiliser des tableaux de petites tailles)
    b. Afficher les temps d'exécution pour les trois algorithmes. Utiliser des tailles assez grandes (de l'ordre de 100 000) et conclure.