TP 2

1. Introduction aux tableaux

Exercice 1. Ecrire et tester les fonctions suivantes :

1. `void affiche_tableau(int* tab,int n)` prenant en paramètre un tableau `tab` de longueur `n` et affichant son contenu (ex. `[1,-2,4]`).
2. `int min(int* tab,int n)` retournant le plus petit élément du tableau `tab` (de taille `n`).
3. `int max(int* tab,int n)` retournant le plus petit élément du tableau `tab` (de taille `n`).
4. `int somme(int* tab,int n)` retournant la somme des éléments du tableau `tab` (de taille `n`).

Exercice 2. Ecrire et tester les fonctions suivantes :

1. `bool est_pair(int* tab,int n)` testant si le tableau `tab` contient que des entiers pairs.
2. `bool est_trie(int* tab,int n)` testant si le tableau `tab` est trié dans l'ordre croissant.
3. `int* copie(int* tab,int n)` qui retourne une copie du tableau `tab`.

2. Algorithmes de tri

Les différentes méthodes de tri pourront être testées sur les tableaux `tab10`, `tab20`, `tab100` et `tab1000` fournis et de tailles respectives 10, 20, 100 et 1000.

Exercice 3.

1. Ecrire une fonction `void tri_insertion(int* tab,int n)` triant le tableau `tab` à l'aide de l'algorithme du tri par insertion.
2. Modifier votre fonction en `void tri_insertion_v2(int* tab,int n)` afin de connaître le nombre de comparaisons d'entiers utilisées ainsi que le nombre de lectures/écritures faites sur le tableau.
3. Noter les valeurs obtenues dans le tableau suivant :

insertion	tab10	tab20	tab100	tab1000
comparaisons				
lectures/écriture				

Exercice 4.

1. Ecrire une fonction `void tri_selection(int* tab,int n)` triant le tableau `tab` à l'aide de l'algorithme du tri par sélection.
2. Modifier votre fonction en `void tri_selection_v2(int* tab,int n)` afin de connaître le nombre de comparaisons d'entiers utilisées ainsi que le nombre de lectures/écritures faites sur le tableau.
3. Noter les valeurs obtenues dans le tableau suivant :

selection	tab10	tab20	tab100	tab1000
comparaisons				
lectures/écritures				

Exercice 5.

1. Ecrire une fonction `void tri_fusion(int* tab,int n)` triant le tableau `t` à l'aide de l'algorithme du tri fusion.
2. Modifier votre fonction en `void tri_fusion_v2(int* tab,int n)` afin de connaître le nombre de comparaisons d'entiers utilisées ainsi que le nombre de lectures/écritures faites sur le tableau.
3. Noter les valeurs obtenues dans le tableau suivant :

fusion	tab10	tab20	tab100	tab1000
comparaisons				
lectures/écritures				

Exercice 6. Comparer les nombres de comparaisons et de lectures/écritures obtenus aux exercices précédents.

3. Recherche par dichotomie

Exercice 7. La recherche par dichotomie consiste à rechercher la position d'un élément dans un tableau trié. Le principe est le suivant : on compare l'élément avec la valeur se trouvant au milieu du tableau. Si les valeurs sont égales, on a fini. Sinon, on recherche par dichotomie l'élément dans la moitié pertinente du tableau.

1. Ecrire une fonction `int recherche(int* tab,int n,int e)` qui retourne la position de `e` dans le tableau trié `tab` de taille `l` si `e` est présent et `-1` sinon.
2. Combien de comparaisons devrons nous faire dans le pire des cas pour rechercher un élément dans un tableau trié de taille n. Et si le tableau n'est pas trié ?