Algorithmique et Structures de données

TD n° 3 : Complexité des algorithmes, algorithmes de tri, recherche par dichotomie

Exercice 1 – *Tableau trié ?*

1. Réaliser une fonction qui teste que le tableau passé en paramètre est trié. Combien de comparaisons d'éléments du tableau sont-elles effectuées ?
2. Réaliser une fonction qui prend deux tableaux en paramètres et qui teste que le second tableau est une version triée du premier. Déterminer la complexité de l'algorithme en nombre de comparaisons.

Exercice 2 – *Tri par insertion (séquentielle)*

L'algorithme de tri par insertion présenté dans le cours insère un élément dans une tranche trié $t[a, b]$ par une succession d'échanges, chaque échange se faisant en trois affectations. Si la tranche a k éléments, l'insertion peut « coûter » jusqu'à $3k$ affectations d'éléments. Concevoir un algorithme avec lequel le nombre maximal d'affectations est $k + 2$.

Exercice 3 – *Recherche dichotomique*

Une recherche dichotomique consiste à profiter de l'ordre (croissant/décroissant) du tableau pour accélérer la recherche en procédant comme suit :

– On considère l'élément figurent au 'milieu' du tableau. Si l'élément cherché lui est égal, la recherche est terminée. S'il lui est inférieur, on en conclut que l'élément cherché ne peut se trouver que dans la première moitié du tableau ; dans le cas contraire, on en conclut qu'il se trouve dans la seconde moitié.
– On recommence alors l'opération sur la moitié concernée, puis sur la moitié de cette moitié, et ainsi de suite … jusqu'à ce que l'une des conditions suivantes soit satisfaire :
  - On a trouvé l'élément cherché,
  - On est sûr qu'il ne figure pas dans le tableau.

1. Implémenter (version itérative puis récursive) l'algorithme de recherche par dichotomie.
2. Déterminer la complexité de la recherche par dichotomie en nombre de comparaisons.
3. En utilisant l'algorithme de recherche dichotomique, proposer un algorithme qui demande mois de comparaisons d'éléments pour la deuxième question du premier l'exercice.

Algorithmique et Structures de données

Travail pratique n° 3 : Les tris

1. Réaliser une fonction pour le remplissage aléatoire d'un tableau d'entiers. Pour cela, utiliser la fonction `random()` définie dans `stdlib.h`.
   ```
   void rand_vector(int v[], int n)
   ```
2. Implémenter l'algorithme de tri Bulle, de manière itérative puis récursive.
   ```
   void bubble_sort(int v[], int n)
   void bubble_sort_rec(int v[], int n)
   ```
3. Implémenter l'algorithme de tri par fusion.
   ```
   void fusion(int v[], int l, int m, int r)
   void mergre_sort(int v[], int l, int r)
   ```
4. Compter le nombre de comparaisons d'éléments, et le nombre de déplacement d'éléments effectuées par chaque algorithme de tri; ajouter des instructions dans les sous-programmes de tris pour réaliser ces calculs.
5. Comparer les temps d'exécution de ces algorithmes de tri (utiliser en C la fonction `clock` définie dans `time.h`).
6. Appliquer les deux algorithmes de tri sur des tableaux (remplis aléatoirement) de grande taille ($n = 1000, 10000, 100000, 100000$).