

Algorithmes de Tri et Recherche Dichotomique

Contenu

I) Algorithmes de tri :	2
Introduction :	2
1) Tri à bulles (Tri par propagation) :	2
a. Principe :	2
b. Algorithme :	2
c. Trace :	3
d. Exécution :	3
2) Tri par insertion :	3
a. Principe :	3
b. Algorithme :	3
c. Trace :	4
d. Exécution :	4
II) Recherche dichotomique :	4
Introduction :	4
1) Principe :	4
2) Algorithme :	5
1) Trace :	5
2) Exécution :	6

I) Algorithmes de tri :

Introduction :

Un algorithme de tri est un algorithme qui permet d'organiser ou de classer un ensemble d'objets selon une relation d'ordre déterminée. Les objets à trier sont des éléments hiérarchiquement constitués. Par exemple les nombres entiers sont des éléments hiérarchiquement reliés par la relation d'ordre usuelle « est inférieur ou égal à », ils peuvent donc être triés suivant cet ordre.

L'ensemble à trier est souvent donnée sous forme de tableau ou de liste, afin de permettre un accès facile et direct aux différents éléments de l'ensemble, pour les adapter aux procédures d'analyse nécessaires aux algorithmes.

Généralement, les algorithmes de tri peuvent être définis indépendamment de l'ensemble auquel appartiennent les éléments à trier et de la relation d'ordre qui leur est associée. Par exemple, un même algorithme peut être à la fois utilisé pour trier des réels par ordre croissant ou décroissant, et des chaînes de caractères selon leur ordre lexicographique.

Les algorithmes de tri sont souvent étudiés dans les cours d'algorithmique pour introduire des notions comme la complexité algorithmique (Notion d'étude de l'espace et du temps consommé par les algorithmes) ou la terminaison (Propriété stipulant l'arrêt des calculs décrits par les algorithmes).

1) Tri à bulles (Tri par propagation) :

a. Principe :

Consiste à faire remonter progressivement les plus grands éléments d'un tableau, comme les bulles d'air remontent à la surface d'un liquide. L'algorithme parcourt le tableau, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.

b. Algorithme

```
// version non optimisée
debut
entier i, j, T[N], tmp
pour i=N-1 → i=1 (pas = -1) faire
    pour j=0 → j=i (pas = 1) faire
        si ( T[j] > T[j+1] ) alors
            tmp ← t[j]
            T[j] ← T[j+1]
            T[j+1] ← tmp // On échange les éléments de rang j et j+1
        fin si
    fin pour
fin pour
fin
```

L'algorithme ci-dessus n'est pas optimal, car on risque d'avoir des tours de boucles inutiles quand l'échange s'arrête, pour l'optimiser, la boucle doit s'arrêter après la fin de l'échange.

```
// version optimisée
debut
entier i, j, T[N], tmp
booléen échange
échange ← vrai
tant que ( i>0 ET échange ) faire
    échange ← faux
    pour j=1 → j=i-1 (pas = 1) faire
        si ( T[j] > T[j+1] ) alors
            tmp ← T[j]
            T[j] ← T[j+1]
            T[j+1] ← tmp
            échange ← vrai
        fin si
    fin pour
    i ← i-1
fin tant que
Fin
```

c. Trace :

Prenons la liste de chiffres « 5 1 4 2 8 » et trions-la de manière croissante en utilisant l'algorithme de tri à bulles. Pour chaque étape, les éléments comparés sont écrits en gras.

Première étape:

(**5** 1 4 2 8) → (**1** 5 4 2 8) Les éléments 5 et 1 sont comparés, et comme $5 > 1$, l'algorithme les intervertit.
 (**1** 5 4 2 8) → (1 **4** 5 2 8) Intersion car $5 > 4$.
 (1 4 **5** 2 8) → (1 4 **2** 5 8) Intersion car $5 > 2$.
 (1 4 2 **5** 8) → (1 4 2 **5** 8) Comme $5 < 8$, les éléments ne sont pas échangés.

Deuxième étape:

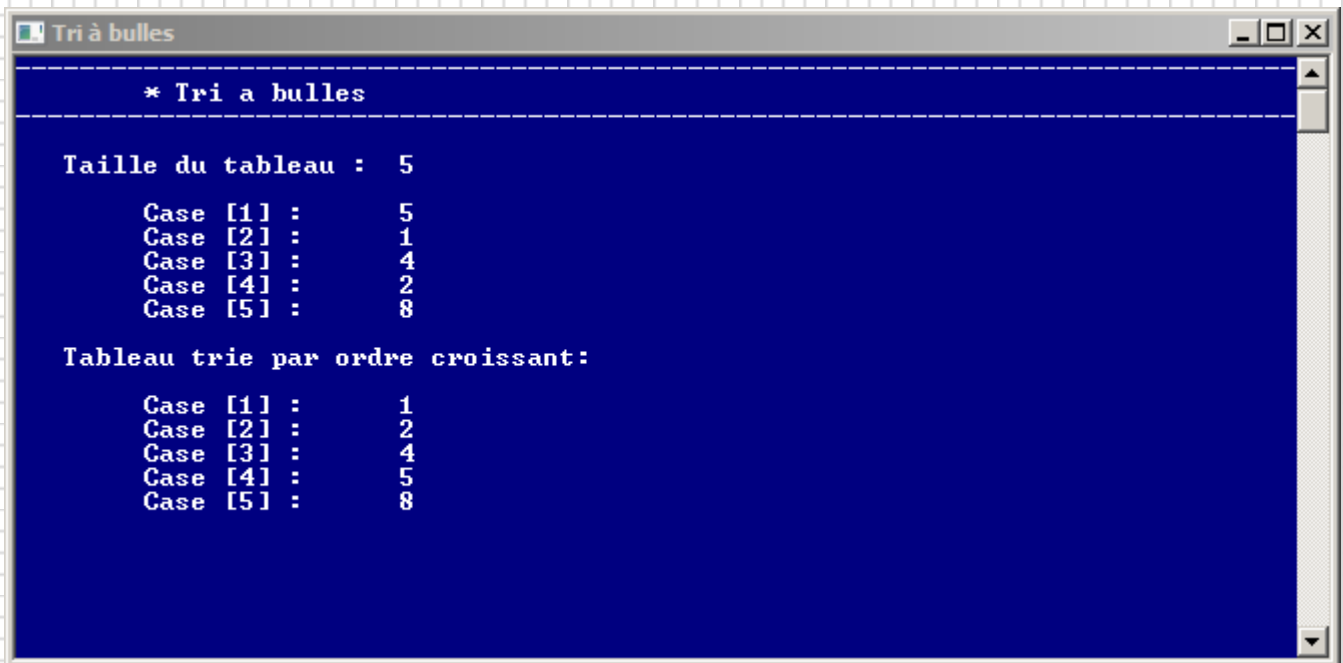
(**1** 4 2 5 8) → (**1** 4 2 5 8) Même principe qu'à l'étape 1.
 (**1** 4 2 5 8) → (1 **2** 4 5 8)
 (1 2 **4** 5 8) → (1 2 **4** 5 8)

À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.

Troisième étape:

(**1** 2 4 5 8) → (**1** 2 4 5 8)
 (**1** 2 4 5 8) → (**1** 2 4 5 8)

Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme.

d. Exécution :**2) Tri par insertion :****a. Principe :**

Consiste à parcourir le tableau à trier du début à la fin. Au moment où on considère un élément déterminé du tableau, les éléments qui précèdent cet élément sont déjà triés. L'objectif est d'insérer l'élément considéré à sa place parmi ceux qui le précèdent. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion. En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire remonter l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

b. Algorithme :

```

debut
entier i, j, T[N], x
  pour i=1 → i=n-1 (pas = 1) faire
    x ← T[i]
    j ← i
    tant que ( j>0 ET T[j-1]>x ) faire
      T[j] ← T[j-1]
      j ← j-1
    fin tant que

```

```

    T[j] ← x
  fin pour
fin

```

c. Trace :

Prenons la liste de chiffres « 9 6 1 4 8 » et trions-la de manière croissante en utilisant l'algorithme de tri par insertion. Pour chaque étape, les éléments considérés sont écrits en gras.

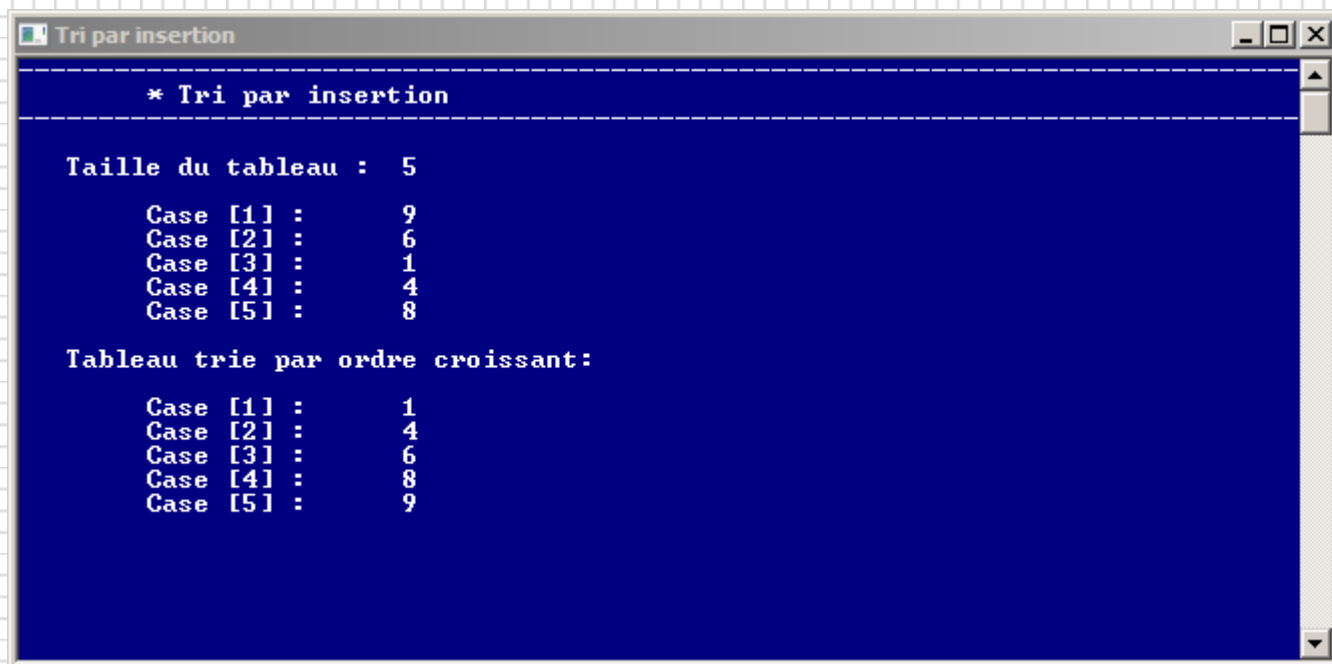
(**9** 6 1 4 8) → (**6** 9 1 4 8) L'élément 6 est comparé à son précédent 9, et comme $6 < 9$, 6 est remonté à la 1^{ère} position.

(**6** 9 **1** 4 8) → (**1** 6 9 4 8) Remontée de 1 à la 1^{ère} position car $1 < 9$ et $1 < 6$.

(**1** 6 9 **4** 8) → (**1** 4 6 9 8) Remontée de 4 à la 2^{ème} position car $(4 < 9 \text{ et } 4 < 6)$ et $1 < 4$.

(**1** 4 6 9 **8**) → (**1** 4 6 **8** 9) Remontée de 8 à la 4^{ème} position car 8 est supérieur à tout sauf 9.

d. Exécution :



II) Recherche dichotomique :

Introduction :

La recherche dichotomique ou dichotomie (« couper en deux » en grec) est un processus itératif ou récursif de recherche où, à chaque étape, on coupe en deux parties (pas forcément égales) un espace de recherche qui devient restreint à l'une de ces deux parties.

On suppose bien sûr qu'il existe un test relativement simple permettant à chaque étape de déterminer l'une des deux parties dans laquelle se trouve une solution. Pour optimiser le nombre d'itérations nécessaires, on s'arrangera pour choisir à chaque étape deux parties sensiblement de la même « taille » (pour un concept de « taille » approprié au problème), le nombre total d'itérations nécessaires à la complétion de l'algorithme étant alors logarithmique en la taille totale du problème initial.

L'algorithme s'applique typiquement à la recherche d'un élément dans un ensemble fini ordonné et organisé en séquence. La fonction de « taille » du problème sera alors le cardinal de l'espace (fini) de recherche, et à chaque étape, on coupera l'espace de recherche en deux parties de même taille (à un élément près) de part et d'autre de l'élément médian.

1) Principe :

Pour trouver rapidement un mot dans un dictionnaire, On ouvre généralement le livre au milieu et l'on tombe sur des mots commençant par une certaine lettre qui se trouve au centre du dictionnaire. Quand on voit d'après l'ordre des lettres que le mot recherché ne peut figurer dans la première moitié du livre, on restreint donc la recherche à la seconde moitié.

On met donc de côté la première moitié et on effectue le même raisonnement sur la seconde et ainsi de suite sur chaque nouvelle partie, on réduit progressivement la liste en allant soit dans la partie gauche, soit dans la partie droite, pour parvenir inéluctablement à la page contenant le mot cherché.

Le principe de recherche dichotomique peut se généraliser à tout type de problème dès lors que les objets du champ de recherche puissent former une séquence et qu'il soit possible d'effectuer une comparaison relative à l'ordre dans la séquence.

Implémenter l'algorithme sur ordinateur est facile grâce à la récursivité, mais il peut s'implémenter également de façon itérative.

2) Algorithme :

```

debut

// déclarations
entier debut, fin, val, mil, N, T[N]
booléen trouve

// tri
algorithme_de_tri(T[N])
// le tableau doit être trié avant la recherche, on utilise donc un des algorithmes de tri

// initialisations
debut ← 0
fin ← N
trouve ← faux

lire(val)

//Boucle de recherche
faire
    mil ← (debut + fin) / 2

    si ( T[mil] = val ) alors
        trouve ← vrai
    sinon

        si ( val > T[mil] ) alors
            debut ← mil+1
        sinon
            fin ← mil-1
        fin si

    fin si

    tant que ( trouve = faux ET debut ≤ fin )
// La condition début inférieur ou égal à fin permet d'éviter de faire
// une boucle infinie si 'val' n'existe pas dans le tableau.

// affichage du résultat
si ( trouve ) alors
    afficher("La valeur " val " est trouvé au rang " mil)
sinon
    afficher("La valeur " val " n'est pas dans le tableau")
fin si

fin

```

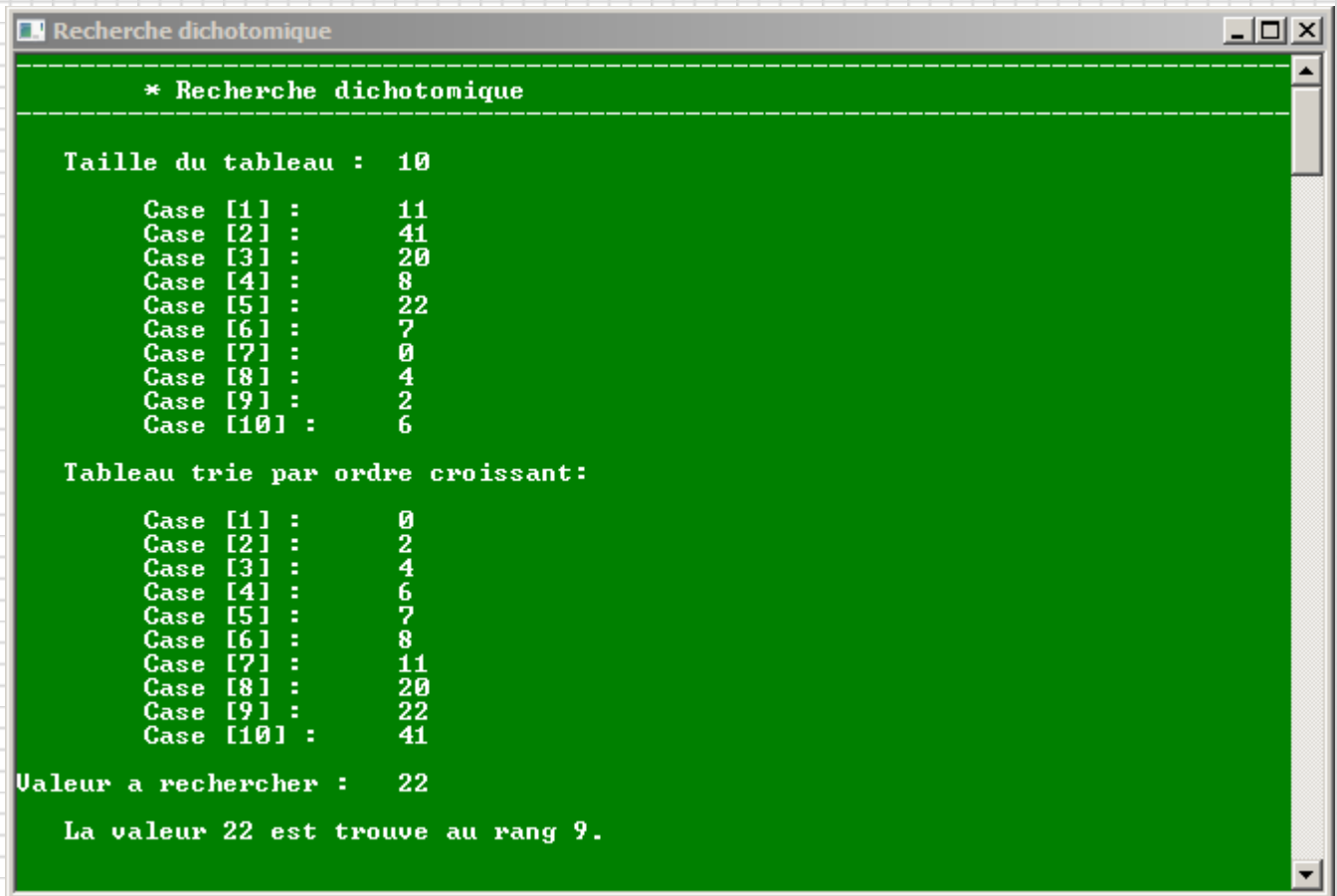
1) Trace :

Trouvons la valeur « 22 » dans la liste de chiffre croissante « 0 2 4 6 7 8 11 20 22 41 » en utilisant l'algorithme de recherche dichotomique. Pour chaque étape, les éléments considérés sont écrits en gras.

(0 2 4 6 7 **8** 11 20 22 41) → (11 20 22 41) La valeur du milieu est 8, comme 8<12, la recherche recommence après 8 (de 11).

(11 **20** 22 41) → (22 41) La valeur assimilable au milieu est 20, on recommence après 20 (de 22) car 20<22.

(**22** 41) → (**22**) La valeur assimilable au milieu est 22 et c'est celle recherchée.

2) Exécution :

```
* Recherche dichotomique

Taille du tableau : 10

Case [1] : 11
Case [2] : 41
Case [3] : 20
Case [4] : 8
Case [5] : 22
Case [6] : 7
Case [7] : 0
Case [8] : 4
Case [9] : 2
Case [10] : 6

Tableau trie par ordre croissant:

Case [1] : 0
Case [2] : 2
Case [3] : 4
Case [4] : 6
Case [5] : 7
Case [6] : 8
Case [7] : 11
Case [8] : 20
Case [9] : 22
Case [10] : 41

Valeur a rechercher : 22

La valeur 22 est trouve au rang 9.
```