09-00 - Trier un tableau

Plusieurs méthodes plus ou moins efficace pour ordonner un tableau

1. Rechercher un élément dans un tableau trié
   1. 1er algorithme

procédure recherche\_el(tableau t, entier n, el)

i=0 1

tant que t[i]<el et i<= n-1 2

i ←i+1 1

fin tant que

si t[i]=el, alors message ( el a été trouvé ) 1 1

sinon message (el n'est pas dans le tableau ) 1

Le nombre d’opérations est : Un = 3n + 3

La complexité de cet algorithme est : O(n)

* 1. Rechercher un élément par une méthode dichotomique

procédure recherche\_par\_dichotomie (tableau t,el)

n←longueur de t 1

G ← 0 1

D ← n-1 1

milieu ← partie entière de ( G + D) / 2 ) 1

tant que t[milieu] ≠ el et G ≤ D 2

si t[milieu] < el alors G ← milieu + 1 1 1

sinon D ← milieu-1 1

milieu ← partie entière de ( G + D) / 2 ) 1

fin tant que

si t[milieu] = el alors message ( el a été trouvé ) 1 1

sinon message (el n'est pas dans le tableau ) 1

a) Soit t ← [ 5, 2 , 3 , 1 , 8 , 4 , 10 , 7 ] Donner un assert de test.

assert recherche\_par\_dichotomie(t,10) == « 10 a été trouvé », « erreur assert»

Dérouler l’algorithme

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | G | D | milieu | message |
| 8 | 0 | 7 | 3 |  |
|  | 4 | 7 | 5 |  |
|  | 6 | 7 | 6 | 10 a été trouvé |

Pour recherche\_par\_dichotomie(t,9) :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | G | D | Milieu | Message |
| 8 | 0 | 7 | 3 |  |
|  | 4 | 7 | 5 |  |
|  | 6 | 7 | 6 |  |
|  | 7 | 7 | 7 |  |
|  | 8 | 7 | 7 | 9 n’est pas dans le tableau |
|  |  |  |  |  |
|  |  |  |  |  |

b) Le nombre d'opérations de cet algorithme est : 3 + 3 + 5[partieEntiere(log2(n)) + 1 ] + 2

c) Le cout de cet algorithme est d’ordre : O(log2(n))

# Tri d'un tableau par sélection

* 1. Principe :

Rechercher le minimum du tableau et le placer en début de liste.

* 1. Algorithme

procédure tri\_selection(tableau t)

n ← longueur de t

pour i de 0 à n - 2

min ← i

pour j de i + 1 à n-1

si t[j] < t[min], alors min ← j

fin pour

si min ≠ i, alors permuter t[i] et t[min]

fin pour

fin procédure

a) Soit t ← [ 5, 2 , 3 , 1 , 8 , 4 ] Donner un assert de test.

assert tri\_selection(t) == [1,2,3,4,5,8]

Donner tous les tableaux t intermédiaires avant d’obtenir le tableau trié.

5 2 3 1 8 4

1 2 3 5 8 4

1 2 3 5 8 4

1 2 3 5 8 4

1 2 3 4 8 5

1 2 3 4 5 8

b) Le nombre d’opérations est : 1 + n(n-1) / 2

c) La complexité de cet algorithme est : O(n2)le cout est quadratique

1. Tri à bulle d'un tableau

procédure tri\_a\_bulle(tableau t, entier n)

echange ← Vrai

tant que echange = Vrai faire

echange = Faux

pour i de 1 à n-1

si t[i]< t[i-1] alors echange (t[i]; t[i-1])

echange ← Vrai

fin si

fin pour

fin tant que

### Soit t ← [ 5, 2 , 3 , 1 , 8 , 4 , 10 , 7 ]

Assert tri\_a\_bulle(t) == [1,2,3,4,5,7,8,10]

Donner tous les tableaux t intermédiaires avant d’obtenir le tableau trié.

5 2 3 1 8 4 10 7

2 5 3 1 8 4 10 7

2 3 5 1 8 4 10 7

2 3 1 5 8 4 10 7

2 3 1 5 8 4 10 7

2 3 1 5 4 8 10 7

2 3 1 5 4 8 10 7

2 3 1 5 4 8 7 10

2 1 3 5 4 8 7 10

2 1 3 4 5 8 7 10  
1 2 3 4 5 8 7 10  
1 2 3 4 5 7 8 10

b) Le nombre d'opérations de cet algorithme est : 1 + 5\*n(n-1)+1

c) Le cout de cet algorithme est d’ordre : O(n2)

# Tri par insertion

procédure tri\_insertion(tableau t)

n ← longueur de t

pour i de 1 à n – 1

x ← t[i]

j ← i

tant que j >= 1 et t[j - 1] > x

t[j] ← t[j - 1]

j ← j - 1

fin tant que

t[j] ← x

fin pour

### Soit t ← [ 5, 2 , 3 , 1 , 8 , 4 , 10 , 7 ]

Assert tri\_insertion(t) == [1, 2, 3, 4, 5, 7, 8, 10]

Donner tous les tableaux t intermédiaires avant d’obtenir le tableau trié.

[ 5, 2 , 3 , 1 , 8 , 4 , 10 , 7 ]

[5, 5, 3, 1, 8, 4, 10, 7]

[2, 5, 3, 1, 8, 4, 10, 7]

[2, 5, 5, 1, 8, 4, 10, 7]

[2, 3, 5, 1, 8, 4, 10, 7]

[2, 3, 5, 5, 8, 4, 10, 7]

[2, 3, 3, 5, 8, 4, 10, 7]

[2, 2, 3, 5, 8, 4, 10, 7]

[1, 2, 3, 5, 8, 4, 10, 7]

[1, 2, 3, 5, 8, 4, 10, 7]

[1, 2, 3, 5, 8, 8, 10, 7]

[1, 2, 3, 5, 5, 8, 10, 7]

[1, 2, 3, 4, 5, 8, 10, 7]

[1, 2, 3, 4, 5, 8, 10, 7]

[1, 2, 3, 4, 5, 8, 10, 10]

[1, 2, 3, 4, 5, 8, 8, 10]

[1, 2, 3, 4, 5, 7, 8, 10]

b) le nombre d'opérations de cet algorithme est : 1 + (n-1)(n+3)

c) La complexité de cet algorithme est : O(n2)

# Tri d’un tableau par fusion

Principe : On construit un 3eme tableau à partir de 2 tableaux déjà triés

procédure tri\_par fusion(tableau t1, tableau t2)

n1 ← longueur de t1

n2 ← longueur de t2

x←0

y←0

pour i de 0 à n1 + n2 -1 faire

Si x< n1 et y < n2

Si t1[x]<=t2[y] alors

tab3[i] ← t1[x]

x←x+1

Sinon

tab3[i] ←t2[y]

y←y+1

fin si.

Sinon Si x >=n1 alors

tab3[i] ← t2[y]

y←y+1

sinon

tab3[i] ← t1[x]

x←x+1

fin si.

fin pour.

a) Soit . t1 ← [1 , 4 , 5 , 6 ] t2 ← [ 2 , 3 , 7 , 8 ] Donner un assert de test.

assert tri\_par\_fusion(t1,t2) == [1,2,3,4,5,6,7,8]

b) Dérouler l’algorithme en faisant un tableau.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | n1 | n2 | tab3 |
| 0 | 0 | 4 | 4 |  |
| 1 | 0 |  |  | 1 |
| 1 | 1 |  |  | 1 ,2 |
| 1 | 2 |  |  | 1,2,3 |
| 2 | 2 |  |  | 1,2,3,4 |
| 3 | 2 |  |  | 1,2,3,4,5 |
| 4 | 2 |  |  | 1,2,3,4,5,6 |
| 4 | 3 |  |  | 1,2,3,4,5,6,7 |
| 4 | 4 |  |  | 1,2,3,4,5,6,7,8 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

c) Le nombre d'opérations de cet algorithme est : n log(n)

d) La complexité de cet algorithme est : O(n)

VI) TP à faire

1) Répondre à toutes les questions précédentes.

2) Après avoir calculé la complexité des algorithmes, classer les 3 algorithmes tri par sélection, tri à bulles, tri par insertion, par ordre d’efficacité

Tri à bulle < tri par sélection < tri par insertion

Pire cas : O(n^2) O(n^2) O(n^2)

Cas moyen : O(n^2) O(n^2) O(n^2)

Meilleur cas : O(n) O(n^2) O(n)

3) Écrire l’algorithme de la fonction permutation.

Procédure permutation (tableau)

n ← longueur tableau

pour i de 1 à n-1

nbAleatoire ← entierAlea (1 , n-1)

tableau[i]← tableau[nbAleatoire]

tableau[nbAleatoire]← tableau[i]

4) Écrire les 6 algorithmes en python. Il faudra aussi écrire en python la fonction permutation.

5) Écrire on rajoutera des assert de test (au moins 3)

6) Rajouter le commentaire pour expliquer la fonction. Ce commentaire doit être visible avec help(*fonction*)

7) Écrire une fonction qui remplira un tableau de n nombres aléatoires compris entre 1 et n

8) Le programme principal devra à partir d’un tableau aléatoire, afficher les temps de chaque algorithme pour trier ce tableau.

On utilisera la fonction time du module time qui donne la date exacte au 10 millionième de seconde

Exemple de résultat :

Pour le tri à bulle, le temps d'exécution est : 0.2344 secondes avec un tableau de longueur 1000

Pour le tri par sélection, le temps d'exécution est : 0.0625 s avec un tableau de longueur 1000