CHAPITRE

7 Tri est sélection

Table des matières

1 Introduction :

1.1 Pourquoi trier ?

Voici un répertoire téléphonique généré par un programme Python :

```
0693250066 0607159107 0635460054 0608283002 0685858770 0683794395 0677112046 0696318689 0663517552 0661130363
0604591765 0634381005 0624246783 0657289793 0631534764 0678511067 0673613761 0614005301 0619378292 0672631573
0689829341 0646779312 0659983193 0658986046 0605132436 0614522265 0623968380 0611932364 0636293766 0694719557
0646521399 0655214349 0612088157 0601234816 0664402976 0628956979 0665188480 0604675511 0693077985 0651223034
0699512670 0637605964 0691164524 0679142432 0652454242 0601424763 0642745020 0659044340 0662794455 0686888884
0696652954 0693979723 0663515650 0628798656 0604092263 0644791917 0652024921 0661696009 0616457151 0612268726
0626461007 0653079654 0604235376 0636184078 0639470282 0623976812 0632703235 0678400583 0634654496 0663642908
0681212323 0678194849 0687453242 0661315840 0651802076 0689393736 0675346176 0616147862 0642578926 0669732822
0652447266 0652822666 0658351046 0607760564 0679577299 0696915307 0606958961 0662727789 0612733366 0635099954
0642125414 0651729365 0621624285 0640035708 0688745735 0621046987 0676555194 0630359269 0693961126 0669340522
0645569116 0660309908 0642030568 0683458761 0626956955 0657356527 0658977236 0616855892 0686026981 0685540960
0671516758 0609141329 0663717196 0671584561 0613198817 0680304779 0619078596 0662195444 0646325641 0697762645
0680417468 0690326180 0647263148 0682857382 0644545556 0600757897 0655477035 0642666772 0644377601 0679803874
0690052697 0635573684 0601520695 0697278870 0613466667 0601670372 0617242660 0633032362 0646717234 0637391170
0647518888 0600857806 0690855560 0605285882 0638472912 0617197491 0695586733 0661831682 0642578738 0632008611
0657499013 0689485838 0611097671 0619630917 0695075071 0642562815 0639506910 0622753389 0699409758 0659125253
0693002097 0645285569 0655784311 0649252127 0640721610 0600154692 0661827751 0673371409 0668855116 0628052757
0676213254 0693537813 0607046856 0695377082 0681632427 0679745570 0684544733 0659390386 0623136778 0620468330
0600549306 0693647295 0691335184 0679663910 0651026561 0639501433 0646593845 0691026896 0609411981 0648539375
0643684469 0649520831 0640242866 0634178040 0617337590 0667330884 0610091532 0695184265 0630623374 0620512765
```

Y a t-il votre numéro de téléphone ? Voici le même répertoire mais ordonné :

```
0600154692 0600549306 0600757897 0600857806 0601234816 0601424763 0601520695 0601670372 0604092263 0604235376
0604591765 0604675511 0605132436 0605285882 0606958961 0607046856 0607159107 0607760564 0608283002 0609141329
0609411981 0610091532 0611097671 0611932364 0612088157 0612268726 0612733366 0613198817 0613466667 0614005301
0614522265 0616147862 0616457151 0616855892 0617197491 0617242660 0617337590 0619078596 0619378292 0619630917
0620468330 0620512765 0621046987 0621624285 0622753389 0623136778 0623968380 0623976812 0624246783 0626461007
0626956955 0628052757 0628798656 0628956979 0630359269 0630623374 0631534764 0632008611 0632703235 0633032362
0634178040 0634381005 0634654496 0635099954 0635460054 0635573684 0636184078 0636293766 0637391170 0637605964
0638472912 0639470282 0639501433 0639506910 0640035708 0640242866 0640721610 0642030568 0642125414 0642562815
0642578738 0642578926 0642666772 0642745020 0643684469 0644377601 0644545556 0644791917 0645285569 0645569116
0646325641 0646521399 0646593845 0646717234 0646779312 0647263148 0647518888 0648539375 0649252127 0649520831
0651026561 0651223034 0651729365 0651802076 0652024921 0652447266 0652454242 0652822666 0653079654 0655214349
0655477035 0655784311 0657289793 0657356527 0657499013 0658351046 0658977236 0658986046 0659044340 0659125253
0659390386 0659983193 0660309908 0661130363 0661315840 0661696009 0661827751 0661831682 0662195444 0662727789
0662794455 0663515650 0663517552 0663642908 0663717196 0664402976 0665188480 0667330884 0668855116 0669340522
0669732822 0671516758 0671584561 0672631573 0673371409 0673613761 0675346176 0676213254 0676555194 0677112046
0678194849 0678400583 0678511067 0679142432 0679577299 0679663910 0679745570 0679803874 0680304779 0680417468
0681212323 0681632427 0682857382 0683458761 0683794395 0684544733 0685540960 0685858770 0686026981 0686888884
0687453242 0688745735 0689393736 0689485838 0689829341 0690052697 0690326180 0690855560 0691026896 0691164524
0691335184 0693002097 0693077985 0693250066 0693537813 0693647295 0693961126 0693979723 0694719557 0695075071
0695184265 0695377082 0695586733 0696318689 0696652954 0696915307 0697278870 0697762645 0699409758 0699512670
```

C'est plus facile pour rechercher, non ?

Cet exemple montre l'intérêt de trier des valeurs, notamment pour en chercher une valeur particulière. Dans ce travail, on s'intéressera aux nombreux algorithmes de tris mais aussi à leur complexité, c'est-à-dire à une mesure qui permet de juger de leur efficacité.

1.2 Recherche dans un tableau ou une liste :

On s'intéresse d'abord à la recherche d'un élément dans un tableau quelconque dont l'algorithme est le suivant :

Algorithme 1 RECHERCHE D'UN ÉLÉMENT DANS UN TABLEAU

Entrée(s) Un tableau T de taille n et un élément e
Sortie(s) Un indice i tel que $T[i] = e$ et -1 sinon

```
pour i allant de 0 à n − 1 faire
    si T[i] = e alors
        retourner i
    fin du si
fin du pour
retourner −1
```

Exercice n°1

❶ Implémenter cet algorithme en Python.

❷ Testez sur des listes que vous aurez construits pas compréhension.

Pour mesurer la complexité en temps de cet algorithme, on compte le nombre de comparaisons à faire. On parle de **complexité dans le pire des cas** quand on s'intéresse aux cas qui donneront le plus de comparaisons.

Exercice n°2

❶ Quel est le pire des cas dans l'algorithme précédent ? Quel est alors le nombre de comparaisons ?

❷ Modifier le programme précédent afin qu'il affiche le nombre de comparaison effectuée.

La complexité s'exprime en fonction de la taille des données proposées en entrée, c'est-à-dire ici à la taille des tableaux.

Exercice n°3
Comment évolue la complexité dans le pire des cas si on double, on triple la taille du tableau ?

On dit que la complexité de l'algorithme est proportionnelle à n, la taille des entrées. On écrit que la complexité est en $O(n)$ pour reprendre une notation mathématique connue [1].
Cette complexité étant jugée trop élevée, on utilise une autre méthode de recherche : la recherche **dichotomique** dans un tableau **trié** :

Algorithme 2 Recherche dichotomique dans un tableau trié

Entrée(s) Un tableau trié T de taille n et un élément e
Sortie(s) Un indice i tel que $T[i] = e$ et -1 sinon

```
min = 0
max = n − 1
tant que min <= max faire
   m = (min + max)/2
   si T[m] = e alors
      retourner m
   sinon
      si T[m] < e alors
         min = m + 1
      sinon
         max = m
      fin du si
   fin du si
fin du tant que
retourner -1
```

Exercice n°4

❶ Implémenter cet algorithme en Python.

❷ Testez le programme sur des listes **triées**.

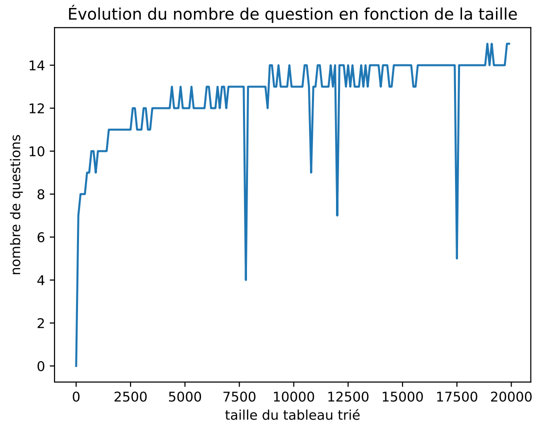
Exercice n°5

① Dans le pire des cas, déterminer le nombre d'itérations (de boucles...) nécessaires pour un tableau trié de $n = 100$.

② Si on double la taille, par exemple $n = 200$, comment évolue le nombre d'itérations nécessaires ?

1. dans le supérieur...

On dit que la complexité dans le pire des cas de l'algorithme précédent est en $\log_2(n)$ et on écrit $O(\log_2(n))$.



2 Des algorithmes de tris

Trier des données n'est pas une mince affaire : il faut non seulement trouver un algorithme qui trie mais aussi s'assurer de son efficacité sur des données de taille très importante. On se propose ici de découvrir deux algorithmes de tris, de comprendre comment ils procèdent et de déterminer leur complexité.

2.1 La tri par sélection

2.1.1 Comprendre...

Exercice n°6
Appliquer cet algorithme au tableau ci-dessous en écrivant les étapes de tri consécutives.

4	8	5	1	17	3	12

2.1.2 Implémenter...

Voici une proposition d'implémentation possible en Python du tri par sélection :

```python
def echange(t:list, i:int, j:int)->list:
        t[i], t[j] = t[j], t[i]

def triselection(tab:list)->list:
    i = 0
    while i < len(tab):
        k = ...
        mini = tab[i]
        for ... in range(..., len(tab)):
            if tab[j] < mini:
                k = ...
                mini = tab[k]
        if k != i:
            echange(tab, ..., ...)
        i += 1
    return tab
```

Exercice n°7

① Compléter ce programme avec les variables i, j et k

② Testez sur un tableau construit par compréhension

2.1.3 Tester...

Il s'agit de mesurer l'efficacité de cet algorithme en déterminant d'abord sa **complexité au pire des cas**.

Exercice n°8
On détermine la complexité dans le pire des cas par le nombre de comparaisons effectuées.

① Montrez que ce nombre est égal à $\dfrac{n(n-1)}{2}$ où n est la taille du tableau.

② En déduire la complexité au pire de cet algorithme.

On se rend compte que dans tous les cas, on effectue toutes les comparaisons évoquées ci-dessus. Aussi, décidons-nous de mesurer cette fois-ci, la complexité par le nombre d'échanges effectués.

Exercice n°9

① Modifier le programme précédent pour qu'il retourne le nombre d'échanges effectués dans son exécution.

② Construire une fonction test prend un entrée deux entiers n et k et qui retourne la liste de k nombres égaux aux nombres d'échanges réalisés sur divers tableaux de taille n.

③ Quelle relation peut-on établir entre n et le nombre moyen d'échanges (on pourra faire un graphique) ?

2.2 Le tri par insertion

Le tri par insertion peut être visualisé de cette façon : visualisationInsertion .py

Exercice n°10
Appliquer cet algorithme au tableau suivant.

4	8	5	1	17	3	12

Il repose sur le principe d'insérer un élément e dans un tableau T déjà trié de taille n. Si nous supposons alors la fonction inserer (T,n,e) déjà codée alors le tri par insertion est alors :

Algorithme 3 TRI PAR INSERTION

Entrée(s) Un tableau T de taille n
Sortie(s) Le tableau T trié
pour i allant de 1 à $n-1$ **faire**
#le tableau T[0 :i-1] est trié
inserer(T,i, T[i])
fin du pour

Exercice n°11

① Tester à la main cette fonction sur le tableau T = [4, 1, 2, 5, 3].

② Tester la fonction suivante sur des exemples construis par compréhension.

```
def tri_par_insertion(tableau) :
    '''Tri en ordre croissant le tableau en utilisation le tri par insertion
    param tableau(list) :: un tableau d'éléments de même type
    return (None) :: procédure (renvoie donc None en Python)
    Effet de bord :: modifie le tableau en permutant les éléments
    '''
    for i in range(1, len(tableau)) :
        val = tableau[i]
        j = i - 1 # On commence par l'élément juste à gauche de la val
        while j >= 0 and tableau[j] > val :
            tableau[j + 1] = tableau[j] # On décale l'élément vers la droite
            j = j - 1 # On passe à l'élément encore à gauche
        tableau[j + 1] = val
```

3 Complexité des algorithmes

Lorsqu'on souhaite déterminer l'efficacité d'un algorithme, il faut prendre le temps de définir ce qu'on entend par « efficacité ». C'est peut-être :

— un algorithme qui « prend » le moins de temps ;
— un algorithme qui réalise le moins d'instructions ;
— un algorithme qui utilise le moins de mémoire.

Déjà, il n'y a rien de plus relatif que le temps sur une machine ! Le temps d'exécution d'un programme est multi-factoriel : pc portable ou pc fixe, compilé ou interprété,... sans compter la bonne volonté du système d'exploitation qui alloue les ressources selon les besoins et les demandes des autres programmes.
Par ailleurs, compter le nombre d'instructions semble une bonne initiative mais il faudra sans doute être plus clair : comptons nous les affectations, les opérations arithmétiques, les comparaisons,...
Enfin, l'intérêt porté sur l'occupation de la mémoire est intéressant : il s'agit d'ailleurs d'**une complexité spatiale** mais nous ne développerons pas cet aspect ici.
Dorénavant, on parlera de **complexité d'un algorithme** pour évoquer sa **complexité temporelle** :il s'agit alors de compter le nombres d'opérations élémentaires (opérations arithmétiques, comparaisons ou affectations) même si le contexte privilégiera ce qu'il faut plutôt compter (dans le tri, on ne compte en général que les comparaisons ou échanges...).
La complexité s'exprime en fonction de la taille n des entrées. Établir la complexité **C** d'un algorithme, c'est donner un ordre de grandeur , en fonction de n, du nombre « d'opérations élémentaires » .
En général, il est plus facile de déterminer la **complexité dans le pire des cas**, c'est-à-dire la complexité dans le cas qui donneraient le plus d'instructions : dans le tri, le pire serait un tableau trié dans un ordre décroissant, pour la recherche d'un élément c'est que l'élément n'y soit pas !