QuickRanking : Algorithme Rapide de Classement

Laurent Laurent Ott Ott

QuickRanking : Algorithme Rapide de Classement

QuickRanking est un algorithme qui permet dans un même traitement, de trier les données et de retourner leur ordre de classement, c'est-à-dire leur rang dans le tri.

Pour des traitements plus rapides, QuickRanking ne déplace pas les données, mais mémorise pour chaque donnée, où est la donnée qui la suit. Cette méthode est expliquée en Annexe 1.
Le code source écrit en VBA est en annexe 2.

Pour évaluer l'efficacité de QuickRanking, j'ai reproduit en annexe 3 le code source de QuickSort, et en annexe 4 une version « enrichie » de QuickSort, que j'ai appelée QuickSort_AndRank, qui à l'instar de QuickRanking, ne déplace pas directement les données mais déplace les références des indices des données. D'où un gain de temps sur les traitements des données alphanumériques volumineuses car l'on déplace des entiers de 4 octets et non plus des chaînes de caractères de longueurs variables. Cela nécessite des capacités mémoire supplémentaires, mais permet du coup d'utiliser ces tableaux de référence pour retourner un ordre de classement en plus du tri, tout comme le fait QuickRanking.

J'ai comparé ces trois algorithmes - QuickSort, QuickSort_AndRank, et QuickRanking - les résultats sont synthétisés dans un tableau en annexe 5. Mon analyse est la suivante :

Si l'ordre de classement n'est pas nécessaire :
- Tri des données numériques : QuickSort est l'algorithme le plus rapide sauf lorsque les données en début de liste sont déjà classées, ou que la valeur minimale ou maximale est souvent représentée. Dans ces cas, QuickRanking prend l'avantage.
- Tri des données Alphanumériques : Plus le nombre de caractères des données à trier est important, plus les algorithmes qui travaillent par référence, QuickSort_AndRank et QuickRanking, sont efficaces. QuickSort ne doit être utilisé que si les capacités mémoire ne permettent pas d'utiliser les deux autres algorithmes qui nécessitent plus de ressources.

Si l'ordre de classement est nécessaire :
QuickRanking et QuickSort_AndRank proposent deux modes de classement pour des données égales : soit le même rang est attribué et il y a donc des ex-aequo (méthode 1), soit la position d'origine des données est utilisée pour hiérarchiser les rangs de façon y a ce qu'il n'y ait pas d'ex-aequo (méthode 2).
- Tri et classement des données numériques : Les deux algorithmes se valent sur les données aléatoires. QuickRanking prend l'avantage lorsque les données en début de liste sont déjà classées, lorsque la liste est composée de peu de valeurs différentes, ou lorsque la valeur minimale ou maximale est souvent représentée.
- Tri et classement des données Alphanumériques : QuickRanking est plus rapide que QuickSort_AndRank. Avantage accentué lorsque la méthode 2 de classement est utilisée, ou sur le traitement des grandes chaînes de caractères. Inversement, l'avantage s'émousse sur les listes très volumineuses, ou pour le traitement des petites chaînes de caractères.

Conclusion : liste idéalement aléatoire ou liste fortement classée, données volumineuses ou de taille réduite, la réalité se situe souvent entre ces extrêmes. Dans la pratique, QuickRanking offre alors une alternative intéressante…

Remarques :
- QuickRanking propose en option des analyses complémentaires pour accélérer les traitements sur les listes où les données ont de nombreuses égalités, ou lorsque les données se suivent. Cette option n'a pas été utilisée dans les tests réalisés car ils sont basés sur des données aléatoires. L'annexe 6 présente une fonction qui détermine, après analyse d'un échantillon des données de la liste, s'il faut ou non activer cette option.
- Le fichier EXCEL joint reprend ces algorithmes.
- Dans les tests présentés, QuickSort et QuickSort_AndRank utilisent un pivot fixe.
- Les tests ont été réalisés sur un PC bureautique en VBA sous EXCEL. Les résultats sont donnés à titre indicatifs, l'objectif étant de faire ressortir une tendance. Ces résultats devront être confirmés dans d'autres langages de programmation et avec d'autres systèmes d'exploitation.
- L'optimisation du code dans cette version 3 permet de réduire les temps de traitement respectivement de 11% et 4% par rapport aux versions précédentes.

Trions les 12 données du tableau ci-dessous pour comprendre cette méthode de tri, qui pour économiser du temps de traitement, ne déplace pas les données, mais mémorise pour chaque donnée où est la donnée qui la suit :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90

Analysons les deux premiers éléments : Ici l'on sait que l'indice 0, qui vaut 3 est la valeur minimale et l'indice 1 qui vaut 9 est la valeur maximale.
Donc l'indice suivant l'indice 0 est l'indice 1. Nous pouvons mémoriser ces informations :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	1	x										

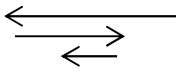
Pour simplifier la lecture du tableau, la valeur minimale est représentée en vert et la valeur maximale est en orange.

Pour classer l'indice 2, qui vaut 5 : Nous lisons la valeur minimale, l'indice 0. La valeur 3 est inférieure à 5, donc nous lisons l'indice suivant, l'indice 1, qui vaut 9. Trop grand cette fois. Donc 5 devient la valeur qui suivra 3 et 9 sera la valeur qui suit 5.
Ce qui met à jour notre tableau des « indices suivants » :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	x	1									

Classons l'indice 3 qui vaut 2, valeur inférieure à notre ancienne valeur minimale qui était l'indice 0 de valeur 3. Donc un échange est effectué. Et l'indice 0 sera la valeur qui suivra l'indice 3.
Pas de changement pour les autres données déjà analysées :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	x	1	0								



Ces flèches indiquent l'ordre de lecture du tableau en partant de la référence minimale, et en lisant les « indices suivants ».

Passons à l'indice 4 qui vaut 15, valeur supérieure à notre ancienne valeur maximale, l'indice 1 qui vaut 9. Donc un échange est effectué. L'indice 4 sera la valeur qui suivra l'indice 1.
Pas de changement pour les autres données déjà analysées :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	1	0	x							

L'indice 5 vaut 7, valeur supérieure à notre valeur minimale de référence, l'indice 3, qui vaut 2. Donc nous lisons l'indice suivant, l'indice 0, qui vaut 3. Toujours inférieur à 7, donc nous lisons l'indice suivant, l'indice 2, qui vaut 5. Nous continuons avec l'indice suivant, 1, qui vaut 9. Cette fois nous pouvons sortir de la boucle et mettre à jour les indices : l'indice suivant le chiffre 7 est donc l'indice 1, et le nouvel indice suivant de l'indice 2 n'est plus 1 mais 5 :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	5	0	x	1						

Mêmes principes pour classer l'indice 6 qui vaut 8 : Après lecture des valeurs du tableau en partant de l'indice de la valeur minimale et en utilisant les « indices suivants », nous trouvons qu'il se situe entre l'indice 5 qui vaut 7 et l'indice 1 qui vaut 9. Donc son « indice suivant » est l'indice 1, et « l'indice suivant » de l'indice 5 est modifié : 1 est remplacé par 6.

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Données :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	5	0	x	6	1					

Remarque : ici un raccourci est possible pour classer rapidement l'indice 6. En effet 8 est supérieur ou égal à la dernière valeur analysée, qui valait 7, et est inférieur ou égale à la valeur de l'indice suivant de la dernière valeur analysée, qui vaut 9. Donc l'indice suivant de l'indice 6 est l'indice suivant de la dernière valeur analysée. Et l'indice suivant de la dernière valeur analysée devient l'indice 6.
Ce test est très performant sur les listes partiellement classées ou s'il y a de nombreuses égalités, mais est contre-productif sur les listes aléatoires. C'est pourquoi ce test est optionnel dans QuickRanking afin de ne pas l'exécuter sur les listes de données que vous savez aléatoires, et ainsi économiser du temps de traitement.

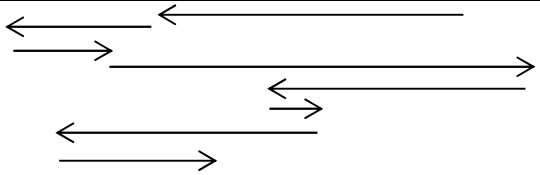
Poursuivez le classement et si vous obtenez ce tableau, c'est que vous avez compris :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x

Pour lire les données dans l'ordre croissant, il faut partir de la référence minimale, l'indice 9, soit la valeur 0, et lire les « indices suivants » : l'indice 3, qui vaut 2, l'indice 0 qui vaut 3, l'indice 2 qui vaut 5, et ainsi de suite…

Les données sont triées sans qu'aucun déplacement n'ait été fait.
Seul problème : pour connaître le classement d'un élément, il faut parfois faire de nombreuses lectures.
Par exemple, pour classer l'indice 12 qui vaut 10, voici le chemin suivi, soit 8 lectures, pour trouver que l'indice 1 est le dernier élément inférieur :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11	12
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90	10
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x	



Vous devinez qu'avec un tableau de plusieurs centaines d'éléments, certaines valeurs à classer vont nécessiter un nombre de lecture impressionnant… ce qui fait exploser les temps de traitement.

Pour gagner du temps, il faut tenir à jour un tableau des données déjà classées :

Indice :	9	3	0	2	10	5	6	1	4	8	7	11
Valeur :	**0**	**2**	**3**	**5**	**5**	**7**	**8**	**9**	**15**	**17**	**20**	**90**

Ce qui permet de faire une recherche dichotomique pour trouver la valeur la plus proche de l'élément que l'on souhaite classer, qui vaut 10, en seulement 4 lectures :

Indice :	9	3	0	2	10	5	6	1	4	8	7	11
Valeur :	**0**	**2**	**3**	**5**	**5**	**7**	**8**	**9**	**15**	**17**	**20**	**90**

Indice :						5	6	1	4	8	7	11
Valeur :						**7**	**8**	**9**	**15**	**17**	**20**	**90**

Indice :						5	6	1				
Valeur :						**7**	**8**	**9**				

Indice :							6	1				
Valeur :							**8**	**9**				

Cette méthode prend toute sa puissance sur un tableau très grand. Par exemple, sur un tableau de 10.000 éléments, 14 lectures seulement permettent de trouver l'élément désiré, soit environ Log(*n*) / Log(2), où *n* est le nombre d'éléments du tableau. A comparer avec les centaines de lectures nécessaires par la lecture un à un des « indices suivants ».

La génération de ce tableau des données déjà classées étant chronophage, elle ne sera réalisée que de temps en temps. Mais même incomplet, ce tableau permet de se rapprocher rapidement de l'élément le plus proche, pour ensuite faire un traitement ordinaire en lisant les « indices suivants » un à un, jusqu'à trouver le bon élément. Considérez ce tableau comme en raccourci pour arriver rapidement proche de la destination finale, et pas forcément à la destination finale.

Il faut arriver à un bon compromis pour que le temps passé à générer ce tableau, soit rentabilisé par le temps gagné en recherche par dichotomie.
J'ai retenu le principe de mettre à jour le tableau lorsque le nombre de lectures des « indexes suivants » équivaut au nombre de données traitées.

Une analyse mathématique devrait permettre de calculer la génération optimale.

Reprenons le tableau déjà vu :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x

La lecture des indices donne le tri des éléments : 0, 2, 3, 5, 5, 7, 8, 9, 15, 17, 20, 90.
Cette lecture permet aussi d'obtenir l'ordre de classement des éléments :

Le premier élément est l'indice 9. Dans un tableau annexe qui sert d'ordre de classement, l'on met 1 pour l'indice 9.

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x
Classement :										1		

L'indice suivant est l'indice 3, l'on met 2 dans l'ordre de classement de l'indice 3.

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x
Classement :				2						1		

L'indice suivant est l'indice 0, l'on met 3 dans l'ordre de classement de l'indice 0.
Et ainsi de suite pour obtenir ce tableau :

Indice :	0	1	2	3	4	5	6	7	8	9	10	11
Valeur :	3	9	5	2	15	7	8	20	17	0	5	90
Indice suivant :	2	4	10	0	8	6	1	11	7	3	5	x
Classement :	3	8	4	2	9	6	7	11	10	1	5	12

L'indice 0, qui vaut 3, est le 3ème élément de la liste classée.
L'indice 1 qui vaut 9 est le 8ème élément de la liste classée.
L'indice 2 qui vaut 5 est le 4ème élément de la liste classée…

L'ordre de classement est pratique pour remplir un tableau de ce type :

Participant	Epreuve 1	Classement	Epreuve 2	Classement	Total	Classement
Mathieu	15,24	3	12,65	4	27,89	3
Sylvie	14,15	4	17,45	1	31,60	2
Nathalie	16,24	2	8,59	5	24,83	5
Edouard	17,50	1	14,15	2	31,65	1
Luc	13,67	5	13,80	3	27,47	4

Mais l'exercice peut se révéler plus compliqué dans certains cas, comme pour le tableau ci-dessous :

Indice :	0	1	2	3	4	5	6	7	8
Valeur :	8	8	9	8	10	8	8	10	5
Indice suivant :	1	2	4	0	7	3	5	x	6
Classement :	5	6	7	4	8	3	2	9	1

Les éléments de valeurs 8, des indices 0, 1, 3, 5 et 6, ont un ordre de classement incohérent. Rien ne peut justifier que l'indice 6 soit classé 2ème alors que l'indice 5 est classé 3ème.

Dans la pratique, soit l'on attribue le même classement à toutes les valeurs égales :

Indice :	0	1	2	3	4	5	6	7	8
Valeur :	8	8	9	8	10	8	8	10	5
Indice suivant :	1	2	4	0	7	3	5	x	6
Classement :	2	2	7	2	8	2	2	8	1

Ici il y a un premier, et 5 deuxièmes ex-aequo. L'élément de valeur 9 (indice 2) est 7ème.

Soit l'on conserve l'ordre d'origine pour classer les valeurs égales :

Indice :	0	1	2	3	4	5	6	7	8
Valeur :	8	8	9	8	10	8	8	10	5
Indice suivant :	1	2	4	0	7	3	5	x	6
Classement :	2	3	7	4	8	5	6	9	1

Ici chaque élément à un rang distinct. Il n'y a pas d'ex-aequo même pour les éléments de même valeur. En cas d'égalité, c'est l'élément à l'indice le plus bas qui est priorisé.

QuickRanking sait gérer ces deux méthodes de classement, ainsi que l'ordre croissant ou décroissant.

La fonction accepte trois arguments, en plus du tableau à trier :

- *OrdreCroissant* : s'il vaut **True**, le tri se fait par ordre croissant, s'il vaut **False**, le tri est décroissant.
- *ModeClassement* indique le mode de classement à retourner : 0 pour aucun classement, 1 pour appliquer le même rang aux données égales, 2 pour attribuer un rang différent aux données égales en respectant leur position d'origine.
- *NiveauTest* : s'il vaut **False**, aucune analyse supplémentaire n'est réalisée. S'il vaut **True** des tests supplémentaires sont réalisés en se basant sur le dernier élément analysé : recherche des suites et des doublons pour un traitement immédiat, recherche de la borne minimale et maximale du tableau utilisé pour la recherche dichotomique.

Exemple d'appel de la fonction QuickRanking en VBA sous EXCEL, après chargement des données à trier placées en colonne A, lignes 2 et suivantes. Le tri est restitué en colonne B et le classement par la méthode 1 en colonne C.

```
'--------------------------------------------------------------------------------
Sub Démonstration_QuickRanking()
'--------------------------------------------------------------------------------
Dim i As Long, Données() As Variant, Rang() As Long

While Cells(i + 2, 1) <> ""      ' Boucle sur les lignes de la colonne A.
    ReDim Preserve Données(i)    ' Redimensionne le tableau des données à trier.
    Données(i) = Cells(i + 2, 1) ' Mémorise la valeur de la cellule.
    i = i + 1                    ' Passe à la ligne suivante.
Wend

Rang = QuickRanking(Données(), True, 1) ' Tri le tableau Données et retourne le classement.

For i = LBound(Données) To UBound(Données) ' Boucle sur les éléments.
    Cells(i + 2, 2) = Données(i)           ' Affiche le tri en B
    Cells(i + 2, 3) = Rang(i)              ' Affiche le rang en C
Next i

End Sub
```

```
'-----------------------------------------------------------------------------
Public Function QuickRanking(ByRef TabDonnées() As Variant, _
                Optional OrdreCroissant As Boolean = True, _
                Optional ModeClassement As Byte = 1, _
                Optional NiveauTest As Boolean = False) As Variant
'-----------------------------------------------------------------------------
' TabDonnées : Tri les données passées en argument et modifie TabDonnées.
' OrdreCroissant : Si vaut True alors ordre croissant, sinon ordre décroissant.
' ModeClassement : 0 = Tri, Pas de classement.
'                  1 = Tri + Classement des données, les données égales ont le même ordre.
'                  2 = Tri + Classement des données, l'ordre des données égales
'                      respecte l'ordre d'origine.
' NiveauTest : False = Pas de test complémentaire,
'              True  = Contrôle les égalités et les suites immédiates.
'-----------------------------------------------------------------------------
' Bornes du tableau des données d'origine:
Dim TabDébut As Long, TabFin As Long
On Error Resume Next ' Si aucune donnée à trier.
TabDébut = LBound(TabDonnées)
TabFin = UBound(TabDonnées)

' Initialisation du tableau du classement des données:
ReDim Ref(TabDébut To TabFin) As Long

' Si rien à trier alors quitte:
If Abs(TabFin - TabDébut) < 1 Then QuickRanking = Ref(): Exit Function

' Initialisation des variables pour le traitement de tri:
Dim Tps As Variant
Dim i As Long, n As Long, j As Long, Anc As Long, l As Long
Dim RefMini As Long, RefMaxi As Long, MaxiRac As Long, MiniRac As Long
Dim NbPassage As Long, Début As Long, Fin As Long
Dim NbRechercheDicho As Long

' Initialisation du tableau des données déjà classées:
ReDim TabTps(TabDébut To TabFin) As Long
MaxiRac = TabDébut
NbPassage = TabFin + 1

' Configure le classement des 2 premiers éléments:
If TabDonnées(TabDébut) > TabDonnées(TabDébut + 1) Then n = 1
RefMini = TabDébut + n
RefMaxi = TabDébut + 1 - n
Ref(TabDébut) = TabDébut + 1
Ref(TabDébut + 1) = RefMaxi

' Boucle sur les autres éléments à classer:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
For n = 2 + TabDébut To TabFin
    Tps = TabDonnées(n)

   ' Controle le débordement du mini:
    While TabDonnées(RefMini) >= Tps
        Ref(n) = RefMini             ' La données suivante de n est l'ancien minimum.
        RefMini = n                  ' Mémorise qui est le nouveau minimum.
        TabTps(TabDébut) = n         ' Le 1er élément du tableau de recheche dicho.
        MiniRac = TabDébut           ' Le minimum pour la mise à jour du tableau de recherche dicho.
        Anc = TabDébut               ' Position du dernier élément analysé dans le tableau de recherche dicho.
        GoTo Element_Suivant         ' Fin du traitement de n.
    Wend

    ' Controle le débordement du maxi:
    While Tps >= TabDonnées(RefMaxi)
        Ref(RefMaxi) = n             ' La donnée suivante de l'ancien maximum est n.
        Ref(n) = n                   ' La donnée suivante de n est n.
        RefMaxi = n                  ' Mémorise qui est le nouveau maximum.
        MaxiRac = MaxiRac + 1        ' Dernière positon dans le tableau de recherche dicho.
        TabTps(MaxiRac) = n          ' Le tableau de recherche dicho peut être alimenté.
        Anc = MaxiRac                ' Position du dernier élément analysé dans le tableau de recherche dicho.
        GoTo Element_Suivant         ' Fin du traitement de n.
    Wend
```

```
    ' Mise à jour du tableau des données déjà classées:
    While NbPassage > n
        i = TabTps(MiniRac)                     ' Boucle depuis la position du plus petit élément analysé,
        If MiniRac = TabDébut Then i = RefMini ' ou boucle depuis la position du minimum.
        For j = MiniRac To n
            TabTps(j) = i                       ' Mémorise la position de l'élément.
            i = Ref(i)                          ' Position de l'élément suivant.
        Next j
        MaxiRac = n - 1                         ' Le dernier élément n'est pas utilisé.
        MiniRac = MaxiRac                       ' Efface la position du plus petit élément analysé.
        NbPassage = 0                           ' Efface le nombre de passages pour mise à jour du tableau.
        NbRechercheDicho = Log(n) / Log(2)      ' Nombre maximum de recherches possibles dans le tableau dicho.
        Début = TabDébut: Fin = MaxiRac         ' Bornes pour la recherche Dichotomique.
        GoTo RechercheDichotomique
    Wend

    ' Bornes pour la Recherche Dichotomique dans le tableau des données déjà classées:
    Début = TabDébut: Fin = MaxiRac

    ' S'il faut faire des tests complémentaires:
    If NiveauTest = True Then
        ' Contrôle les égalités et les suites immédiates:
        If Tps >= TabDonnées(n - 1) Then            ' Si n est >= dernier élément analysé.
            Début = Anc                             ' Borne de début pour la recherche dicho.
            While TabDonnées(Ref(n - 1)) >= Tps     ' Si n est <= élément suivant du dernier élément analysé.
                Ref(n) = Ref(n - 1)                 ' Echange de la donnée suivante de n et de l'ancien élément.
                Ref(n - 1) = n                      ' n devient la donnée suivante de l'ancien élément.
                GoTo Element_Suivant                ' Fin du traitement de n.
            Wend
        Else
            Fin = Anc                               ' Borne de fin pour la recherche dicho.
        End If
    End If

    ' Recherche Dichotomique dans le tableau des données déjà classées:
RechercheDichotomique:

      For j = 2 To NbRechercheDicho  ' Plus rapide que Do...Loop While Début + 2 < Fin
          i = (Début + Fin) / 2      ' Calcule le milieu.
          If Tps > TabDonnées(TabTps(i)) Then
              Début = i    ' Nouvelle borne de début.
          Else
              Fin = i      ' Nouvelle borne de fin.
          End If
      Next j

      Anc = Début     ' Solution.
      i = TabTps(Anc) ' Plus proche donnée inférieure connue.
      While Anc < MiniRac: MiniRac = Anc: Wend ' Plus rapide que If Anc < MiniRac Then MiniRac = Anc

      ' Boucle sur les indices suivants pour trouver le classement du nouvel élément:
      Do
          j = i
          i = Ref(i)
          NbPassage = NbPassage + 1   ' Compte le nombre de passages infructeux.
      Loop While Tps > TabDonnées(i)

      Ref(n) = Ref(j)   ' Qui est la donnée suivante de n.
      Ref(j) = n        ' n devient la donnée suivante de l'ancien élément.

  Element_Suivant:
  Next n
  ' Fait une copie temporaire du tableau d'origine:
  ' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ReDim Mémo(TabDébut To TabFin) As Variant
  For n = TabDébut To TabFin
      Mémo(n) = TabDonnées(n)
  Next n
```

```
' Initialisation du tableau du classement si demandé:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement > 0 Then
    Erase TabTps()
    ReDim Pos(TabDébut To TabFin) As Long
    Dim Egalités() As Variant
End If

' Classe les données dans l'ordre croissant:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If OrdreCroissant = True Then
    i = RefMini
    For n = TabDébut To TabFin
        TabDonnées(n) = Mémo(i)
        i = Ref(i)
    Next n

    ' S'il faut retourner le Classement où les égalités ont le même classement:
    If ModeClassement = 1 Then
        i = RefMini: Anc = i: NbPassage = 1
        For n = TabDébut To TabFin
            Pos(i) = NbPassage: NbPassage = NbPassage + 1
            If Mémo(i) = Mémo(Anc) Then Pos(i) = Pos(Anc)
            Anc = i
            i = Ref(i)
        Next n
        QuickRanking = Pos(): Exit Function
    End If
    ' S'il faut retourner le Classement où les égalités distinguent l'ordre d'origine:
    If ModeClassement = 2 Then
        i = RefMini: Anc = i: j = TabDébut: NbPassage = 1
        For n = TabDébut To TabFin
            ReDim Preserve Egalités(TabDébut To j)
            Egalités(j) = i
            Anc = i
            i = Ref(i)
            j = j + 1
            If Mémo(i) <> Mémo(Anc) Then
                If j > TabDébut + 1 Then
                    Call QuickSort(Egalités(), LBound(Egalités), UBound(Egalités))
                    ' Plus rapide que Call QuickRanking(Egalités(), True, 0)
                End If
                For l = TabDébut To j - 1
                    Pos(Egalités(l)) = NbPassage
                    NbPassage = NbPassage + 1
                Next l
                j = TabDébut
            End If
        Next n
        If j > TabDébut + 1 Then
            Call QuickSort(Egalités(), LBound(Egalités), UBound(Egalités))
            ' Plus rapide que Call QuickRanking(Egalités(), True, 0)
        End If
        For l = TabDébut To j - 1
            Pos(Egalités(l)) = NbPassage
            NbPassage = NbPassage + 1
        Next l
        QuickRanking = Pos(): Exit Function
    End If

    QuickRanking = TabTps()
    Exit Function

End If
```

```
' Classe les données dans l'ordre Décroissant:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
i = RefMini
For n = TabFin To TabDébut Step -1
    TabDonnées(n) = Mémo(i)
    i = Ref(i)
Next n

' S'il faut retourner le Classement où les égalités ont le même classement:
If ModeClassement = 1 Then
    i = RefMini: Anc = i: NbPassage = TabFin - TabDébut + 1
    For n = TabFin To TabDébut Step -1
        Pos(i) = NbPassage: NbPassage = NbPassage - 1
        If Mémo(i) = Mémo(Anc) Then Pos(i) = Pos(Anc)
        Anc = i
        i = Ref(i)
    Next n
    QuickRanking = Pos(): Exit Function
End If

' S'il faut retourner le Classement où les égalités distinguent l'ordre d'origine:
If ModeClassement = 2 Then
    i = RefMini: Anc = i: j = TabDébut: NbPassage = TabFin - TabDébut + 1
    For n = TabDébut To TabFin
        ReDim Preserve Egalités(TabDébut To j)
        Egalités(j) = i
        Anc = i
        i = Ref(i)
        j = j + 1
        If Mémo(i) <> Mémo(Anc) Or n = TabFin Then
            If j > TabDébut + 1 Then
                Call QuickSort(Egalités(), LBound(Egalités), UBound(Egalités))
                ' Plus rapide que Call QuickRanking(Egalités(), True, 0)
            End If
            For l = TabDébut To j - 1
                Pos(Egalités(l)) = NbPassage
                NbPassage = NbPassage - 1
            Next l
            j = TabDébut
        End If
    Next n
    QuickRanking = Pos(): Exit Function
End If

QuickRanking = TabTps()
End Function

'-------------------------------------------------------------------------------
'-------------------------------------------------------------------------------
```

```
'--------------------------------------------------------------------------------
Public Sub QuickSort(ByRef TabDonnées() As Variant, ByVal Gauche As Long, ByVal Droite As Long)
'--------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Variant, Pivot As Variant

i = Gauche
j = Droite
Pivot = TabDonnées((Gauche + Droite) / 2)

Do
    While Pivot > TabDonnées(i): i = i + 1: Wend
    While TabDonnées(j) > Pivot: j = j - 1: Wend

    If j + 1 > i Then
        Temp = TabDonnées(i)
        TabDonnées(i) = TabDonnées(j)
        TabDonnées(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QuickSort(TabDonnées(), Gauche, j)
If i < Droite Then Call QuickSort(TabDonnées(), i, Droite)

End Sub
```

Vous pouvez optimiser cet algorithme pour le traitement des entiers numériques en remplaçant les déclarations de variables « Variant » par « Long ».

Annexe 4 : L'algorithme QuickSort_AndRank (sur la base de l'algorithme QuickSort)

Permettant de retourner un ordre de classement.

```
'-------------------------------------------------------------------------------
Public Function QuickSort_AndRank(ByRef TabDonnées() As Variant, _
                Optional OrdreCroissant As Boolean = True, _
                Optional ModeClassement As Byte = 1, _
                Optional PivotFixe As Boolean = True) As Variant
'-------------------------------------------------------------------------------
Dim i As Long, Mini As Long, Maxi As Long
Dim Anc As Long, Classement As Long

Mini = LBound(TabDonnées)
Maxi = UBound(TabDonnées)
ReDim Ref(Mini To Maxi) As Long
ReDim Mémo(Mini To Maxi) As Variant
Dim Pos() As Long

' Mémorise les données avant de les trier:
For i = Mini To Maxi
    Ref(i) = i
    Mémo(i) = TabDonnées(i)
Next i

' Trie les données:
If OrdreCroissant = True Then
    Call QS(TabDonnées(), Ref(), Mini, Maxi, PivotFixe)
Else
    Call QSDEC(TabDonnées(), Ref(), Mini, Maxi, PivotFixe)
End If

' Alimente TabDonnées du Tri:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 0 Then
    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))
    Next i
    QuickSort_AndRank = Pos()
    Exit Function
End If

' Alimente TabDonnées du Tri et retourne un classement où les égalités ont le même classement:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 1 Then
    ReDim Pos(Mini To Maxi) As Long
    Anc = Mini
    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))
        Classement = Classement + 1
        Pos(Ref(i)) = Classement
        If TabDonnées(i) = TabDonnées(Anc) Then Pos(Ref(i)) = Pos(Ref(Anc))
        Anc = i
    Next i
    QuickSort_AndRank = Pos()
    Exit Function
End If

' Initialise les variables pour la méthode 2:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Dim Egalités() As Variant
Dim j As Long, l As Long, AncVal As Variant
ReDim Pos(Mini To Maxi) As Long
```

```
' Alimente TabDonnées dans l'ordre croissant du Tri et
' retourne un classement où les égalités distinguent l'ordre d'origine:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If OrdreCroissant = True Then

    ' Alimente TabDonnées du Tri et Retourne un classement où les égalités
    ' distinguent l'ordre d'origine:
    If ModeClassement = 2 Then

        j = Mini: AncVal = Mémo(Ref(j)):

        For i = Mini To Maxi
            TabDonnées(i) = Mémo(Ref(i))
            If TabDonnées(i) = AncVal Then ' Si nouvelle égalité alors:
                ReDim Preserve Egalités(Mini To j) ' Augmente la taille du tableau.
                Egalités(j) = Ref(i) ' Ajoute la donnée dans le tableau des égalités.
                j = j + 1            ' Compte le nombre d'égalités.
            Else ' Si plus d'égalité:
                If j > Mini + 1 Then ' Si plusieurs égalité alors:
                    Call QuickSort(Egalités(), LBound(Egalités), UBound(Egalités))
                End If
                For l = Mini To j - 1 ' Boucle sur les égalités:
                    Classement = Classement + 1  ' Incrémente le classement.
                    Pos(Egalités(l)) = Classement ' Indique le classement.
                Next l
                ' Place cette donnée dans le tableau des égalités:
                j = Mini
                ReDim Egalités(Mini To j)
                Egalités(j) = Ref(i)
                j = j + 1
            End If
            AncVal = Mémo(Ref(i)) ' Nouvelle référence de comparaison.
        Next i

        ' Vide le tableau des égalités:
        If j > Mini + 1 Then ' Si plusieurs égalité alors:
            Call QuickSort(Egalités(), LBound(Egalités), UBound(Egalités))
        End If
        For l = Mini To j - 1 ' Boucle sur les égalités:
            Classement = Classement + 1  ' Incrémente le classement.
            Pos(Egalités(l)) = Classement ' Indique le classement.
        Next l
    End If

    QuickSort_AndRank = Pos()
    Exit Function
End If
```

```
' Alimente TabDonnées dans l'ordre Décroissant du Tri et
' retourne un classement où les égalités distinguent l'ordre d'origine:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 2 Then

    j = Mini: AncVal = Mémo(Ref(j)):

    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))

        If TabDonnées(i) = AncVal Then ' Si nouvelle égalité alors:
            ReDim Preserve Egalités(Mini To j) ' Augmente la taille du tableau.
            Egalités(j) = Ref(i) ' Ajoute la donnée dans le tableau des égalités.
            j = j + 1            ' Compte le nombre d'égalités.
        Else ' Si plus d'égalité:
            If j > Mini + 1 Then ' Si plusieurs égalité alors:
                Call QuickSortDESC(Egalités(), LBound(Egalités), UBound(Egalités))
            End If
            For l = Mini To j - 1 ' Boucle sur les égalités:
                Classement = Classement + 1  ' Incrémente le classement.
                Pos(Egalités(l)) = Classement ' Indique le classement.
            Next l
            ' Place cette donnée dans le tableau des égalités:
            j = Mini
            ReDim Egalités(Mini To j)
            Egalités(j) = Ref(i)
            j = j + 1
        End If
        AncVal = Mémo(Ref(i)) ' Nouvelle référence de comparaison.
    Next i

    ' Vide le tableau des égalités:
    If j > Mini + 1 Then ' Si plusieurs égalité alors:
        Call QuickSortDESC(Egalités(), LBound(Egalités), UBound(Egalités))
    End If
    For l = Mini To j - 1 ' Boucle sur les égalités:
        Classement = Classement + 1  ' Incrémente le classement.
        Pos(Egalités(l)) = Classement ' Indique le classement.
    Next l
End If

QuickSort_AndRank = Pos()
End Function
'-------------------------------------------------------------------------------
'-------------------------------------------------------------------------------
```

```
'-------------------------------------------------------------------------------
Private Sub QS(ByRef TabDonnées() As Variant, ByRef Ref() As Long, _
               ByVal Gauche As Long, ByVal Droite As Long, ByVal PivotFixe As Boolean)
'-------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Long, ValQS As Variant

i = Gauche
j = Droite
If PivotFixe = True Then
    ValQS = TabDonnées(Ref((Gauche + Droite) / 2))
Else
    ValQS = TabDonnées(Ref(Gauche + (Rnd() * (Droite - Gauche))))
End If

Do
    While ValQS > TabDonnées(Ref(i)): i = i + 1: Wend
    While ValQS < TabDonnées(Ref(j)): j = j - 1: Wend

    If j + 1 > i Then
        Temp = Ref(i)
        Ref(i) = Ref(j)
        Ref(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QS(TabDonnées(), Ref(), Gauche, j, PivotFixe)
If i < Droite Then Call QS(TabDonnées(), Ref(), i, Droite, PivotFixe)

End Sub


'-------------------------------------------------------------------------------
Private Sub QSDEC(ByRef TabDonnées() As Variant, ByRef Ref() As Long, _
                  ByVal Gauche As Long, ByVal Droite As Long, ByVal PivotFixe As Boolean)
'-------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Long, ValQS As Variant

i = Gauche
j = Droite

If PivotFixe = True Then
    ValQS = TabDonnées(Ref((Gauche + Droite) / 2))
Else
    ValQS = TabDonnées(Ref(Gauche + (Rnd() * (Droite - Gauche))))
End If

Do
    While ValQS < TabDonnées(Ref(i)): i = i + 1: Wend
    While ValQS > TabDonnées(Ref(j)): j = j - 1: Wend

    If j + 1 > i Then
        Temp = Ref(i)
        Ref(i) = Ref(j)
        Ref(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QSDEC(TabDonnées(), Ref(), Gauche, j, PivotFixe)
If i < Droite Then Call QSDEC(TabDonnées(), Ref(), i, Droite, PivotFixe)

End Sub
```

Annexe 5 : Comparaisons des trois algorithmes

	QuickSort		QuickSort_AndRank			QuickRanking					
	Tri sans classement		Tri sans classement	Tri et classement Méthode 1	Tri et classement Méthode 2	Tri sans classement		Tri et classement Méthode 1		Tri et classement Méthode 2	
100 000 entiers :	(A)					(B1)		(B2)		(B3)	
Répartition totalement aléatoire	0,19	-2%	0,19	0,20	0,26	0,19	0%	0,20	0%	0,28	4%
Premiers 20% classées, puis répartition aléatoire	0,18	-1%	0,19	0,20	0,26	0,16	-14%	0,17	-12%	0,24	-7%
Répartition aléatoire sauf derniers 20% classées	0,18	-1%	0,18	0,20	0,26	0,19	2%	0,20	3%	0,27	5%
500 000 entiers :											
Répartition aléatoire, valeurs entre 1 et 10 000	1,02	-12%	1,17	1,26	1,88	1,25	7%	1,40	11%	2,00	7%
Répartition aléatoire, valeurs entre 1 et 100	1,12	-9%	1,22	1,35	2,33	1,10	-10%	1,23	-9%	2,07	-11%
100 000 alphabétiques de 1 à 100 caractères :											
Répartition totalement aléatoire	0,93	12%	0,83	0,87	0,92	0,74	-11%	0,79	-9%	0,81	-13%
Premiers 20% classées, puis répartition aléatoire	0,94	12%	0,84	0,87	0,98	0,65	-23%	0,69	-21%	0,71	-28%
Répartition aléatoire sauf derniers 20% classées	0,94	12%	0,84	0,87	0,92	0,72	-13%	0,77	-11%	0,79	-15%
Exemple de tris sur des répartitions aléatoires :											
Liste de 36680 communes de France	0,29	11%	0,26	0,28	0,30	0,23	-15%	0,24	-14%	0,25	-18%
100 000 alphabétiques de 20 caractères	0,82	0%	0,81	0,87	0,91	0,72	-12%	0,77	-12%	0,78	-14%
500 000 alphabétiques de 20 caractères	4,54	-6%	4,82	5,00	5,24	4,49	-7%	4,79	-4%	4,91	-6%
100 000 alphabétiques de 1 à 250 caractères	1,13	16%	0,97	1,00	1,08	0,83	-14%	0,88	-12%	0,89	-17%
500 000 alphabétiques de 1 à 250 caractères	6,55	12%	5,84	6,05	6,30	5,20	-11%	5,50	-9%	5,69	-10%
1 000 000 alphabétiques de 1 à 250 caractères	13,64	7%	12,70	13,05	13,23	11,51	-9%	11,79	-10%	11,96	-10%
100 000 alphabétiques de 1 000 caractères	1,95	13%	1,72	1,76	1,82	1,15	-33%	1,20	-32%	1,21	-33%

Méthode de classement 1 : les données égales ont le même ordre.
Méthode de classement 2 : l'ordre des données égales respecte l'ordre d'origine.
(A) : comparaison entre QuickSort et QuickSort_AndRank.
(B) : comparaison entre QuickRanking et QuickSort_AndRank : B1 = Tri sans classement, B2 = Tri et classement méthode 1, B3 = Tri et classement méthode 2.
QuickRanking n'utilise pas l'option d'analyses supplémentaires. Cette option n'étant pas utile sur les données aléatoires.

Annexe 6 : L'analyse d'un échantillon des données de la liste permet de déterminer s'il faut ou non activer l'option de tests complémentaires.

QuickRanking propose en option des analyses complémentaires pour accélérer les traitements sur les listes où les données ont de nombreuses égalités, ou lorsque les données se suivent.
Ces analyses sont performantes sur les listes partiellement classées ou s'il y a de nombreuses égalités, mais sont contre-productives sur les listes aléatoires.
C'est pourquoi ces analyses sont optionnelles dans QuickRanking afin de ne pas les exécuter sur les listes de données que vous savez aléatoires, et ainsi économiser du temps de traitement : Si l'argument NiveauTest vaut **False**, aucune analyse supplémentaire n'est réalisée. S'il vaut **True** des tests supplémentaires sont réalisés en se basant sur le dernier élément analysé : recherche des suites et des doublons pour un traitement immédiat, recherche de la borne minimale et maximale du tableau utilisé pour la recherche dichotomique.

Le dilemme se pose donc lorsque vous ne savez pas à l'avance si la liste des données à analyser sera ou non aléatoire. Car dans certains cas l'activation de l'option accélérera incroyablement les traitements, ou au contraire les ralentira fortement.
La solution que je propose, est de prendre un échantillon des données de la liste, pris au hasard, et de l'analyser d'abord avec l'option activée, puis avec l'option désactivée. En comparant la durée de ces deux traitements, l'on peut ainsi estimer s'il faut ou non activer l'option.
Plus la taille de l'échantillon est grande et plus le test sera représentatif, mais plus il prendra de temps. Ainsi, j'ai limité la taille de l'échantillon à 10% de la taille de la liste d'origine. Inversement si l'échantillon fait moins de 100 éléments, il est jugé non significatif et la fonction retourne **True** sans faire de test. Par défaut la taille de l'échantillon est 1/1000 de la liste.

```
'-------------------------------------------------------------------------------
' En VBA il faut Utiliser l'API QueryPerformanceCounter pour calculer la durée d'un traitement:
Declare Function QueryPerformanceCounter Lib "kernel32" (cyTickCount As Currency) As Long

'-------------------------------------------------------------------------------
Function TesterNiveau(MonTableau() As Variant, Optional ByVal PcEchantillon As Double = 0.1) As Boolean
'-------------------------------------------------------------------------------
Dim Début As Long, Fin As Long, TailleEchantillon As Long
Dim DT As Currency, DF As Currency, Durée As Currency, cyTicks As Currency
Début = LBound(MonTableau())
Fin = UBound(MonTableau())

' Initialisation des variables:
TailleEchantillon = (Fin - Début) * PcEchantillon / 100
Dim i As Long, l As Long

' Controle la taille de l'échantillon pris au hasard dans la liste:
If TailleEchantillon > Fin / 10 Then TailleEchantillon = Fin / 10
If TailleEchantillon < 100 Then TesterNiveau = True: Exit Function
ReDim MonTest(0 To TailleEchantillon) As Variant
Do
    i = Rnd() * Fin
Loop While i + TailleEchantillon > Fin

' Calcule la durée avec l'option à Vrai:
For l = 0 To TailleEchantillon: MonTest(l) = MonTableau(l): Next l
QueryPerformanceCounter DT
Classement = QuickRanking(MonTest(), True, 0, True)  ' Avec les Tests
QueryPerformanceCounter Durée
DT = Durée - DT

' Calcule la durée avec l'option à Faux:
For l = 0 To TailleEchantillon: MonTest(l) = MonTableau(l): Next l
QueryPerformanceCounter DF
Classement = QuickRanking(MonTest(), True, 0, False) ' Sans les Tests
QueryPerformanceCounter Durée
DF = Durée - DF

' Retourne Vrai si l'option Vrai est plus rapide, sinon retourne Faux:
If DF > DT Then TesterNiveau = True
End Function
'-------------------------------------------------------------------------------
```