ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

Algorithmique Avancée

Tri Rapide
(Quick sort)

Animé par : Dr. ibrahim GUELZIM
Email : ib.guelzim@gmail.com

Sommaire

- Rappels
  - Introduction et notions générales
  - Analyse et conception d'algorithmes
  - Complexité d'algorithmes classiques : 3 Tris de tableaux, 2 recherches dans un tableau, Schéma de Hörner
  - Preuves d'algorithmes
- Autres algorithmes de tri :
  - Tri par fusion
  - Tri par Tas
- Complexité moyenne :
  - **Application au Tri rapide**
  - Structures de Données Probabilistes :
    - Notions sur les Tables de Hachage et Fonctions de Hachage,
    - Bloom Filter,
    - Count Min Sketch
- Programmation dynamique
- Traitements de chaines de Caractères :
  - Recherche de chaine de caractères
  - Compression de données

Tri Rapide

- Exercice : Soit T un Tableau de n entiers naturels,
  - Écrire une solution qui met tous les éléments pairs au début de la liste et tous les éléments impairs à la fin de la liste
- Solution (python) :

```python
T =  [9, 6, 5, 13, 0, 10, 15, 22, 12]
indP = 0; indImp = len(T)-1
while indP < indImp:
    if T[indP]%2 == 0:  # si l'élément actuel est pair, passer au suivant
        indP += 1
    else:                # sinon, mettre l'élément actuel à la fin de la liste
        T[indP] , T[indImp] = T[indImp], T[indP]  # permuter
        indImp -= 1
print(T) # Affiche : [ 12, 6, 22, 10, 0, 15, 13, 5, 9 ]
```

- L'ordre interne des éléments pairs ou impairs n'est pas important

Tri Rapide

- Idée Simple & Astusieuse:

  Positionner chaque élément à sa place finale dans le tableau trié
  - Tri d'un tableau T de taille n : T[0 , n-1]
  - Plus généralement T[0 , n-1] = T[d , f] (d = 0 et f = n-1)

  - Choisir un élément du tableau : Pivot, noté x

  - Le choix est arbitraire

  - Notre Convention :
    - Dernier élément (x = T[f] = T[n-1])

4	1	7	8	3	2	10	9	**5**

    - OU, le premier dans d'autres conventions (x = T[d] = T[0])

  - Trouver la position finale du pivot (Lorsque le tableau sera trié)
  - Comment ?

Tri Rapide

- Q : Quelle est la position Finale du pivot T[f] ?
- Méthode appliquée au tableau T[d, f-1] (= T[0, n-2] de taille n-1) :
  - Range tous les éléments **inférieurs** ≤ au pivot au **début** du tableau
  - Range tous les éléments supérieurs ≥ au pivot à la fin du tableau

4	1	3	2	7	8	10	9	5

  - Mise du pivot à sa position finale : permutation

4	1	3	2	5	8	10	9	7

  - Méthode appelée : Partition (T, d, f)
  - Remarque : les éléments **inférieurs** et **supérieurs** ne sont pas forcéments triés
  - Ainsi, La position du pivot est finale et égale = **p**

Tri Rapide : Idée Partition :: positions éléments ≤ avant éléments ≥

4	1	7	8	3	2	10	9	5
4	1	7	8	3	2	10	9	5
4	1	7	8	3	2	10	9	5
4	1	7	8	3	2	10	9	5
4	1	7	8	3	2	10	9	5
4	1	3	8	7	2	10	9	5
4	1	3	2	7	8	10	9	5
4	1	3	2	7	8	10	9	5
4	1	3	2	7	8	10	9	5
4	1	3	2	5	8	10	9	7

- Une fois le pivot placé à la position p, on fait appel à la récursivité pour :
  - Le sous tableau à gauche T[d, p-1]
  - Le sous tableau à droite T[p+1, f]

Tri Rapide

- Pseudo Code :

```
Fonction TriRapide ( T , d , f )
Début
        Si f-d ≤ 0 alors
                retourner T
        Sinon
                x = T[ f ]
                p = partition ( T, d, f, x = T[f] )
                TriRapide ( T, d, p-1 )          // Tri de la partie à gauche du pivot
                TriRapide ( T, p+1, f )          // Tri de la partie à droite du pivot
        FinSi
Fin
```

Remarque : l'appel initial pour un tableau T de taille n est **TriRapide (T , 0 , n-1)**

Tri Rapide

- Caractéristiques / Tri Rapide :
  - In place : n'a pas besoin d'un espace auxiliaire comme le tri par fusion
  - Minimise le mouvement des données
  - Selon le principe Diviser pour régner

Tri Rapide

- Partitionnement du Tableau : Partition de Lamuto
  - Partitionner le tableau T[T[0] T[1] … T[n-2] x]
  - À l'itération j , le tableau est partitionné en 3 régions :

  [T[0] … T[i] T[i+1] … T[j-1] T[j] … T[n-2] x]

  - Région Inférieure RI ≤ x

  - Région Supérieure RS ≥ x

  - Région Non Traitée RN

  - Remarque : $x \notin RI$, $x \notin RS$, $x \notin RN$

Tri Rapide

- Partition de Lamuto :
  - Début par l'initialisation : RI = RS = Ø , RN = T[d , f-1] tels que:
    - i est la dernière position de RI , i = d-1
    - j première position de RN , j = i + 1
  - Si T est de taille n, on commence par d = 0 , f = n-1
  - $j^{ème}$ itération : T = [T[0] … T[i] T[i+1] T[i+2] … T[j-1] T[j] T[j+1] … T[n-2] T[n-1]]
    - L'élémént à traiter : T[j]
    - Si T[j] ≥ x (x = T[n-1]) :
      - T[j] doit ∈ RS
      - Tableau devient T : [T[0] … T[i] T[i+1] T[i+2] … T[j-1] T[j] T[j+1] … T[n-2] T[n-1]]
      - j = j + 1
    - Sinon (T[j] < x)
      - T[j] doit ∈ RI
      - Echanger T[j] et T[i+1]
      - Tableau devient T : [T[0] … T[i] T[j] T[i+2] … T[j-1] T[i+1] T[j+1] … T[n-2] T[n-1]]
      - i = i + 1 , j = j + 1

Tri Rapide

- Partition de Lamuto :
  - Réitérer tant que RN ≠ ∅
  - Après la dernière itération: RN = ∅ , RI et RS deviennent :
    T = [T[0] … T[i] T[i+1] T[i+2] … T[n-2] x]
  - Dernière opération : échanger x et T[i+1]
    T = [T[0] … T[i] x T[i+2] … T[n-2] T[i+1]]
    retourner la position p = i+1

Tri Rapide

- Partition de Lamuto : Pseudo Code

```
Fonction PartitionLamuto ( T, d , f)
Début
    x = T[ f ]  # pivot
    i = d - 1,
    j = i + 1
    TantQue ( j < f )    // RN ≠ Ø
        Si T [ j ] < x alors
            permuter ( T[ i + 1 ] ,  T [ j ] )
            i = i + 1
        FinSi
        j = j + 1
    FinTantQue
    Permuter (T[ i + 1 ] , T[ f ] )
    retourner i + 1
Fin
```

<u>Temps d'execution d'une itération de la partition pour un Tableau de taille n</u>

Faire n-1 fois :

comparaison avec x

permuter, si nécessaire

Tps(n) = O(n)

Tri Rapide

- Partition : solution 2

```
Fonction Partition ( T, d , f)
Début
    x = T[ f ] // pivot
    indInf = d, indSup = f-1
    TantQue (indInf ≤  indSup )
        Si T [indInf ] < x alors
            indInf +=  1
        Sinon
            permuter (T [indInf] , T [indSup])
            indSup -=  1
        FinSi
    FinTantQue
    Permuter (T[indSup + 1 ] , T[ f ] )
    retourner indSup + 1
Fin
```

Fin Partie 1

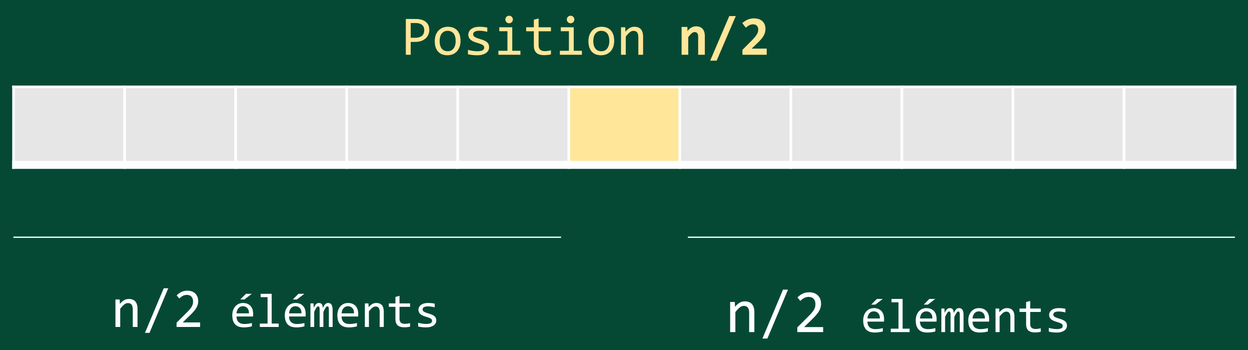
Tri Rapide

- Complexité (temporelle)
  - Cas 1 : partition balancée (position pivot au milieu)
    - $T(n) = T(n/2) + T(n/2) + \Theta(n)$

    → Déjà vu (cf tri par fusion)

    → $T(n) = O(n.Log(n))$

    → Meilleur des cas



Tri Rapide

- Complexité (temporelle)
  - Pire des cas :
    - T trié initialement
    - Tous les éléménts sont inférieures au pivot T[n-1]
    - Faire n fois la permutation de T[i+1] et T[j]
    - Partition est un O(n)
    - Exemple d'une itération

T = [1 2 4 5 6 8 15] // i = -1, j = 0, Permuter T[0] et T[0]
T = [1 2 4 5 6 8 15] // i = 0, j = 1, Permuter T[1] et T[1]
T = [1 2 4 5 6 8 15] // i = 1, j = 2, Permuter T[2] et T[2]
T = [1 2 4 5 6 8 15] // i = 2, j = 3, Permuter T[3] et T[3]
T = [1 2 4 5 6 8 15] // i = 3, j = 4, Permuter T[4] et T[4]
T = [1 2 4 5 6 8 15] // i = 4, j = 5, Permuter T[5] et T[5]
T = [1 2 4 5 6 8 15] // i = 5, j = 6, j == f → arrêt
T = [1 2 4 5 6 8 15] // i = 5, f = 6, Permuter T[f] et T[i+1],
// càd Permuter T[6] et T[6] → Pivot à sa position Finale

Tri Rapide

- Calcul de la Complexité temporelle
  - Pire des cas :
    - T trié initialement
    - Tous les éléménts sont inférieures au pivot T[n-1]
    - Faire n fois la permutation de T[i+1] et T[j]
    - Partition est un $\Theta(n)$

      T = [1 2 4 5 6 8 15]   // i = 6, j = 1, Pivot à sa position Finale

    - $Tps(n) = Tps(n-1) + \Theta(n) = Tps(n-1) + Cn$

      $= Tps(n-2) + \Theta(n-1) + Cn = Tps(n-2) + C(n-1) + Cn$

      $= Tps(n-3) + C[(n-2) + (n-1) + n]$

      $= \ldots$

      $= Tps(1) + C[1 + \ldots + (n-1) + n] = n(n+1)/2$

  - Pire des cas = $O(n^2)$
  - Questions :
    - est ce que le pire des cas est celui qui se produit le plus ?
    - est ce que le meilleur des cas est celui qui se produit le plus ?

Tri Rapide

- Calcul de la Complexité temporelle : Pire des cas
  - Exemple : Soit la fonction suivante :

```
Fonction Geom ( p )
Début
    Si aleat( ) < p alors                          // aleat() génére des nbr entre 0 et 1
            retourner 1 + Geom ( p )
    Sinon
            retourner 1
    FinSi
Fin
```

- Remarque :
  - la proba que aleat() < p est égale à p
  - la proba que aleat() ≥ p est égale à 1 - p
- Question : Quelle est la complexité pire des cas de la fonction Geom ?
- Réponse : Théoriquement = + ∞

→ Intérêt à calculer la complexité moyenne (dans le cas moyen)

Tri Rapide

- Calcul de la Complexité Moyenne :
  - Complexité moyenne de la fonction Geom

$$Tps(p) = p \,.\, [\,1 + Tps(p)\,] + (1 - p) \,.\, 1$$

$$Tps(p) \,.\, [\,1 - p\,] = p + 1 - p$$

$$Tps(p) = 1 / (1 - p)$$

Tri Rapide

myCompiler

Français Récent Login S'inscrire

Entrez un titre...

$$Tps(p) = 1 / (1 - p)$$ ✓

Python

Exécuter Enregistrer

```python
import random
def Geom ( p ):
    if (random.random() < p) :
        return 1 + Geom ( p )
    else:
        return 1

p = 0.8
k = 10
moy = 0

for k in range(k):
    nbr_it = Geom(p)
    print("nbr iteration pour p = ",p, " est :",nbr_it)
    moy = moy + nbr_it
moy = moy / k
print(f"\nnbr MOYEN d'iterations pour p = {p} est : {moy:.2f}")
```

Entrée du programme

Sortie du programme

```
nbr iteration pour p =  0.8  est : 5
nbr iteration pour p =  0.8  est : 9
nbr iteration pour p =  0.8  est : 7
nbr iteration pour p =  0.8  est : 6
nbr iteration pour p =  0.8  est : 1
nbr iteration pour p =  0.8  est : 10
nbr iteration pour p =  0.8  est : 2
nbr iteration pour p =  0.8  est : 5
nbr iteration pour p =  0.8  est : 1
nbr iteration pour p =  0.8  est : 1

nbr MOYEN d'iterations pour p = 0.8 est : 5.22
```

Tri Rapide

myCompiler   Français   Récent   Login   S'inscrire

Entrez un titre...

Tps(p) = 1 / (1 – p) ✓

Python   Exécuter   Enregistrer

```python
import random
def Geom ( p ):
    if (random.random() < p) :
        return 1 + Geom ( p )
    else:
        return 1

p = 0.95
k = 10
moy = 0

for k in range(k):
    nbr_it = Geom(p)
    print("nbr iteration pour p = ",p, " est :",nbr_it)
    moy = moy + nbr_it
moy = moy / k
print(f"\nnbr MOYEN d'iterations pour p = {p} est : {moy:.2f}")
```

Entrée du programme

Sortie du programme

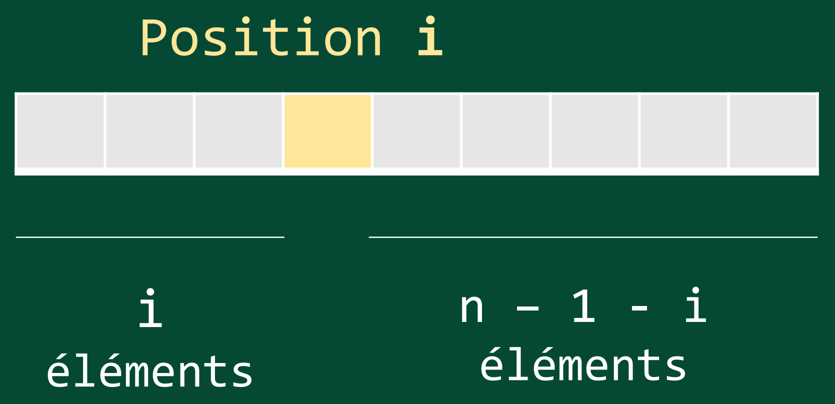
```
nbr iteration pour p =  0.95  est : 28
nbr iteration pour p =  0.95  est : 18
nbr iteration pour p =  0.95  est : 30
nbr iteration pour p =  0.95  est : 11
nbr iteration pour p =  0.95  est : 5
nbr iteration pour p =  0.95  est : 10
nbr iteration pour p =  0.95  est : 8
nbr iteration pour p =  0.95  est : 20
nbr iteration pour p =  0.95  est : 11
nbr iteration pour p =  0.95  est : 38

nbr MOYEN d'iterations pour p = 0.95 est : 19.89
```

Tri Rapide

- Complexité moyenne du Tri Rapide Tr(n) =
  - Tr(n - 1) + T(0) + Θ(n),   proba = 1 / n
  - Tr(n - 2) + T(1) + Θ(n),   proba = 1 / n
  - Tr(n - 3) + T(2) + Θ(n),   proba = 1 / n
  - …
  - Tr(n - 1 - i) + T(i) + Θ(n),   proba = 1 / n
  - …
  - Tr(0) + T(n - 1) + Θ(n) ,   proba = 1 / n

Remarque:

Θ(n) coût pour que la partition mette le pivot à la position i



Tri Rapide

- Calcul de la Complexité Moyenne : (on remplace Θ(n) par C.n)
  - Tr(n) = $\frac{1}{n} \sum_{i=1}^{n} [\, Tr(i-1) + Tr(n-i) + C.n \,]$
  - Tr(n) = $\frac{2}{n} \sum_{i=0}^{n-1} \mathrm{Tr(i)} + \frac{1}{n}$. (n.C.n)

    = $\frac{2}{n} \sum_{i=0}^{n-1} \mathrm{Tr(i)}$ + C.n

A : n . Tr(n) = 2 [Tr(0) + … + Tr(n - 1)] + C . n²,
B : (n – 1) . Tr(n - 1) = 2 [Tr(0) + … + Tr(n - 2)] + C . (n – 1)²,

A – B : n.Tr(n) - (n – 1) . Tr(n – 1) = 2 . Tr(n – 1) + C . (2n – 1),
n.Tr(n) = (n + 1) . Tr(n – 1) + C . (2n – 1),
Tr(n) = $\frac{n+1}{n}$. Tr(n – 1) + C . (2 – 1/n), //On peut remplacer 2C ← C.(2 – 1/n)
Tr(n) = $\frac{n+1}{n}$. Tr(n – 1) + 2.C,

Tri Rapide

- D'une manière récursive

$$\mathrm{Tr}(n) = \frac{n+1}{n} \,.\, \mathrm{Tr}(n-1) + 2C,$$

$$\mathrm{Tr}(n) = \frac{n+1}{n} \,.\, \left[\frac{n}{n-1} \,.\, \mathrm{Tr}(n-2) + 2C \right] + 2C,$$

$$= \frac{n+1}{n-1} \,.\, \mathrm{Tr}(n-2) + \frac{n+1}{n} \,.\, 2C + 2C,$$

$$= \frac{n+1}{n-2} \,.\, \mathrm{Tr}(n-3) + \frac{n+1}{n-1} \,.\, 2C + \frac{n+1}{n} \,.\, 2C + 2C,$$

$$= \frac{n+1}{n-j+1} \,.\, \mathrm{Tr}(n-j) + \sum_{i=0}^{j-1} \frac{n+1}{n-i} \,.\, 2C + 2C,$$

…

$$= \frac{n+1}{n-n+1} \,.\, \mathrm{Tr}(n-n) + \sum_{i=0}^{n-1} \frac{n+1}{n-i} \,.\, 2C + 2C, \qquad \text{or } T(0) = 0$$

$$= \sum_{i=0}^{n-1} \frac{n+1}{n-i} \,.\, 2C + 2C,$$

Tri Rapide

$$Tr(n) = \sum_{i=0}^{n-1} \frac{n+1}{n-i} . 2C + 2C,$$

$$= 2C.(n+1)\left[\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1\right] + 2C$$

Or $\left[\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1\right] = Log(n) + \beta$, tq $\beta > 0$

$= \Theta(Log(n))$

- $Tr(n) = 2C.(n+1) . \Theta(Log(n)) + 2C$

$= \Theta(\mathbf{n . Log(n)})$ = Complexité Moyenne du Tri Rapide

References

- Introduction à l'algorithmique. Cours et exercices. Cormen et al. 2e édition. (En 3rd Edition)

- Algorithms, FOURTH EDITION, Robert Sedgewick and Kevin Wayne. Princeton University.

- https://jeffe.cs.illinois.edu/teaching/algorithms/