

République Algérienne Démocratique Et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université 20 Août 1955 Skikda
Faculté Des Sciences
Département D'informatique



Rapport de Algorithmique et Complexité Des App :Master 1
Professionnelle Option : Génie Logiciel
et Application Avancée

THEME



A LGORITHMES DE T RI

Réalisé Par :

❖ Boulkenafet Samir

Encadré par :

❖ Pr.Bouchehame.B

Sommaire

Introduction	3
1-Tri par Sélection.....	4
2-Tri par insertion.....	4
3-Tri par propagation.....	5
4- Tri par fusion.....	6
5-Tri par Rapide.	7
6-Tri par ABR.....	8
7-Tri par TAS.....	9
8-Tri de shell (Shellsort)	10
9-Tri par base (Radix sort).....	11
10-Tri de paquets (bucket sort).....	12
11-Tri par comptage (counting sort).....	12
12- Tri par peigne(comb sort).....	13
13- Tri cocktaile(shaker sort).....	14
14-Tri pair-impair(odd-even-sort).....	15
15- Tri stupide (Bogosort).....	16
Conclusion	17
Bibliographie	

Introduction:

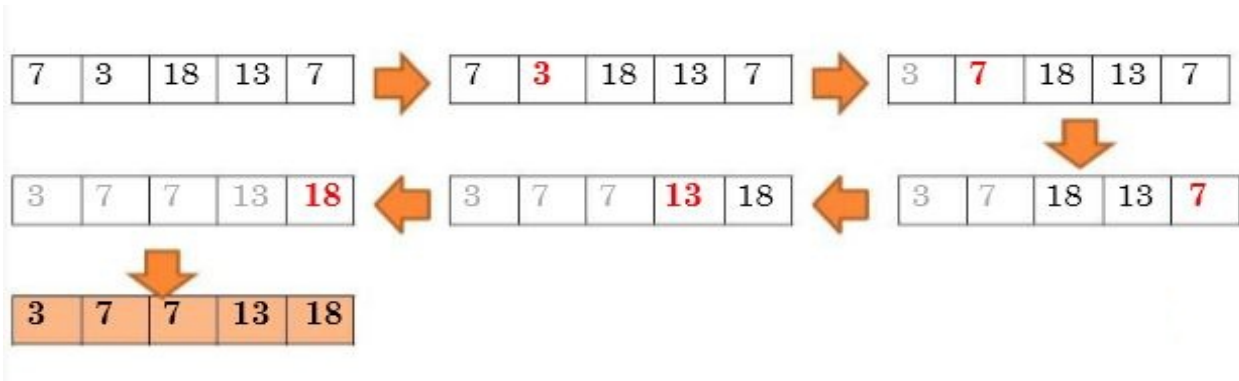
Étant donné un tableau d'entiers T (n : sa taille), l'algorithme du tri permet d'organiser les éléments du tableau selon un ordre déterminé (croissant par exemple). Plusieurs algorithmes de tri existent: tri par sélection, par insertion, par bulle, par fusion, rapide, par ABR et par TAS et ect.....

L'objectif de ce rapport est de concevoir de quelque algorithmes de tri , ensuite, de calculer leur complexité.

T RI PAR S ÉLECTION

- PRINCIPE

Le principe est de rechercher la plus petite valeur et la placer au début du tableau, puis la plus petite valeur dans les valeurs restantes et la placer à la deuxième position et ainsi de suite...



Algorithme :

i, j: entier ; tmp, small : entier ; t : tableau entier [n] ;

Début

Pour i de 1 à n-1 faire

small \leftarrow i;

Pour j de i+1 à n faire

Si $t[j] < t[\text{small}]$ alors

small \leftarrow j ;

Fin si

Fin pour j

tmp \leftarrow t[small];

t[small] \leftarrow t[i];

t[i] \leftarrow tmp;

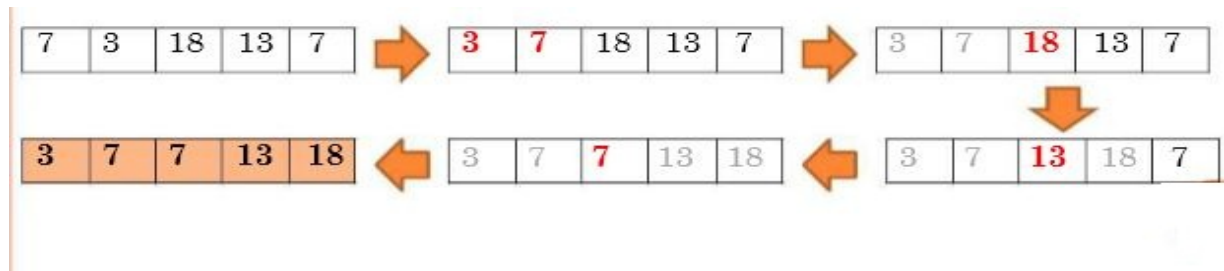
Fin pour i

Fin

T RI PAR I NSERTION

- PRINCIPE

Le principe est d'ordonner les deux premiers éléments et d'insérer le 3 e élément de manière à ce que les 3 premiers éléments soient triés, ensuite d'insérer le 4 e élément à sa place et ainsi de suite. A la fin de la i e itération, les i premiers éléments de T sont triés et rangés au début du tableau T'



Algorithme :

i, j: entier ; var: entier ; t : tableau entier [n] ;

Début

Pour i de 0 à n-1 faire

var ← t[i];

j ← i-1;

tant que(j <= 0 et t[j] > var)

t[j+1] ← t[j];

j ← j-1;

Fin tant que

t[j+1] ← var;

Fin pour i

Fin

TRI PAR PROPAGATION

• PRINCIPE

L'algorithme de tri par propagation (ou à bulles) consiste à faire remonter progressivement les plus grands éléments d'un tableau. Son principe est d'inverser deux éléments successifs s'ils ne sont pas classés dans le bon ordre et de recommencer jusqu'à ce qu'on ne peut plus permuter.



Algorithme :

Procédure tri_Bulle (tab : tableau entier [N]) i, k : entier ; tmp : entier)

Pour i de N à 2 faire

Pour k de 1 à i-1 faire

Si (tab[k] > tab[k+1]) alors

tmp ← tab[k];

tab[k] ← tab[k+1];

tab[k+1] ← tmp;

Fin si

Fin pour

Fin

T RI PAR F USION

- **PRINCIPE**

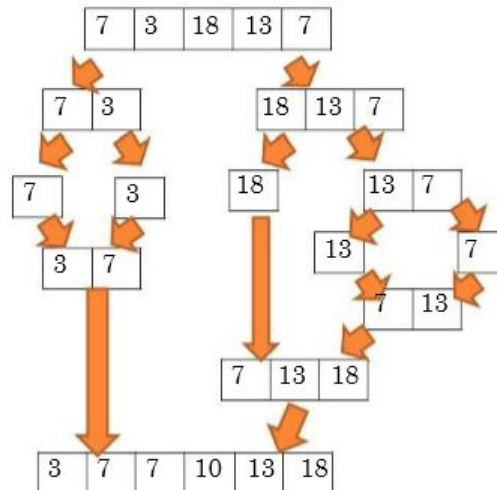
Le principe est de trier deux tableaux de taille $N/2$ et ensuite de les fusionner de sorte à ce que le tableau final soit trié.

DIVISER: diviser le tableau en deux tableaux:

$T[\text{debut}..\text{fin}] = T1[\text{debut}..\text{milieu}] + T2[\text{milieu}+1..\text{fin}]$

REGNER: trier (par fusion) les deux tableaux

COMBINER: combiner les 2 tableaux de telle manière que le tableau T reste trié



Algorithme :

Procédure trifusion ($T1, T2: \text{Tab}; N1, N2: \text{Entier}; \text{Var } T: \text{Tab}$)

$i \leftarrow 1; j \leftarrow 1; k \leftarrow 0;$

Répéter

$k \leftarrow k + 1;$

Si ($T1[i] < T2[j]$)

alors $T[k] \leftarrow T1[i]; i \leftarrow i + 1;$

sinon $T[k] \leftarrow T2[j]$

$j \leftarrow j + 1;$

Fin Si

jus'qu'a ($i > N1$ OU $j > N2$)

Si ($i > N1$) Alors

pour c de j à $N2$ Faire

$k \leftarrow k + 1;$

$T[k] \leftarrow T2[c];$ Fin pour;

Sinon pour c de i à $N1$ Faire $k \leftarrow k + 1;$

$T[k] \leftarrow T1[c];$ Fin pour; Fin Si; Fin;

T R I R A P I D E

- PRINCIPE

Le principe est de choisir une valeur dans le tableau appelée pivot (par exemple la première valeur du tableau) et de déplacer avant elle toutes celles qui lui sont inférieures et après elle toutes celles qui lui sont supérieures. réitérer le procédé avec la tranche de tableau inférieure et la tranche de tableau supérieure à ce pivot.

DIVISER: Diviser le tableau en deux tableaux selon le pivot choisi : T1[debut..pivot] et T2[pivot+1..fin]

REGNER: trier (par trie rapide) les deux tableaux

COMBINER: combiner les 2 tableaux:

T [debut..fin] = T1[debut..pivot] + T2[pivot+1..fin] est trié

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

15, 10, 19, 13, 5, 3, 12, 7, 16, 20, 35, 40, 25, 38
à trier! à trier!

Algorithme :

Tri_Rapide (T, deb, fin)

Si (deb < fin) alors

inter =Partionner (T, deb, fin);

Tri_Rapide (T, deb, inter) ;

Tri_Rapide (T, inter+1, fin);

Fin si.

Partionner (T, deb, fin)

x = T(deb) ;i = deb-1 ; j= fin+1

Tant que (1)

Répéter { j=j-1; } Jusqu'à T(j) <= x

Répéter { i =i+1 ;} Jusqu'à T(i) >= x

si (i < j) **permuter** (T(i), T(j))

sinon retourner j ;

Fin Si.

Fin.

TRI PAR ABR

- Définition

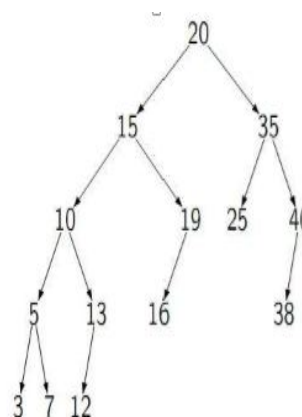
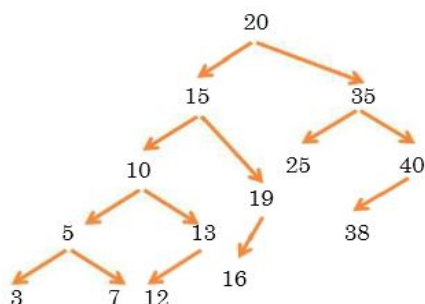
Un Arbre Binaire de Recherche (ABR) est un arbre binaire, dans lequel chaque noeud contient un entier en respectant la propriété suivante :

- Inférieur (ou égal) aux entiers de son sous-arbre gauche
- Supérieur strictement aux entiers de son sous-arbre droit

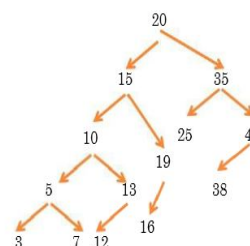
- PRINCIPE

1. Insérer toutes les éléments du tableau dans un ABR

20	15	10	35	19	13	5		3	12	7	16	40	25	38		
----	----	----	----	----	----	---	--	---	----	---	----	----	----	----	--	--



2. Parcourir l'ABR en ordre infixe: fils gauche, noeud, fils droit



Algorithme :

Tri_ABR(T:Tableau, n: entier, AR: noeud)

Debut

SI (n>0) alors

Insérer (ARB, T[n])

Tri_ABR(T, n-1, AR)

SINON n=0; // l'arbre est construit en totalité

Parcours_Infixe (AR,T);

FIN SI

Fin

Insérer (AR: noeud , x: entier): noeud

debut

SI (AR=null) alors //L'arbre est vide

AR Noeud(x,null,null) // créer la racine (le premier noeud)

SINON

SI x<=valeur(AR) alors

AR.FG Insérer(x, FG(AR)) // Insérer à gauche

SINON

AR.FD Insérer(x, FD(AR)) // Insérer à droite

FSI Insère AR Fin.

3	5	7	10	12	13	15	16	19	20	25	35	38	40		
---	---	---	----	----	----	----	----	----	----	----	----	----	----	--	--

Soit indice une variable globale initialisé à 1

Parcours_Infixe (AR: noeud, T: Tableau)

Debut

Si (ARR#null) alors //Arbre n'est pas vide

Parcours_Infixe(FG(AR,T, indice))

T[indice] :=valeur (AR) //Écrire la valeur dans le tableau

Indice :=indice + 1;

Parcours_Infixe(FD(ARR), T, indice) FFSI FFin

T RI PAR TAS

- Définition

Un TAS un arbre binaire qui vérifie les deux propriétés suivantes :
propriété structurelle: arbre binaire complet (ou parfait), Tous les niveaux sont totalement remplis sauf le dernier qui est rempli de la gauche vers la droite

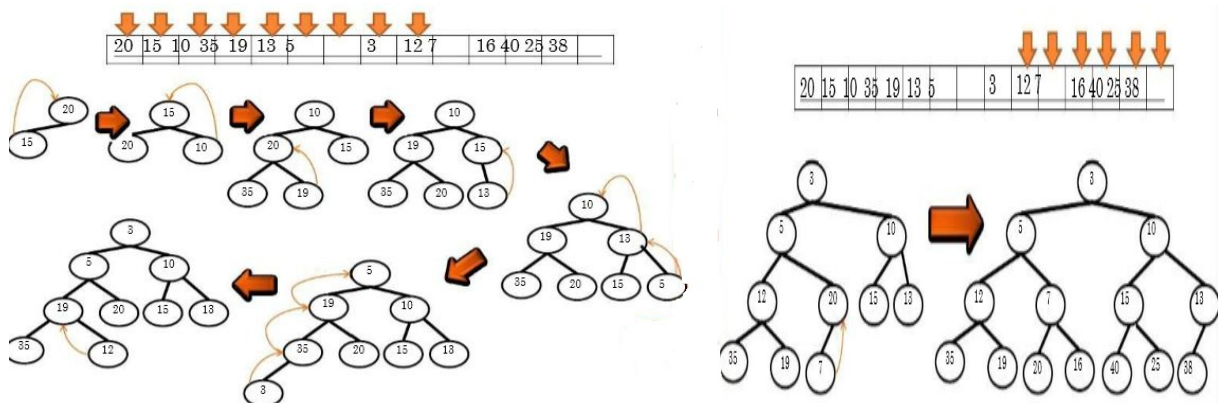
Propriété d'ordre :

TAS min : valeur (père) valeur (fils)

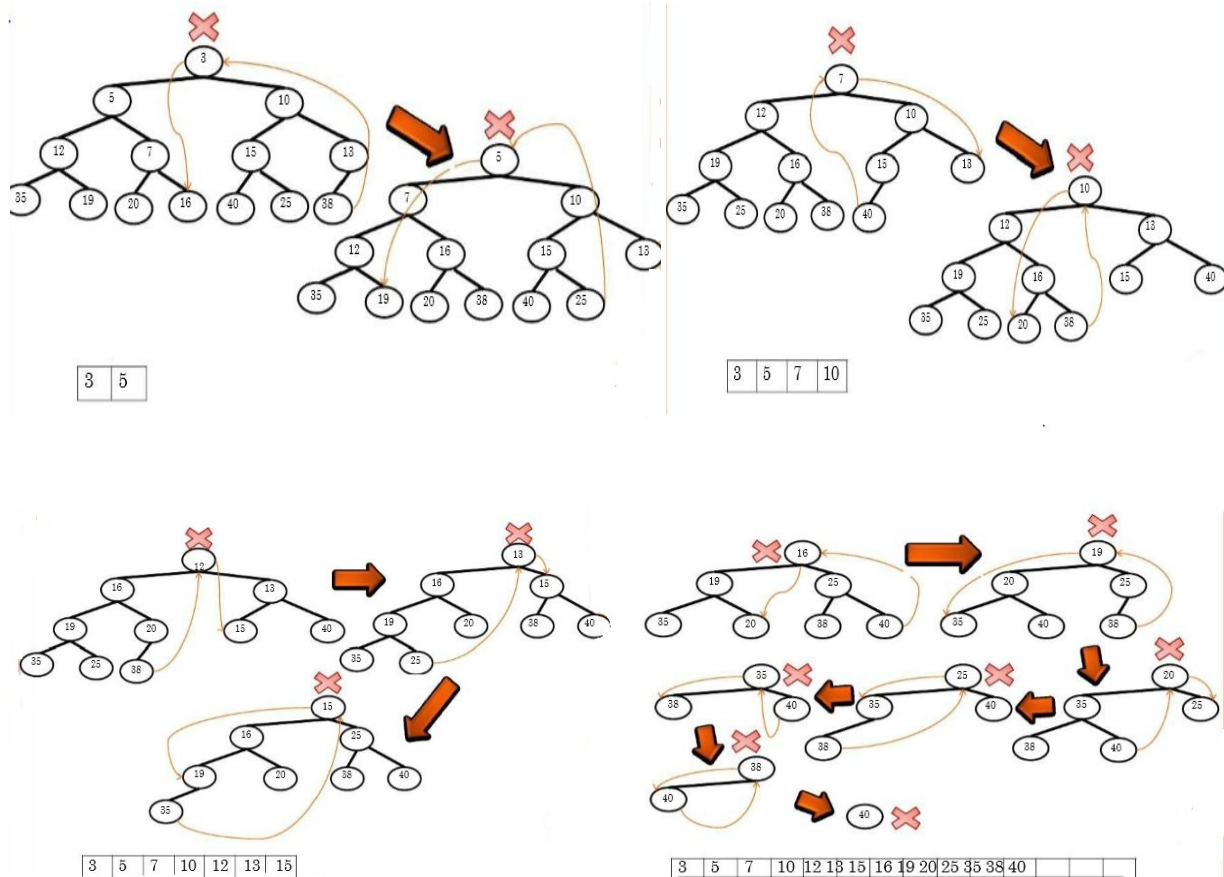
TAS max : valeur (père) valeur (fils)

- PRINCIPE

1. transformer le tableau en un TAS_MIN



2. Extraire n fois le minimum du tas:



Algorithme :

```

i ,phase:=1;
Tri_TASmin (T, TAS: Tableau, n, i, phase: entier)
Début
Si phase = 1 alors //construire le TAS
Si (i<=n) alors
Insérer_TAS (Tas, i-1,T[i])
Tri_TASmin (T, TAS, i++, n, phase)
Sinon
Tri_TASmin(T, TAS, 0, n, 2) // On passe à
la phase 2
Sinon // Phase 2: Extraire les minimums
Si (i<n) alors
T[i+1]:=TAS[i]
Extraire_Minimum (TAS, n-i)
Tri_TASmin (T, TAS, i++, n, phase)
Fsi
Fin

```

Procédure **Insérer_TAS** (Tas: tableau, n, x: entier)

```

Début
n :=n+1
i := n
Tas[i ] x
Tantque (i/2 > 0 et Tas[i/2] > x)
faire
permuter (Tas, i, i/2)
i :=i/2
Fin.

```

Extraire_Minimum (Tas: tableau, n : entier)

```

Début
Tas [1]:= T[n];
min 1; Sortie vrai
TQ (non sortie) faire
i :=min;g :=2*i ;d:=2*i+1;
Si g < n et Tas[g] < Tas[min] alors
min g;
Si d < n et Tas[d] < Tas[min] alors
min d;
Si min i alors Permuter (Tas, i, min)
Sinon Sortie vrai;
Fin

```

T RI PAR SHell

- Définition

-C'est une variante du tri par insertion

-Shell propose une suite définie par : $u_1=1$ et $u_{n+1}= 3*n +1$ pour déterminer la valeur du pas.

-trie chaque liste d'éléments séparés par p positions chacun avec un tri par insertion.)

72	61	44	80	70	85	21	23	51	87	74	94	20	17	56
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Algorithme :

DEF PROC TRI_SHELL (var T : tab ; n : entier)

P←0

Tant que p>0 faire

P ←3*p+1 ;

Fin Tant que

P←p div 3;

Pour i de p à n faire

Aux←T[i];

J← i ;

Tant que (j>p-1) et (T[j-p]>aux) Faire

T[j] ←T[j-p] ;

j←j-p;

Fin Tant que

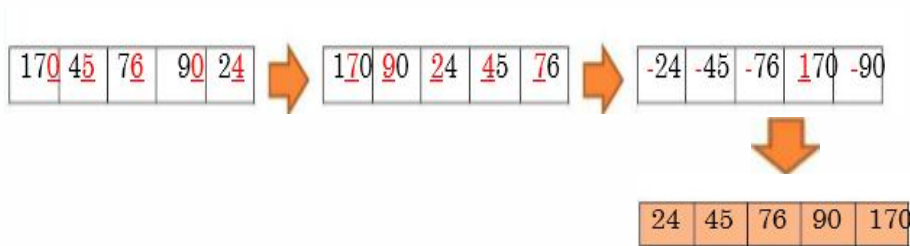
T[j]←aux ;Fin pour Fin Tant que Fin.

T RI PAR BAS (Radix Sort)

- Définition

-cet algorithme est utilisé seulement pour assortir le nombre

-nous assortissons les nombres chiffre moins considérable à la plupart du chiffre considérable



Algorithme :

```
procedure radixsort(int arr[],int n)
```

```
int m =getmax(arr,n);//trouve les maxuimum comptent pour savoir nombre de chiffres .
```

```
pour (int exp=1;m/exp>0;exp*=10)
```

```
countSort(arr,n,exp);//le boucle applique countsort au chiffre énième des éléments.
```

```
Fin Pour
```

```
Fin
```

T RI PAR PAQUETS (Bucket Sort)

- Définition

In ces paquets de l'algorithme du triage sont créés pour mettre des éléments dans eux. Alors nous appliquons quelque algorithme du triage (sorte de l'insertion) pour assortir les éléments dans chaque paquet.

finallement sortez et joignez-les devant être assortis la collection

0.78	0.17	0.39	0.26	0.72	0.94	0.21	0.12	0.23	0.68
------	------	------	------	------	------	------	------	------	------



tab[0] à paquet7
tab[1] à paquet1
tab[2] à paquet3
tab[3] à paquet2
tab[4] à paquet7
tab[5] à paquet9
tab[6] à paquet2
tab[7] à paquet1
tab[8] à paquet2
tab[9] à paquet6

0.12	0.17	0.21	0.23	0.26	0.39	0.68	0.72	0.78	0.94
------	------	------	------	------	------	------	------	------	------

tableau sortir

Algorithme :

```
procedure bucketSort(float arr[], int n)
{
vector<float> b[n]; //creat n paquet vides
for (int i=0; i<n; i++){
int bi = n*arr[i]; //mettez des éléments de la collection dans diffrent paquets
b[bi] .push_back( arr[i]);
for (int i=0; i<n; i++){ //paquet de l'indivdual de la sorte
sort(b[i].begin(), b[i].end());
int index = 0; } //enchaînez tout le paquets dans arr []
for (int i = 0; i < n; i++){
for (int j = 0; j < b[i].size(); j++) {
arr[index++] := b[i][j];}
Fin
}}
```

T RI PARCompatage (Counting Sort)

- Définition

Le tri par compatage est une technique du triage basée sur les clefs entre une gamme spécifique

le fonctionne en comptant le nombre d'objets qui ont des valeurs clés distinctes. faire alors quelque arithmétique pour calculer la place de chaque objet dans la séquence de la production

-1 pour simplicité, considérez des données dans la gamme de 0 à 9

1	4	1	2	7	5	2
---	---	---	---	---	---	---

-2 créez une tableau du compte pour entreposer le compte de chaque objet unique

index	0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0

-3 comptez chaque élément dans le tableau donné et placez le compte l'index approprié

0	1	2	3	4	5	6	7	8	9
0	2	2	0	1	1	0	1	0	0

-4 modifiez le tableau du compte en ajoutant les comptes antérieurs

0	1	2	3	4	5	6	7	8	9
0	2	2	0	1	1	0	1	0	0
<div><div></div><div>2 + 2</div></div>									

0	1	2	3	4	5	6	7	8	9
0	2	4	4	5	6	6	7	7	7

5-depuis que nous avons sept entrée nous créons une collection avec sept places. les valeurs correspondantes représentent les places dans le tableau du compte .

	1	4	1	2	7	5	2		
0	1	2	3	4	5	6	7	8	9
0	2	4	4	5	6	6	7	7	7
1	2	3	4	5	6	7			
	1								

-nous plasons les objets dans leur position correct et diminuons le compte par un

1	2	3	4	5	6	7
1	1	2	2	4	5	7

Algorithme :

```

procedur countSort(char arr[]) {
char outputl [strlen(arr)];
int count[RANGE+1 ], i;
memset(count, 0, sizeof(count));
for(i = 0; arr[i]; ++i)//compte les exemples
++count[arr[i]];
for(i=1;i<=RANGE;++i){//modifes il dans compte de la somme
count[i]+=count[i-1];}
for(i=0;arr[i];++i){//place les objets à sa place correcte et diminue le compte par un
output[count[arr[i]-1];
-count[arr[i]];
}
for(i=0;arr[i];++i)//copiez la collection de la production à arr afin que l'arr contienne
maintenant des caractères assortis
arr[i]=output[i]
}

```

T RI PAR PEIGNE (Comb Sort)

- Définition

la sorte du peigne améliore sur sorte de la bulle qui utilise intervalle de dimension plus que 1. l'intervalle commence avec une grande valeur et se rétrécit par un facteur de 1,3 dans chaque itération atteint la valeur 1.

1-laissez les éléments de la tableau être :

8	4	1	56	3	-44	23	-6	28	0
---	---	---	----	---	-----	----	----	----	---

2-intervalle = $10/1,3=7$,indice = 1,échangé T[1] et T[7]

-6	4	1	56	3	-44	23	8	28	0
----	---	---	----	---	-----	----	---	----	---

3-continué la comparaison de tous les indices avec intervalle 7

-6	4	0	56	3	-44	23	8	28	1
----	---	---	----	---	-----	----	---	----	---

4-a chaque fois intervalle suivante == le résultat de intervalle précédente

7/1,3=5

5/1,3=3

3/1,3=2

2/1,3=1

5-continué la comparaison de tous les indices avec tous les intervalle jusqu'à le tableau trié

-44	-6	0	1	3	4	8	23	28	56
-----	----	---	---	---	---	---	----	----	----

Algorithme :

```
procedur combSort(int a[], int n)
```

```
{
```

```
int gap= n;
```

```
bool swapped = true;
```

```
while (gap!= 1 || swapped == true)
```

```
{
```

```
//continuez à courir pendant que l'intervalle est plus que 1 et dernière itération a causé un échange
```

```
gap= getNextGap(gap);
```

```
swapped=false;
```

```
for(int i=0;i<n-gap;i++)//comparez tous les éléments avec intervalle courant
```

```
{
```

```
if(a[i]>a[i+gap])
```

```
{
```

```
swap(a[i],a[i+gap]);
```

```
swapped=true;
```

```
}
```

```
}
```

```
}
```

TRI COCKTAIL

- **PRINCIPE**

Son principe est identique à celui du tri à bulles, sauf qu'il change de direction à chaque passe. C'est une légère amélioration car il permet non seulement aux plus grands éléments de migrer vers la fin de la série mais également aux plus petits éléments de migrer vers le début.

Algorithme :

```
PROCEDURE tri_shaker ( TABLEAU a[1:n])
sens ← 'avant', debut ← 1, fin ← n-1, en_cours ← 1
REPETER
permut ← FAUX:
REPETER
SI a[en_cours] > a[en_cours + 1] ALORS
echanger a[en_cours] et a[en_cours + 1]
permut ← VRAI;
FIN SI
SI (sens='avant') ALORS
en_cours ← en_cours + 1 ;
SINON
en_cours ← en_cours - 1;
FIN SI
TANT QUE ((sens='avant') ET (en_cours<fin)) OU ((sens='arriere') ET
(en_cours>debut))
SI (sens='avant') ALORS
sens ← 'arriere';
fin ← fin - 1;
SINON
sens ← 'avant';
debut ← debut + 1;
FIN SI
TANT QUE permut = VRAI;
FIN PROCEDURE
```

T RI PAIR-IMPAIR**• PRINCIPE**

Son principe est basé sur le tri à bulles, Il opère en comparant tous les couples d'éléments aux positions paires et impaires consécutives dans une liste et, si un couple est dans le mauvais ordre (le premier élément est supérieur au second), il en échange les deux élément.

Algorithm

```
PROCEDURE tri pair-impair(list x[])
trié = faux;
tant que non trié
trié = vrai; //comparaisons impaires-paires :
pour ( x = 1; x < list.length-1; x += 2)
si list[x] > list[x+1]
échanger list[x] et list[x+1];
trié = faux; Fin Si Fin Pour.
```

```
//comparaisons paires-impaires :  
( x := 0; x < list.length-1; x += 2)
```

```
si list[x] > list[x+1]  
échanger list[x] et list[x+1];  
trié = faux; Fin si Fin Tan Que Fin.
```

T RI STUPIDE

- **PRINCIPE**

Il est présenté pour des raisons pédagogiques, par comparaison aux méthodes de tri traditionnelles, ou comme exercice.

Le tri stupide consiste à vérifier si les éléments sont ordonnés et s'ils ne le sont pas à mélanger aléatoirement les éléments, puis à répéter l'opération.

Algorithme :

```
fonction tri_stupide (liste)
```

```
    tant que la liste n'est pas triée
```

```
        mélanger aléatoirement les éléments de la liste
```

Conclusion:

Algorithme de Tri	Complexité au Pire	Stable
Tri par Sélection	$T(n) = O(n^2)$	non
Tri par insertion	$T(n) = O(n^2)$	oui
Tri par ABR	$T_{ARB}(n) = O(n^2)$	oui
Tri par TAS	$O(T_{TAS}) = O(n \log^2(n))$	non
Tri par Rapide	$T(n) = O(n \log n)$	non
Tri par propagation	$T(n) = O(n^2)$	non
Tri par fusion	$T(n) = O(n \log^2 n)$	oui
Tri Shell	$T(n) = O(n \log^2(n))$	non
Tri base	$T(n) = O(n)$	oui
Tri comptage	$T(n) = O(n)$	oui
Tri peigne	$T(n) = O(n^2)$	non
Tri pair-impair	$T(n) = O(n^2)$	oui
Tri cocktail	$T(n) = O(n^2)$	oui
Tri paquets	$T(n) = O(n)$	oui
Tri stupide	$T(n) = O(n)$	non

Bibliographie

- [1] Frédéric Vivien, Algorithmique avancée, École Normale Supérieure de Lyon, 2002.,pp. 93. Disponible sur <http://perso.ens-lyon.fr/frederic.vivien/Enseignement/Algo-2001-2002/Cours.pdf>
- [2] Renaud Dumont, Algorithmique P2, HeapSort et files de priorité, 2010, pp 31,Disponible sur www.montefiore.ulg.ac.be/~dumont/pdf/ac7.pdf
- [3] François Laroussinie Algorithmes de tri, Université Paris Diderot (Paris 7), 2010, pp * 110. Disponible sur www.liafa.jussieu.fr/~francoisl/IREM/tri.pdf
- [4]slim masfar Algorithmique et Complexité,2012 pp 104 disponible sur <http://p835.phpnet.org/testremorque/upload/catalogue/coursalgorithmi.pdf>
- [5] oliver Bournez cour 7 arbre de recherche TAS disponible sur <http://www.enseignement.polytechnique.fr/informatique/INF421/Amphi-b/cours7-handout2x2.pdf>
-