

## Partie 8 : Tri itératif de tableau

Hypothèses de travail :

- ▶ On manipule des entiers.
- ▶ On cherche à ordonner une séquences de N entiers.
- ▶ On considère que ces entiers sont dans un tableau.
- ▶ On considère que l'ordre recherché est l'ordre croissant.

AlgoProg 2015-16 183

## Tri itératif de tableau

Critères d'analyses des algorithmes :

- ▶ Complexité en temps
- ▶ Complexité en mémoire
- ▶ Stabilité

◦ Exemple :

	0	1	2		i				N-1	
T	2	5	1	3	6	1	3	5	3	2
	1	1	2	2	3	3	3	5	5	6

AlgoProg 2015-16 184

## Tri itératif de tableau

Idées :

	0	1	2		i				N-1	
T	2	5	1	3	6	1	3	5	3	2

AlgoProg 2015-16 185

## Tri itératif de tableau

Idées :

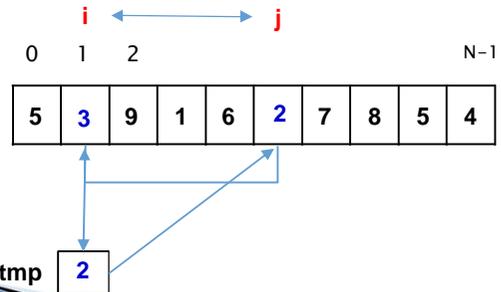
	0	1	2		i				N-1	
T	2	5	1	3	6	1	3	5	3	2
	1	2	5	3	6	1	3	5	3	2

AlgoProg 2015-16 186

## Tri par sélection

Utilise la fonction

```
void echanger (int t[], int i, int j)
```



AlgoProg 2015-16 187

## Tri par sélection

Utilise la fonction

```
void echanger (int t[], int i, int j){
    int tmp;
    tmp=t[i];
    t[i]=t[j];
    t[j]=tmp; }
```

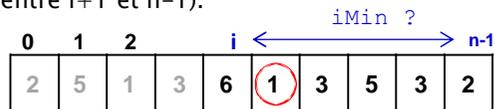
Temps de calcul :  
(1)

AlgoProg 2015-16 188

## Tri par sélection

Basé sur la sélection du minimum :

- ▶ adapter l'algorithme pour l'utiliser sur un sous tableau (entre  $i+1$  et  $n-1$ ).



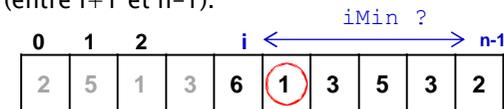
```
int indiceMin(int t[],int n, int i){
    int iMin=i;
    for(int j=i+1;j<=n-1;j++){
        if(t[j]<t[iMin])
            iMin=j; }
    return (iMin);
}
```

AlgoProg 2015-16 189

## Tri par sélection

Basé sur la sélection du minimum :

- ▶ adapter l'algorithme pour l'utiliser sur un sous tableau (entre  $i+1$  et  $n-1$ ).



```
int indiceMin(int t[],int n, int i){
    int iMin=i;
    for(j=i+1;j<=n-1;j++){
        if(t[j]<t[iMin])
            iMin=j; }
    return (iMin);
}
```

Temps de calcul ?  
 $\Theta(n - i)$

AlgoProg 2015-16 190

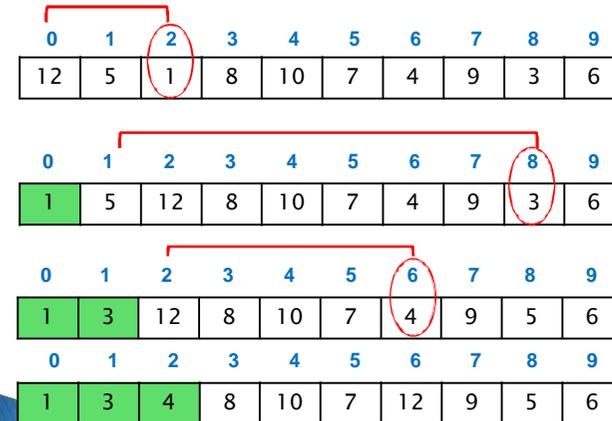
## Tri par sélection

Exemple :

<https://www.youtube.com/user/AlgoRythmics/videos>



## Tri par sélection



AlgoProg 2015-16 192

## Tri par sélection

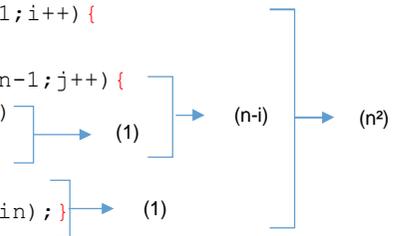
```
void triSelection (int t[],int n){
    int iMin;
    for (int i=0;i<=n-1;i++){
        iMin=i;
        for(int j=i+1;j<=n-1;j++){
            if(t[j]<t[iMin])
                iMin=j;}
        if(i!=iMin)
            echanger (t, i, iMin);
    }
}
```

Complexité en temps ?  
Complexité en mémoire ?  
Stabilité ?

AlgoProg 2015-16 193

## Tri par sélection

```
void triSelection (int t[],int n){
    int iMin;
    for (int i=0;i<=n-1;i++){
        iMin=i;
        for(int j=i+1;j<=n-1;j++){
            if(t[j]<t[iMin])
                iMin=j;}
        if(i!=iMin)
            echanger (t, i, iMin);
    }
}
```



Complexité en temps : (n<sup>2</sup>)  
Complexité en mémoire : (1)  
Stabilité ?

AlgoProg 2015-16 194

## Propriété :

Soit  $T$  un tableau d'entiers trié entre les indices  $i$  et  $j$  ( $i <= j$ ).

Soit  $e$  un entier quelconque, alors on a l'une des propriétés suivantes :

- ▶  $e \leq T[i]$
- ▶ il existe un unique entier  $k$  dans  $[i..j-1]$  tel que  $T[k] < e \leq T[k+1]$
- ▶  $e > T[j]$

On déduit de cette propriété deux algorithmes permettant de trier un tableau.

## Propriété :

0	1	2	3	4	5	6	7	8	9
1	3	4	8	10	7	12	9	5	6

Idées ?

## Tri Insertion

C'est le tri des joueurs de cartes : insérer une nouvelle carte parmi celles déjà triées.

### Hypothèse :

- ▶ Les éléments du tableau entre les indices 0 et  $i-1$  sont triés entre eux.
- ▶ Les éléments de  $t$  d'indice supérieur ou égal à  $i$  n'ont jamais été observés.

On doit mettre  $t[i]$  à sa place.

## Tri insertion

Exemple :

<https://www.youtube.com/user/AlgoRythmics/videos>



## Tri insertion

0	1	2	3	4	5	6	7	8	9
12	5	1	8	10	7	4	9	3	6

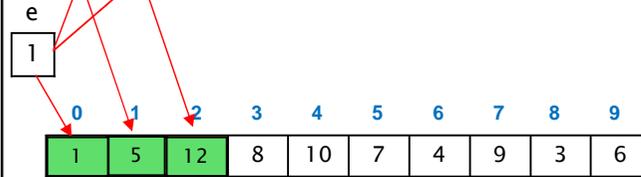
e  
5  
12 > 5 ? Oui donc décalage de 12 vers la droite  
Placement de 5 en première place

0	1	2	3	4	5	6	7	8	9
5	12	1	8	10	7	4	9	3	6

AlgoProg 2015-16 199

## Tri insertion

0	1	2	3	4	5	6	7	8	9
5	12	1	8	10	7	4	9	3	6



AlgoProg 2015-16 200

## Tri insertion

0	1	2	3	4	5	6	7	8	9
1	5	12	8	10	7	4	9	3	6

e  
8

0	1	2	3	4	5	6	7	8	9
1	5	8	12	10	7	4	9	3	6

...

AlgoProg 2015-16 201

## Tri Insertion

```
void triInsertion(int t[],int n){
    int i, j, e;
    for (i=1;i<=n-1;i++){
        e=t[i];
        j=i-1;
        while (j>=0 && t[j]>e){
            t[j+1]=t[j];
            j=j-1; }
        t[j+1]=e;
    }
}
```

Complexité en temps au mieux? Au pire ?

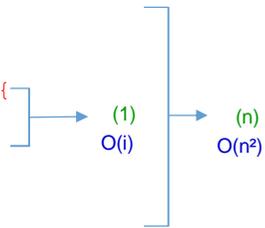
Complexité en mémoire ?

Stabilité ?

AlgoProg 2015-16 202

## Tri Insertion

```
void triInsertion(int t[],int n){
  int i, j, e;
  for (i=1;i<=n-1;i++){
    e=t[i];
    j=i-1;
    while (j>=0 && t[j]>e){
      t[j+1]=t[j];
      j=j-1; }
    t[j+1]=e;
  }
}
```



Complexité en temps au mieux? Au pire? (n) O(n<sup>2</sup>)  
 Complexité en mémoire? (1)  
 Stabilité? Oui car test strict

## Tri à bulles

Les bulles les plus grosses remontent en premier dans une boisson gazeuse.

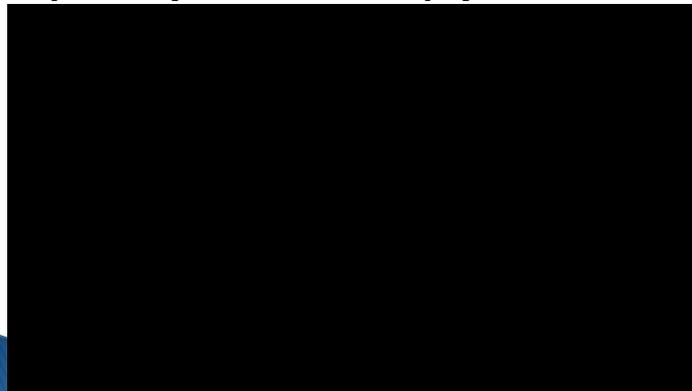


### Hypothèse :

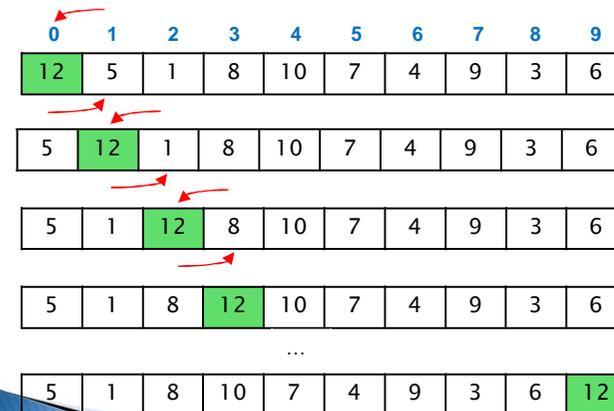
- Le tableau est trié entre les indices i et n-1.
- On échange deux à deux les éléments de t qui ne sont pas correctement ordonnés entre les indices 0 et i-1.

## Tri à bulles

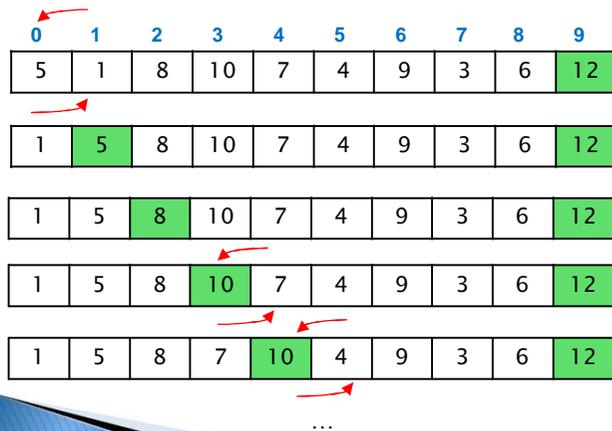
Exemple :  
<https://www.youtube.com/user/AlgoRythmics/videos>



## Tri à bulles



## Tri à bulles



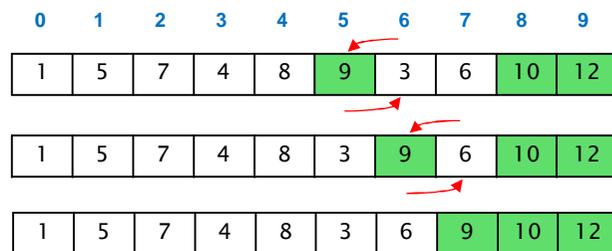
AlgoProg 2015-16 207

## Tri à bulles



AlgoProg 2015-16 208

## Tri à bulles



AlgoProg 2015-16 209

## Tri à bulles

```
void triBulle(int t[],int n){
    int i, j, e;
    for (i=n-1;i>=1;i--){
        for (j=0; j<=i-1; j++){
            if(t[j]>t[j+1])
                echange(t,j,j+1);
        }
    }
}
```

Complexité en temps ?  
Complexité en mémoire ?  
Stabilité ?

AlgoProg 2015-16 210

## Tri à bulles

```
void triBulle(int t[],int n){
  int i, j, e;
  for (i=n-1;i>=1;i--){
    for (j=0; j<=i-1; j++){
      if(t[j]>t[j+1])
        echange(t,j,j+1);
    }
  }
}
```

Complexité en temps ?  $(n^2)$   
 Complexité en mémoire ?  $(1)$   
 Stabilité ? Oui car test strict

AlgoProg 2015-16 211

## Tris itératifs

Tri	Complexité en temps	Complexité mémoire	Stabilité
Sélection	$\Theta(n^2)$	$\Theta(1)$	non
Insertion	$\Omega(n)$ $O(n^2)$	$\Theta(1)$	oui
À bulles	$\Theta(n^2)$	$\Theta(1)$	oui

AlgoProg 2015-16 212

## Approche incrémentale

1. Supposer que l'algorithme fonctionne et qu'une partie des données ont été traitées avec succès.
2. Définir les actions à mener pour traiter la donnée suivante (incrémenter l'action).
3. Définir les conditions de début et de fin de l'algorithme

Exemple ?

AlgoProg 2015-16 213

## Recherche dans un tableau

```
int
recherche(int t[], int n, int x){
  int i=0;
  bool trouve=false;
  while(trouve==false && i<n){
    if (t[i]==x)
      trouve=true;
    else
      i++;
  }
  if (trouve==true)
    return (i);
  return (-1);
}
```

AlgoProg 2015-16 214

## Recherche dans un tableau

```
int
recherche(int t[], int n, int x){
    int i=0;
    bool trouve=false;
    while(!trouve && i<n){
        if (t[i]==x)
            trouve=true;
        else
            i++;
    }
    if (trouve)
        return(i);
    return(-1);
}
```

AlgoProg 2015-16 215

## Recherche dans un tableau

```
int
recherche(int t[], int n, int x){
    int i=0;
    bool trouve=false;
    while(!trouve && i<n){
        if (t[i]==x)
            trouve=true;
        else
            i++;
    }
    if (trouve)
        return(i);
    return(-1);
}
```

Complexité en temps ? (1)  $O(n)$   
Complexité en mémoire ? (1)  
Retourne le 1<sup>er</sup> si doublon : oui

AlgoProg 2015-16 216

## Recherche dans un tableau

```
int
rechercheDichotomique(int t[], int n, int x){
    int g = 0;
    int d = n-1;
    int m;
    bool trouve=false;
    while(!trouve && g<=d){
        m = (g+d)/2;
        if (t[m]==x)
            trouve=true;
        else
            if (t[m]<x)
                g=m+1;
            else
                d=m-1;
    }
    if (trouve)
        return(m);
    return(-1);
}
```

k fois  
que vaut k ?

Complexité en temps ? (1)  $O(\log_2(n))$   
Complexité en mémoire ? (1)  
Retourne le 1<sup>er</sup> si doublon : non

AlgoProg 2015-16 217