Tri par insertion

```
fonction InsertionSort(A,n)

 pour i de 2 à n boucler
  tmp ← A(i)
  j ← i
  tant que j-1 ≥ 1 et A(j-1) > tmp boucler
   A(j) ← A(j-1)
   décrémenter j de 1
  fin tant que
  A(j) ← tmp
 fin pour i
```

Attention, tant que A(j-1) > tmp, donc s'il s'agit des mêmes éléments, le nouveau est inséré après celui qui est déjà bien placé

Tri à bulles

```
fonction BubbleSort(A,n)
        (tableau A de n éléments)

 pour i de 1 à n-1 boucler
  pour j de 1 à n-i boucler
   si A(j+1) < A(j), alors
    permuter A(j) et A(j+1)
   fin si
  fin pour j
 fin pour i
```

Attention au fait que le tableau soit modifié à chaque pas !

Tri par sélection

```
fonction SelectionSort(A,n)

 pour i de 1 à n-1 boucler

  imin ← i
  pour j de i+1 à n boucler
   si A(j) < A(imin), alors
    imin ← j
   fin si
  fin pour j

  permuter A(i) et A(imin)

 fin pour i
```

On trouve le min (1er si 2+ occurences), on le swap avec l'élément à la position i actuelle, on incrémente i

fonction f(N) O(N²)
Pour i allant de 1 à N
Pour j allant de i*i + j*j à N*N
Afficher un caractère

Stable : BubbleSort (avec <), InsertionSort, MergeSort, BaseSort, TriComptage
Non-stable : SelectionSort, QuickSort

En place : QuickSort, Bubble/Insertion/Selection-Sort
Pas en place : MergeSort, BaseSort, TriComptage

Tri fusion

```
fonction Fusionner(A,p,q,r)
 L ← copie du tableau A de p à q
 R ← copie du tableau A de q+1 à r
 insérer une sentinelle ∞ en fin de L et R

 i ← 1, j ← 1
 pour k de p à r boucler
  si L(i) ≤ R(j), alors
   A(k) ← L(i)
   incrémenter i
  sinon
   A(k) ← R(j)
   incrémenter j
  fin si
 fin pour
```

```
fonction TriFusion(A,lo,hi)

 si hi <= lo, alors
  retourner
 fin si

 mid ← lo + (hi-lo)/2
       Pour que ce soit adapté à des pointeurs/adresses,
       on ne peut pas simplement faire (lo+hi)/2
 TriFusion(A,lo,mid)
 TriFusion(A,mid+1,hi)

 Fusionner(A,lo,mid,hi)
```

On scinde, on scinde, jusqu'à avoir une taille de 2 ou 1, on fusionne, on fusionne, on fait pareil pour l'autre partie, récursivement

Tri rapide

```
fonction Partition(A,lo,hi)
 i ← lo-1, j ← hi

 boucler
  répéter incrémenter i
  tant que A(i) < A(hi)

  répéter décrémenter j
  tant que j>lo et A(hi) < A(j)

  si i ≥ j, alors sortir boucle
  fin si

  permuter A(i) et A(j)
 fin boucler

 permuter A(i) et A(hi)
 retourner i
```

```
fonction TriRapide(A,lo,hi)
 si lo < hi, alors
  p ← choisir l'élément pivot
  permuter A(hi) et A(p)
  i ← Partition(A,lo,hi)

  TriRapide(A,lo,i-1)
  TriRapide(A,i+1,hi)
 fin si
```

```
fonction TriRapide(A,lo,hi)
 tant que lo < hi
  p ← choisir l'élément pivot
  permuter A(hi) et A(p)
  i ← Partition(A,lo,hi)

  si i-lo < hi-i, alors
   TriRapide(A,lo,i-1)
   lo ← i+1
  sinon,
   TriRapide(A,i+1,hi)
   hi ← i-1
  fin si
 fin tant que
```

Sélection rapide

```
fonction SelectionRapide(A,n,k)
 lo ← 1
 hi ← n

 tant que hi > lo
  i ← partition(A,lo,hi)
  si i < k, alors
   lo ← i+1
  sinon, si i > k, alors
   hi ← i-1
  sinon  (i = k)
   retourner A(k)
 fin tant que

 retourner A(k)
```

Tri comptage

```
fonction TriComptage(A,n,b,clé):

 C ← tableau de b compteurs à zéro
 pour tout e dans A
  C[clé(e)] += 1

 idx ← 1
 pour i de 1 à b
  tmp ← C[i]
  C[i] ← idx
  idx += tmp

 B = tableau de même taille que A

 pour tout e dans A
  B[C[clé(e)]] ← déplacer e
  C[clé(e)] += 1

 return B
```

Tri par base

```
fonction triParBase(T, d):
   Pour i allant de d à 1
       Trier avec un tri
       stable le tableau T
       selon le i-ème chiffre
```

indices tris

BubbleSort : fin triée, début non trié, lettres changées et certaines manquantes.

SelectionSort : début trié, fin ressemble à l'original avec lettres échangées.

InsertionSort : partie triée, partie identique, élément en trop (en train d'être décalé), élément manquant (en train d'être copié).

MergeSort : demi tableau gauche trié, éventuellement moitié du demi tableau droite trié.

QuickSort : partie droite peut sembler identiques, certains éléments échangés.

qsort

Variation du tri rapide.
On peut lui passer l'adresse du 1er élément d'un tableau, le nombre d'éléments à trier, la taille d'un élément du tableau, ainsi que l'adresse de la fonction de comparaison.

sort

Variation du tri rapide.
On lui passe un itérateur de début, de fin et une fonction de comparaison.

nth_element

On cherche le nième plus petit élément. L'algorithme effectue un tri (rapide, quickselect) jusqu'à ce que cet élément soit correctement placé. On peut ensuite le récupérer avec les [ ].

Les éléments à gauche de celui-ci sont < et ceux à droite sont >.

stable_sort

Utilise le tri fusion. Faire des "groupes" par rapport à la fonction "compare".

	Meilleur des cas	Moyenne	Pire des cas
Bubble sort	O(n)	$O(n^2)$	
Selection sort	$O(n^2)$		
Insertion sort	O(n)	$O(n^2)$	
Merge sort std::sort, std::stable_sort	O(nlog(n))		
Quick sort	O(nlog(n))		$O(n^2)$
Counting sort (comptage)	O(n+b)		
Radix sort (base)	O(d·(n+b))		
Euclide (pgcd), std::upper_bound	O(log(n))		
Quick select, std::nth_element, std::min_element	O(n)		

Tri par insertion (complément) : A chaque élément, il va le comparer avec tous ses précédents, tant qu'il n'a pas rencontré plus petit ou égal, il continue à chercher l'emplacement en déplaçant à chaque pas l'élément testé en avant.

Tri à bulles (complément) : On fait un nombre de tours égal au nombre d'éléments - 1. A chaque tour, on commence au début, on compare l'élément qu'on a en mains avec le suivant, s'il est plus grand que le suivant, on les swap, sinon on passe à l'élément suivant et ainsi de suite.

Tri par sélection (complément) : A chaque tour, on cherche le minimum et on le place au début.

Tri fusion (complément) : On coupe en 2 (si N impair, normalement on répartit pair/impair) récursivement jusqu'à obtenir un élément seul, on remonte en triant les morceaux et on fusionne chaque "paire" de morceaux.

Tri rapide (complément) : si le pivot fourni n'est pas tout à droite, le placer tout à droite (1 étape complète). Maintenant, l'indice j est l'élément à gauche du pivot, i est au début du tableau, tant que i est plus petit que le pivot (strictement), on l'avance, sinon on s'arrête et on cherche un j (en décrémentant) qui est plus petit ou égal au pivot. Dès que i et j se croisent, on arrête et on échange le pivot avec l'élément situé à l'endroit où ils se sont croisés. On recommence en prenant l'élément à gauche du pivot comme nouveau pivot et on traite la partie gauche du tableau. Ensuite on reprendra avec la partie droite du tableau en reprenant l'élément tout à droite comme pivot, et ainsi de suite. On fait ça jusqu'à ce que chaque position du tableau ait reçu un pivot (signifiant que l'élément est bien placé). **NB :** dès qu'on a swap 2 éléments, on ne les touche plus, on incrémente i et décrémente j.

Tri comptage :

Tri par base :

lower_bound

Retourne un itérateur pointant sur le 1er élément dans [first, last) qui n'est pas plus petit que val (donc plus petit ou égal).

upper_bound

Retourne un itérateur pointant sur le 1er élément dans [first, last) qui est plus grand que val (strictement).