Complexité algorithmique

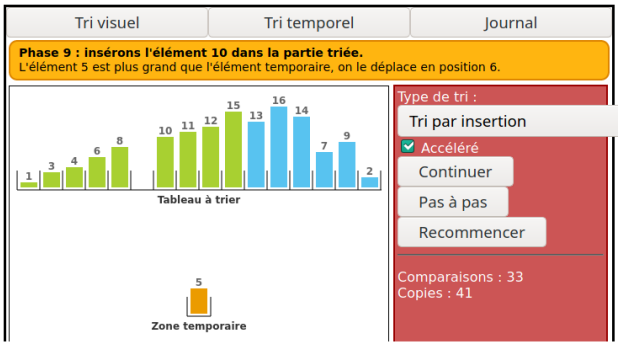
TPx : Tri par insertion

Un algorithme de tri permet d'organiser une collection d'objets d'un même ensemble selon une relation d'ordre prédéterminée (e.g. ranger les entiers d'une liste en ordre croissant).

Plusieurs algorithmes sont disponibles pour réaliser une opération de tri. La présente activité se concentre sur la technique de tri par insertion.

Partie 1 : Introduction au tri par insertion / Objectif :Observer le mécanisme de tri

A l'aide d'un navigateur internet, **ouvrir** l'animation de tri en ligne du le site interstice https://interstices.info/les-algorithmes-de-tri/



Procéder à plusieurs simulations du tri insertion, en mode « accéléré » et en mode « pas à pas ».

Remettre dans l'ordre les différentes actions de l'algorithme :

La liste est constituée d'une partie triée (à gauche) et une partie non triée (à droite)

Intitulé	***Ordre d'éxecution***
• *Si l'élement de la liste triée est supérieur à l'élément mémorisé, le décaler vers la droite dans l'espace vide,* • *Si l'élément de la liste triée est inférieur à l'élément mémorisé, insérer l'élément mémorisé dans l'espace vide,*	4
• *Libérer l'espace du 1er élément de la partie non trié du tableau,*	2
• *Répéter l'opération jusqu'a obtenir une partie non trié vide.*	5
• *Parcourir la partie triée de la liste en sens inverse,*	3
• *Mémoriser dans la zone mémoire temporaire le 1er élément de la partie non trié du tableau,*	1

Analyser le mécanisme de tri insertion pour les données ci-dessous et **compléter** l'étape n°5.

Etape n°1 : La liste triée comporte que l'élément d'indice n°1

Liste	47	18	8	21	34	
Indice	1	2	3	4	5	Zone temporaire

Etape n°2 :

Copie de l'élément d'indice n°2 dans la zone temporaire

Liste	47		8	21	34	18
Indice	1	2	3	4	5	Zone temporaire

L'élément à l'indice n°1 est supérieur à l'élément en zone temporaire, il est déplacé à l'indice n°2

Liste		47	8	21	34	18
Indice	1	2	3	4	5	Zone temporaire

Il n'y a plus d'éléments à déplacer, l'élément en zone temporaire est le plus petit, il est déplacé à l'indice n°1

Liste	18	47	8	21	34	
Indice	1	2	3	4	5	Zone temporaire

Etape n°3 :

Copie de l'élément d'indice n°3 dans la zone temporaire

Liste	18	47		21	34
Indice	1	2	3	4	5

8
Zone temporaire

L'élément à l'indice n°2 est supérieur à l'élément en zone temporaire, il est déplacé à l'indice n°3

Liste	18		47	21	34
Indice	1	2	3	4	5

8
Zone temporaire

L'élément à l'indice n°1 est supérieur à l'élément en zone temporaire, il est déplacé à l'indice n°2

Liste		18	47	21	34
Indice	1	2	3	4	5

8
Zone temporaire

Il n'y a plus d'éléments à déplacer, l'élément en zone temporaire est le plus petit, il est déplacé à l'indice n°1

Liste	8	18	47	21	34
Indice	1	2	3	4	5

Zone temporaire

Etape n°4 :

Copie de l'élément d'indice n°4 dans la zone temporaire

Liste	8	18	47		34
Indice	1	2	3	4	5

21
Zone temporaire

L'élément à l'indice n°3 est supérieur à l'élément en zone temporaire, il est déplacé à l'indice n°4

Liste	8	18		47	34
Indice	1	2	3	4	5

21
Zone temporaire

L'élément à l'indice n°2 est inférieur à l'élément en zone temporaire, l'élément en zone temporaire est inséré

Liste	8	18	21	47	34
Indice	1	2	3	4	5

Zone temporaire

Etape n°5 : (à compléter)

Copie de l'élément d'indice n°**5** dans la zone temporaire

Liste	8	18	21	47	
Indice	1	2	3	4	5

34
Zone temporaire

L'élément à l'indice °**4** est supérieur à l'élément en zone temporaire, il est déplacé à l'indice n°**5**

Liste	8	18	21		47
Indice	1	2	3	4	5

34
Zone temporaire

L'élément à l'indice n°**3** est inférieur à l'élément en zone temporaire, l'élément en zone temporaire est inséré

Liste	8	18	21	34	47
Indice	1	2	3	4	5

Zone temporaire

Etape n°6 : Le dernier indice (indice n°5) a été traité, le tableau est trié et l'algorithme se termine

Partie 2 : Spécification de l'algorithme

L'algorithme ci-contre propose une solution sur « **comment** » réaliser un tri par insertion mais il n'apporte aucune information sur la description du problème à traiter (le « **quoi** »).

Afin de décrire le problème traité, Il est nécessaire de le **spécifier** à l'aide de paramètres d'entrée et de sortie, de pré-conditions et de post-conditions :

Compléter la spécification ci-dessous à l'aide des termes suivants :

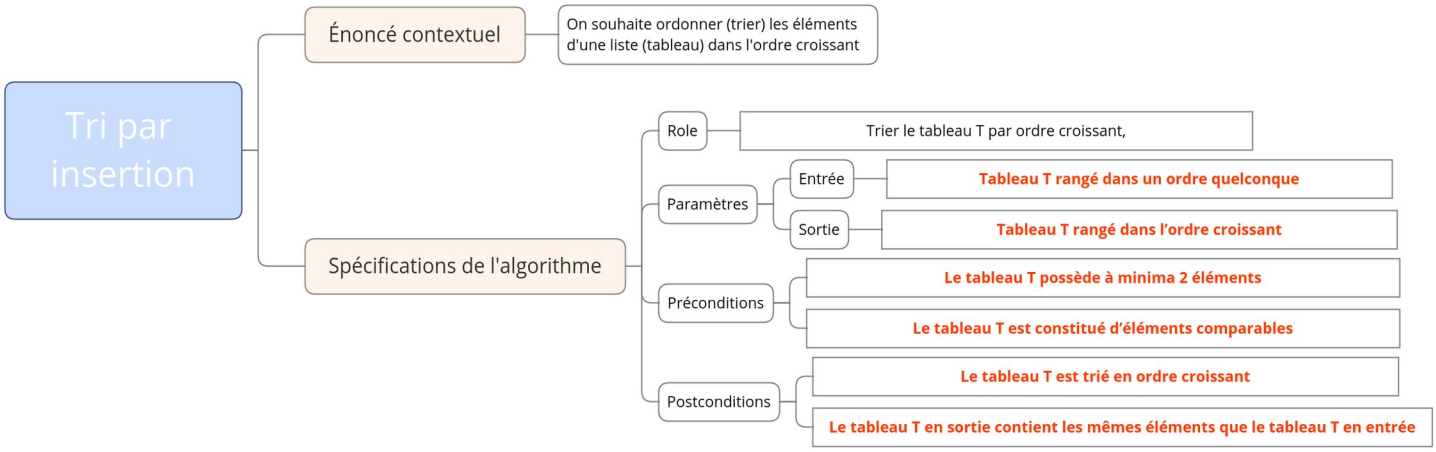
- Tableau T rangé dans un ordre quelconque,
- Tableau T rangé dans l'ordre croissant,
- Le tableau T possède à minima 2 éléments,
- Le tableau T est constitué d'éléments comparables,
- Le tableau T est trié en ordre croissant,
- Le tableau T en sortie contient les mêmes éléments que le tableau T en entrée

```
Début Tri_insertion
  Tab = [18,47,8,21,34]
  pour i de 1 à n faire
     j=i
     memoire=Tab[i]

     tant que (j>0) and memoire<Tab[j-1]:
        Tab[j-1],Tabl[j] = Tab[j],Tab[j-1]
        j = j - 1
     fin_tant_que

     Tabl[j]= memoire

  fin_pour
fin Tri_Insertion
```



Partie 3 : Preuve de correction : Est-ce-que l'algorithme donne le résultat attendu ?

Reprendre les différentes étapes de la partie 1, **compléter** ci-dessous l'état du tableau et **répondre** à la question : « à l'étape n°x du traitement de l'algorithme, les x premier(s) élément(s) du tableau sont-ils triés ? »

Etape n°1	État du tableau	47	18	8	21	34
	Le premier élément du tableau est trié : ☑ Vrai ☐ Faux					
Etape n°2	État du tableau	18	47	8	21	34
	Le 2 premiers éléments du tableau sont triés : ☑ Vrai ☐ Faux					
Etape n°3	État du tableau	8	18	47	21	34
	Le 3 premiers éléments du tableau sont triés : ☑ Vrai ☐ Faux					
Etape n°4	État du tableau	8	18	21	47	34
	Le 4 premiers éléments du tableau sont triés : ☑ Vrai ☐ Faux					
Etape n°5	État du tableau	8	18	21	34	47
	Le 5 premiers éléments du tableau sont triés : ☑ Vrai ☐ Faux					

Conclure, l'algorithme produit-t-il le résultat attendu ?

L'algorithme renvoie, dans sa partie gauche, puis progressivement dans sa totalité, un tableau trié en ordre croissant. Il donne le résultat attendu.

Partie 4 : Codage en Python

Traduire l'algorithme ci-dessous en code Python.

Algorithme

```
Début Tri_insertion
  Tab = [18,47,8,21,34]
  pour i de 1 à n faire
      j=i
      memoire=Tab[i]

      tant que (j>0) and memoire<Tab[j-1]:
          Tab[j-1],Tabl[j] = Tab[j],Tab[j-1]
          j = j - 1
      fin_tant_que

      Tabl[j]= memoire

  fin_pour
fin Tri_Insertion
```

Code Python

```
Tab = [18,47,8,21,34]

for i in range(1,len(Tab)):

    j=i
    memoire=Tab[i]

    while (j>0) and encours<Tableau[j-1]:
        Tab[j-1],Tabl[j] = Tab[j],Tab[j-1]

        j = j - 1

    Tab[j]= memoire
```

Procéder aux ajustements pour vérifier le bon fonctionnement de votre code.

Partie 5 : Etude de la complexité : est-ce que l'algorithme est efficace ?

Un programme Python de tri par insertion est proposé ci-dessous. Il propose les évolutions suivantes :

- Le tableau Tab à trier est généré « en compréhension » avec l'instruction « *Tableau = [randint(0, 1000000) for i in range(10000)]* » ce qui permet de créer un tableau de 20 valeurs aléatoire comprises entre 0 et 1 000 000 exclu.
- L'instruction Python « *Tableau.sort(reverse=True)* » permet de ranger le tableau dans l'ordre décroissant, afin de se placer dans le pire cas pour notre algorithme.
- Il affiche en fin de programme, la taille du tableau et le nombre d'affectation réalisé.
- Pour le reste, il est identique à l'algorithme proposé précédemment.

```
from random import randint

Tableau = [randint(0, 1000000) for i in range(10000)]
Tableau.sort(reverse=True)

nb_affectation = 0

for i in range(1,len(Tableau)):

    j=i

    encours=Tableau[i]
    nb_affectation += 1

    while (j>0) and encours<Tableau[j-1]:

        Tableau[j] = Tableau[j-1]
        nb_affectation += 1

        j-=1

    Tableau[j]=encours

print('taille n=',len(Tableau)," nb affectation =",nb_affectation)
```

Saisir et **exécuter** ce programme pour les différentes tailles N de tableau ci-dessous. Que constatez-vous ? :

Taille N du tableau	10	50	100	200	1000
Nombre d'affectation	**54**	**1274**	**5049**	**20 099**	**500 499**
Calcul : $\frac{1}{2}\cdot(n^2+n)-1$	**54**	**1274**	**5049**	**20 099**	**500 499**

Quand la taille du tableau augmente, le temps de traitement du tri augmente. L'augmentation du temps de traitement se semble pas proportionnelle avec la taille du tableau.

Sur le graphe ci-dessous, la taille du tableau N est positionné en abscisse, et le nombre d'affectation en ordonnée
Proposer une classe de complexité « a^n, n^3, n^2, n.log(n), n ou log(n) » pour votre algorithme.

La complexité est d'ordre quadratique ou N^2