Semaine du 7 au 11 décembre

Séance 1 (mardi 8 décembre ou jeudi 10 décembre) :

Au regard de l'importance des méthodes algorithmiques, nous avons repris la méthodo autour de l'utilisation du tableau pour comprendre un algorithme déjà écrit.

- [ ] Revoyez cette méthode, sur l'exemple sur le cas complexe avec la boucle Tant que.
- [ ] Essayez de refaire ce cas sans regarder le cours.

Nous avons également repris la complexité qui semblait avoir posé problème à la majeure partie des élèves. Par rapport au cours de la semaine dernière, nous avons également pris le temps de montrer que les opérations élémentaires prennent à peu près toute le même temps de calcul. Cela permet de justifier que le nombre d'affectations, de comparaisons ou d'opérations arithmétiques d'un algorithme peuvent être additionner ensemble pour compter le nombre global d'opérations.

- [ ] Sur un éditeur Python (**trinket.io** par exemple), essayez de mesurer le temps que prend une affectation, une comparaison et une opération arithmétique (+, -, * ou ÷).
  Pour cela, vous devrez importer la bibliothèque **timeit**.

  Pour connaitre le temps (en secondes) utilisé pour faire un test d'égalité, on écrira l'instruction:

  ```
  print(timeit.timeit('3==3', number=1000)/1000)
  ```

  Cela permettra de faire une moyenne sur 1000 test d'égalité. **Remarquez que l'instruction à tester est écrite entre guillemets.**

  Complétez le tableau ci-dessous et concluez sur la pertinence d'additionner toutes les opérations élémentaires ensemble, sans distinction.

	Temps (en s)	Temps (en ns)
Opération arithmétique		
Affectation		
Comparaison		

Finalement, nous avons fait les exercices 1 et 2 de la feuille d'exercice. Pour faire ces exercices, il faut bien prendre des exemples afin de comprendre ce qu'il se passe.

- [ ] Exercice 1 à faire
- [ ] Exercice 2 à faire : essayez d'imaginer que chaque élément de votre tableau est une boite avec un couvercle. À l'intérieur est caché un nombre. Vous ne pouvez ouvrir que deux boites en même temps.
  Comment faites-vous pour savoir si vos boites sont dans l'ordre ? Cela va vous donner votre algorithme.

Correction page suivante

Exercice 1 :

a) On prend un exemple : T = [7, 6, 4]. Ce pourrait être des coordonnées de vecteurs dans l'espace.
On prend moyenne = 0
On fait le tableau.

i	T[i]	moyenne
0	7	0+7 = 7
1	6	7+6 = 13
2	4	13+4 = 17

On renvoie 17, c'est la somme des éléments de mon tableau.

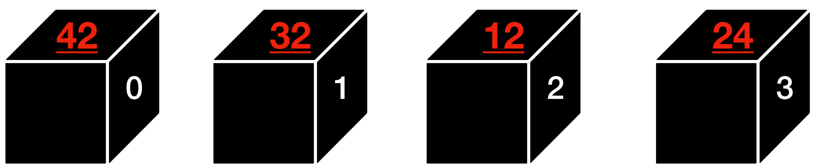
b) Complexité :
On compte toutes les opérations élémentaires (et on les ajoute ensemble). J'ai une affectation suivie de 2 opérations élémentaires qui sont répétées 3 fois, soit 7 opérations. Je décide de ne pas compter le i de la boucle…

En tout état de cause, cela me donne un nombre constant d'opérations, peu importe le problème que je cherche à résoudre. Cela s'appelle une **complexité constante.**

Exercice 2 :

Aide : Je ne peux comparer que deux éléments à la fois. Pour savoir si tout est dans l'ordre sans perdre trop de temps, je vais appliquer la méthode ci-dessous.

Continuous l'exemple avec mes boites (les numéros en rouge sont les valeurs, invisibles tant que l'on n'ouvre pas la boite).



Je prends la boite numéro 0 et je l'ouvre : 42.
Je prends la suivante. C'est la boite numéro 1, je l'ouvre : 32
Est-ce que 32 > 42 ? Non, donc la boite 0 et la boite 1 sont dans l'ordre.

Je prends la boite numéro 1 et je l'ouvre : 32.
Je prends la suivante. C'est la boite numéro 2, je l'ouvre : 12
Est-ce que 12 > 32 ? Non, donc la boite 1 et la boite 2 sont dans l'ordre.

etc…

Vous devez voir une structure de boucle, avec des tests.

Correction page suivante

Aide 2 :
Vous allez avoir besoin d'une variable booléenne (prenant la valeur VRAI/FAUX) nous indiquant si le tableau est trié dans l'ordre décroissant ou non. Appelons cette variable tri.
On va d'abord être optimiste et dire que le tableau est trié dans l'ordre décroissant : tri = VRAI.

Solution version langage français :
tri = VRAI
Pour i allant de 0 jusqu'à la taille du tableau **- 2**, on fait :
    si la valeur suivante est supérieure à la valeur numéro i :
        tri = FAUX
renvoyer tri

a) Solution version pseudo-code :
tri = True
Pour i allant de 0 jusqu'à la taille du tableau **- 2**, on fait :
    si T[i+1] > T[i] :
        tri = False
renvoyer tri

b) T= [17,9,8,11]
tri = True
On fait le tableau car on a une boucle :

i	T[i]	T[i+1]	T[i+1]>T[i]	tri
0	17	9	FAUX	VRAI
1	9	8	FAUX	VRAI
2	8	11	VRAI	FAUX

On renvoie FAUX : T n'est pas dans l'ordre décroissant.

c) Pour la complexité, nous avons quelque chose de très semblable au cours.
1 affectation pour commencer
boucle n-1 fois répétée : 1 comparaison et **dans le pire des cas, on fait l'affectation tri = False à chaque fois.** Donc 2 opérations.
Au total $1+2 \times (n-1) = 2n-1$ opérations élémentaires.
On va donc dire que la complexité est **linéaire** ou en $\mathcal{O}(n)$.

Correction page suivante