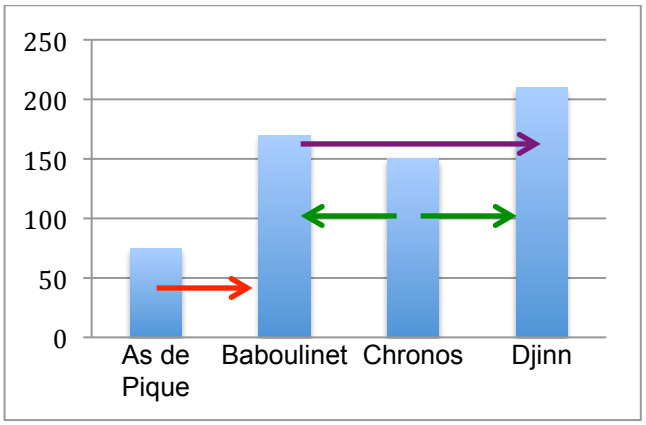
A la fin du premier acte du blockbuster « Rapide et Impossible XIII ¾ : Les Gardiens de la Justice Reloaded », le méchant Dr. Mablanquex hypnotise tous les gentils de la force-ligue-équipe des héros. Il les fait se mettre en ligne sur le bord d'une falaise, et leur ordonne que chacun tire sur le héros le plus proche, à droite et à gauche de celui qui tire, plus grand que le tireur, à son signal (les héros touchés sont ainsi tués). Les héros ont tous des tailles différentes.

Exemple :

Les héros As de Pique, Baboulinet, Chronos, Djinn ont pour tailles respectives 75, 170, 150, 210 cm.



As de Pique ne tire sur personne à gauche, et tire sur Baboulinet à droite ;
Baboulinet ne tire sur personne à gauche, et tire sur Djinn à droite ;
Chronos tire sur Baboulinet à gauche et sur Djinn à droite ;
Djinn ne tire sur personne, ni à gauche ni à droite.

Le but de cet exercice est d'écrire un programme avec la méthode « diviser pour régner », pour savoir qui tire sur qui.

On représente les tailles par un tableau `taille` d'entiers strictement positifs, deux à deux distincts, indexée de 0 à $n-1$.

On construit deux tableaux `tire_droite` et `tire_gauche`, qui donne pour chaque héros, l'indice du héros sur lequel il tire à gauche et à droite.

1.

a. On donne le tableau `taille = [185, 116, 54, 150, 60, 229, 170]`.
Donner les tableaux `tire_droite` et `tire_gauche`. On codera « aucun » par `None`.

b. Idem avec le tableau `taille = [185, 116]`

c. Idem avec le tableau `taille = [185]`

2. Un algorithme de résolution du problème par force brute est :

```
Pour i de 0 à n – 1
        # on cherche la cible à gauche
        tire_gauche[i] ← None
        Pour j de 1 à i – 1
                Si taille[j] > taille[i]
                        tire_gauche[i] ← j

        # on cherche la cible à droite
        tire_droite[i] ← None
        Pour j de n – 1 à i + 1 avec un pas de –1
                Si taille[j] > taille[i]
                        tire_droite[i] ← j
```

Quelle est la complexité de cet algorithme (on ne demande pas de calculs précis) ?

3. Stratégie de résolution avec « diviser pour régner »

La résolution de ce problème avec une méthode « diviser pour régner » divise le tableau des tailles successivement, d'un indice de gauche `g` jusqu'à un indice de droite `d`, résout le problème sur chaque tableau une fois que la taille est suffisamment petite, puis combine les résultats obtenus. Les tableaux `tire_droite` et `tire_gauche` sont créés dans le programme principal et modifiés lors de la résolution progressive du problème.
Les spécifications de la fonction de divsion et de fusion sont :

```
def quiTireSurQui(taille, tire_gauche, tire_droite, g, d):
      """
      Etant donné un tableau d'entiers strictement positifs
      tous différents, renvoie pour chaque élément du tableau l'indice
      de l'entier le plus grand "visible" à sa droite, ainsi qu'à sa
      gauche. Un élément du tableau ne peut pas "voir" les entiers
      cachés par des entiers plus grands que lui.
      Exemple :
            taille = [185, 173, 116, 54, 150, 51, 60, 180, 229, 170]
            L'élément 150 d'indice 4 verra
            à sa gauche 173 d'indice 1
            et à sa droite 180 d'indice 7
            d'où gauche[4] =  1 et droite[4] = 7
      @param taille : tableau d'entiers strictement positifs
      @param tire_gauche, tire_droite : tableaux d'indices du tableau
      taille, ou None
      @param g, d : indices du tableau taille avec g ≤ d
      @return : None
      """
def fusion(taille, tire_gauche, tire_droite, g, d):
      """
      Combine les résultats obtenus lors de la phase de division
      @param taille : tableau d'entiers strictement positifs
      @param tire_gauche, tire_droite : tableaux d'indices du tableau
      taille, ou None
      @param g, d : indices du tableau taille avec g ≤ d
      @return : None
      """
```

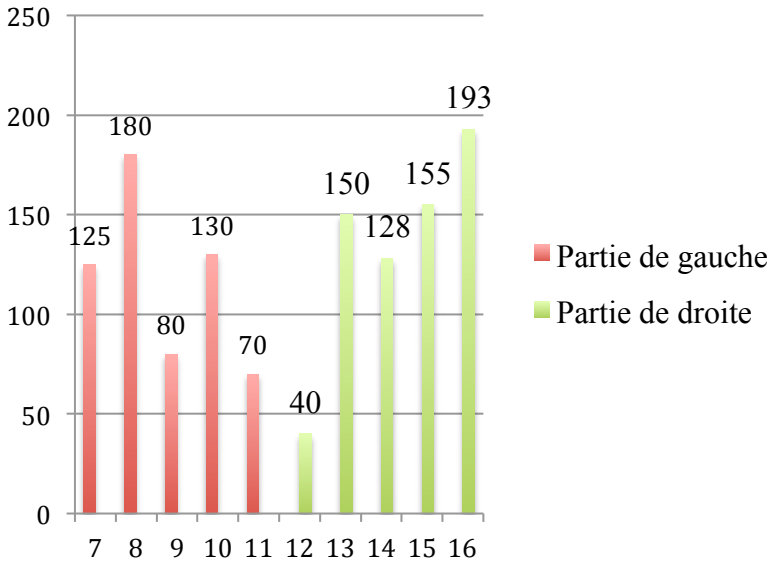
a. Préliminaire.
Qu'est-ce qu'un objet mutable ? En quoi cela permet-il de ne pas renvoyer les tableaux `tire_droite` et `tire_gauche` lors du retour des fonctions `quiTireSurQui` et `fusion` ?

b. Phase de division : cas terminaux
La division s'arrête lorsque la partie du tableau des tailles sur laquelle on résout le problème est de longueur 1 ou 2.
Compléter en Python la fonction `quiTireSurQui()` en traitant ces deux cas.

c. Phase de divisions : cas non terminaux
Lorsque la taille du tableau est supérieure ou égale à 3, la fonction `quiTireSurQui()` calcule l'indice médian entre `g` et `d`. Puis elle se rappelle sur la moitié gauche de la partie en cours d'exploration du tableau des tailles. Elle se rappelle également sur la moitié droite. Enfin, elle appelle la fonction `fusion()`, avec en paramètres les tableaux `taille`, `tire_droite` et `tire_gauche` , et les indices `g` et `d`.
Compléter le code Python de la fonction `quiTireSurQui()` en tenant compte de ces indications. On écrira le code à la suite du code de la question b, sans recopier celle-ci.

d. Phase de combinaison : étude sur un exemple de la construction du tableau `tire_droite`.
La combinaison des résultats de deux appels, à gauche et à droite, se fait avec un parcours des tableaux résultants en partant de l'indice médian.
Dans l'exemple suivant, on se place dans une phase de combinaison ayant lieu en « milieu » d'exécution du programme. Les résultats de `tire_droite` ont été calculés sur la partie gauche,

d'indice 7 à 11, et sur la partie droite, d'indice 12 à 16. On souhaite fusionner les deux résultats pour obtenir le tableau `tire_droite` , cohérent avec toutes les données de `taille` d'indice 7 à 16.



	Indice *i* de parcours de la partie gauche	Indice *j* de parcours de la partie droite	Extrait du tableau `tire_droite` (indices 7 à 16)
Itération 0 / initialisation	`i = 11`	`j = 12`	`[8, None, 10, None, None, x, x, x, x, x]`
Itération 1	`i = 11`	`j = 13`	`[8, None, 10, None, 13, x, x, x, x, x]`
Itération 2	`i = 10`	`j = 13`	`[8, None, 10, 13, 13, x, x, x, x, x]`
Itération 3	`i = 9`	`j = 13`	`[8, None, 10, 13, 13, x, x, x, x, x]`
…			

i. Pourquoi les valeurs de `tire_droite` d'indices 12 à 16 ne sont pas à traiter dans cette phase de combinaison ?
ii. Compléter le tableau (*recopié en annexe donnée en fin de sujet, à rendre avec la copie*) jusqu'à ce que cette phase de combinaison soit terminée.
iii. Dans le tableau `tire_droite`, on ne modifie éventuellement que certaines valeurs. Quelles sont les valeurs que l'on modifie éventuellement ?
iv. Coder la fonction `fusion(taille, tire_gauche, tire_droite, g, d)`. On traitera uniquement le tableau `tire_droite`, le raisonnement étant similaire pour le tableau `tire_gauche`.

e. Quelle est la complexité de l'algorithme utilisé avec la stratégie « diviser pour régner » ?

f. Le Dr. Mablanquex itère le procédé jusqu'à ce qu'il ne reste plus qu'un seul super-héros (Squishy le petit écureuil en l'occurrence). Écrire une fonction `final(taille)` qui donne la suite des super-héros éliminés à chaque round, jusqu'à ce qu'il n'en reste qu'un seul.

Pour ceux qui voudraient savoir la fin de cette histoire, sachez que Squishy retourne dans le passé, et remplace les super-héros par des sculptures. Le Dr Mablanquex étant très myope ne se rend compte de rien. C'était le scénario prévu pour Avengers Endgame, mais comme il a été piraté et rendu public, les studios Marvel/Disney ont été obligés de se rabattre sur quelque chose de bien moins original, avec une interminable baston finale pour remplir.