Consignes. Les calculatrices sont interdites. Numérotez vos feuilles et faites apparaître les questions dans l'ordre du sujet. Si vous repérez une erreur d'énoncé, signalez la sur votre copie et poursuivez votre composition.

Organisation de l'énoncé. Ce sujet est constitué de cinq parties. La partie 1 est formée de préliminaires utiles dans le reste du sujet, mais les programmes qui y sont définis peuvent être admis pour traiter les parties suivantes. Les parties 2, 3, 4 et 5 sont indépendantes.

Notations. Dans tout le sujet, la variable « `L: list[int]` » désigne une liste d'entiers et $n \in \mathbb{N}$ est le nombre d'éléments dans `L`. L'objectif est d'étudier différents algorithmes permettant de trier `L` par ordre croissant. Par exemple à partir de `L = [10, -16, -13, 15, 13, -17, -8, -10, 5, -8, -5, -14, -13]`, on souhaite obtenir `[-17, -16, -14, -13, -13, -10, -8, -8, -5, 5, 10, 13, 15]`.

1 Préliminaires

Pour répondre aux questions des autres parties, il sera nécessaire d'utiliser certaines fonctions des modules `math`, `random`, `time` et `matplotlib.pyplot`.

1. (a) Écrire deux lignes de code permettant d'importer les modules `random` et `time`. Après exécution de ces 2 lignes, on veut que le programme 1 puisse s'exécuter sans erreur.
   (b) Écrire une ligne de code permettant d'importer la fonction `floor` du module `math`. Après exécution de cette ligne, on veut que le programme 2 puisse s'exécuter sans erreur.
   (c) Écrire une ligne de code permettant d'importer le module `matplotlib.pyplot` en lui donnant `plt` comme alias. Après exécution de cette ligne, on veut que le programme 3 puisse s'exécuter sans erreur.

```
# Programme 1
tpsIni = time.perf_counter()
L = []
for _ in range(123):
    L.append(random.randint(-753, 159))
tpsTotal = time.perf_counter() - tpsIni
print(tpsTotal)
```

```
# Programme 2
x = floor(45/13)
print(x) # Affiche 3
```

```
# Programme 3
plt.figure()
X = [x for x in range(-10, 11)]
Y = [(x-1)*(x+1) for x in X]
plt.plot(X,Y)
plt.show()
```

À partir de maintenant et jusqu'à la fin de l'énoncé, on suppose que les programmes donnés dans la question 1 ont été exécutés.

2 Tri par comptage

Dans cette partie, les éléments de `L` sont tous compris entre 0 et 100. Pour toute liste `L`, notons φ`(L)` la liste de taille 101 telle que φ`(L)` contient, à l'indice `i`, le nombre d'occurrences de `i` dans `L`. Par exemple :

`L`	`[99, 2, 2, 8, 5, 5, 4, 6, 2, 1, 2, 8, 7, 100, 99, 1, 6, 6, 0]`
φ`(L)`	`[1, 2, 4, 0, 1, 2, 3, 1, 2, 0, 0, 0, 0, ... , 0, 0, 0, 2, 1]`

2. Pour construire φ`(L)` à partir de `L`, on écrit la fonction ci-contre.
   (a) Donner la signature de la fonction `phi`.
   (b) Montrer que la fonction `phi` termine pour toute entrée `L`.
   (c) Déterminer en la justifiant la complexité de la fonction `phi`.

```
1  def phi(L):
2      C = [0] * 101
3      for i in range(len(L)):
4          C[L[i]] += 1
5      return C
```

3. Montrer que pour tout `L`, l'appel à `phi(L)` renvoie φ`(L)`. On attend une preuve complète et détaillée utilisant un invariant de boucle.

Le ***tri par comptage*** consiste à construire φ`(L)`, puis à en déduire une liste triée `T` contenant les mêmes éléments que `L`. Avec l'exemple précédent, on obtient :

`T = [0, 1, 1, 2, 2, 2, 2, 4, 5, 5, 6, 6, 6, 7, 8, 8, 99, 99, 100]`

4. (a) Écrire une fonction `triComptage(L: list[int]) -> list[int]` qui trie `L` à l'aide d'un tri par comptage.
   (b) Montrer la terminaison de votre fonction.
   (c) Donner la complexité de votre fonction.

3 Tri par base binaire

Dans cette partie, la liste `L` contient uniquement des entiers positifs ou nuls. Le ***tri par base binaire*** consiste à trier `L` en utilisant la représentation binaire de ses éléments.

5. (a) Sans utiliser la fonction `bin` de Python, écrire une fonction `intToBin(n: int) -> str` qui renvoie la représentation binaire de `n` en commençant par le bit de poids faible. Si l'entier en entrée n'est pas positif ou nul, votre fonction déclenchera une erreur. Par exemple :

n	0	1	2	3	4	20	57
intToBin(n)	""	"1"	"01"	"11"	"001"	"00101"	"100111"

   (b) Écrire une fonction `listToBin(L: list[int]) -> list[str]` qui applique la fonction `intToBin` à chacun des éléments de `L`. Par exemple :

L	[3, 57, 0, 20, 1, 2, 4]
listToBin(L)	["11", "100111", "", "00101", "1", "01", "001"]

Soit `M = listToBin(L)`. Pour l'instant, les chaînes de caractères de `M` ne sont pas toutes de la même longueur, ce qui poserait problème dans la suite. Pour corriger ça, on ajoute des zéros à droite des éléments de `M` pour faire en sorte qu'ils aient tous la même taille.

6. (a) Écrire une fonction `tailleMax(M: list[str]) -> int` qui renvoie le maximum des tailles des éléments de `M`. Si `M` est vide, votre fonction renverra `-1`.

   (b) En déduire une fonction `completer(M: list[str]) -> list[str]` qui ajoute des zéros à droite des éléments de `M`. Dans la liste renvoyée, toutes les chaînes de caractères doivent avoir la même taille. Par exemple :

M	["11", "100111", "", "00101", "1", "01", "001"]
completer(M)	["110000", "100111", "000000", "001010", "100000", "010000", "001000"]

7. Soit `N = completer(listToBin(L))`.

   (a) Écrire une fonction `separerBit(N: list[str], i: int) -> (list[str], list[str])` qui renvoie un couple (`N0`, `N1`), où `N0` est la liste des éléments de `N` dont le bit d'indice `i` vaut zéro et `N1` est la liste des éléments de `N` dont le bit d'indice `i` vaut un. On pourra supposer sans le vérifier que `i` est un indice valide pour les éléments de `N`. Voir l'exemple ci-dessous.

   (b) En déduire une fonction `triBinStr(N: list[str]) -> list[str]` qui trie la liste donnée en entrée en interprétant ses éléments comme des entiers écrits en base 2. On expliquera succinctement la procédure utilisée. Voir l'exemple ci-dessous.

N	["110000", "100111", "000000", "001010", "100000", "010000", "001000"]
separerBit(N, 1)	(["100111", "000000", "001010", "100000", "001000"], ["110000", "010000"])
triBinStr(N)	["000000", "100000", "010000", "110000", "001000", "001010", "100111"]

Pour terminer le tri par base binaire, il ne reste plus qu'à convertir les chaînes de caractères vers les entiers dont elles sont les représentations en base 2.

8. (a) Sans utiliser la fonction `int` de Python, écrire une fonction `binToInt(s: str) -> int` qui renvoie l'entier dont la représentation binaire est `s`. Par exemple :

s	"000000"	"100000"	"010000"	"110000"	"001000"	"001010"	"100111"
binToInt(s)	0	1	2	3	4	20	57

   (b) À l'aide des fonctions précédentes, écrire une fonction `triBin(L: list[int]) -> list[int]` qui trie `L` à l'aide d'un tri par base binaire.

4 Tri à peigne

4.1 Implémentation

Rappel : `L` désigne la liste à trier et $n \in \mathbb{N}$ son nombre d'éléments. Le ***tri à peigne*** se décompose en plusieurs étapes, chaque étape étant caractérisée par un entier $p \in \mathbb{N}^*$ appelé la ***longueur du peigne***. Pour l'instant, intéressons-nous à l'étape associée à un peigne de longueur p. Pour cela on pose :

$$I_p = \Big\{ i \in [\![0, n-1]\!] : i + p \in [\![0, n-1]\!] \Big\},$$

puis :

→ On parcourt successivement les entiers $i \in I_p$.

→ Pour chaque i, on note x l'élément d'indice i de `L` et y celui d'indice $i+p$.

→ Si x est strictement plus grand que y, on échange x et y, sinon on ne fait rien.

9. Écrire une fonction `peigne(L: list[int], p: int) -> bool` qui applique la procédure décrite ci-dessus. Votre fonction doit modifier directement `L` et ne pas renvoyer de liste. Le booléen en sortie vaut `True` si au moins un échange a été effectué et `False` sinon. Votre fonction déclenchera une erreur si `p` $\leqslant$ `0`.

Le principe du tri peigne est d'appeler la fonction `peigne` en boucle jusqu'à obtention d'une liste triée. Pour cela on dispose d'un réel $R \in]1\,;+\infty[$ appelé le ***facteur de réduction***. Soit $(p_i)_{i\in\mathbb{N}}$ la suite d'entiers définie par :

$$p_0 = n \qquad \text{et} \qquad \forall i \in \mathbb{N} : p_{i+1} = \max\left(1, \lfloor p_i/R \rfloor\right)$$

Lors d'un tri peigne, la fonction `peigne` est appelée avec p_1, puis p_2, puis p_3 ... jusqu'à ce que la liste soit triée.

10. (a) Pourquoi n'est il pas nécessaire d'appeler la fonction `peigne` avec p_0 ?

    (b) Montrer que l'appel à `peigne(L, 1)` renvoie `False` si et seulement si `L` est triée. On attend une justification rapide ; utiliser un invariant de boucle n'est pas nécessaire.

    (c) Écrire une fonction `triPeigne(L: list[int], R: float) -> NoneType` qui trie `L` à l'aide du tri peigne avec un facteur de réduction égal à `R`. Votre fonction doit modifier `L` et ne rien renvoyer. On arrêtera l'exécution lorsque la liste est triée en utilisant le critère obtenu à la question précédente. Attention : la fonction `peigne` doit être appelée avec p_1, puis p_2, puis p_3, ... en particulier, vous ne devez pas intercaler d'appel à `peigne(L, 1)`.

11. (a) On s'intéresse au programme suivant :

```
print(triPeigne([1, 5, 2, -5], 1.05062025))
```

    Expliquer pourquoi ce programme ne permet pas de tester si la fonction `triPeigne` trie correctement la liste `[1, 5, 2, -5]`.

    (b) Donner une version corrigée du programme précédent.

4.2 Optimisation du facteur de réduction

On souhaite maintenant déterminer le facteur de réduction R donnant un temps d'exécution le plus court possible. Pour cela on va générer des listes aléatoires et comparer les temps d'exécution pour différentes valeurs de R. Dans la suite, on pourra utiliser la fonction « `random.randint(a: int, b: int) -> int` » qui renvoie un entier aléatoire de l'intervalle $[\![$`a, b`$]\!]$.

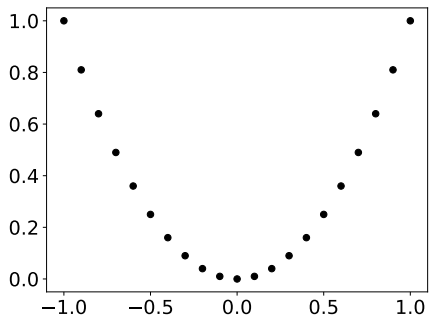
12. Écrire une fonction `listeAlea() -> list[int]` qui renvoie une liste dont la longueur est choisie aléatoirement entre 500 et 1000 et dont chaque élément est choisi aléatoirement entre -1000 et 1000.

La fonction « `time.perf_counter() -> float` » renvoie le nombre de secondes écoulées depuis le démarrage de l'ordinateur.

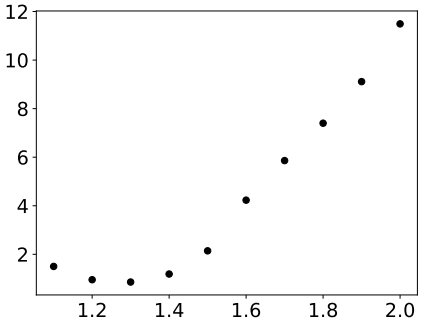
13. Écrire une fonction `chrono(R: float) -> float` qui génère 800 listes avec la fonction `listeAlea`, les trie avec la fonction `triPeigne`, puis renvoie le temps d'exécution total pour trier ces 800 listes. Le temps nécessaire pour générer les listes (c'est à dire les appels à `listeAlea`) ne seront pas comptabilisées.

Pour tracer le temps d'exécution de `listeAlea` en fonction de R, on va se servir du module `matplotlib.pyplot` dont voici un exemple d'utilisation :

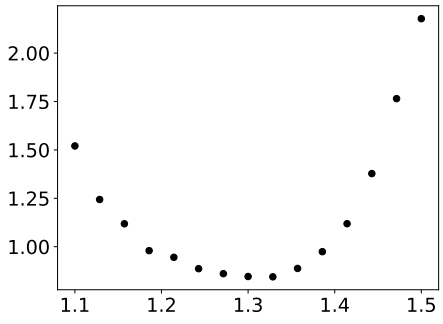
```
X = [-1 + 0.1*i for i in range(21)]
Y = [x*x for x in X]
plt.figure()
plt.plot(X, Y, "o", color = "black")
plt.show()
```



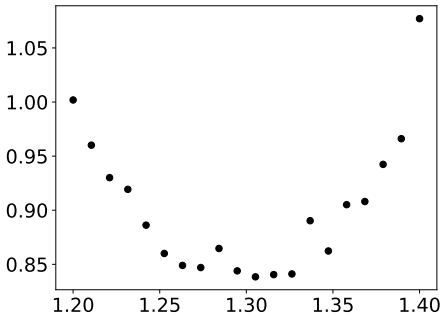
14. Écrire une fonction `traceChrono(a: float, b: float, m: int) -> NoneType` qui affiche un graphique représentant `chrono(R)` en fonction de `R`. La variable `R` prendra `m` valeurs différentes régulièrement espacées dans l'intervalle $[\![$`a ; b`$]\!]$. On pourra supposer sans le vérifier que `1 < a < b` et `m` $\geqslant$ `2`. Par exemple :



`traceChrono(1.1, 2, 10)`



`traceChrono(1.1, 1.5, 15)`



`traceChrono(1.2, 1.4, 20)`

15. Quel facteur de réduction utiliseriez vous dans un tri à peigne ? Justifier.

5 Tri par paquet

Dans cette partie, nous aurons besoin de calculer le minimum et le maximum d'une liste. Les listes que nous manipulerons contiendront des entiers, mais aussi des `None`. Ainsi, étant donnée une liste « `P: list[int ou None]` » contenant des entiers et/ou des `None`, le maximum (resp. minimum) de `P` est le maximum (resp. minimum) de `P` en excluant les éléments égaux à `None`. Par exemple, si :

```
P = [-8, 3, 1, -3, None, 7, 6, -8, None, 0, -1, -9, 4, 7, -7, None]
```

alors le maximum de `P` est `7` et son minimum est `-9`.

16. (a) Écrire une fonction `indiceMin(P: list[int ou NoneType]) -> int ou NoneType` qui renvoie l'indice `i` tel que `P[i]` est le minimum de `P`. Si `P` contient uniquement des `None`, votre fonction renverra `None`.

    (b) Dans cette question, on suppose que « `L: list[int]` » ne contient aucun `None` et est non vide. Déterminer en le justifiant ce que renvoie la fonction `bornes`.

```
def bornes(L):
    i = indiceMin(L)
    M = [-e for e in L]
    j = indiceMin(M)
    return L[i], L[j]
```

```
def copie(L):
    """L: list[int] -> list[int]"""
    return ...
```

On s'intéresse d'abord à un cas particulier du tri par paquet : si `L` est vide ou bien si tous ses éléments sont égaux, alors il suffit de renvoyer une copie de `L`.

17. (a) Compléter la fonction `copie` en remplaçant les points de suspension.

    (b) Quelle est la complexité de cette fonction ?

Jusqu'à la fin de cette partie, $\texttt{m} \in \mathbb{N}^*$ est un entier naturel non nul et `L` désigne la liste à trier. Dans le cas où `L` contient au moins deux entiers distincts, le ***tri par paquet*** consiste à partitionner `L` en `m` listes $\texttt{P}_0, \texttt{P}_1, \ldots, \texttt{P}_{\texttt{m-1}}$ telles que :

$$\forall (\texttt{i},\texttt{j}) \in [\![0, \texttt{m}-1]\!]^2, \forall \texttt{x} \in \texttt{P}_\texttt{i}, \forall \texttt{y} \in \texttt{P}_\texttt{j},\ \texttt{si i < j alors x < y;}$$

puis à appliquer un tri par sélection sur l'ensemble des $\texttt{P}_\texttt{i}$. Dans la suite, la liste $\texttt{P}_\texttt{i}$ sera appelée le ***paquet numéro*** `i`. Soit a le minimum de `L`, soit b son maximum et $\ell = \frac{b-a}{m}$. Alors le paquet numéro $i \in [\![0, m-1]\!]$ contient tous les éléments de `L` appartenant à I_i où :

$$\forall i \in [\![0, m-2]\!] : I_i = [\, a + i\ell \,;\, a + (i+1)\ell \,[\qquad \text{et pour } i = m-1 : I_i = [\, a + i\ell \,;\, a + (i+1)\ell \,]$$

Pour terminer le tri par paquet, on définit une liste « `T: list[int]` » initialement vide, puis on y ajoute tous les éléments du paquet numéro 0 (en commençant par le minimum, puis le deuxième minimum, etc ...), puis tous les éléments du paquet numéro 1, puis tous les éléments du paquet numéro 2, etc ...

18. Dans cette question on suppose que la liste `L` contient au moins deux éléments distincts.

    (a) Soit x un élément de `L` et m, a, b les réels définis ci-dessus. Expliquer comment déterminer en temps constant l'unique indice i tel que x appartient au paquet numéro i.

    (b) À l'aide de votre réponse à la question précédente, écrire une fonction :

    `partition(L: list[int], m: int) -> list[list[int]]`

    qui renvoie une liste `P` de taille `m` telle que `P[i]` contient le paquet numéro `i`.

19. Écrire une fonction `triPaquets(L: list[int], m: int) -> list[int]` qui trie `L` en utilisant un tri par paquets. Dans cette question, `L` est quelconque (elle ne contient pas nécessairement deux entiers distincts). Lorsqu'un entier est extrait de l'un des paquets `P[i]`, il sera remplacé par un `None` dans `P[i]`.