

Dans ce fichier, vous respecterez les consignes suivantes :

- Écrivez d'abord en commentaires (ligne débutant par #), le titre du TP, votre nom et prénom.
- Commencez chaque question par son numéro écrit en commentaires.
- Les questions demandant une réponse écrite seront rédigées en commentaires.
- Les questions demandant une réponse sous forme de fonction ou de script respecteront pointilleusement les noms de variables et de fonctions demandés.

Le nom de votre fichier python sera **impérativement** `dupont-jean-TpNote.py`, où `dupont` est à remplacer par votre nom et `jean` par votre prénom, les deux étant en **minuscules** (même la première lettre) et **sans caractère accentué**.

Dans la mesure du possible, on justifiera le code demandé par des invariants.

I Programmation

1. Écrire une fonction `random_list(n, k)` construisant une liste de n entiers tirés au hasard dans l'intervalle `range(k)`. On pourra utiliser la fonction `randrange` du module `random` à cet effet.
2. Écrire une fonction `counting_sort(k, t)` prenant en argument une liste `t` d'entiers appartenant à `range(k)` et retournant une copie triée de ce tableau. On utilisera impérativement l'algorithme suivant :
 - On construit un tableau `u` de taille `k` initialisé avec des 0.
 - On parcourt `t`. Pour chaque valeur `x` trouvée, on incrémente `u[x]`. À la fin du parcours, pour tout entier `i` de `range(k)`, `u[i]` contient donc le nombre d'occurrences de `i` dans `t`.
 - Il est alors facile de construire un tableau `r` trié répondant à la question posée.
3. Écrire suivant le même principe une fonction `bucket_sort(f, k, t)` retournant une copie de `t` triée suivant le critère `f`. Plus précisément, `f` doit être une fonction prenant ses valeurs dans `range(k)` et les éléments de la liste résultat sont triés par ordre croissant de leurs images par `f`.

De plus, on fera en sorte que le tri soit *stable*, c'est-à-dire que pour tout couple de valeurs `x` et `y` ayant même image par `f`, `x` et `y` apparaissent dans le même ordre dans `t` et dans la liste triée.

4. On se donne la fonction suivante pour trier une liste d'entiers.

PYTHON

```
def radix_sort(k, t):  
    """Retourne une copie de t triée par ordre croissant.  
    t doit contenir des entiers appartenant à range(k**2) """  
    def lp(x):  
        return x // k  
    def rp(x):  
        return x % k  
    u = bucket_sort(rp, k, t)  
    r = bucket_sort(lp, k, u)  
    return r
```

Vérifier (expérimentalement) sur une petite liste d'entier que cette fonction trie effectivement.

5. Écrire une fonction `radix_sort2(k, t)` inspirée de la fonction `radix_sort` qui, pour un tableau `t` contenant des entiers appartenant à `range(k**3)` renvoie une copie de `t` triée par ordre croissant.

II Étude de complexité

1. Expliquer (brièvement) et sous quelle hypothèse pourquoi la complexité de `random_list(n, k)` est un $O(n)$.
2. Justifier que la complexité de `counting_sort(n, k)` est un $O(\text{Max}(n, k))$.
3. Justifier que la complexité de `bucket_sort(n, k)` est un $O(\text{Max}(n, k))$.
4. Quelle est la complexité de `radix_sort`?
5. En admettant que la fonction `bucket_sort` répond bien à l'énoncé, justifier que `radix_sort` trie le tableau donné en argument. On pourra commencer par regarder le cas où `k` vaut 10 pour comprendre ce qu'il se passe et se poser la question : à quelle(s) condition(s) sur `lp(x), lp(y), rp(x), rp(y)` a-t-on $x \leq y$?