Spécialité Numérique et Sciences Informatiques
DS N°1

L'usage de la calculatrice n'est pas autorisé

1 Heure

Exercice 1 : Le Bucket Sort ou tri par paquet (15 points)

Dans cet exercice, on supposera que tous les nombres du tableau que l'on souhaite trier sont compris entre 0 **et** 100

Principe du tri par paquet

Imaginez qu'il faut trier 200 copies d'élèves par ordre alphabétique. Le tri par insertion ou le tri par sélection essaieraient de traiter le tas complet tout de suite. Le tri par paquet commence par faire 26 tas en fonction de la première lettre du nom, puis tri séparément chacun des paquets ainsi réalisés. Les paquets sont ensuite empilés dans l'ordre.

Programmer un tri par paquet

1. Pour trier chacun des paquets nous allons avoir besoin d'une fonction de tri. On se propose de réaliser un tri par insertion.

   1.1 On propose la fonction ci-dessous, recopiez sur votre copie le code en le complétant.

```
def tri_par_insertion(T) :
  for i in range (.......) :
    k = i
    clef = T[i]
    while  ....... and clef < T[k-1] :
      T[...] = T[....]
      k = .....
    T[k] = clef
  return T
```

   1.2 Quelle est la complexité temporelle minimale d'un tel algorithme ? Dans quel cas ?

   1.3 Quelle est la compléxité temporelle maximale ? Dans quel cas ?

2. Fabrication des paquets

   On propose la fonction `bucket(tab)` ci-dessous :

```
def bucket(tab) :
  L = [ [] for i in range(10)]
  for i in tab :
    j = i // 10
    L[j].append(i)
  return L
```

2.1 Combien de paquets sont constitués par la fonction `bucket(tab)` ?

2.2 Que retournera l'appel `bucket([45, 1, 78, 87, 7, 54, 41])` ?

2.3 Quelle est la compléxité temporelle de cette fonction ?

3. Concaténation des listes

La concaténation de listes est une opération où les éléments d'une liste sont ajoutés à la fin d'une autre liste. La méthode la plus conventionnelle pour effectuer la concaténation de liste, l'utilisation de l'opérateur « + » permet d'ajouter facilement l'ensemble d'une liste derrière l'autre liste et donc d'effectuer la concaténation, mais le problème avec l'opérateur + pour la concaténation de liste est qu'il crée une nouvelle liste pour chaque opération de concaténation de liste. Cela peut être très inefficace si vous utilisez l'opérateur + plusieurs fois dans une boucle.

3.1 Proposez une fonction `contatene(L1, L2)` qui renvoie une liste `L1` qui est la concaténation des deux listes `L1` et `L2`. On n'utilisera pas l'opérateur +, mais on utilisera la méthode `.append(elt)`.

Exemples d'utilisation :

```
>>> concatene([1, 2, 3], [5, 6])
[1, 2, 3, 5, 6]
```

3.2 Si l'on considère que la concaténation de deux listes : `L1` contenant n éléments et `L2` contenant p éléments prend un temps t, combien de temps prendra la concaténation de :

- `L1` contenant $2n$ éléments et `L2` contenant p éléments
- `L1` contenant n éléments et `L2` contenant $2p$ éléments
- `L1` contenant $3n$ éléments et `L2` contenant $3p$ éléments

4. Une Fonction de tri par paquet

4.1 En utilisant les trois fonctions précédemment programmées `tri_par-insertion`, `bucket` et `concatene` et en vous appuyant sur le principe du tri par paquets décrit en début d'exercice, proposer une fonction `bucket_sort(tab)` qui implémente ce tri.

4.2 En considérant les deux listes ci dessous et en leur appelant la fonction `bucket_sort` pour les trier, `L1 = [78, 14, 21, 45, 8, 36]` et `L2 = [14, 11, 17, 19, 13, 12]`, laquelle des deux sera la plus longue à trier ?

4.3 Le tri par paquet est-il un tri en place ? Est-il un tri stable ?

Exercice 2 : Les arrêts de bus (5 points)

Sur une route, il n maisons aux cordonnées $x_1, x_2, x_2, ...x_n$ positives ou nulles qui représentent la position des maisons exprimées en mètre.

Une ligne de bus va passer par cette route, et il faut placer les arrêts de bus. La contrainte imposée pour placer ces arrêts est que pour chaque maison, il y ait un arrêt à moins de 100 m ($d \leq 100$).

Il s'agit donc de trouver le nombre minimal m et les coordonnées de tels arrêts $a_1, a_2, a_3..., a_m$.

1. Pour des maisons aux coordonnées $x_1 = 180, x_2 = 200, x_3 = 370, x_4 = 390, x_5 = 590$, trouver une solution optimale pour placer les arrêts (nombre et position).
2. A quelle condition, sera-t-il impossible de trouver une solution répondant à la contrainte ? En déduire une précondition sur la liste passée en paramètre.
3. Pour placer les arrêts on peut donc proposer l'algorithme glouton ci-dessous.

Algorithme 1 : Placement des arrêts de bus

Entrées : Un tableau $x : x[0 : n]$, position des maisons, déjà trié, précondition respectée

Sorties : Un tableau $a : a[0 : m]$, position des arrêts

début

```
arrêts ← 0 ;
distance ← 100 ;
dernier ← −1 ;
pour i = 0 jusqu'à n faire
    si x_i > dernier alors
        a_arrêts = x_i + 100 ;
        dernier ← a_arrêts + distance ;
        arrêts ← arrêts + 1 ;
```

Implémenter cet algorithme en python sous la forme d'une fonction `arret_bus(tab)` où `tab` est une liste déjà triée, satisfaisant la précondition ennoncée à la question précédente représentant la position des maisons. Cette fonction doit renvoyer une liste représentant la position des arrêts de bus.