Algorithmique
TD n° 3 : Implementation et comparaison

Exercice 1 : recherche du k-ème élément

1. Dans un script python implémenter les différentes versions de l'algorithme de recherche du k-ème élément vues en cours.

Pour rappel voici les algorithmes de différentes versions :

```
Procedure Naif(T, k)
begin
  pour chaque i = 1...k faire
    im = indiceMin(T)
      si i < k alors
        T[im] := ∞;
      fin
    fin
    return T[im];
end
```

```
Procedure Trie(T, k)
begin
  trier(T);
  return T[k];
end
```

```
Procedure Tas(T,k)
begin
  MiseEnTas(T);
  pour chaque i = 1...k faire
    v = ExtraireMin(T)
  fin
  return v
end
```

La fonction indicePivot(T,bg,bd) pivote le tableau T entre bg et bd et renvoie l'indice du pivot de sorte que tous les éléments à gauche du pivot sont plus petits que le pivot et les éléments à droite plus grands.

```
Procedure QuickSelect(T, k, bg, bd)
begin
  ip := indicePivot(T, bg, bd)
  rgpivot := ip - bg + 1
  si k < rgpivot alors
    return QuickSelect(T, k, bg, ip - 1)
  sinon si k > rgpivot alors
     return QuickSelec(T, k - rgpivot, ip + 1, bd)
  sinon
    return T[ip]
  end
end
```

2. Rappeler la complexité de ces 4 algorithmes

3. Grace au code suivant évaluer les temps d'exécution de chaque algorithme :

```
import random
import timeit

data = list(range(0, 1000))
random.shuffle(data)
k = random.randrange(1000)

timeit.timeit('Naif(data.copy(), k)', number=1, globals=globals())
timeit.timeit('Trie(data.copy(), k)', number=1, globals=globals())
timeit.timeit('Tas(data.copy(), k)', number=1, globals=globals())
timeit.timeit('QuickSelect(data.copy(), k, 0, len(data)-1)', number=1, globals=globals())
```

timeit renvoie le temps d'execution de la fonction en secondes. Est-ce que ces temps d'execution correspondent a ce que l'on attend ? Si non, pourquoi.

4. Modifier la taille du tableau `data` et observer comment ces temps d'execution évoluent. Cela correspond-t-il a l'attendu ?

Exercice 2 : fusionner k tableaux d'entriers triés

On veut fusionner k tableaux d'entiers triés. Chaque tableau de taille n. Un algorithme naif est de fusionner le premier tableau avec le deuxième, de fusionner le résultat avec le troisième, etc.

1. Implémenter cet algorithme naïf

2. Donner un algorithme *diviser pour régner* et l'implémeter.

3. Évaluer le temps de calcul de ces algorithmes grace aux outils vus dans l'exercice précédant. Est-ce que vos implémentations ont des temps de calcul qui correspondent aux complexités théoriques ?

Exercice 3 : Tri en surfusion

On veut maintenant généraliser l'idée du tri fusion. A chaque étape, au lieu de diviser le tableau en 2, on souhaite le diviser en k. On pourra ensuite fusionner les k sous tableaux en utilisant les résultats de l'exercice précédent.

1. Implémenter le tri fusion modifié pour diviser en k sous tableaux.

2. pour un k fixé, quelle est la complexité asymptotique de votre algorithme ?

3. Évaluer le temps de calcul pour plusieurs valeurs de k (par exemple de 2 à 10). Comment varie-t-il ? Est-ce attendu au regard de la complexité asymptotique ?