# CM 1

## QUESTION DE REFLEXION

### *Fibonacci*

On a programmé la fonction de Fibonacci en récursif.

On mesure le temps mis pour faire le calcul, on obtient :

```
n=5 fibo=5 temps en ms =0
n=10 fibo=55 temps en ms =0
n=15 fibo=610 temps en ms =0
n=20 fibo=6765 temps en ms =0
n=25 fibo=75025 temps en ms =0
n=30 fibo=832040 temps en ms =3
n=35 fibo=9227465 temps en ms = ? (40)
n=40 fibo=102334155 temps en ms = ? 405
n=45 fibo=1134903170 temps en ms = ? 4470
n=50 fibo=-298632863 temps en ms =49901
```

Sauriez-vous répondre aux questions suivantes (et comment faites-vous ?)

- Etes-vous d'accord pour mesurer le temps de fibonacci (100) sur une machine de l'ENSEM (sous-entendu j'attends la fin du calcul pour repartir ?)
- Si j'ai deux heures à disposition, quelle valeur de fibonacci (en récursif) puis-je calculer ?

Quelle loi sous-jacente exprime la relation entre n et le temps mis à calculer la valeur?

Pouvez-vous expliquer comment vous avez obtenu cette loi (classe de complexité) ?

En quoi le fait de savoir que la fonction est récursive influence l'analyse ?

### *Tri*

Soit un algorithme de tri (peu importe lequel), le principe exprimé ci-dessus est-il encore valable ?

Pourquoi ? Si non expliquez comment vous pouvez essayer d'estimer la loi qui régit la taille des données et le temps de calcul

Pouvez-vous faire la relation avec les différentes notions de complexité (exacte, pire cas, moyenne, etc) ?

### *Mise en pratique et réflexion*

Pour répondre à cette question il n'est pas utile de connaître le tri par tas. Il s'agit ici de réfléchir … ☺

Si vous êtes intéressés, voyez page 343 à 347 de Gaudel et al.

Une première version (algo 1) est proposée, puis une seconde (algo 2).

**Algo 1**

*Soit t[1..n] de element les données*

```
p<--0
Tant que p <n faire
    ajouter(t,p,t[p+1]) // on ajoute le nouveau
    p++
FinTantQue

Tant que p>1 faire
    min <--détruire(t,p)
    p--
    t[p+1]<--min
FinTantQue
```

**Algo 2**

*Soit t[1..n] de element les données*

```
Pour p de n div 2 à 1 faire
    Ordonner(t,p)
FinTantQue

p<--n

Tant que p>1 faire
    min <--détruire(t,p)
    p--
    t[p+1]<--min
FinTantQue
```

La classe de complexité de la partie grisée italique est en $O(N \log_2 N)$ avec N la taille des données.

Que peut-on dire des ***classes de complexités*** respectives de ces deux algorithmes (algo 1 et algo 2), comment les comparer (laquelle est plus faible que l'autre, etc.) ? A quelle condition le second sera plus performant que le premier ?