OPTION CALCUL FORMEL
TP COMPLEXITÉ SAGE

CHRISTOPHE RITZENTHALER

1. Complexité et tris

Exercice 1. (1) Implémenter le tri par sélection et le tri fusion.

(2) Comparer les temps de ces deux méthodes. On fera les tests sur des listes aléatoires de longueur 100 à 1000 avec un pas de 10. Pour afficher deux graphes p1 et p2 superposés, on utilisera la fonction `show(p1+p2)`.

2. Exponentiation rapide

Exercice 2. (1) Implémenter les algorithmes d'exponentiation rapide.

Dans certains protocoles cryptographiques, il est important qu'on ne puisse distinguer les opérations en fonction des bits de n. Ceci est réalisé grâce à l'échelle de Montgomery que l'on décrit ci-dessous sur un exemple $n = 314$.

i	7	6	5	4	3	2	1	0
n_i	0	0	1	1	1	0	1	0
(x_1, x_2)	(x^2, x^3)	(x^4, x^5)	(x^9, x^{10})	(x^{19}, x^{20})	(x^{39}, x^{40})	(x^{78}, x^{79})	(x^{157}, x^{158})	(x^{314}, x^{315})

Il est en effet clair que les lorsque $n_i = 0$, le terme $x_1 \to x_1^2$ et sinon $x_1 \to x_1 x_2 = x_1^2 \cdot x$.

(2) Implémenter un algorithme qui correspond à cette méthode.

Lorsque l'inversion est rapide dans G, il peut être bon d'utiliser x^{-1} dans les calculs. Par exemple si $n = 2^k - 1$ alors au lieu de calculer avec $n = (1, \dots, 1)_2$ (et donc de faire $k-1$ carrés et $k-1$ multiplications) on peut calculer x^{2^k} en k carrés et une multiplication par x^{-1}. On utilise donc des développements signés $n = (n_{l-1}, \dots, n_0)_s = \sum_{i=0}^{l-1} n_i 2^i$ où $n_i \in \{0, 1, -1\}$. Ceci est utile si le poids de Hamming du développement signé est inférieur au poids de Hamming du développement binaire. Une possibilité (pas optimale) est la suivante et permet d'obtenir une représentation NAF (non-adjacent form)

```
1  def naf(n):
2      L=[]
3      E=n
4      while E>=1:
5          if E%2==1:
6              z=2-(E%4)
7              L.append(z)
8          else:
9              z=0
10             L.append(z)
11         E=(E-z)/2
12         print E,z
13     L.reverse()
14     return L
```

(3) Programmer cette écriture et l'exponentiation rapide associée.

3. Séries formellles, division euclidienne

Exercice 3. Implémenter l'algorithme de calcul de la division euclidienne pour les polynômes basé sur l'inversion des séries formelles. On utilisera `PowerSeriesRing(QQ)` pour définir les séries formelles. L'opération de troncature est `truncate()`.

Exercice 4. Soit F une série formelle de terme constant nul. Écrire un programme qui calcule $\log(1+F)$ en utilisant l'expression

$$\log(1+F) = \int \frac{F'}{1+F}.$$