EA4 – Éléments d'algorithmique
TD n° 1 : opérations arithmétiques

Exercice 1 : des limites du progrès...

Il y a 25 ans, un ordinateur faisait dix millions d'opérations par seconde et implémentait un algorithme de tri demandant $50 \times n \times \log_{10} n$ opérations pour un tableau d'entrée de taille n. On souhaite le comparer à un ordinateur 100 fois plus rapide, mais sur lequel tourne un (moins bon) algorithme de tri demandant n^2 opérations. Quels sont les temps de calcul de chacun pour une entrée de taille $n = 10^6$? et $n = 10^7$?
Inversement, quelle est la taille maximale d'un tableau qui peut être traité en 1 heure dans chacune des deux configurations ?

Exercice 2 : multiplication du paysan russe

Considérons l'algorithme de multiplication suivant, dit « *du paysan russe* » :

```
def multiplication_russe(m, n):
  res = 0
  while n != 0 :
    if n%2 == 1 : res += m
    m *= 2
    n //= 2
  return res
```

1. Tester cet algorithme pour le couple de valeurs `m = 13, n = 14`.

2. Montrer que cet algorithme effectue bien la multiplication de `m` par `n`.

3. Calculer le nombre d'additions, de multiplications par 2 et de divisions par 2 effectuées au pire lors de l'exécution de cet algorithme en fonction de `n`.

4. En base 2, comment se traduit la multiplication par 2 ? la division par 2 ? Comment tester si un nombre binaire est impair ?
Écrire les nombres 13 et 14 en base 2 et appliquer l'algorithme de multiplication ci-dessus à ces nombres.

5. Comparer avec l'algorithme de multiplication usuel.

Exercice 3 : exponentiation binaire, ou *exponentiation rapide*

De même que la multiplication par un entier n, $a \mapsto a \times n$, correspond à une addition répétée n fois (ou $n - 1$, selon la valeur considérée comme initiale), l'exponentiation d'exposant n, $a \mapsto a^n$, correspond à une multiplication répétée n fois.
En déduire un algorithme `exponentiation_rapide(m, n)` permettant de calculer m^n en $O(\log n)$ additions ou multiplications.

Exercice 4 : évaluation de polynômes

1. Proposer un algorithme le plus naïf possible qui, étant donné un polynôme P et une valeur x, calcule $P(x)$. On supposera que P est décrit par un tableau contenant, en case d'indice i, le cœfficient du monôme de degré i.
2. Appliquer votre algorithme au polynôme $P(x) = x^4 + x^3 + x^2 + x + 1$ avec $x = 3$. Que constatez-vous ?
3. Quelle est le nombre d'additions et de multiplications effectuées lors de l'exécution de cet algorithme sur un polynôme de degré n ?
4. Comment peut-on améliorer la complexité de cet algorithme en utilisant l'exponentiation binaire ?
5. Montrer que l'algorithme suivant résout le même problème :

```
def horner(P, x) :
  res = 0
  for coeff in P[::-1] :      # effectue un parcours du tableau à l'envers
    res = res * x + coeff
  return res
```

   Quelle est le nombre d'additions et de multiplications d'entiers effectuées lors de l'exécution de cet algorithme sur un polynôme de degré n ?

Exercice 5 : nombres premiers

1. Proposer un algorithme `est_premier(p)` *le plus naïf possible* permettant de déterminer si un entier `p` est premier. Quelle est sa complexité ? Comment peut-on l'améliorer ?

On considère maintenant l'algorithme suivant :

```
def eratosthene(n) :
  tab = [False, False] + [True]*(n-1) # tab = [False,False,True,True,...,True]
  for i in range(2, n+1) :
    if tab[i] :
      for k in range(2*i, n+1, i) : # depuis 2i, par pas de i, sans dépasser n
        tab[k] = False
  return tab
```

2. Exécuter l'algorithme pour `n = 10`.
3. Que représente le tableau calculé par `eratosthene(n)` ? Justifier.
4. Calculer un majorant du nombre d'additions et de multiplications d'entiers effectuées par l'algorithme pour une entrée `n`.
5. Pour vous convaincre de l'impact pratique des différences de complexité, comparer les temps de calcul de `[ p for p in range(10**6) if est_premier(p) ]` et de `[ p for p,b in enumerate(eratosthene(10**6)) if b ]`.