Preuve et Analyse des Algorithmes
corrigé

12 juin 2013

1. Temps de calcul. Une implantation d'un algorithme de temps de calcul $n\sqrt{n}$ traite une instance de taille $n = 64$ en 8 micro-secondes.

1. Quel sera le temps de calcul pour une instance de taille 512 ?

$$T(512) = T(8 \times 64) = 8\sqrt{8}T(64) = 128\sqrt{2} \sim 180\mu\text{sec}$$

2. Donner un exemple d'algorithme ayant cette complexité.
   Nous en avons vu un seul : algorithme de Karatsuba.

2. Comparaison des algorithmes de tris. Pour un entier n, et algorithme de tri X, on note :

$$t_X(n) = \frac{\bar{T}_X(2n)}{\bar{T}_X(n)}, \quad v_X(n) = \frac{\bar{V}_X(2n)}{\bar{V}_X(n)}$$

où $\bar{T}_X(n)$ (respectivement $\bar{V}_X(n)$) désigne la moyenne du temps de calcul (respectivement du volume de mémoire auxiliaire) utilisé pour le traitement par X d'une instance de taille n.

Attention, l'espace occupé par le tableau à trier ne doit pas être comptabilisé dans le calcul du volume de mémoire.

1. Déterminer $t_L(n)$, $t_R(n)$, $t_S(n)$ pour les algortihmes de tri : linéaire, rapide (quick-sort), et sélection.

$$t_L(n) = \frac{2n}{n} = 2, \quad t_R(n) = \frac{2n\log(2n)}{n\log n} = 2 + \frac{2\log 2}{\log n}, \quad t_S(n) = \frac{4n^2}{n^2} = 4.$$

2. Déterminer les fonctions $v_X(n)$ pour les mêmes algorithmes.

$$v_L(n) = \frac{2n}{n} = 2, \quad v_R(n) = \frac{\log(2n)}{\log n} = 1 + \frac{\log 2}{\log n}, \quad v_S(n) = \frac{1}{1} = 1.$$

3. Quelles conclusions tirer de ces petits calculs ?
   Le tri rapide est le meilleur compromis temps/mémoire.

```
typedef struct {
  double real, img;
} cmp;

cmp K_prd( cmp x, cmp y)
{ cmp p;
  double  r, i, t;
  t = ( x.img + x.real )
     * ( y.real + y.im );
  i =  x.img  * y.img ;
  r =  x.real * y.real;
  p.img = r - i;
  p.real= s - r - i;
  return p;
}
```

3. Optimisation à la Karatsuba.

1. Que calcule la fonction `prd` ?
   Elle calcule le produit de deux nombres complexes.
2. S'inspirer de la méthode de Karatsuba pour utiliser une multiplication de moins.
3. On suppose que pour deux `double`, le temps de multiplication est λ-fois plus important que celui d'une addition ou d'une soustraction. Comparer les performances en fonction de λ.
   On néglige les temps d'affectation. L'algorithme standard à un coût de $4\lambda+2$ additions. L'algorithme modifié a un coût de $3\lambda + 5$ additions, il est plus intéressant dès $\lambda > 3$.

4. Implantation du tri linéaire.

On suppose une implanation du trilinéaire en langage C

```
typedef unsigned long long ullong;
void trilin( ullong *t , ullong n );
```

1. Quelles sont les préconditions d'utilisation ?
   Il faut utiliser un tableau auxiliaire de taille n, les valeurs du tableau à trier sont inférieures à n.
2. Estimer le temps de calcul d'une instance de taille 2^{20} sur la machine `obelix`.
   Les opérations mises en jeu sont simple, on peut estimer utiliser une centaine d'opérations élémentaires par itération. Vu la fréquence d'horloge, je propose un temps de calcul de l'ordre de
   $$\frac{2^{20} \times 2^7}{3000 \times 10^6} \sim \frac{2^{27}}{2^{31}} = \frac{1}{16}\text{sec}$$
3. pour la taille 2^{24} ?
   16 fois plus, soit 1 seconde.
4. pour la taille 2^{26} ? 64 fois plus soit 1 minute. Notons que le programme utilisera $2^{27} \times 2^3 = 2^{30}$ octets de mémoire. Il y a un risque important de dysfonctionnement mémoire (swap) !

```
[drm@obelix exam-12]$ lscpu
Architecture: i686    Mode: 32-bit Vitesse proc. en MHz: 3000.000

[drm@obelix exam-12]$ free -b
             total       used       free     shared
Mem:    1582931968  837398528  745533440          0
-/+ buffers/cache:  389427200 1193504768
Swap:   3187666944          0 3187666944

[drm@obelix exam-12]$ echo 'l(1582931968)/l(2)'  | bc -l
30.559952105962841097 56
```

FIG. 1 – capacités de la machine `obelix`

5. Analyse d'une fonction récursive. On note $K(n)$, le nombre de comparaisons réalisées par la fonction récursive proc quand elle est appliquée à un tableau de n entiers.

1. Donner une formule récursive du temps de calcul.
   Il existe une constante K telle que, pour $n > 1$,
   $$K(n) = 2K(n/2) + K,$$
2. Donner $K(n)$ en fonction de n.
   linéaire
3. Mézaufait, que fait cete fonction ?
   recherche récursive de minimum.

```
int proc( int* t, int n )                    1
{  int x, y, p;                              2
   if ( n > 1 ) {                            3
      p = n / 2 + ( n & 1 );                 4
      x = proc( t,   n/2    );               5
      y = proc( t + n/2, p);                 6
      if ( x > y )  x = y;                   7
    }                                        8
   else x = *t;                              9
   return x;                                 10
}                                            11
```

6. Ordre d'une permutation. On considère les permutations de l'ensemble $\{0, 1, \ldots, n-1\}$. Une permutation π est représentée naturellement par un tableau de taille n. Pour un entier $x < n$, on définit la suite d'entiers $(x_n)_{n\geq 0}$

$$x_0 := x, \qquad \forall n > 0 \quad x_n = \pi(x_{n-1}),$$

on note alors $r(x)$ le plus petit entier strictement positif tel que $x_0 = x_{r(x)}$. L'ordre de π est le PPCM des valeurs de r, un tel ppcm peut être calculé de manière itérative :

$$\text{PPCM}(a_1, a_2, a_3, \ldots) = \text{PPCM}(\text{PPCM}(\text{PPCM}(a_1, a_2), a_3), \ldots)$$

1. Déterminer les valeurs de r, puis l'ordre de la permutation, correspondant au tableau de taille 15 :

   0 2 4 6 1 10 12 14 8 3 5 7 9 11 13

   On calcule les images sucéssives : $0 \to 0$ i.e. $r(0) = 1$, $1 \to 2 \to 4 \to 1$ i.e. $r(1) = 3$, $3 \to 6 \to 12 \to 9 \to 3$ i.e. $r(3) = 4$, $5 \to 10 \to 5$ i.e. $r(5) = 2$, $7 \to 14 \to 13 \to 11 \to 73$ i.e. $r(7) = 4$, $8 \to 8$ i.e. $r(8) = 1$. Le PPCM vaut 12.
2. L'entier $r(x)$ est effectivement bien défini, pourquoi ?
   La suite x_n prend un nombre fini de valeurs, il existe $i < j$ tels que $x_i = x_j$, et, commme π est une bijection, $\pi^{j-i}(x_0) = x_0$.

3. Ecrire une fonction `ordre(p, n)` qui calcule l'ordre de la permutation p.

```
int ordre( int* p, int n )                                    1
{  int res = 1, r;                                            2
   int vu[n];                                                 3
   int x, y;                                                  4
   for( x = 0; x < n; x++ )                                   5
       vu[x] = 0;                                             6
   for( x = 0; x < n; x++ )                                   7
     if ( ! vu[x]  ) {                                        8
        r = 0;                                                9
        y = x;                                                10
        do {                                                  11
              y = pi[y];                                      12
           vu[y] = 1;                                         13
           r++;                                               14
        } while ( y!=x );                                     15
        res = ppcm(res, r );                                  16
     }                                                        17
   return res;                                                18
}                                                             19
```