Algorithmique TD1

Magistère Informatique 1ère Année

17 Septembre 2008

Exercice 1 Le programme suivant calcule le PGCD de deux entiers strictement positifs :

```
int pgcd(int a, int b){
    int x = a ; int y = b ;
    while (x != y)
       if (x > y)
           x = x-y ;
       else
       y = y-x ;
    return x ;
}
```

Montrer que l'algorithme pgcd termine et qu'il est correct. Montrer que l'algorithme (d'Euclide) suivant calcule aussi le PGCD de deux entiers :

```
int euclide(int a, int b){
    int r ; int x = a ; int y = b ;
    while (y != 0){
       r = x%y ;
       x = y;
       y = r;
    }
    return x ;
}
```

Exercice 2 La suite de Fibonacci définie par :

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_n = u_{n-2} + u_{n-1}, \quad pour \quad n > 1 \end{cases}$$

est une suite linéaire récurrente d'ordre 2. On peut calculer chacun de ses termes en temps constant, en calculant la racine de l'équation caractéristique associée $r^2 - r - 1 = 0$, mais considérons ici un calcul moins sophistiqué pour chaque terme u_n

```
int fib(int n){
    int t ; int x = 0 ; int y = 1 ; int i = 2 ;
    if (n == 0)
       return x ;
    if (n == 1)
```

```
       return y ;
    while (i <= n){
       t = x;
       x = y;
       y = y + t;
       ++i ;
    }
    return y ;
}
```

Montrer que l'algorithme fib termine et qu'il est correct.

Exercice 3 Le programme suivant calcule la factorielle d'un entier :

```
int fact(int n){
    int r ;
    if (n == 0)
       r = 1;
    else
       r = n*fact(n-1) ;
    return r ;
}
```

Montrer que l'algorithme fact termine et qu'il est correct.

Exercice 4 Prouver la terminaison de la fonction d'Ackermann implémentée par :

```
int ack(int m, int n){
    if (m == 0)
       return n+1 ;
    else
       if (n == 0)
           return ack(m-1, 1) ;
       else
           return ack(m-1, ack(m, n-1)) ;
}
```

Exercice 5 Prouver la terminaison de la fonction 91 de McCarthy implémentée par :

```
int f91(int x){
     if (x > 100)
        return r-10 ;
     else
        return f91(f91(x+11)) ;
}
```

Exercice 6 Prouver la correction et la terminaison de la recherche dichotomique :

```
fonction chercheDico (t:tableau[1..n] de reels; n:entier; c:reel):entier
Debut
  i <- 1; j <- n
  tq i < j faire
    m <- (i+j) div 2
    si c <= t[m] alors j <- m sinon i <- m+1 fsi
  ftq
  retourner i
Fin
```

Exercice 7 Soit l'algorithme suivant de la procédure de fusion du tri fusion :

```
procedure fusion(t: tableau[1..n] de reels; i,j,k: entiers)
Debut
     s:tableau[i..j]
     a <- i; b <- k+1; c <- i
     tq a <= k et b <= j faire
        si t[a] <= t[b] alors s[c] <- t[a]; a++; c++
                        sinon s[c] <- t[b]; b++; c++
        finsi
     ftq
     tq a <= k faire s[c] <- t[a]; a++; c++ ftq
     tq b <= j faire s[c] <- t[b]; b++; c++ ftq

     t[i..j] <- s[i..j]
fin
```

1. Montrer que la procédure *fusion* termine.
2. Montrer que la procédure *fusion* est correcte.

Exercice 8 Soit l'algorithme suivant du tri par sélection :

```
procedure tri_selection(t : tableau[1..n] de reels ; g, d : entiers)
debut
     si d > g
     alors
        m <- min_index(t, g, d)
        e
        echanger(t, m, g)
        tri_selection(t, g+1, d)
fin
```

1. Écrire l'algorithme de la fonction *min_index* :

   `procedure min_index(tableau[1..n] de reels ; g, d : entiers)`

   qui retourne l'indice du plus petit élément de $t[g..d]$.
2. Montrer que votre fonction *min_index* est correcte et termine.
3. Montrer la correction ainsi que la terminaison de *tri_selection*.

Exercice 9 L'algorithme du tri rapide est le suivant :

```
procedure partition (t : tableau[1..n] de reels ; i, j : entiers)
debut
    echanger(t, i, (i+j) div 2)
    pivot <- t[i] ; g <- i+1 ; d <-j
    tantque g < d faire
        tantque g <= j et t[g] <= pivot faire
            g++
        fintantque
        tantque d >= i et t[d] > pivot faire
            d--
        fintantque
        si g < d
        alors
            e
            echanger(t, g, d)
        finsi
    fintantque
    si t[d] > pivot
    alors
        d--
    finsi
    echanger(t, i, d)
    retourner d
fin

procedure tri_rapide (t : tableau[1..n] de reels ; g, d : entiers)
debut
    si g < d
    alors
        k <- partition(t, g, d)
        tri_rapide(t, g, k-1)
        tri_rapide(t, k+1, d)
fin
```

1. Montrer que la procédure *partition* est correcte et termine.
2. Montrer que la procédure *tri_rapide* est correcte et termine.