24/9/2019

Serie 1

EX1

1) Premier (p) //booléen vrai si p est premier, faux sinon.

```
Début
    i := 2 ;
  tantque (pmodi ≠ 0) alors
        i := i+1
  ftq
    si (i ≠ p) alors retourner (faux);
       sinon retourner (vrai);
    fsi
  fin
```

Autre Meth

ou :

```
tant que (pmodi ≠ 0) et (i ≤ √p) faire
        i := i+1;
ftq
si (i > √p) alors retourner (vrai)
  sinon retourner (faux)
fsi
fin
```

Implémentation
Pb → Algo → prog
Analyse
codage

p = 13

13 mod 2 = 1 ≠ 0
13 mod 3 = 1 ≠ 0
13 mod 4 = 1 ≠ 0
...
13 = 0

Abstraction:
n → [nbp] → B

1

2)

```
nbs_premiers (n)
debut
    cpt := 0;
    pour p := 2 à n faire
        si Premier(p) alors cpt++
    fsi
    fpour
    retourner (cpt);
fin
```

EX2

a)

```
plus_petit_entier (n)
debut
    p := 0 ; x := 1;
    tant que (x ≤ n) faire
        p := p + 1;
        x := x * 2;
    ftq
    retourne (p);
fin
```

b) p est le nbr d'itérations de la boucle tant que p vérifie les itérations suivantes

$$2^{p-1} \leq n < 2^p$$

$$p - 1 \leq \log_2(n) < p$$

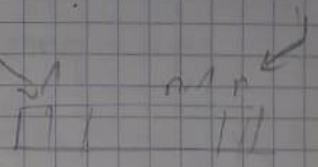
$$E[\log_2(n)] = p - 1$$

$$p = E[\log_2(n)] + 1$$

EX3

a)

```
Inverse (T, n)
début
    i := 1 ; j := n ;
    tantque (i < j) faire
        x := T[i] ;
        T[i] := T[j] ;
        T[j] := x ;
        i++
        j--
    ftq
    retourne (T) ;
fin
```



b)

```
Inverse (T, n)
début
    pour i := 1 à n/2 faire
        x := T[i] ;
        T[i] := T[n - i + 1] ;
        T[n - i + 1] := x ;
    fp
    retourne (T)
fin
```

3

1/10/2019

b)

```
Somme_binaire (T1, T2, n)
debut
    r := 0 ;
    pour i := 1 à n faire
        $ := T1[i] + T2[i] + r
        T3[i] := $ mod 2 ;
        r := $ div 2 ;
    fpour
    retourne (T3)
Fin
```

T_1 : n | n-1 | ... | 2 | 1

T_2 : n | n-1 | ... | 2 | 1

T_3 :

$11-7|_{10} = 0\,1\,1\,1\,0\,1\,0\,1$

$22|_{10} = 0\,0\,0\,1\,0\,1\,1\,0$

$1\,0\,0\,0\,0\,1\,0\,1\,1$

respectivement $T_1[i]$, $T_2[i]$, $T_3[i]$

$O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(2^n)$

Ex 4

a)

1) L'Algorithme A_1 —temps d'exécution→ $T_1(n) = 9(n^2) = O(n^2)$

" " A_2 → $T_2(n) = 100n + 96 = O(n)$

Donc l'algorithme A_2 est meilleur par / à A_2 selon l'ordre asymptotique grand O (pour n assez grand)

4

3) $T_1(n) = 9n^2$ et $T_2(n) = 100n + 96$

$T_1(n) - T_2(n) = 9n^2 - 100n - 96$

$T_3(m) = 9m^2 - 100m - 96$

$\Delta = 100^2 + 4 \times 9 \times 96$

$\Delta = 13456 \Rightarrow \sqrt{\Delta} = 16$

$m_1 = -0,8$ $m_2 = 12$

n	0		12		$+\infty$
T_3		−	0	+	

a) $T_1(10) = 900$

$T_2(10) = 1096$

donc l'hypothèse n'est pas vérifiée pour $n = 10$

$\Rightarrow$ Chercher le rang n_0 à partir duquel l'Algo A_2 est meilleur par rapport à A_1

donc $n_1 = 12 \Rightarrow$ Algorithme T_2 meilleur par rapport à T_1

4)

```
Alg 1
Debut
debut
    A1 ;
    A2 ;
Fin
```

La complexité de l'Algo 1 est la somme des 2 complexités des Algos A_1 et A_2

$T_1(n) + T_2(n) = O(n^2) + O(n)$

$= O(\max(n^2, n))$

$= O(n^2)$

b) $\log(n!) = O(n \log n)$

$\log n! = \log 1 + \log 2 + \dots + \log n$

$\leq \log n + \log n + \dots + \log n$

$\leq n \log n$

5

8/10/2019

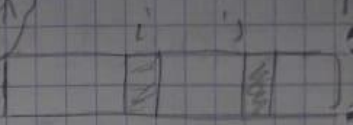
EXS

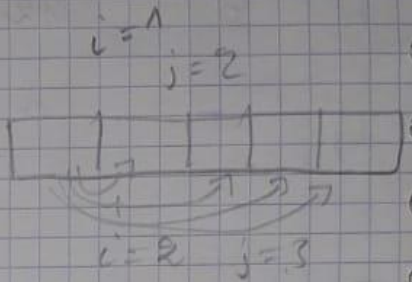
a)

```
Recher_Som (T, n, v)
debut
    pour i := 1 à n-1 faire
        pour j := i+1 à n faire
            si (T[i] + T[j] = v) alors
                retourne (i, j);
            fsi
        fpour
    fpour
    retourne (0,0);
fin
```

$O(n-i)$

nb d'itération = $n - i - 1 + 1$





$$O\left(\sum_{i=1}^{n-1} (n-i)\right) = O((n-1)+\dots+1)$$
$$= O\left(\frac{n(n-1)}{2}\right) = O(n^2)$$

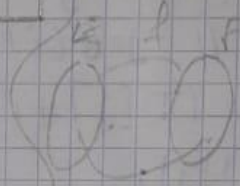
b)

```
Rech_som (T, n, v)
debut
    pour i := 1 à n-1 faire
        j := Rech_elt (T, i+1, n, v-T[i])
        si (j ≠ 0) alors retourner (i, j);
        fsi
    fpour
    retourne (0,0);
fin
```

6

```
Rech_elt (T, k, l, n)
debut
    pour j := k à l faire
        si (T[j] = n) retourne (j);
        fsi
    fpour
    retourner(0);
fin
```

surj



Img(f) = F

[1, 2, ..., n]

T[i] ∈ [1, n]

si n'importe pas surjective

EX6

a) Permutation (T, n)

```
debut
    pour i := 1 à n faire
        si (T[i] < 1 ou T[i] > n)
        alors retourner (faux);
        fsi
        pour j := i+1 à n faire
            si (T[i] = T[j]) retourner (faux);
            fsi
        fpour
    fpour
fin
```

surject

injection

O(1)

O(n-i)

2 boucles => quadratique

7

$$O\left(\sum_{i=1}^{n-1} 1+(n-i)\right) = O\left(n + \frac{(n-1)\,n}{2}\right) = O(n^2)$$

15/10/2019

b)

	1	2	3	4	5	6	7	8
T	6	2	4	1	7	3	5	8

les différents cycle constituent une partition d'ensemble $\{1,2,..n\}$

$\text{Card}\{1,2,...,n\} = \sum_{i=1}^{m} \text{Card}\, C_i$

$\{1,6,3,4\}\{2\}\{5,7\}\{8\}$

$4 + 1 + 2 + 1 = 8$

cycle de $i = \{i, \sigma(i), \sigma^2(i), ..., \sigma^{k-1}(i)\}$

où k représente le + plus entier vérifiant $\sigma^k(i) = i$

```
Longueur_cycle (i,T)
début
    k:= 1   // Car cycle de i contient i
    j:= i   // j = σ^0(i)
    tant que (T[j] ≠ i) faire
        j:= T[j];
        k:= k + 1;
    ftq
    retourner (k);
fin
```

cycle de 1

$1 = \sigma^0(1)$

$\sigma = \sigma^1(1)$

$3 = \sigma(6) = \sigma^2(1)$

$4 = \sigma^3(1)$

$1 = \sigma^4(1)$

i	k
1	1
6	2
3	3
4	4

cycle 2

$T[2] = 2 = 2$

Complexité linéaire $O(n)$

c)

```
Nbre_cycle (T,n)                                   | marq
début                                              |
    (pour i:= 1 à n faire                          |
        marq[i] := faux;) O(n)                     |
    fpour                                          |
    nc := 0 ;  // Nbre de cycles                   |
    pour i:= 1 à n faire                           | O(n)
        si (non marq[i]) alors                     |
```

répétition des cycles

	1	2	3	4	5	6	7	8
marq	faux	faux	f	f	f	f	f	f

vrai

cycle 1

8

```
        j = i ; marq[i] := Vrai ;
ar)     tant que (T[j] ≠ i) faire
Σ_{i=1}^{n} ℓ_i = n        j := T[j] ;
            marq[j] := Vrai ;
        ft.q
    fsi  nc := nc + 1 ;
fpour
    retourner(nc) ;
fin
```

Complexité :

Les différents cycles constituent une partition de l'ensemble $\{1, 2, \dots n\}$ donc $\{1, 2, n\}$

$$= \sum_{i=1}^{m} \text{Card } C_i = n \quad (8)$$

$\Rightarrow$ L'algo a une complexité linéaire de $O(n)$

9