Evaluation du modûle (susceptible de varier en fonction des conditions sanitaires)

10 points : QCM (moodle)
40 points : partiel en présentiel
50 points : examens en présentiel

Etude d'un algo récursif :

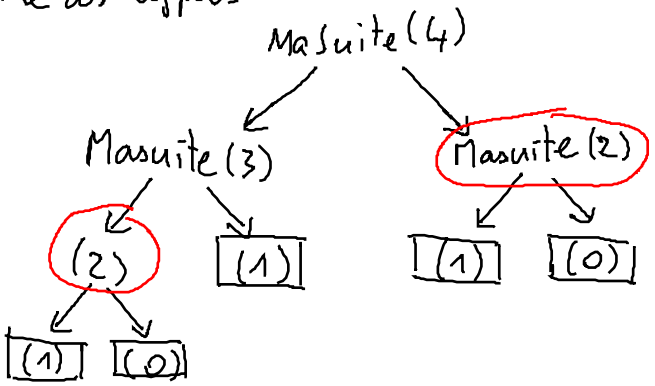
1) Déroulement de l'algo
2) Cas de base } termin
3) Induction } validité
4) Conclusion

Séance 4 de TD

Correction TD3 exercice 2

$$u_n = u_{n-1} + u_{n-2} + 7n$$

1) Arbre des appels



Problème de complexité : les mêmes calculs sont répétés dans plusieurs branches
Astuce : stocker les valeurs calculées dans un tableau

2) P(n) : "MaSuite(n) renvoie u_n"

Base : n = 2 : MaSuite(2) = MaSuite(1) + MaSuite(0) + 7×2
= 7 + 3 + 14
= 24

Base alternative : n = 0, MaSuite(0) = 3 = u_0
n = 1, MaSuite(1) = 7 = u_1

Induction : Supposons que P(n_0) est vraie $\forall n_0 \leq n-1$
Montrons que P(n)

MaSuite(n) renvoie MaSuite($\underbrace{n-1}_{\leq n-1}$) + MaSuite($\underbrace{n-2}_{\leq n-1}$) + 7n

qui, par hypothèse, vaut $u_{n-1} + u_{n-2} + 7n = u_n$

Donc MaSuite(n) termine et renvoie bien u_n

Feuille de TD 4 : Complexité

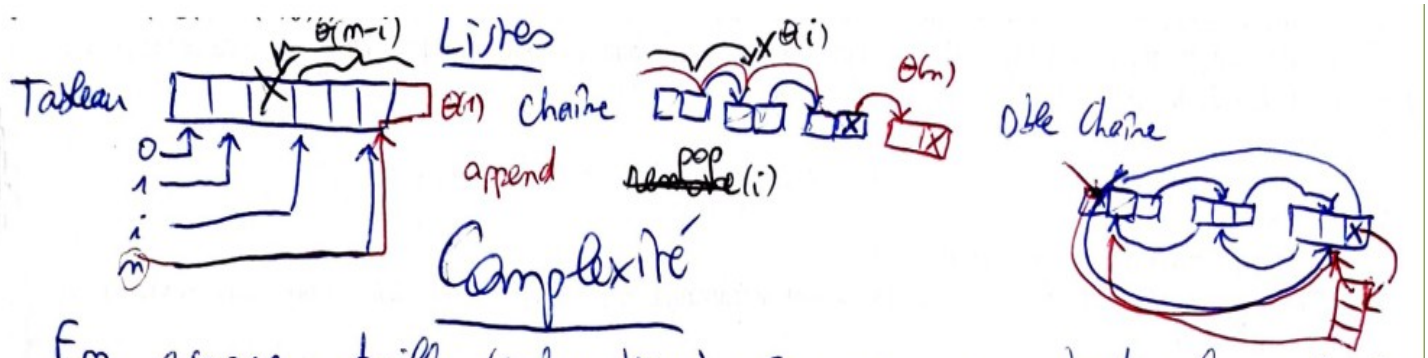
Rappel de cours

Complexité en espace : taille (informatique) nécessaire pour stocker le calcul
Complexité en temps : nombre d'opérations élémentaires pour réaliser le calcul
- indépendant de l'ordinateur donc à un facteur près
- on s'intéresse au pire des cas

} Notations de Landau

O Θ Ω

↑ pire cas ↑ meilleur cas



Complexité

En espace : taille (informatique) nécessaire pour stocker le calcul.

En temps : temps de calcul asymptotique en opérations élémentaires
- indépendant de l'ordinateur ou d'un facteur : notations de Landau
- pire des cas, car parfois le problème peut être favorable. O et pas Θ

exemples : logarithmique $\log n$, linéaire n, $n \log n$, quadratique n^2, cubique n^3, polynomial n^{130}, exponentiel 2^n, factoriel $n! \sim n^n$

⚠ En pratique, un algo linéaire n'est pas forcément plus rapide qu'un polynomial
ex : $O(n) : 10^6 \times n$ $O(\text{poly}(n)) : 10^{-6} n^{1.2}$

Calcul :

Itératif
complexité d'une boucle
somme sur toutes les boucles

Récursif
Complexité de base
Formule de récurrence
↓
suites récurrentes linéaires

⚠ Comme TD 2, la récursion peut faire faire plein de fois le même calcul et mener à complexité exponentielle.
Fib : $f_n = f_{n-1} + f_{n-2} = O(n)$
REC : F(n) : return F(n-1) + F(n-2)
F(n-2) + F(n-3)

Exo 1 (vert, magenta)

1) def somme_Tab(T)

```
res = 0                    Θ(1)
for x in T:                } n boucles
    res = res + x          Θ(1) } Θ(n)
return res
```

2) def RechercheMin(T, d, f):

```
imin = d                   Θ(1)
i = d+1                    Θ(1)
while i ≤ f                f-d boucles
    if tab[i] < tab[imin]:     Θ(f-d)
        imin = i
    i = i+1
return imin
```

(Ex : d=0, f=4, i=1,2,3,4, 4 valeurs = f-d)

1 comparaison, 1 affectation, 1 addition } opérations élémentaires. Θ(1)

3)

```
def TriParSelection(tab):
    i=0; n=len(tab)                Θ(1)
    while (i!= n):
        k = RechercheMin(tab, i, n-1)    Θ(n-1-i)
        if (k!=i):                       } Θ(1) car on
            z = tab[i]                   } utilise un tableau (et pas une liste chaînée)
            tab[i]=tab[k]                }
            tab[k]=z                     }
        i=i+1
```

Pour la boucle i, on note c_i le nombre d'opérations

$c_i = (n-1-i) \times A$

Opérations totales : $C = \sum_{i=0}^{n-1} c_i = A \sum_{i=0}^{n-1} n-1-i = A \sum_{j=0}^{n-1} j = A \frac{n(n-1)}{2}$

on pose $j = n-1-i$

pour i=0, j vaut n-1 ; 1 ; ... ; n-1

n-2 ; ... ; 0

$C \propto n(n-1)$

donc $C = \boxed{\Theta(n^2)}$

Exo 2 : (Bleu, Jaune)

1) Notons (u_n) le nombre de multiplications de fact(n)

$u_0 = 0$ car fact(0) retourne directement 1.

$u_n = 1 + u_{n-1} = 1 + 1 + u_{n-2} \pm \dots \pm n + u_0 = n$

D'où une complexité $\Theta(n)$

2) Notons (c_n) le nombre de comparaisons pour Recherche Min en notant $n = f - d$

$c_0 = 1$ comparaison car $f = d$

$c_n = c_{f-d} = 1 + c_{f-1-d} + 1 = 2 + c_{n-1}$

$= 2 + (2 + c_{n-2}) = \dots = 2n + c_0 = 2n + 1$

Complexité $\Theta(n)$ càd $\Theta(f - d)$

3)

```
def RechercheRec(elem, tab, n) :
    if n==0 :
        return -1
    if tab[n-1]==elem :
        return n-1
    return RechercheRec(elem, tab, n-1)
```

(meilleur cas)

Meilleur cas : l'élément est à la fin, 1 opération : $\Omega(1)$

Pire cas : on recherche dans tous les sous-tableaux s'il n'y a pas l'élément.

On note c_n le nb de comparaison pour Recherche(n).

$c_0 = 1$

$c_n \leqslant \underset{n==0}{1} + \underset{tab[n-1]==elem}{1} + c_{n-1} = 2 + c_{n-1}$

$\leqslant 2 + (2 + c_{n-2}) \leqslant \dots \leqslant 2 \times n + c_0$

$\leqslant 2n + 1$

D'où une complexité $\mathcal{O}(n)$

Exo 3 : Jaune, Magenta Exo 4 : Vert, Bleu → 11h55

Exo 4 : Recherche dans tableau trié

1) Complexité séquentielle :

meilleur $\Omega(1)$ pire $O(n)$

2) Déroulement :

```
 >>> RechercheDicho(40,T,0,6)
A l ppel T[ 0 , 6 ]= [1, 6, 12, 17, 34, 45, 65] et elem= 40
A l ppel T[ 4 , 6 ]= [34, 45, 65] et elem= 40
A l ppel T[ 4 , 4 ]= [34] et elem= 40
A l ppel T[ 5 , 4 ]= [] et elem= 40
En sortie res= -1
En sortie res= -1
En sortie res= -1
En sortie res= -1
```

```
>>> RechercheDicho(12,T,0,6)
A l ppel T[ 0 , 6 ]= [1, 6, 12, 17, 34, 45, 65] et elem= 12
A l ppel T[ 0 , 2 ]= [1, 6, 12] et elem= 12
A l ppel T[ 2 , 2 ]= [12] et elem= 12
En sortie res= 2
En sortie res= 2
En sortie res= 2
```

3) Terminaison et validité

$P(k)$: " RechercheDicho(elem, T, d, f) termine et retourne un indice $i \in [\![d, f]\!]$ tel que $T[i] = elem$ si possible, sinon -1, avec $k = f - d + 1$ "

Base : pour $k \leq 0$, on a $f + 1 \leq d$, le tableau est vide donc il ne contient pas elem, ça termine en retournant -1

Induction : Supposons que $\forall\, l < k$, $P(l)$ est vraie.
Montrons que $P(k)$ est vraie aussi.

Soient d et f tels que $k = f - d + 1$ et $0 \leq d \leq f \leq n-1$
On a bien $d \leq f$, donc on définit $i = \left\lfloor \frac{d+f}{2} \right\rfloor$
on a $d \leq i \leq f$.

Trois cas :
- $T[i] = elem$, alors on termine et renvoie i
- $T[i] > elem$, on appelle RechercheDicho(d, i)
  comme $\underbrace{i+1-d}_{l} < \underbrace{f-d+1}_{k}$

Par hypothèse de récurrence forte, P(l) est vraie, donc Recherchedicho termine et renvoie le bon indice

- T[i] < elem, on appelle Recherchedicho(i, f) et de même, ça termine et c'est valide.

4) Complexité :

meilleur cas : $\Omega(1)$

pire cas : notons C_n le nb de comparaisons pour trouver elem dans un tableau de taille n.

Base : $C_1 = 1$

Induction : $C_n \leq 1 + 1 + 1 + C_{\lfloor \frac{n}{2} \rfloor} = 3 + C_{\lfloor \frac{n}{2} \rfloor}$

$\leq 3 + \left(3 + C_{\lfloor \frac{n}{4} \rfloor}\right) \leq \dots \leq 3k + C_{\lfloor \frac{n}{2^k} \rfloor}$

On s'arrête quand $\lfloor \frac{n}{2^k} \rfloor = 1$ càd $k = \lfloor \log_2 n \rfloor$

$C_n = 1 + 3\lfloor \log_2 n \rfloor$

Donc la complexité est $O(\log n)$

Solution:
Cette question tombe tout le temps en examen. Inutile de dire ce qui se passe, juste donner les affichages ! !

```
A l ppel T[ 0 , 6 ]= [3, 1, 8, 5, 1, 4, 3]
A l ppel T[ 1 , 6 ]= [3, 8, 5, 1, 4, 3]
A l ppel T[ 2 , 6 ]= [8, 5, 3, 4, 3]
A l ppel T[ 3 , 6 ]= [5, 8, 4, 3]
A l ppel T[ 4 , 6 ]= [8, 4, 5]
A l ppel T[ 5 , 6 ]= [8, 5]
En sortie T[ 5 , 6 ]= [5, 8]
En sortie T[ 4 , 6 ]= [4, 5, 8]
En sortie T[ 3 , 6 ]= [3, 4, 5, 8]
En sortie T[ 2 , 6 ]= [3, 3, 4, 5, 8]
En sortie T[ 1 , 6 ]= [1, 3, 3, 4, 5, 8]
En sortie T[ 0 , 6 ]= [1, 1, 3, 3, 4, 5, 8]
```