Algorithmique
M2 Data Science

cours 2
divide and conquer
diviser pour régner
les algorithmes récursifs

Vincent Runge

Laboratoire de Mathématiques et Modélisation d'Évry (LaMME)
université evry val-d'essonne
université PARIS-SACLAY

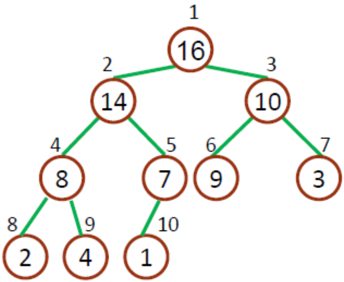
Section 1

L'exemple du tri par tas

Qu'est ce qu'un tas?

Le tas = un arbre binaire (équilibré et tassé à gauche). On conserve le vecteur MAIS les deux éléments en dessous d'un noeud en position j sont aux positions $2j$ et $2j+1$ (plus de parcours linéaire)

Propriété d'un tas "tamisé": $v[j] > v[2j]$ et $v[j] > v[2j+1]$



Vecteur $v = (16, 14, 10, 8, 7, 9, 3, 2, 4, 1)$

Tri par tas (gif du tri par tas)

```
1  heap_sort <- function(v)
2  {
3    n <- length(v)
4      for(i in (n%/%2):1)
5      {
6        v <- build_heap_recursive(v, i, n)
7      }
8      for(i in n:2)
9      {
10       temp <- v[i]
11       v[i] <- v[1]
12       v[1] <- temp
13       v <- build_heap_recursive(v, 1, i-1)
14     }
15   return(v)
16 }
```

```
1  build_heap_recursive <- function(heap, i, n)
2  {
3    l <- 2*i
4    if(l <= n)
5    {
6      if((l < n) && (heap[l] < heap[l+1])){l <- l + 1}
7                                        #choose the right son
8      if(heap[i] < heap[l]) #switch the node values
9      {
10       temp <- heap[i]
11       heap[i] <- heap[l]
12       heap[l] <- temp
13       heap <- build_heap_recursive(heap, l, n)
14     }
15   }
16   return(heap)
17 }
```

- gauche lignes 4,5,6,7 : création du tas (“heapify”)
- gauche ligne 8 : La plus grande valeur est maintenant à la racine!

1. isoler la racine $v[1]$ en l'échangeant avec la dernière valeur du vecteur $v[n]$ (ligne 10,11, 12)
2. reconstruire le tas (= tamiser = "heapify") (ligne 13)

Complexité : tri par insertion + rappels

- $T(n) = \Omega(n)$ (cas vecteur trié ordre croissant)
- $T(n) = O(n^2)$ (cas vecteur trié ordre décroissant)

Rappel : Comparaisons asymptotiques

- $u_n = O(v_n)$ s'il existe $M > 0$ tel que un $\frac{u_n}{v_n} \leq M$ à partir d'un certain rang
- $u_n = \Omega(v_n)$ s'il existe $m > 0$ tel que un $\frac{u_n}{v_n} \geq m$ à partir d'un certain rang
- $u_n = \Theta(v_n)$ s'il existe $m, M > 0$ tels que $m \leq \frac{u_n}{v_n} \leq M$ à partir d'un certain rang.

Ainsi :

- $T(n) = \Omega(n)$: $T(n) \geq m \times n$ pour une constant m et n assez grand
- $T(n) = O(n^2)$: $T(n) \leq M \times n^2$ pour une constant M et n assez grand

Complexité : tris pas tas

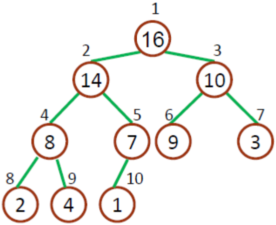
$$T(n) = \Theta(n \log(n)) \text{ (complexité quasi-linéaire)}$$

Soit :

$$m \times n \log(n) \leq T(n) \leq M \times n \log(n)$$

pour deux constantes $m > 0$ et $M > 0$ et n assez grand

hauteur d'un tas



- le noeud 5 est à la hauteur 1
- la racine est à la hauteur 3
- la hauteur maximale avec n noeuds est donnée par $\lfloor log2(n) \rfloor$

ici :

$$2^3 < 10 < 2^4\,, \quad 3 < log2(10) < 4\,, \quad \lfloor log2(n) \rfloor = 3$$

Au plus $\lceil \frac{n}{2^h} \rceil$ noeuds à la hauteur h

Temps création d'un tas à partir d'un vecteur

$$\text{TEMPS} = \sum_{h=0}^{\lfloor log2(n) \rfloor} (\text{nombre de noeuds hauteur } h) \times (\text{nb max d'échanges})$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \lceil \frac{n}{2^h} \rceil O(h) = O\Big(n \sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h}\Big)$$

Temps création d'un tas à partir d'un vecteur

$$\text{TEMPS} = \sum_{h=0}^{\lfloor log2(n) \rfloor} (\text{nombre de noeuds hauteur } h) \times (\text{nb max d'échanges})$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \lceil \frac{n}{2^h} \rceil O(h) = O\Big(n \sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h}\Big)$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} (h+1)\Big(\frac{1}{2}\Big)^h$$

Temps création d'un tas à partir d'un vecteur

$$\text{TEMPS} = \sum_{h=0}^{\lfloor log2(n) \rfloor} (\text{nombre de noeuds hauteur } h) \times (\text{nb max d'échanges})$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \lceil \frac{n}{2^h} \rceil O(h) = O\Big(n \sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h}\Big)$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} (h+1)\Big(\frac{1}{2}\Big)^h$$

$$\sum_{h=0}^{+\infty} (h+1)\Big(\frac{1}{2}\Big)^h \leq \Big(\sum_{h=0}^{+\infty} x^{h+1}\Big)'\Big|_{x=\frac{1}{2}} = \Big(\frac{x}{1-x}\Big)'\Big|_{x=\frac{1}{2}} = \Big(\frac{1}{(1-x)^2}\Big)\Big|_{x=\frac{1}{2}}$$

Temps création d'un tas à partir d'un vecteur

$$\text{TEMPS} = \sum_{h=0}^{\lfloor log2(n) \rfloor} (\text{nombre de noeuds hauteur } h) \times (\text{nb max d'échanges})$$

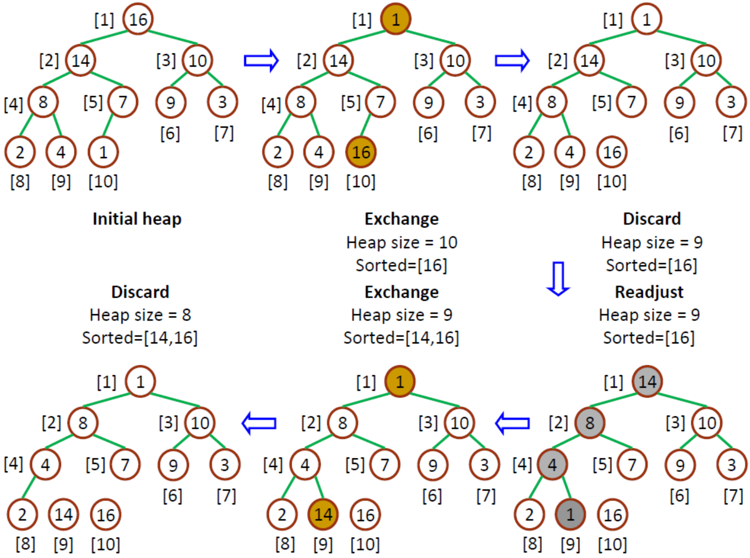
$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \lceil \frac{n}{2^h} \rceil O(h) = O\Big(n \sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h}\Big)$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} \frac{h}{2^h} \leq \sum_{h=0}^{+\infty} (h+1)\Big(\frac{1}{2}\Big)^h$$

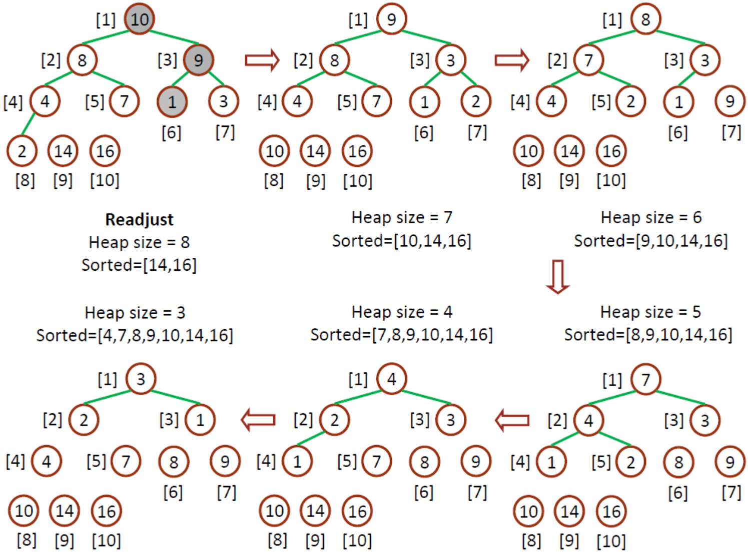
$$\sum_{h=0}^{+\infty} (h+1)\Big(\frac{1}{2}\Big)^h \leq \Big(\sum_{h=0}^{+\infty} x^{h+1}\Big)'\Big|_{x=\frac{1}{2}} = \Big(\frac{x}{1-x}\Big)'\Big|_{x=\frac{1}{2}} = \Big(\frac{1}{(1-x)^2}\Big)\Big|_{x=\frac{1}{2}}$$

$$\sum_{h=0}^{\lfloor log2(n) \rfloor} \lceil \frac{n}{2^h} \rceil O(h) \leq O\Big(n \frac{1}{(1-\frac{1}{2})^2}\Big) = O(4n)$$

tri : boucle i de n à 2 (1)



tri : boucle i de n à 2 (2)



Complexité

- Etape 1 : création du tas $T_1(n) = O(n)$
- Etape 2 : tri avec tas $T_2(n) = \Theta(n \times log(n))$

En effet, on traite $n - 1 = \Theta(n)$ positions dans le vecteur avec au plus $log(n)$ échanges dans le tas. En effet (dans le pire des cas):

$$T_2(n) \leq T_2(\frac{2n}{3}) + \Theta(1) = \Theta(log(n))$$

- $\Theta(1)$ est le temps de l'échange entre les noeuds (2 comparaisons)
- On utilise le master theorem pour trouver la solution $\Theta(log(n))$

Complexité en temps $T(n) = T_1(n) + T_2(n) = \Theta(n \times log(n))$

En fait, c'est $O(n \times log(n))$ et non pas Θ si le vecteur contient des valeurs identiques.

Section 2

Le *master theorem*

Diviser pour régner

Récurrence sur le temps d'exécution :

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Problème initial de taille n.

1. diviser : découpage en a problèmes de taille n/b de même nature
2. régner : découpage récursif jusqu'à un problème élémentaire (cas de base) de résolution facile
3. combiner les sous-problèmes pour reconstruire le problème initial

Remarque : temps pour diviser et recombiner : $f(n)$

Diviser pour régner (un exemple pour être précis)

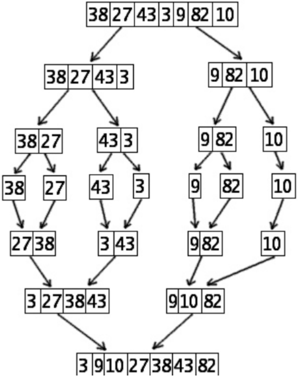
Cas du **tri fusion** avec $a = b = 2$ on devrait écrire :

$$T(n) = \begin{cases} \Theta(1), & \text{si } n = 1 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + \Theta(n), & \text{si } n > 1 \end{cases}$$

Qu'on réécrit

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

- On ignore la condition aux limites $\Theta(1)$ qui n'a pas de conséquence sur l'ordre de la complexité
- Le *master theorem* étant valable quel que soit la partie entière (sup ou inf) on se permet de l'enlever



Master theorem

Enoncé

Si on a la relation de récurrence suivante avec $a \geq 1$ et $b > 1$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

avec $\frac{n}{b}$ signifiant $\lfloor \frac{n}{b} \rfloor$ ou $\lceil \frac{n}{b} \rceil$ ALORS

1. Si $f(n) = O(n^{log_b a - \epsilon})$ avec $\epsilon > 0$ alors $T(n) = O(n^{log_b a})$
2. Si $f(n) = \Theta(n^{log_b a})$ alors $T(n) = \Theta(n^{log_b a} log(n))$
3. Si $f(n) = \Omega(n^{log_b a + \epsilon})$ avec $\epsilon > 0$ et $a \times f(\frac{n}{b}) \leq c \times f(n)$ avec $c < 1$ alors $T(n) = \Theta(f(n))$

Dans les cas 1 et 2, la relation de récurrence est assez importante par rapport au temps associé à $f(n)$ pour avoir une influence sur la complexité.

Enoncé avec $a = b = 2$

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n)$$

1) Si $f(n) = O(n^{1-\epsilon})$ avec $\epsilon > 0$ alors $T(n) = O(n)$
2) Si $f(n) = \Theta(n)$ alors $T(n) = \Theta(n \times log(n))$
3) Si $f(n) = \Omega(n^{1+\epsilon})$ avec $\epsilon > 0$ et $2 \times f(\frac{n}{2}) \leq c \times f(n)$ avec $c < 1$ alors $T(n) = \Theta(f(n))$

Enoncé (preuve du tri par tas) avec $a = 1$ et $b = 3/2$

Rappel

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- Si $f(n) = O(n^{log_b a - \epsilon})$ alors $T(n) = O(n^{log_b a})$
- Si $f(n) = \Theta(n^{log_b a})$ alors $T(n) = \Theta(n^{log_b a} log(n))$
- Si $f(n) = \Omega(n^{log_b a + \epsilon})$ alors $T(n) = \Theta(f(n))$

$$T(n) = T\left(\frac{n}{3/2}\right) + \Theta(1)$$

$log_b a = log_{3/2} 1 = 0$ donc $f(n) = \Theta(1) = \Theta(n^0) = \Theta(n^{log_b a})$

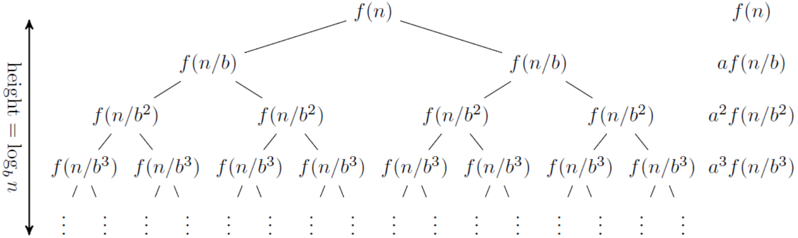
et ainsi $T(n) = \Theta(n^0 log(n)) = \Theta(log(n))$

Section 3

La preuve du théorème

arbre

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$



Un exemple plus original : l'algorithme de Strassen

Multiplication matricielle "rapide" $\mathbf{C} = \mathbf{AB}$ $\mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^p \times 2^p}$. Temps avec la méthode standard : $\Theta(n^3)$ avec $n = 2^p$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

avec $\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{p-1} \times 2^{p-1}}$

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

$$\begin{aligned} \mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

Pour aller plus loin (Akra-Bazzi)

Théorème d'Akra-Bazzi

$$T(x) = g(x) + \sum_{i=1}^{k} a_i T(b_i x + h_i(x)) \quad \text{pour } x \geq x_0,$$

Alors $T(x)$ admet l'estimation suivante de son comportement asymptotique

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\right)$$

Un exemple :

On considère $T(n)$ définie par

$$T(n) = n^2 + \frac{7}{4}T\left(\left\lfloor \frac{1}{2}n \right\rfloor\right) + T\left(\left\lceil \frac{3}{4}n \right\rceil\right) \quad \text{pour } n > 3$$

et $T(n) = 1$ pour $0 \leq n \leq 3$. On prend $p = 2$ de sorte que

$$\frac{7}{4}\left(\frac{1}{2}\right)^p + \left(\frac{3}{4}\right)^p = 1.$$

La formule s'évalue alors comme suit :

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\right) = \Theta\left(x^2\left(1 + \int_1^x \frac{u^2}{u^3}\,\mathrm{d}u\right)\right) = \Theta(x^2(1 + \ln x)) = \Theta(x^2 \log x)$$

Section 4

TD / TP

Appliquer le “master theorem”

1. $$T(n) = 2T\left(\frac{n}{2}\right) + n^4$$

2. $$T(n) = T\left(\frac{7n}{10}\right) + n$$

3. $$T(n) = 16T\left(\frac{n}{4}\right) + n^2$$

4. $$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$$

5. $$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

Des exemples de diviser pour régner

compléter le fichier avec 3 exemples d'algorithme récursif (donner un lien).

ATTENTION : au moins l'un des 3 exemples NE DOIT PAS être du type $\left(T(n) = 2T\left(\frac{n}{2}\right) + f(n)\right)$

Choix optimal d'un projet

- Vendredi prochain, je vous proposerai des sujets pour l'évaluation. La question se pose de **comment répartir ces sujets en fonction de vos préférences** ?
- Pour n étudiants et m projets, on demande aux étudiants de choisir 3 projets par ordre de préférence (1, 2 et 3). **Comment répartir au mieux les projets pour maximiser la satisfaction des étudiants?**

<u>Exercice :</u> vous avez 20 min pour trouver un algorithme sur internet !

remplir le tableur avec son choix

Ensuite, vous aurez jusqu'au vendredi 3 décembre (date du cours suivant) pour choisir votre préférence, le choix majoritaire l'emportera! (Si égalité, je ferai le choix final)

Le package exemple

tri par tas versus tri par insertion dans le package **M2algorithmique**

```
devtools::install_github("vrunge/M2algorithmique")
library(M2algorithmique)
```

Résoudre le “Maximum subarray problem” en se basant sur le package exemple **M2algorithmique**.

Page wikipédia

Objectifs :

- installer un git, créer un github
- créer un 1er projet simple avec ce problème

Section 5

Rcpp package (a few tips)

Rcpp package (code)

- fonctions R dans le dossier R
- fonctions C++ dans le dossier src

Pour le C++, habillage **obligatoire** :

1. // [[Rcpp::export]] Pour l'export en R
2. Possibilité d'utiliser des *include*

```
include <Rcpp.h> //to use the NumericVector object
using namespace Rcpp; //to use the NumericVector object
include<vector>
```

- Faire attention au fichier NAMESPACE

Rcpp package (information)

Les informations autour du code sont **capitales** dans un package

- Ecrire un fichier de DESCRIPTION complet
- Un README.md qui détaille l'installation et l'utilisation du package
- L'aide en .Rd des fonctions est dans le dossier man

Les fichiers .Rd sont générés automatiquement à partir du code devant les fonctions en utilisant le package roxygen2

Aide d'une fonction R

```r
#' Insertion sort algorithm
#'
#' @description Sorting by insertion
#' @param v an unsorted vector of numeric data
#' @return the sorted vector
```

Aide d'une fonction Rcpp

```cpp
//' Insertion sort algorithm using C++
//'
//' @param v an unsorted vector of numeric data
//' @return a sorted vector
//' @export
```

Faire attention aux fichiers **.Rbuildignore** et **.gitignore** pour une bonne compilation du package et un git bien géré.