Complexité des algorithmes

Serge Miguet

Algorithmes et structures de données

$\mathcal{P}$: un problème

$\mathcal{M}$: une méthode pour résoudre le problème $\mathcal{P}$

Algorithme : description de la méthode $\mathcal{M}$ dans un *langage algorithmique*

du nom du mathématicien perse *Al Khuwarizmi* (780 - 850)

Algorithmes et structures de données

$\mathcal{P}$: un problème

$\mathcal{M}$: une méthode pour résoudre le problème $\mathcal{P}$

Algorithme : description de la méthode $\mathcal{M}$ dans un *langage algorithmique*

du nom du mathématicien perse *Al Khuwarizmi* (780 - 850)

Structure de données : manière d'organiser et de stocker les données (supposée rendre efficace certaines opérations)

Structures algorithmiques

Structures de contrôle

- séquence
- embranchement (ou sélection)
- boucle (ou itération)

Supports pour les structures de données

- constantes, variables
- tableaux
- structures récursives (listes, arbres, graphes)

Complexité des algorithmes

On veut

- Evaluer l'efficacité de la méthode $\mathcal{M}$
- Comparer $\mathcal{M}$ avec une autre méthode $\mathcal{M}'$

indépendamment de l'environnement (machine, système, compilateur, . . .)

Complexité des algorithmes

Evaluation du nombre d'opérations élémentaires en fonction

- de la taille des données (par exemple le nombre d'éléments à trier)
- de la nature des données (provoquant par exemple la sortie d'une boucle)

Notations :

- n : taille des données,
- $T(n)$: nombre d'opérations élémentaires

Configurations caractéristiques

- meilleur cas,
- **pire des cas**,
- cas moyen.

Evaluation de $T(n)$ (séquence)

Somme des coûts.

$$\left.\begin{array}{ll} \text{Traitement1} & T_1(n) \\ \\ \text{Traitement2} & T_2(n) \end{array}\right\} \quad T(n) = T_1(n) + T_2(n)$$

Evaluation de $T(n)$ (embranchement)

Max des coûts.

si $<$ condition $>$ **alors**	$T_c(n)$
Traitement1	$T_1(n)$
sinon	
Traitement2	$T_2(n)$

$$T_c(n) + max(T_1(n), T_2(n))$$

Evaluation de $T(n)$ (boucle)

Somme des coûts des passages successifs

tant que $<$ condition $>$ **faire**	$C_i(n)$
Traitement	$T_i(n)$
fin faire	

$$\sum_{i=1}^{k}(C_i(n) + T_i(n)) + C_{k+1}(n)$$

$T_i(n)$: coût de la $i^{\text{ème}}$ itération

souvent défini par une équation récursive

Evaluation de $T(n)$ (fonctions récursives)

fonction FUNCTIONRECURSIVE (n)

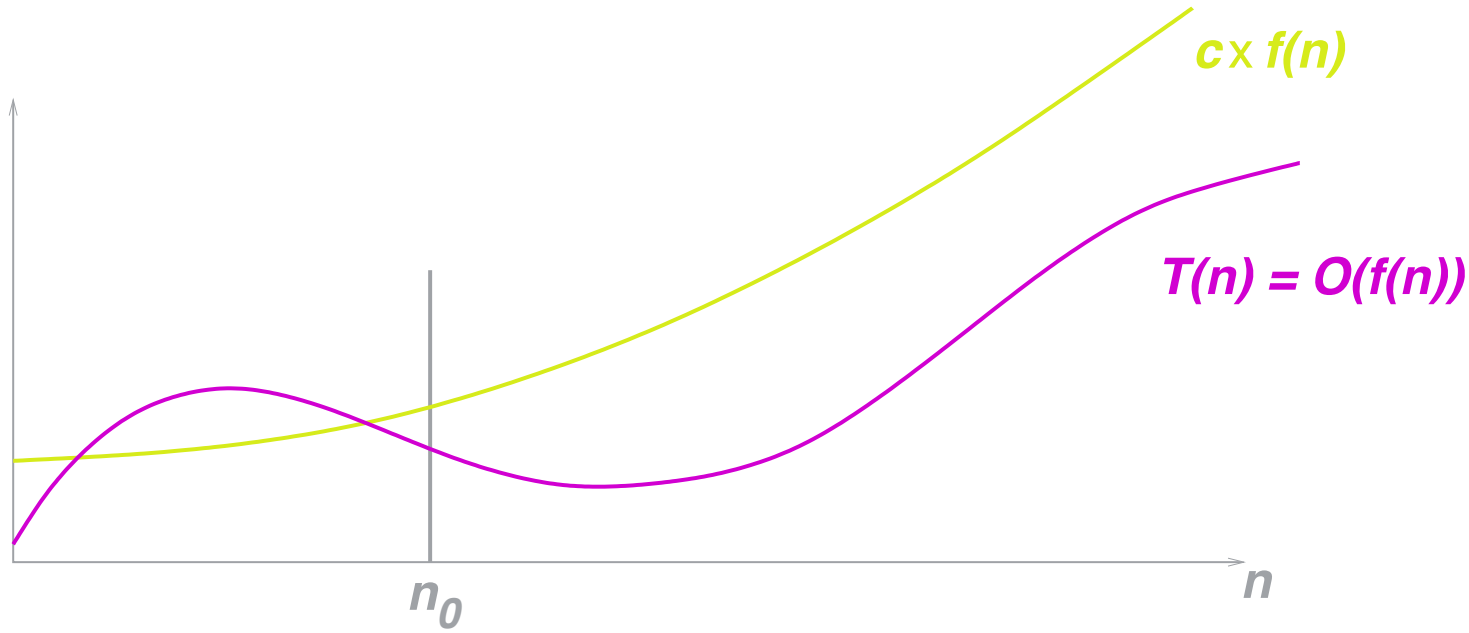
1 **si** $(n > 1)$ **alors**
2 FUNCTIONRECURSIVE$(n/2)$, coût $T(n/2)$
3 Traitement(n), coût $C(n)$
4 FUNCTIONRECURSIVE$(n/2)$, coût $T(n/2)$

Equation récursive

$$T(n) = 1 + 2 \times T(n/2) + C(n)$$

Exemple :
si $C(n) = 1$ alors $T(n) = K \times n$
si $C(n) = n$ alors $T(n) = K \times n \times \log n$

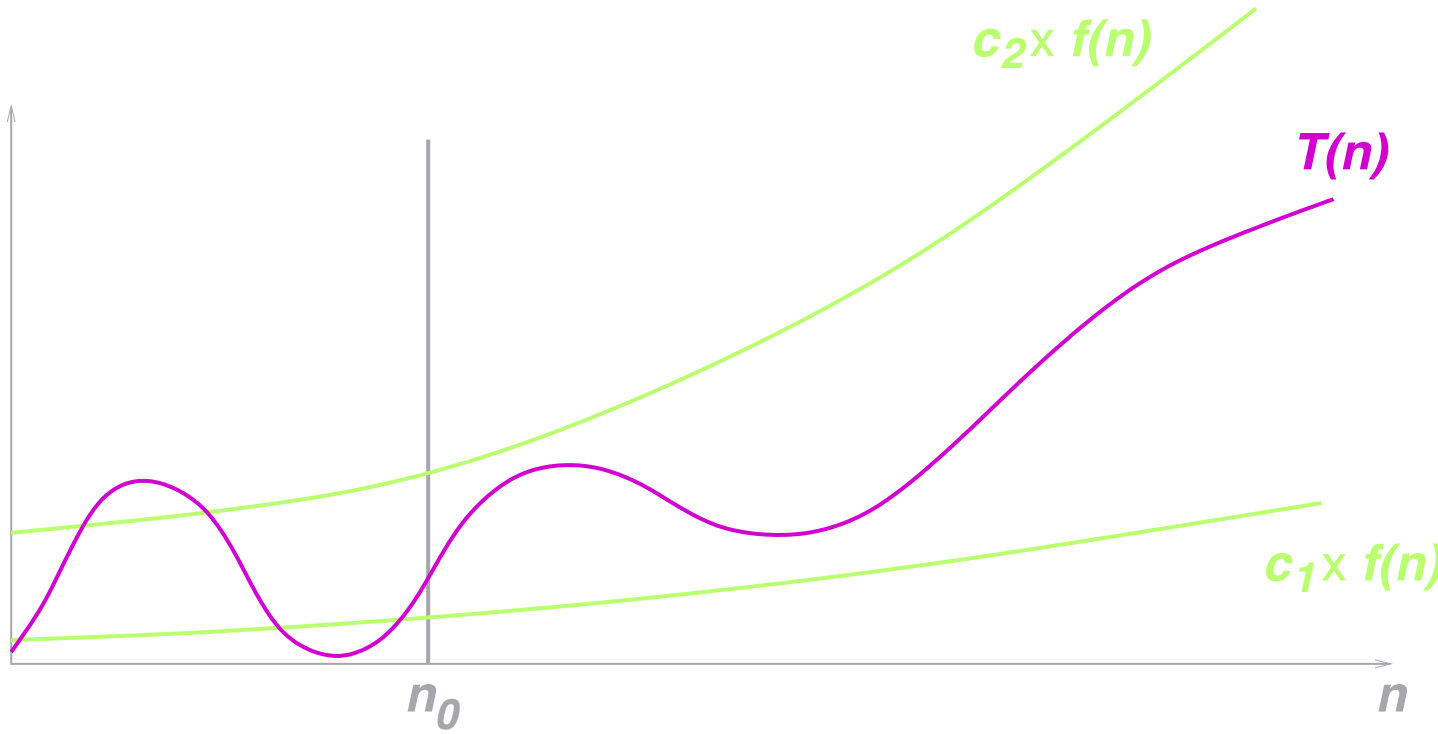
Notation de Landau $O(f(n))$



Caractérise le comportement asymptotique (i.e. quand $n \to \infty$).

$$T(n) = O(f(n)) \text{ si } \exists c\ \exists n_0 \text{ tels que } \forall n > n_0,\ T(n) \leq c \times f(n)$$

Notation $\Theta(f(n))$



$T(n) = \Theta(f(n))$ si $\exists c_1, c_2, n_0$ tels que

$$\forall n > n_0, \quad c_1 \times f(n) \leq T(n) \leq c_2 \times f(n)$$

Exemples

$$T(n) = n^3 + 2\,n^2 + 4\,n + 2 = O(n^3)$$

(si $n \geq 1$ alors $T(n) \leq 9 \times n^3$)

Exemples

$$T(n) = n^3 + 2\,n^2 + 4\,n + 2 = O(n^3)$$

(si $n \geq 1$ alors $T(n) \leq 9 \times n^3$)

$$T(n) = n \log n + 12\,n + 2 = O(n \log n)$$

Exemples

$$T(n) = n^3 + 2\,n^2 + 4\,n + 2 = O(n^3)$$

(si $n \geq 1$ alors $T(n) \leq 9 \times n^3$)

$$T(n) = n \log n + 12\,n + 2 = O(n \log n)$$

$$T(n) = 2\,n^{10} + n^7 + 12\,n^4 + \frac{2^n}{100} = O(2^n)$$

Les principales classes de complexité

$O(1)$	temps constant
$O(\log n)$	logarithmique
$O(n)$	linéaire
$O(n \times \log n)$	tris (par comparaisons)
$O(n^2)$	quadratique, polynomial
$O(2^n)$	exponentiel (problèmes très difficiles)

Exemple : permutation

fonction PERMUTATION (S, i, j)

1	$tmp := S[i]$,	coût c_1
2	$S[i] := S[j]$,	coût c_2
3	$S[j] := tmp$,	coût c_3
4	**renvoyer** S	coût c_4

Coût total

$$T(n) = c_1 + c_2 + c_3 + c_4 = O(1)$$

Exemple : recherche séquentielle

fonction RECHERCHE_SEQUENTIELLE$(x, S[1, \ldots, n])$

1 $i := 1,$ (c_1)

2 **tant que** $((i < n)$ **et** $(S[i] \neq x))$ **faire** (c_2)

3 $\quad i := i + 1,$ (c_3)

4 **renvoyer** $(S[i] = x)$ (c_4)

Pire des cas : n fois la boucle

$$T(n) = c_1 + c_2 + \sum_{i=1}^{n}(c_3 + c_2) + c_4 = O(n)$$

Exemple : tri à bulle

fonction TRI_A_BULLES $(S[1, \ldots, n])$

1	**pour** $i := n$ **à** 2 **faire**	
2	**pour** $j := 1$ **à** $i - 1$ **faire**	
3	**si** $(S[j] > S[j+1])$ **alors**	$i - 1$ fois
4	PERMUTER$(S, j, j+1)$,	C_{perm}

$$T(n) = C_{perm} \times \sum_{i=1}^{n-1} i = \frac{C_{perm} \times n \times (n-1)}{2} = O(n^2)$$

Equations récursives

Boucles itératives, fonctions récursives (approches de type diviser pour régner notamment)

Cas général

$$T(n) = a \times T(n/b) + f(n)$$

- méthode par substitution,
- méthode par développement itératif,
- méthode générale.

Méthode par substitution

Principe : on vérifie une **intuition**

$$T(n) = a \times T(n/b) + f(n) \text{ et } T(1) = c$$

Hypothèse

$$T(n) = g(n) \ \ (\text{intuition})$$

Conclusion

$$a \times g(n/b) + f(n) = g(n) \text{ et } g(1) = c$$

à démontrer en fixant les constantes

Méthode par substitution [recherche dichotomique]

fonction RECHERCHE_DICHOTOMIQUE $(x, S[1, \ldots, n])$

1	$g := 0$, $d := n + 1$,	$g < position \leq d$
2	**tant que** $(g < d - 1)$ **faire**	$termine\ quand\ g = d - 1$
3	**si** $(x > S[(g + d)/2])$ **alors**	$g < (g + d)/2 < d$
4	$g := (g + d)/2$,	$et\ donc\ g < position \leq d$
5	**sinon**	
6	$d := (g + d)/2$,	$et\ donc\ g < position \leq d$
7	**renvoyer** d,	$g < position \leq d\ et\ g = d - 1$

Nombre d'itérations $T(n) = 1 + T(n/2)$

Méthode par substitution [recherche dichotomique]

$$T(n) = 1 + T(n/2) \text{ et } T(1) = 1^\star$$

Intuition $T(n) = O(\log_2 n)$

Hypothèse $T(n) = a \times \log_2 n + c$

donc $T(n/2) = a \times \log_2 n - a + c$

en substituant on a $T(n) = 1 + a \times \log_2 n - a + c = a \times \log_2 n + c$

donc $1 - a = 0$ et $a = 1$, et puisque $T(1) = 1$ donc $c = 1$

Conclusion $T(n) = \log_2 n + 1 = O(\log_2 n)$

* s'il y a un élément on fera une itération

Méthode itérative : rappel sommations

$$\sum_{i=1}^{n-1} i = \frac{n \times (n-1)}{2} = O(n^2)$$

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$$

en particulier quand x vaut 2

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Méthode itérative [tri par fusion]

fonction TRI_PAR_FUSION (S)

1 **si** $(|S| \leq 1)$ **alors**
2 **renvoyer** S
3 *décomposer* S *en* S_1 *et* S_2, (n)
3 $S_1 :=$ TRI_PAR_FUSION(S_1), $(T(\lceil n/2 \rceil))$
4 $S_2 :=$ TRI_PAR_FUSION(S_2), $(T(\lfloor n/2 \rfloor))$
5 $S :=$ FUSIONNER(S_1, S_2), (n)
6 **renvoyer** S

$$T(n) = 1 + n + 2 \times T(n/2) + n \quad \text{et} \quad T(1) = 1$$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

$$T(n) = \sum_{i=0}^{\log\, n-1}(2\,n + 2^i) + 2^{\log\, n} \times T(1)$$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

$$T(n) = \sum_{i=0}^{\log\, n-1}(2\,n + 2^i) + 2^{\log\, n} \times T(1)$$

$$T(n) = 2n\,\log\, n + \sum_{i=0}^{\log\, n-1} 2^i + n$$

Méthode itérative [tri par fusion]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

$$T(n) = \sum_{i=0}^{\log\, n-1}(2\,n + 2^i) + 2^{\log\, n} \times T(1)$$

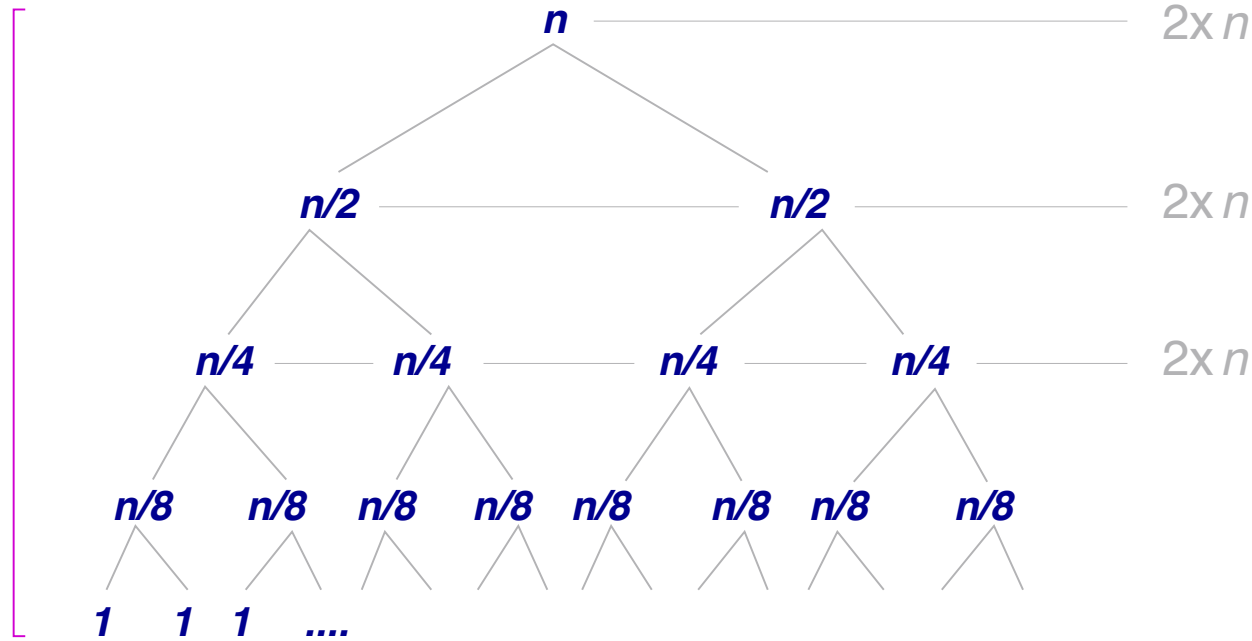
$$T(n) = 2n\,\log\, n + \sum_{i=0}^{\log\, n-1} 2^i + n$$

et comme $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$, on a $\sum_{i=0}^{\log\, n-1} 2^i = 2^{\log\, n} - 1$

$$T(n) = 2n\,\log\, n + 2n - 1 = O(n\,\log\, n)$$

Méthode itérative [tri par fusion]

hauteur $\lceil \log_2 n \rceil$



$O(k)$ traitements pour décomposer et fusionner une suite de longueur k

$$T(n) = 2 \times n \times \lceil \log_2 n \rceil$$

Méthode générale [Equations récursives]

$$T(n) = a \times T(n/b) + f(n)$$

avec $a \geq 1$, $b > 1$ et $f(n)$ est positive asymptotiquement.

- si $\exists \epsilon > 0,\ f(n) = O(n^{\log_b\ a-\epsilon})$ alors $T(n) = \Theta(n^{\log_b\ a})$,
- si $f(n) = \Theta(n^{\log_b\ a})$ alors $T(n) = \Theta(n^{\log_b\ a} \times \lg\ n)$,
- si $\exists \epsilon > 0,\ f(n) = \Omega(n^{\log_b\ a+\epsilon})$ et
  si $\exists c < 1,\ \exists n_0,\ \forall n > n_0,\ a \times f(n/b) \leq c \times f(n)$ alors $T(n) = \Theta(f(n))$

Méthode générale [Equations récursives]

$$T(n) = a \times T(n/b) + f(n)$$

avec $a \geq 1$, $b > 1$ et $f(n)$ est positive asymptotiquement.

- si $\exists \epsilon > 0,\ f(n) = O(n^{\log_b\ a-\epsilon})$ alors $T(n) = \Theta(n^{\log_b\ a})$,
- si $f(n) = \Theta(n^{\log_b\ a})$ alors $T(n) = \Theta(n^{\log_b\ a} \times \lg\ n)$,
- si $\exists \epsilon > 0,\ f(n) = \Omega(n^{\log_b\ a+\epsilon})$ et
  si $\exists c < 1,\ \exists n_0,\ \forall n > n_0,\ a \times f(n/b) \leq c \times f(n)$ alors $T(n) = \Theta(f(n))$

Le tri par fusion ($T(n) = 2n + 1 + 2\ T(n/2)$) est dans le cas 2 :

$a = b = 2$ et $f(n) = 2n + 1 = \Theta(n)$ et $T(n) = \Theta(n\ \log\ n)$

Méthode par substitution [tri par fusion]

$$T(n) = 2n + 1 + 2\,T(n/2) \text{ et } T(1) = 1$$

Hypothèse : $T(n) = O(n \log n) = an \log n + bn + c$

et donc $T(n/2) = a/2\; n \log n + (b - a)n/2 + c$

$$\begin{aligned} T(n) &= 2n + 1 + 2T(n/2) = 2n + 1 + an \log n + (b - a)n + 2c \\ &= an \log n + (b - a + 2)n + 2c + 1 \end{aligned}$$

(1) $b = b - a + 2$ et $a = 2$
(2) $c = 2c + 1$ et $c = -1$
(3) $T(1) = b + c = 1$ et donc $b = 2$

et finalement $T(n) = 2n \log n + 2n - 1 = O(n \log n)$

Temps de calcul

$Taille$	$\log_2 n$	n	$n \log_2 n$	n^2	2^n
10	$0.003\,ms$	$0.01\,ms$	$0.03\,ms$	$0.1\,ms$	$1\,ms$
100	$0.006\,ms$	$0.1\,ms$	$0.6\,ms$	$10\,ms$	$10^{14}\,siecles$
1000	$0.01\,ms$	$1\,ms$	$10\,ms$	$1\,s$	
10^4	$0.013\,ms$	$10\,ms$	$0.1\,s$	$100\,s$	
10^5	$0.016\,ms$	$100\,ms$	$1.6\,s$	$3\,heures$	
10^6	$0.02\,ms$	$1\,s$	$20\,s$	$10\,jours$	

pour un processeur qui effectue 10^6 traitements par seconde

Temps de calcul [simulation]

nTs	2^n	n^2	$n \log_2 n$	n	$\log_2 n$
10^6	20	1000	63000	10^6	10^{300000}
10^7	23	3162	600000	10^7	$10^{3000000}$
10^9	30	31000	4.10^7	10^9	
10^{12}	40	10^6	3.10^{10}		

$nTs =$ nombre d'instructions par seconde