- *Introduction to Algorithms* T.Cormen, C. Leiserson, R. Rivest, C. Stein http://ressources.unisciel.fr/algoprog/s00aaroot/aa00module1/res/%5BCormen-AL2011%5DIntroduction_To_Algorithms-A3.pdf
- *The Algorithm Design Manual* Steven Skiena https://www.algorist.com/

Contenu

- Rappels : Complexités des algorithmes
- Complexités des algorithmes récursifs
- TAS
- ABR équilibré (AVL)
- Algorithmes des graphes
- Programmation Dynamique

Complexité des Algorithmes

On s'intéresse à deux complexités

- en espace mémoire nécessaire pour l'exécution de l'algorithme ;
- en temps d'exécution de l'algorithme.

Pour le calcul du temps d'exécution :

1. Déterminer le paramètre représentant la taille du problème (le nombre d'éléments dans un tableau, le nombre de sommets et/ou le nombres d'arêtes dans un graphe)
2. Déterminer l'instruction qui prend relativement plus de temps d'exécution
   (la comparaison, l'échange)
3. Calculer le nombre d'instructions choisis pour **une taille fixée** du problème
   (notée souvent par n ou N).

Dans le cas où les complexités dépendent de la disposition des données, on définit

- la complexité dans **le pire des cas**
- la complexité dans **le meilleur des cas**

On peut également définir **la complexité moyenne**
si on a un modèle probabiliste pour la disposition des données.

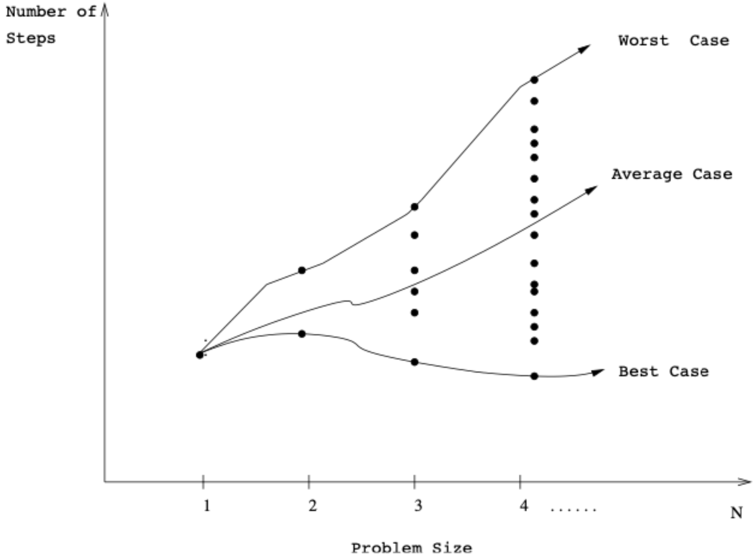


Figure – Figure prise du livre The Algorithm Design Manual

```
TriInsert(A)
  //Les indices du tableau A sont de 1 à A.taille = n
  for j=2 to n do
    key = A[j]
    // insérer A[j] dans le sous-tableau trié A[1]···A[j−1]
    i=j-1
    while i > 0 AND A[i] > key do
      A[i+1] = A[i]
      i=i-1
    end while
    A[i+1]=key
  end for
```

Analyse de complexité Tri par Insertion

1. $n =$ le nombre d'éléments du tableau.
2. On compte le nombre de comparaisons $(A[i] > key)$
3. - Dans le meilleur des cas, le tableau est déjà trié $(A[i] \leq key)$ et on n'entame pas la boucle *while* et on fait 1 seule comparaison pour chaque itération de j. On remplace la boucle j par une somme :

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

   - Dans le pire des cas, le tableau est trié dans le sens décroissant. On exécute toutes les itérations possibles de *while* pour chaque itération de j. On aura donc deux boucles imbriquées :

$$\sum_{j=2}^{n}(\sum_{i=j-1}^{1} 1) = \sum_{j=2}^{n}(\sum_{i=1}^{j-1} 1) = \sum_{j=2}^{n}(j-1) = \sum_{j=2}^{n} j - \sum_{j=2}^{n} 1$$

$$= \frac{n(n+1)}{2} - 1 - (n-1)$$

$$= n^2/2 + n/2 - n = n^2/2 - n/2$$

Tri par Sélection

TriSelect(A)

```
//Les indices du tableau A sont de 1 à A.taille = n
for  j=1 to n-1 do
   min=j
   //trouver min du sous-tableau j ··· n
   for i=j+1 to n do
      if A[i] < A[min] then
         min=i
      end if
   end for
   A[j] ⇔ A[min]
end for
```

Question de TD

Grandeurs Asymptotiques

- On s'intéresse à la complexité des algorithmes pour de grandes tailles de données ($\lim_{n\to\infty}$) (ordres de grandeurs des complexités).
- Soient $f(n)$ et $g(n)$ des fonctions croissantes dans R^+ (représentant des complexités des algorithmes).

Definition (notation O)

On dit que $f(n)$ est dominée par $g(n)$ ou $g(n)$ fournit un encadrement supérieur pour $f(n)$ (noté par $f(n) = O(g(n))$), s'il existe les constants positives c_1 et n_0 tels que

$$0 \leq f(n) \leq c_1 g(n), \ \ \forall n \geq n_0$$

$f(n) = O(g(n))$ veut dire $g(n)$ **croît plus vite que** $f(n)$

Exemple : $2n^2 + 56 = O(n^3)$

Definition (Ω notation)

On dit que $g(n)$ fournit un encadrement inférieur pour $f(n)$ (noté par $f(n) = \Omega(g(n))$), s'il existe les constants positives c_2 et n_0 tels que

$$0 \leq c_2 g(n) \leq f(n), \quad \forall n \geq n_0$$

$f(n) = \Omega(g(n))$ veut dire $g(n)$ **croît moins vite que** $f(n)$
Egalement, $g(n) = O(f(n)))$
Exemple : $n^3 = \Omega(n^2)$

Definition (Θ notation)

On dit que $f(n)$ et $g(n)$ ont le même ordre de grandeur (noté par $f(n) = \Theta(g(n))$), s'il existe les constants positives c_1, c_2 et n_0 tels que

$$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n), \quad \forall n \geq n_0$$

Exemple : $12n^3 + 5n^2 + 9 = \Theta(n^3)$ $f(n) = n^3 + n^2 = \Theta(n^3))$.

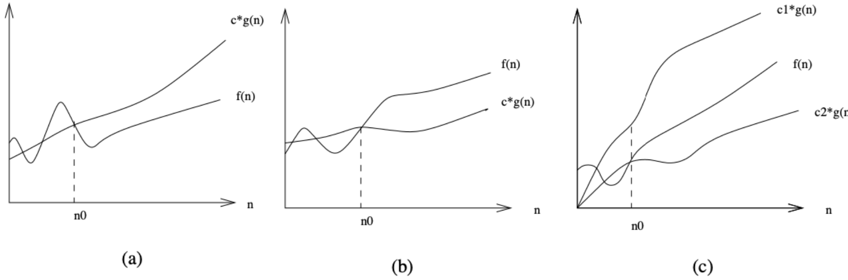


Figure – Figure prise du livre The Algorithm Design Manual

a) $f(n) = O(g(n))$
b) $f(n) = \Omega(g(n))$
c) $f(n) = \Theta(g(n))$

Complexité Asymptotique

- Θ est une relation d'équivalence.
  - $f(n) = \Theta(f(n))$ *Reflexive*
  - Si $f(n) = \Theta(g(n)) \rightarrow g(n) = \Theta(f(n))$*Symétrique*
  - Si $f(n) = \Theta(g(n))$ et $g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n))$*Transitive*
- $f(n) = \Theta(g(n)) \rightarrow g(n) = \Theta(f(n))$.
- Si $f(n) = \Omega(g(n))$ et $f(n) = 0(g(n)) \rightarrow f(n) = \Theta(g(n))$

Exemple : La complexité dans le pire des cas du *Tri par Insertion* est $\Theta(n^2)$. La complexité dans le meilleur des cas du *Tri par Insertion* est $\Theta(n)$.

Dans le cas des limites sont définies, **les conditions suffisantes :**

- *g(n) croît plus vite que f(n)*
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \to f(n) = O(g(n))$
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \to g(n) = \Omega(f(n))$
- *g(n) croît moins vite que f(n)*
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \to g(n) = O(f(n))$
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \to f(n) = \Omega(g(n))$
- *g(n) et f(n) ont même ordre de grandeur*
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = constant > 0 \to g(n) = \Theta(f(n))$
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = constant > 0 \to f(n) = \Theta(g(n))$

Classes de complexité

- complexité constante, $\Theta(c)$
- complexité logarithmique, $\Theta(logn)$ (recherche dichotomique)
- complexité linéaire, $\Theta(n)$ (recherche séquentielle)
- $\Theta(nlogn)$ (meilleurs algorithmes de tri)
- $\Theta(n^2)$ (2 boucles imbriquées)
- $\Theta(n^3)$ 3 boucles imbriquées (multiplication des matrices) ....
- $\Theta(2^n)$,$\Theta(n!)$, ... non-polynomiaux

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Figure – Figure prise du livre The Algorithm Design Manual-1000 instructions par sec.

Les égalités utiles

- $\sum_{i=1}^{n} i = \frac{(n+1)n}{2}$
- $\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a}$, $a \neq 1$
- $a^{log_a x} = x$
- $log_a a^x = x$
- $log_a x = \frac{log_b x}{log_b a}$
- $log_a x^n = n log_a x$
- $a^{log_b e} = e^{log_b a}$
- $a^{-1} = \frac{1}{a}$
- $(a^m)^n = a^{mn}$ $\qquad a^m a^n = a^{m+n}$
- "Exponentiel croît plus vite qu'un polynôme"
  $\lim_{\to\infty} \frac{n^b}{a^n} = 0$ $\qquad n^b = O(a^n)$, $a^n = \Omega(n^b)$
- "Logarithme croît moins vite que linéaire"
  $\lim_{\to\infty} \frac{ln(n)}{n} = 0$ $\qquad ln(n) = O(n)$, $n = \Omega(ln(n))$

Diviser pour Régner (Gagner)

- *Diviser :* on divise le problème en sous-problèmes de tailles plus petites
- *Régner :* on résout récursivement ces problèmes. On donne le résultat lorsque la taille est suffisamment petite
- *Combiner :* on combine les solutions des sous-problèmes pour obtenir la solution du problème principal.

Exemple : Tri par Fusion, La recherche Dichotomique

TriFusion(A,p,d)

```
  //Les indices du tableau A sont de 1 à A.taille = n
  //p :premier indice, d : dernier indice
  if  p < d then
    m=(p+d)/2 // DIVISER
    TriFusion(A,p,m) // REGNER
    TriFusion(A,m+1,d) // REGNER
    Fusion(A,p,m,d) // COMBINER
  end if
```

1er appel : TriFusion(A,1,n)

$$A = [5, 2, 4, 7, 1, 3, 2, 6]$$

- Appel 1 : TriFusion(A,1,8) $m_l = 4$
- Appel 2 : TriFusion(A,1,4) $m_l = 2$
- Appel 3 : TriFusion(A,1,2) $m_l = 1$
- Appel 4 : TriFusion(A,1,1) Terminaison Appel 4
  Continuer avec les paramètres Appel 3 :
- Appel 5 : TriFusion(A,2,2) Terminaison Appel 5
- Finir Appel 3 : Fusion(A,1,1,2) Fusion([2], [5])→[2,5]
  Continuer avec les paramètres Appel 2 :
- Appel 6 : TriFusion(A,3,4) $m_l = 3$
- Appel 7 : TriFusion(A,3,3) Terminaison Appel 7
  Continuer avec les paramètres Appel 6 :
- Appel 8 : TriFusion(A,4,4) Terminaison Appel 8
- Finir Appel 6 Fusion(A,3,3,4) Fusion([4], [7]) → [4,7]
  Finir Appel 2 Fusion(A,1,2,4) Fusion([2,5], [4,7]) → [2,4,5,7]
  Continuer avec les paramètres Appel 1 :

Continuer avec les paramètres Appel 1 :

- Appel 9 : TriFusion(A,5,8) $m_l = 6$
- Appel 10 : TriFusion(A,5,6) $m_l = 5$
- Appel 11 : TriFusion(A,5,5) Terminaison Appel 11
  Continuer avec les paramètres Appel 10 :
- Appel 12 : TriFusion(A,6,6) Terminaison Appel 12
- Finir Appel 10 : Fusion(A,5,5,6) Fusion([1], [3]) $\rightarrow$ [1,3]
  Continuer avec les paramètres Appel 9 :
- Appel 13 : TriFusion(A,7,8) $m_l = 7$
- Appel 14 : TriFusion(A,7,7) Terminaison Appel 14
  Continuer avec les paramètres Appel 13 :
- Appel 15 : TriFusion(A,8,8) Terminaison Appel 15
- Finir Appel 13 Fusion(A,7,7,8) Fusion([2], [6]) $\rightarrow$ [2,6]
  Finir Appel 9 Fusion(A,5,6,8) Fusion([1,3], [2,6]) $\rightarrow$ [1,2,3,6]
  Finir Appel 1 Fusion(A,1,4,8)
  Fusion([2,4,5,7], [1,2,3,6]) $\rightarrow$ [1,2,2,3,4,6,7]

Fusion(A,p,m,d)

$n_1 = m - p + 1$
$n_2 = d - m$
//Les tableaux auxiliaires $L[1, \cdots n_1 + 1]$ et $R[1, \cdots n_2 + 1]$
//la taille du tableau L :n_1
//la taille du tableau R :n_2
for $i = 1$ to n_1 **do**
    $L[i] = A[p + i - 1]$
end for
for $j = 1$ to n_2 **do**
    $R[j] = A[m + j]$
end for
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
//on met une "barrière" comme dernier élément
$i = 1$
$j = 1$
for $k = p$ to d **do**
    if $L[i] \leq R[j]$ **then**
        $A[k] = L[i]$
        $i = i + 1$
    else
        $A[k] = R[j]$
        $j = j + 1$
    end if
end for

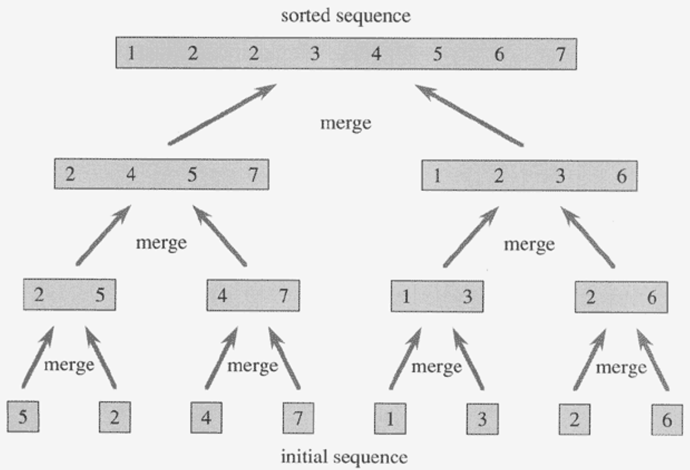


Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of sorted sequences being merged increase as the algorithm progresses from bottom to top.

Complexité Fusion

- La taille du tableau est $n = d - p + 1$
- On utilise les tableaux auxiliares L, R.
  Il faut $\Theta(n)$ de place mémoire supplémentaire.
- Au début de chaque itération de k, le sous-tableau $A[p, \cdots k-1]$ contient les $k-p$ plus petits éléments de $L[1, \cdots n_1+1]$, $R[1, \cdots n_2+1]$ en ordre croissant.

  $L[i]$: le plus petit élément de L qui n'a pas été copié dans A
  $R[j]$: le plus petit élément de R qui n'a pas été copié dans A
- On fait n itérations :
  Complexité de l'algorithme Fusion : $\Theta(n)$.

Complexité Tri Fusion

Soit $T(n)$ la complexité de l'algorithme TriFusion pour un tableau de taille n

Complexité Tri par Fusion (dans le meilleur et le pire des cas) :

$$T(n) = 2T(n/2) + c.n, \quad T(1) = 0$$

$$T(n) = \Theta(nlogn)$$

$c > 0$ est une constante.

Arbre de récursion Tri par Fusion

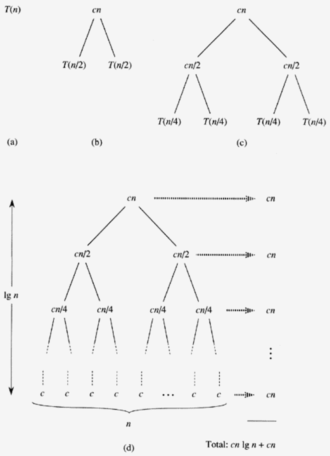


Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of cn. The total cost, therefore, is $cn \lg n + cn$, which is $\Theta(n \lg n)$.

"Master Théorème" / Théoreme de complexité des algorithmes récursifs

$$T(n) = \begin{cases} aT(n/b) + c.n^e & n > 1 \\ 1 & n = 1 \end{cases}$$

où $a \geq 1,\ b > 1,\ e(xposant) \geq 0,\ c(onstante) > 0,\ n = b^k$

- a le nombre des sous-problèmes
- b on divise par b la taille du problème
- $c.n^e$ le coût pour fusionner les solutions des sous-problèmes

1. si $\frac{a}{b^e} < 1, \quad T(n) = \Theta(n^e)$
2. si $\frac{a}{b^e} = 1, \quad T(n) = \Theta(n^e logn)$
3. si $\frac{a}{b^e} > 1, \quad T(n) = \Theta(n^{log_b a})$

$\frac{a}{b^e} < 1 \rightarrow log_b a < e,\ \frac{a}{b^e} = 1 \rightarrow log_b a = e,\ \frac{a}{b^e} > 1 \rightarrow log_b a > e$

Preuve

Dans l'arbre de récursion

- au niveau i, il y a a^i sous-problèmes de taille $\frac{n}{b^i}$
- au niveau i, le coût $a^i \times c(\frac{n}{b^i})^e$
- au niveau 0 (le premier niveau), la taille du problème $n/b^0 = n$
  au niveau k (le dernier niveau) , la taille du problème est
  $n/b^k = 1$.
  $k = log_b n$
- au niveau k (dernier niveau), il y a $a^{log_b n} = n^{log_b a}$ sous problèmes
- Coût total :

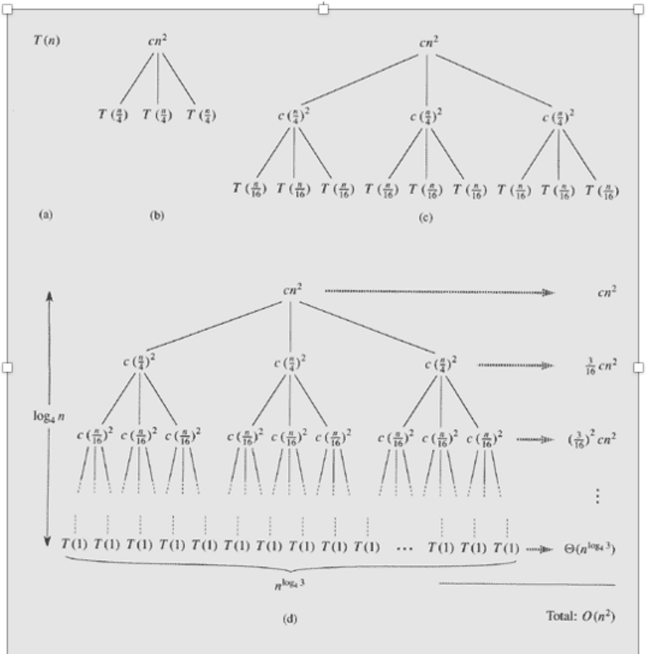
$$\sum_{i=0}^{log_b n} a^i \times c(\frac{n}{b^i})^e = c.n^e \sum_{i=0}^{log_b n} (\frac{a}{b^e})^i$$

2. Si $\frac{a}{b^e} = 1$, alors $c.n^e \sum_{i=0}^{log_b n} (\frac{a}{b^e})^i = c.n^e \sum_{i=0}^{log_b n} 1 = c.n^e(log_b n + 1) = \Theta(n^e logn)$

1. Si $\frac{a}{b^e} < 1$, alors $(\frac{a}{b^e})^0 > (\frac{a}{b^e})^1 > \cdots (\frac{a}{b^e})^i > \cdots > (\frac{a}{b^e})^{log_b n}$
On prend $(\frac{a}{b^e})^0 = 1$ pour la somme car c'est e plus grand terme de la somme :
$c.n^e \sum_{i=0}^{log_b n} (\frac{a}{b^e})^i = n^e.1 = \theta(n^e)$

3. Si $\frac{a}{b^e} > 1$, alors $(\frac{a}{b^e})^0 < (\frac{a}{b^e})^1 \cdots < (\frac{a}{b^e})^i < \cdots < (\frac{a}{b^e})^{log_b n}$
On prend $(\frac{a}{b^e})^{log_b n}$ pour la somme car c'est e plus grand terme de la somme :
$c.n^e \frac{a^{log_b n}}{(b^e)^{log_b n}} = c.n^e \frac{a^{log_b n}}{(b^{log_b n})^e} = c.n^e \frac{a^{log_b n}}{n^e} = c.a^{log_b n} = c.n^{log_b a}$
$= \theta(n^{log_b a})$.

Arbre de récursion- $T(n) = 3T(n/4) + cn2$



Solution par substitution

Soit $n = 4^k \quad T(1) = T(4^0) = 1$

$$
\begin{array}{lllll}
 & T(n) & = 3T(n/4)+ & & cn^2 \\
\hline
 & T(4^k) & = 3T(4^{k-1}) & + & c.4^{2k} \\
3^1 & T(4^{k-1}) & = 3^2T(4^{k-2}) & +3 & c.4^{2(k-1)} \\
3^2 & T(4^{k-2}) & = 3^3T(4^{k-3}) & +3^2 & c.4^{2(k-2)} \\
 & \dots & = \vdots & +\vdots & \vdots \\
3^{k-2} & T(4^2) & = 3^{k-1}T(4^1) & +3^{k-2} & c.4^{2.2} \\
3^{k-1} & T(4^1) & = 3^kT(4^0) & +3^{k-1} & c.4^{2.1} \\
\hline
 & T(4^k) & = 3^k & +c.3^k\sum_{i=1}^{k}(\frac{4^2}{3})^i & \\
 & T(4^k) & = 3^k & +c.3^{k-1}.4^2\sum_{i=0}^{k-1}(\frac{4^2}{3})^i & \\
 & T(4^k) & = 3^k & +c.3^{k-1}.4^2\frac{(\frac{4^2}{3})^k-1}{(\frac{4^2}{3})-1} & \\
 & T(4^k) & = 3^k & +c.3^k.4^2\frac{(\frac{4^2}{3})^k-1}{15} & \\
 & T(4^k) & = 3^k & (1+c'(\frac{4^2}{3})^k - c')) & \\
 & T(4^k) & = 3^k & (c'(\frac{4^2}{3})^k - c'')) &
\end{array}
$$

$k = log_4 n \qquad T(n) = \Theta(n^2)$