

# DIVISER POUR RÉGNER

À la fin de ce chapitre, je sais :

- ✎ énoncer le principe d'un algorithme de type diviser pour régner
- ✎ distinguer les cas d'utilisation de ce principe (sous-problèmes indépendants)
- ✎ évaluer la complexité d'un algorithme diviser pour régner

## A Diviser pour régner

■ **Définition 1 — Algorithme de type diviser pour régner.** L'idée centrale d'un algorithme de type diviser pour régner est de décomposer le problème étudié en plusieurs sous-problèmes de taille réduite. Ces algorithmes peuvent éventuellement être exprimés récursivement et sont souvent très efficaces.

On distingue trois étapes lors de l'exécution d'un tel algorithme :

1. la division du problème en sous-problèmes qu'on espère plus simples à résoudre,
2. la résolution des sous-problèmes, c'est à cette étape que l'on peut faire appel à la récursivité,
3. la combinaison des solutions des sous-problèmes pour construire la solution au problème.

Ⓡ On devrait donc logiquement nommer ces algorithmes diviser, résoudre et combiner!

🇬🇧 **Vocabulary 1 — Divide and conquer** ⇔ diviser pour régner.

Très souvent<sup>1</sup>, les algorithmes de type diviser pour régner sont exprimés récursivement. À partir d'une taille de problème  $\mathcal{P}$  de taille  $n$ , la division en sous-problèmes aboutit soit à  $n = 1$  soit à  $n = s$ ,  $s$  étant alors une taille pour laquelle on sait résoudre le problème facilement et efficacement. Les étapes 7 et 9 de l'algorithme 1 sont facilement descriptibles à l'aide d'un arbre, comme le montre la figure 2. La hauteur de cet arbre peut être appréciée et quantifiée. Elle sert notamment à calculer la complexité liée aux algorithmes récursifs.

---

1. mais pas toujours



FIGURE 1 – Principe de la décomposition d'un problème en sous-problèmes indépendants pour un algorithme de type diviser pour régner.

---

**Algorithme 1** Diviser, résoudre et combiner (Divide And Conquer)

---

```

1: Fonction DRC( $\mathcal{P}$ )                                      $\triangleright \mathcal{P}$  est un problème de taille  $n$ 
2:    $r \leftarrow$  un entier  $\geq 1$                               $\triangleright$  pour générer  $r$  sous-problèmes à chaque étape
3:    $d \leftarrow$  un entier  $> 1$                               $\triangleright$  on divise par  $d$  la taille du problème
4:   si  $n < s$  alors                                          $\triangleright$  Condition d'arrêt,  $s$  est un seuil à déterminer
5:     renvoyer RÉSOUDRE( $n$ )
6:   sinon
7:      $(\mathcal{P}_1, \dots, \mathcal{P}_r) \leftarrow$  Diviser  $\mathcal{P}$  en  $r$  sous problèmes de taille  $n/d$ .
8:     pour  $i$  de 1 à  $r$  répéter
9:        $S_i \leftarrow$  DRC( $\mathcal{P}_i$ )                                $\triangleright$  Appels récursifs
10:    renvoyer COMBINER( $S_1, \dots, S_r$ )

```

---

Diagram illustrating the recursive decomposition of a function  $f(n)$  into smaller subproblems, showing the recurrence relation and the corresponding tree structure.

The recurrence relation is:

$$f(n) + rf(n/d) + r^2f(n/d^2) + \dots + r^L f(n/d^L)$$

The tree structure shows the recursive calls:

- Root node:  $n$
- Level 1:  $\frac{n}{d}$  (3 nodes)
- Level 2:  $\frac{n}{d^2}$  (9 nodes)
- Level 3:  $\frac{n}{d^3}$  (27 nodes)
- Level 4:  $\frac{n}{d^4}$  (81 nodes)
- Level 5:  $\frac{n}{d^5}$  (243 nodes)
- Level 6:  $\frac{n}{d^6}$  (729 nodes)
- Level 7:  $\frac{n}{d^7}$  (2187 nodes)
- Level 8:  $\frac{n}{d^8}$  (6561 nodes)
- Level 9:  $\frac{n}{d^9}$  (19683 nodes)
- Level 10:  $\frac{n}{d^{10}}$  (59049 nodes)
- Level 11:  $\frac{n}{d^{11}}$  (177147 nodes)
- Level 12:  $\frac{n}{d^{12}}$  (531441 nodes)
- Level 13:  $\frac{n}{d^{13}}$  (1594323 nodes)
- Level 14:  $\frac{n}{d^{14}}$  (4782969 nodes)
- Level 15:  $\frac{n}{d^{15}}$  (14348907 nodes)
- Level 16:  $\frac{n}{d^{16}}$  (43046721 nodes)
- Level 17:  $\frac{n}{d^{17}}$  (129140163 nodes)
- Level 18:  $\frac{n}{d^{18}}$  (387420489 nodes)
- Level 19:  $\frac{n}{d^{19}}$  (1162261467 nodes)
- Level 20:  $\frac{n}{d^{20}}$  (3486784401 nodes)
- Level 21:  $\frac{n}{d^{21}}$  (10460353203 nodes)
- Level 22:  $\frac{n}{d^{22}}$  (31381059609 nodes)
- Level 23:  $\frac{n}{d^{23}}$  (94143178827 nodes)
- Level 24:  $\frac{n}{d^{24}}$  (282429536481 nodes)
- Level 25:  $\frac{n}{d^{25}}$  (847288609443 nodes)
- Level 26:  $\frac{n}{d^{26}}$  (2541865828329 nodes)
- Level 27:  $\frac{n}{d^{27}}$  (7625597484987 nodes)
- Level 28:  $\frac{n}{d^{28}}$  (22876792454961 nodes)
- Level 29:  $\frac{n}{d^{29}}$  (68630377364883 nodes)
- Level 30:  $\frac{n}{d^{30}}$  (205891132094649 nodes)
- Level 31:  $\frac{n}{d^{31}}$  (617673396283947 nodes)
- Level 32:  $\frac{n}{d^{32}}$  (1853020188851841 nodes)
- Level 33:  $\frac{n}{d^{33}}$  (5559060566555523 nodes)
- Level 34:  $\frac{n}{d^{34}}$  (16677181699666569 nodes)
- Level 35:  $\frac{n}{d^{35}}$  (50031545098999707 nodes)
- Level 36:  $\frac{n}{d^{36}}$  (150094635296999121 nodes)
- Level 37:  $\frac{n}{d^{37}}$  (450283905890997363 nodes)
- Level 38:  $\frac{n}{d^{38}}$  (1350851717672992089 nodes)
- Level 39:  $\frac{n}{d^{39}}$  (4052555153018976267 nodes)
- Level 40:  $\frac{n}{d^{40}}$  (12157665459056928801 nodes)
- Level 41:  $\frac{n}{d^{41}}$  (36472996377170786403 nodes)
- Level 42:  $\frac{n}{d^{42}}$  (109418989131512359209 nodes)
- Level 43:  $\frac{n}{d^{43}}$  (328256967394537077627 nodes)
- Level 44:  $\frac{n}{d^{44}}$  (984770902183611232881 nodes)
- Level 45:  $\frac{n}{d^{45}}$  (2954312706550833698643 nodes)
- Level 46:  $\frac{n}{d^{46}}$  (8862938119652501095929 nodes)
- Level 47:  $\frac{n}{d^{47}}$  (26588814358957503287787 nodes)
- Level 48:  $\frac{n}{d^{48}}$  (79766443076872509863361 nodes)
- Level 49:  $\frac{n}{d^{49}}$  (239299329230617529580083 nodes)
- Level 50:  $\frac{n}{d^{50}}$  (717897987691852588740249 nodes)
- Level 51:  $\frac{n}{d^{51}}$  (2153693963075557766220747 nodes)
- Level 52:  $\frac{n}{d^{52}}$  (6461081889226673298662241 nodes)
- Level 53:  $\frac{n}{d^{53}}$  (19383245667680019895986723 nodes)
- Level 54:  $\frac{n}{d^{54}}$  (58149737003040059687960169 nodes)
- Level 55:  $\frac{n}{d^{55}}$  (174449211009120179063880507 nodes)
- Level 56:  $\frac{n}{d^{56}}$  (523347633027360537191641521 nodes)
- Level 57:  $\frac{n}{d^{57}}$  (1570042899082081611574924563 nodes)
- Level 58:  $\frac{n}{d^{58}}$  (4710128697246244834724773689 nodes)
- Level 59:  $\frac{n}{d^{59}}$  (14130386091738734504174321067 nodes)
- Level 60:  $\frac{n}{d^{60}}$  (42391158275216203512522963201 nodes)
- Level 61:  $\frac{n}{d^{61}}$  (127173474825648610537568889603 nodes)
- Level 62:  $\frac{n}{d^{62}}$  (381520424476945831612706668809 nodes)
- Level 63:  $\frac{n}{d^{63}}$  (1144561273430837494838119906427 nodes)
- Level 64:  $\frac{n}{d^{64}}$  (3433683820292512484514359719281 nodes)
- Level 65:  $\frac{n}{d^{65}}$  (10301051460877537453543079157843 nodes)
- Level 66:  $\frac{n}{d^{66}}$  (30903154382632612360629237473529 nodes)
- Level 67:  $\frac{n}{d^{67}}$  (92709463147897837081887712420587 nodes)
- Level 68:  $\frac{n}{d^{68}}$  (278128389443693511245663137261761 nodes)
- Level 69:  $\frac{n}{d^{69}}$  (834385168331080533736989411785283 nodes)
- Level 70:  $\frac{n}{d^{70}}$  (2503155504993241501210968235355849 nodes)
- Level 71:  $\frac{n}{d^{71}}$  (7509466514979724503632904706067547 nodes)
- Level 72:  $\frac{n}{d^{72}}$  (22528399544939173510898714118202641 nodes)
- Level 73:  $\frac{n}{d^{73}}$  (67585198634817520532696142354607923 nodes)
- Level 74:  $\frac{n}{d^{74}}$  (202755595904452561598088427063823769 nodes)
- Level 75:  $\frac{n}{d^{75}}$  (608266787713357684794265281191471307 nodes)
- Level 76:  $\frac{n}{d^{76}}$  (1824799363140073054382795843574413921 nodes)
- Level 77:  $\frac{n}{d^{77}}$  (5474398089420219163148387530723241763 nodes)
- Level 78:  $\frac{n}{d^{78}}$  (16423194268260657489445162592169725289 nodes)
- Level 79:  $\frac{n}{d^{79}}$  (49269582804781972468335487776509175867 nodes)
- Level 80:  $\frac{n}{d^{80}}$  (147808748414345917405006463329527527601 nodes)
- Level 81:  $\frac{n}{d^{81}}$  (443426245243037752215019389988582582803 nodes)
- Level 82:  $\frac{n}{d^{82}}$  (1330278735729113256645058169965747748409 nodes)
- Level 83:  $\frac{n}{d^{83}}$  (3990836207187339769935174509897243245227 nodes)
- Level 84:  $\frac{n}{d^{84}}$  (119725086

FIGURE 2 – Structure d'arbre et appels récursifs pour la récurrence :  $T(n) = rT(n/d) + f(n)$  ET  $n/d^k = s$ . On a choisi  $r = 3$  pour l'illustration, c'est à dire chaque nœud possède trois enfants au maximum : on opère trois appels récursifs à chaque étape de l'algorithme.

## B Exemple de la recherche dichotomique

■ **Exemple 1 — Recherche dichotomique.** L'algorithme de recherche dichotomique 2 est un exemple d'algorithme de type diviser pour régner : la division du problème en sous-problèmes est opérée via la ligne 8. La résolution des sous-problèmes est effectuée par des appels récursifs. La combinaison des résultats n'est pas explicite mais s'effectue sur le tableau lui-même grâce aux indices  $g$  et  $d$ .

(R) La recherche dichotomique est donc bien un cas particulier d'algorithme diviser pour régner avec  $r = 1$  et  $d = 2$ , c'est à dire un seul appel récursif par chemin d'exécution et une division de la taille du problème par deux.

---

**Algorithme 2** Recherche récursive d'un élément par dichotomie dans un tableau trié

---

```

1: Fonction REC_DICH( $t, g, d, elem$ )
2:   si  $g = d$  alors                                     ▷ Condition d'arrêt
3:     si  $t[g] = elem$  alors
4:       renvoyer  $g$ 
5:     sinon
6:       renvoyer l'élément n'a pas été trouvé
7:   sinon
8:      $m \leftarrow (g+d)//2$                                    ▷ Diviser
9:     si  $t[m] < elem$  alors
10:      REC_DICH( $t, m+1, d, elem$ )                         ▷ résoudre
11:     sinon
12:      REC_DICH( $t, g, m, elem$ )                             ▷ résoudre

```

---

Grâce à la figure 3, on peut calculer le nombre d'opérations élémentaires  $T(n)$  nécessaires à l'exécution de l'algorithme 2. On fait l'hypothèse que, hors appel récursif, la fonction REC\_DICH nécessite un nombre constant d'opérations  $c$ . On peut expliciter formellement la relation de récurrence qui existe entre  $T(n)$  et  $T(n/2)$  : on a  $T(n) = T(n/2) + c$ . on peut donc écrire :

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

$$= T(n/4) + c + c = T(n/4) + 2c \quad (2)$$

$$= T(n/8) + 3c \quad (3)$$

$$= \dots \quad (4)$$

$$= T(n/2^k) + kc \quad (5)$$

$$\checkmark = T(1) + kc \quad (6)$$

D'après l'algorithme 2, la condition d'arrêt s'effectue en un nombre constant d'opérations :  $T(1) = O(1)$ . Donc on a  $T(n) = O(k)$ . Or, on a  $\frac{n}{2^k} = 1$ . Donc  $k = \log_2 n$  et  $T(n) = O(\log n)$ .



FIGURE 3 – Structure d'arbre et appels récursifs pour la récurrence de la recherche dichotomique :  $T(n) = T(n/2) + c$  et  $\frac{n}{2^k} = 1$ . Hors appel récursif, la fonction opère un nombre constant d'opérations  $c$ .

## C Exemple de l'exponentiation rapide

L'algorithme naïf de l'exponentiation (cf. algorithme 3) qui permet d'obtenir  $a^n$  en multipliant  $a$  par lui-même  $n$  fois n'est pas très efficace : sa complexité étant en  $O(n)$ .

---

### Algorithme 3 Exponentiation naïve $a^n$

---

```

1: Fonction EXP_NAIVE(a,n)
2:   api ← 1
3:   pour i de 0 à n – 1 répéter
4:     api ← api × a
5:   renvoyer api

```

---

Or, l'exponentiation est une opération très récurrente qu'il est nécessaire de pouvoir exécuter le plus rapidement possible. L'exponentiation rapide (cf. algorithme 4) propose une version récursive de type diviser pour régner dont la complexité est en  $O(\log n)$ .

L'analyse de l'algorithme 4 montre que :

- c'est un cas particulier d'algorithme diviser pour régner avec  $r = 1$  et  $d = 2$ , c'est à dire un seul appel récursif par chemin d'exécution et une division de la taille du problème par deux<sup>2</sup>,
- l'évolution du coût ne dépend pas de  $a$  mais de  $n$ , c'est à dire l'exposant.

On peut procéder de la même manière qu'avec l'algorithme 2 pour calculer la complexité et s'appuyer sur l'arbre de la figure 3. Pour simplifier le calcul, on peut considérer que la taille du

---

2. à un près si  $n$  est pair

**Algorithme 4** Exponentiation rapide  $a^n$ 


---

```

1: Fonction EXP_RAPIDE(a,n)
2:   si  $n = 0$  alors                                     ▷ Condition d'arrêt
3:     renvoyer 1
4:   sinon si  $n$  est pair alors
5:      $p \leftarrow \text{EXP\_RAPIDE}(a, n//2)$                  ▷ Appel récursif
6:     renvoyer  $p \times p$ 
7:   sinon
8:      $p \leftarrow \text{EXP\_RAPIDE}(a, (n-1)//2)$              ▷ Appel récursif
9:     renvoyer  $p \times p \times a$ 

```

---

problème est divisée par deux. Le coût hors appel récursif est constant car il s'agit de multiplications. On a donc  $T(n) = O(\log n)$ .

**(R)** L'algorithme 4 n'est pas à récursivité terminale. Sa version itérative est moins évidente. Celle-ci est détaillée par l'algorithme 5

**Algorithme 5** Exponentiation rapide  $a^n$  (version itérative)

---

```

1: Fonction EXP_RAPIDE_ITE( $a, n$ )                             ▷  $a$  et  $n$  sont des entiers naturels
2:   si  $n = 0$  alors
3:     renvoyer 1
4:    $p \leftarrow a$ 
5:    $m \leftarrow 1$ 
6:    $n_{it} \leftarrow \lfloor \log_2(n) \rfloor$ 
7:   pour  $i$  de 1 à  $n_{it}$  répéter
8:      $m \leftarrow n // 2^{n_{it}-i}$ 
9:     si  $m$  est pair alors
10:       $p \leftarrow p \times p$ 
11:     sinon
12:       $p \leftarrow p \times p \times a$ 
13:   renvoyer  $p$ 

```

---

**D Exemple du tri fusion**

Les tris génériques abordés jusqu'à présent, par sélection ou insertion, présentent des complexités polynomiales en  $O(n^2)$  dans le pire des cas. L'algorithme de tri fusion a été inventé par John von Neumann en 1945. C'est un bel exemple d'algorithme de type diviser pour régner avec  $r = 2$  et  $d = 2$ , c'est à dire deux appels récursifs par chemin d'exécution et une division de la taille du problème par deux (cf. figure 4). Il permet de dépasser cette limite et d'obtenir un tri générique de complexité logarithmique. Ce tri est comparatif, il peut s'effectuer en place et

les implémentations peuvent être stables.

Son principe (cf. algorithmes 6 et 7) est simple : transformer le tri d'un tableau à  $n$  éléments en sous-tableaux ne comportant qu'un seul élément<sup>3</sup> puis les recombinaison en un seul tableau en conservant l'ordre. L'algorithme est divisé en deux fonctions :

- TRI\_FUSION qui opère concrètement la division et la résolution des sous-problèmes,
- FUSION qui combine les solutions des sous-problèmes en fusionnant deux sous-tableaux triés.

Pour le calcul de la complexité, on a la relation de récurrence  $T(n) = 2T(n/2) + f(n)$  où  $f(n)$  représente le nombre d'opérations élémentaires nécessaires pour fusionner deux sous-tableaux de taille  $n/2$ . La condition d'arrêt de la récursivité est atteinte pour  $\frac{n}{2^k} = 1$  c'est à dire pour  $k = \log_2 n$ . Par ailleurs, on remarque que  $T(1) = c$  est une opération à coût constant, suite à la condition d'arrêt on ne fait que retourner le tableau non modifié.

$$T(n) = 2T(n/2) + f(n) \quad (7)$$

$$= 4T(n/4) + 2O(n) \quad (8)$$

$$= 8T(n/8) + 8O(n) \quad (9)$$

$$= \dots \quad (10)$$

$$= 2^k T(1) + 2^k O(n) \quad (11)$$

$$= c \log_2 n + O(n) \log_2 n \quad (12)$$

Dans le pire des cas, l'étape de combinaison effectue la combinaison de deux sous-tableaux de dimension  $n/2$  et les trois boucles de la fonction FUSION effectue  $n/2$  tours. On a alors  $f(n) = O(n)$ . C'est pourquoi, la complexité du tri fusion vaut  $T(n) = O(\log_2 n + n \log_2 n) = O(n \log n)$

---

**Algorithme 6** Tri fusion
 

---

```

1: Fonction TRI_FUSION(t, g, d)
2:   si d = g alors                                     ▷ Condition d'arrêt
3:     renvoyer t
4:   m ← (g+d)//2                                         ▷ on découpe au milieu
5:   TRI_FUSION(t, g, m)
6:   TRI_FUSION(t, m+1, d)
7:   FUSION(t, g, m, sup)
  
```

---



---

3. et donc déjà triés!

---

**Algorithme 7** Fusion de sous-tableaux triés
 

---

```

1: Fonction FUSION(t, g, m, d)
2:    $i \leftarrow m - g + 1$ 
3:    $j \leftarrow d - m$ 
4:   G, D deux tableaux de taille i et j
5:   pour k de 1 à i répéter
6:      $G[k] \leftarrow t[g + k - 1]$ 
7:   pour k de 1 à i répéter
8:      $D[k] \leftarrow t[m + k]$ 
9:    $G[i + 1] \leftarrow \text{None}$ 
10:   $D[j + 1] \leftarrow \text{None}$ 
11:   $p \leftarrow 1$ 
12:   $q \leftarrow 1$ 
13:  pour v de g à d répéter
14:    si  $G[p] < D[q]$  alors
15:       $t[v] \leftarrow G[p]$ 
16:       $p \leftarrow p + 1$ 
17:    sinon
18:       $t[v] \leftarrow D[q]$ 
19:       $q \leftarrow q + 1$ 

```

---

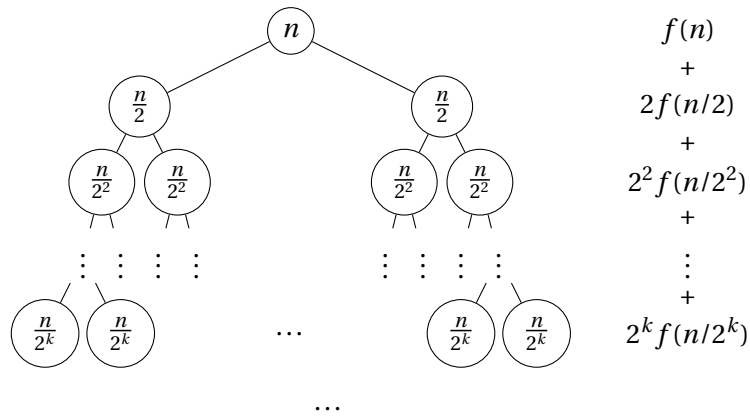


FIGURE 4 – Structure d'arbre et appels récurifs pour le tri fusion :  $T(n) = 2T(n/2) + f(n)$  et  $\frac{n}{2^k} = 1$ . La fonction FUSION opère un nombre d'opérations  $f(n)$ .