Diviser pour régner

Sommaire

Sommaire

Le principe Diviser pour régner

- Pour résoudre un problème donné, une approche utilisée par plusieurs algorithmes consiste à
  1. décomposer le problème en deux (ou plus de deux) sous-problèmes plus simples, en considérant des problèmes (liés au problème à résoudre) mais de taille plus petite ;
  2. résoudre ces sous-problèmes de taille plus petite, donc de résolution plus facile ;
  3. recombiner les solutions des sous-problèmes en une solution du problème initial (ce qui s'avère souvent possible).
- Lorsqu'à l'étape 2 les sous-problèmes sont eux même résolus en réappliquant ce principe de décomposition/résolution/recomposition, on obtient un algorithme *récursif*.
- Ce principe de résolution porte le nom «diviser pour régner» [1]

1. L'étymologie de cette expression est à rattacher à la stratégie militaire qui consiste à vaincre un enemi en le divisant, c'est à dire en semant la division au sein de l'enemi lui-même.

- On va voir dans ce chapitre qu'on peut appliquer ce principe à la réalisation d'algorithmes :
  - de recherche : la recherche dichotomique (récursive) ;
  - de tri : le tri fusion.

Rq : parfois l'étape 3 n'est pas utile, dans la mesure où dans certains cas on peut directement déduire la solution globale à partir d'une solution partielle. C'est par exemple possible pour la recherche dichotomique, car à chaque étape de la division, un seul des sous-problèmes doit en fait être résolu, et sa solution est celle du problème global.

Sommaire

La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50

La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50

↑ g ↑ d

La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

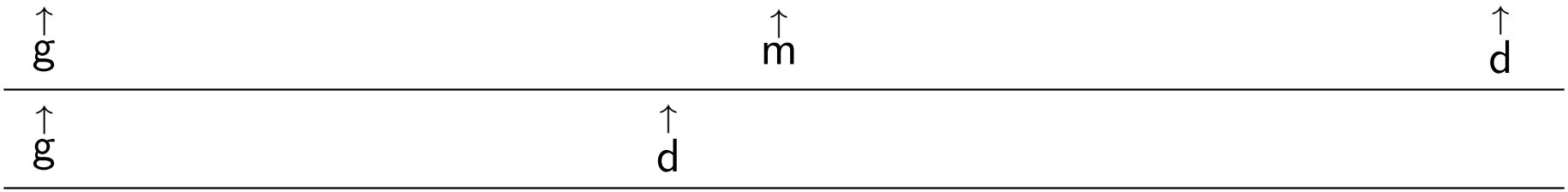
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50

g m d

La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

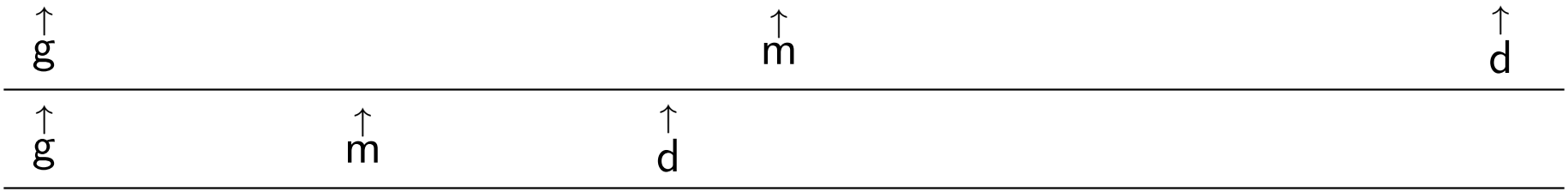
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

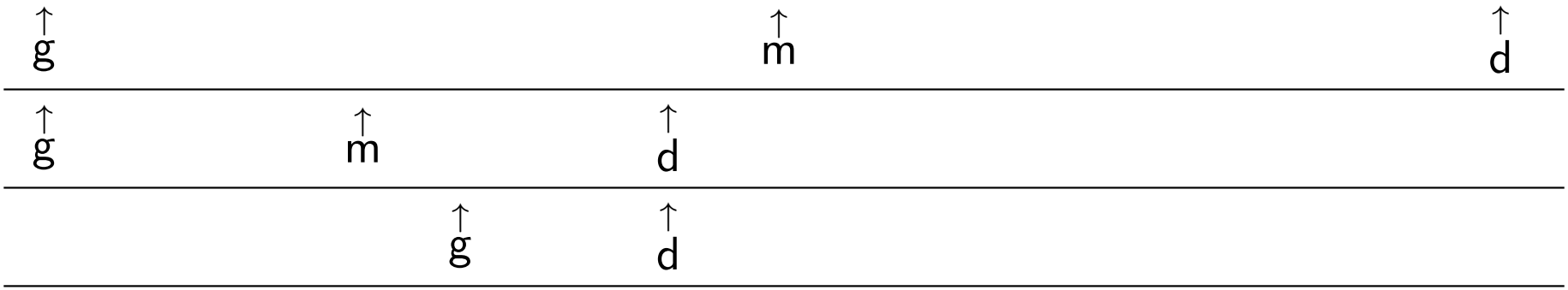
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

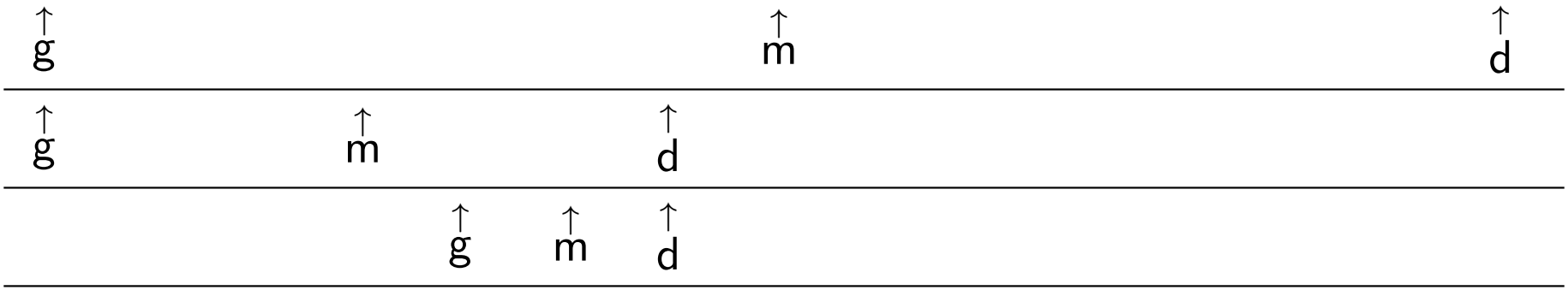
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

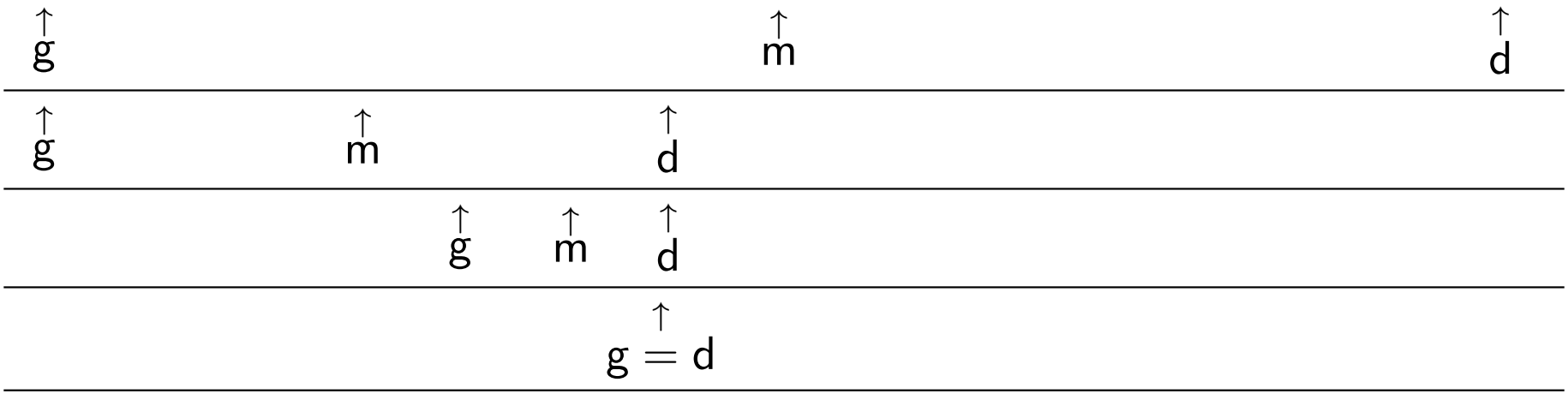
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

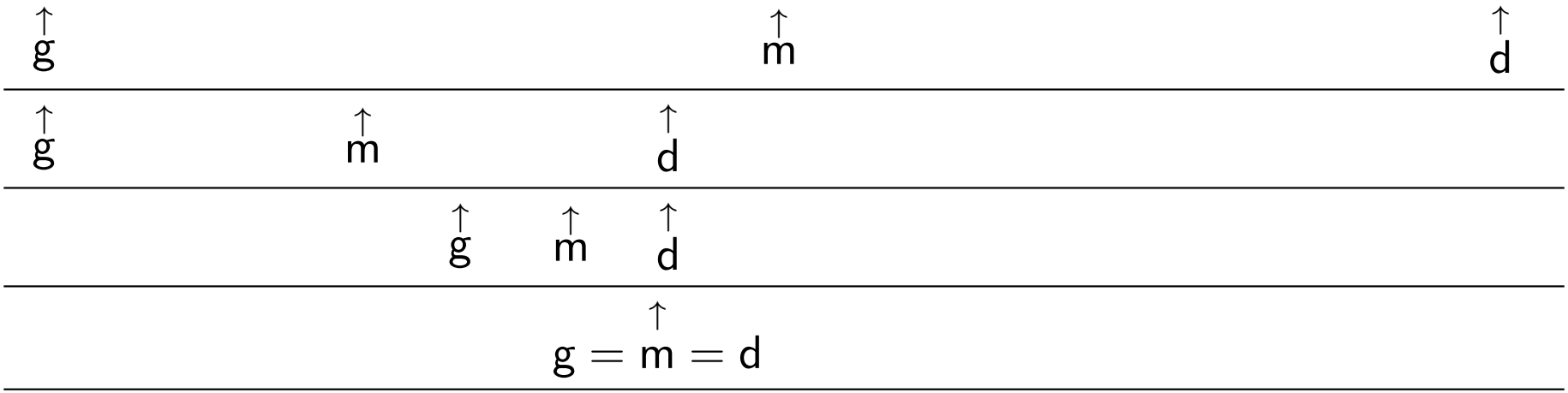
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique

Exemple 1 : recherche de la valeur $v = 14$ dans un tableau t de 15 valeurs trié par ordre croissant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50

↑ g							↑ m							↑ d
↑ g			↑ m			↑ d								
				↑ g	↑ m	↑ d								
						↑ g = m = d								
					↑ d	↑ g								

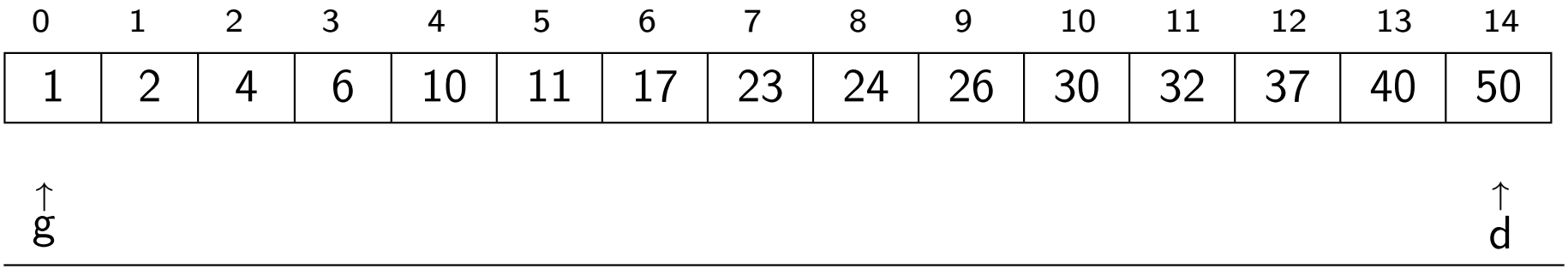
La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50

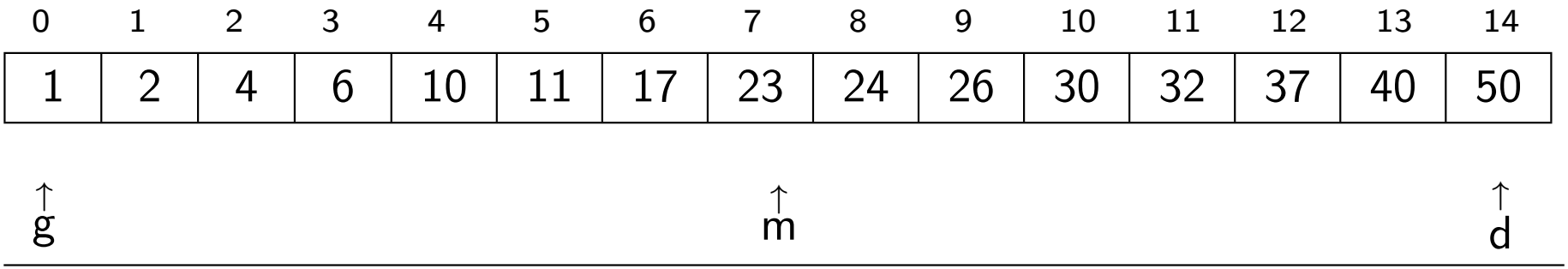
La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :



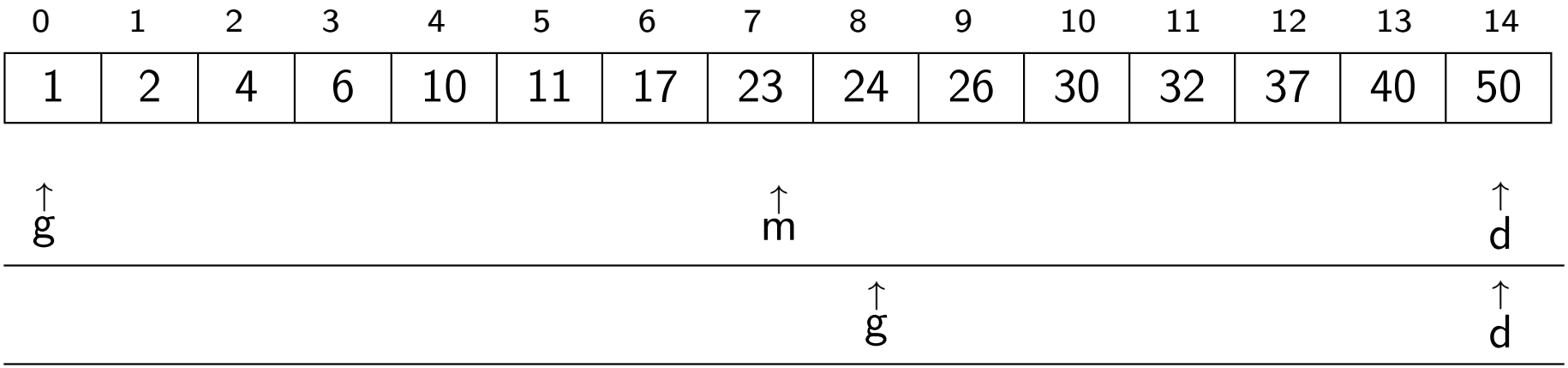
La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :



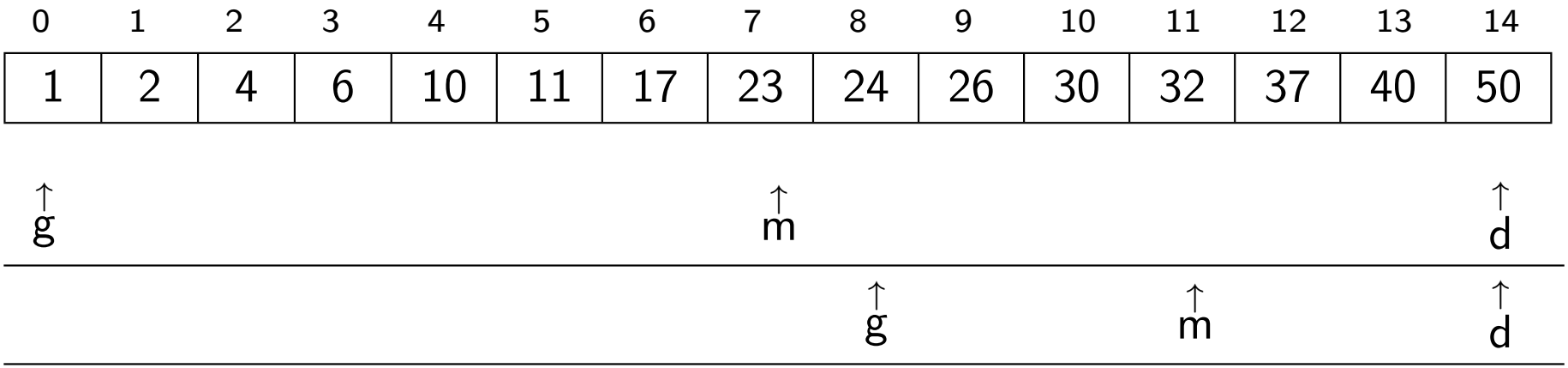
La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :



La recherche dichotomique (suite)

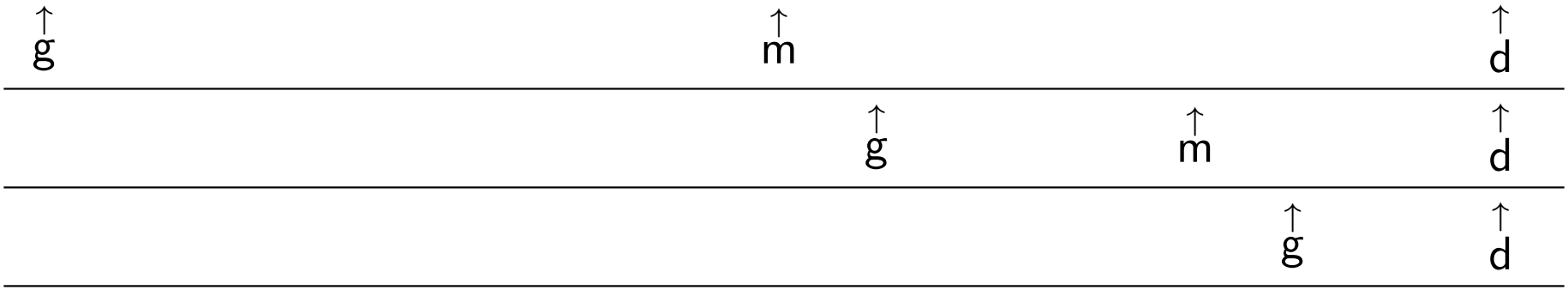
Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :



La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :

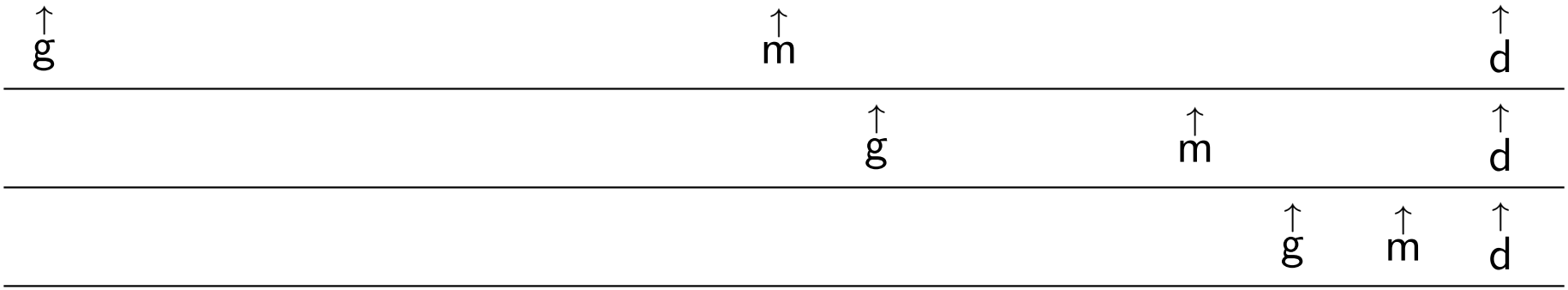
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique (suite)

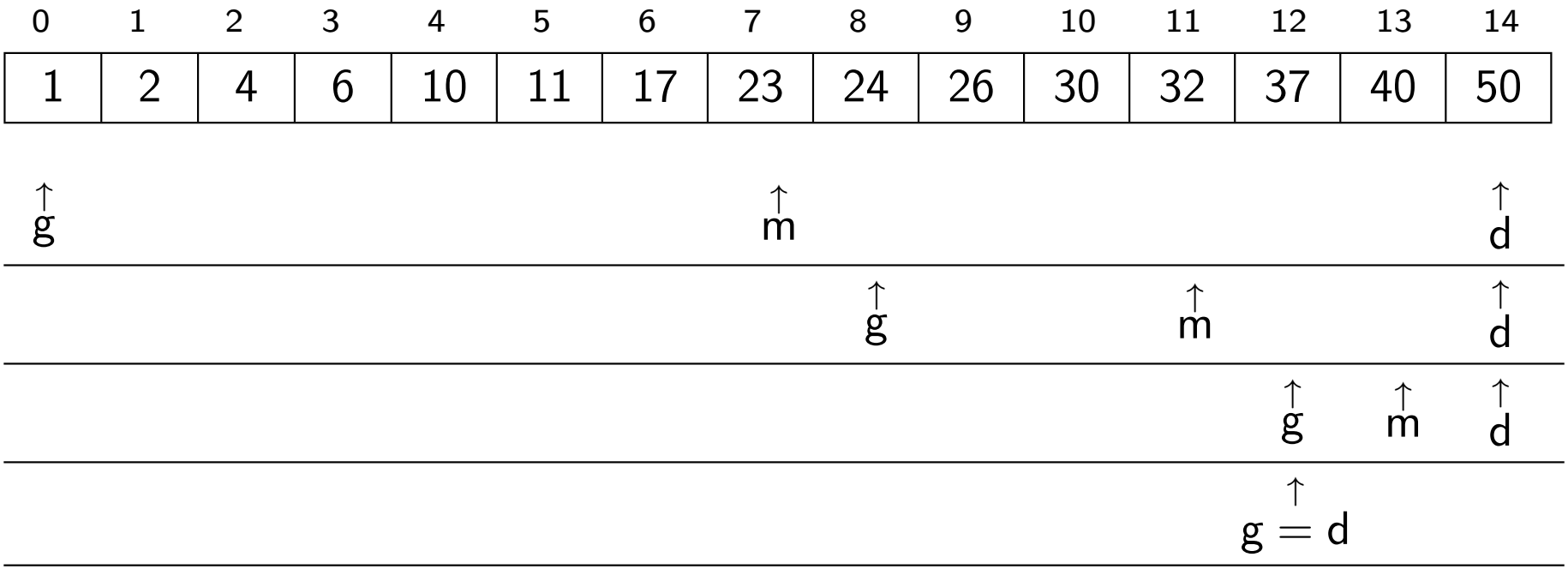
Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



La recherche dichotomique (suite)

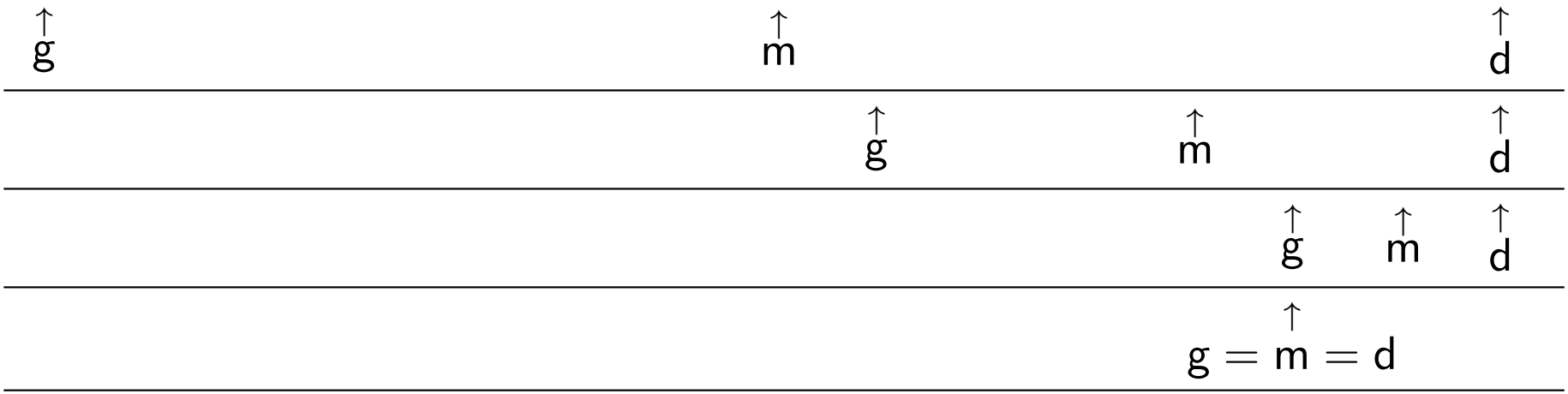
Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :



La recherche dichotomique (suite)

Exemple 2 : recherche de la valeur $v = 37$ dans le tableau t de 15 valeurs précédent :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	6	10	11	17	23	24	26	30	32	37	40	50



Algorithme itératif

Rappel : une implémentation itérative (au programme de Première) de l'algorithme de recherche dichotomique écrite en pseudo-code est la suivante :

```
Fonction DICHO_ITER(t, v)
    ▷ Renvoie l'indice de présence de v dans le tableau t si v est dans t, Nil
      sinon. Méthode par dichotomie itérative. ◁
    g ← 0
    d ← longueur(t) − 1
    tant que g ⩽ d faire
        m ← (g + d)//2                    # Division entière de g + d par 2
        si t[m] > v alors
            d ← m − 1
        sinon si t[m] = v alors
            renvoyer m
        sinon
            g ← m + 1
    renvoyer Nil
```

Ex. 1 Coder et tester l'algorithme précédent en Python.

Remarque : si la valeur v est présente plusieurs fois dans le tableau t, l'algorithme donne *un* des indices de présence de v dans t, mais on ne peut dire à l'avance si cet indice est (par exemple) le plus petit (considérer par exemple les tableaux contenant 1, 5, 5, 10 et 1, 5, 5, 10, 11). En revanche on pourrait modifier (en l'augmentant de quelques lignes) l'algorithme pour qu'il donne toujours (par exemple) le plus petit indice de présence de v dans t.

Algorithme récursif

- On applique le principe Diviser pour régner.
- Une particularité de la recherche dichotomique est que seul un des deux sous-problèmes, associés aux 2 tableaux de taille 2 fois plus petite, doit être résolu.
- Il suffit de déterminer, par comparaison de la valeur v recherchée avec la borme m du milieu du tableau, si la recherche doit être récursivement poursuivie dans le sous-tableau gauche ou droit.
- Écrit en pseudo-code, l'algorithme de recherche par dichotomie récursif (donné page suivante) est basé sur l'appel d'une fonction DICHO_RECUR(t, g, d, v) où :
  - t est le tableau où v est recherchée ;
  - g, d sont respectivement les indices gauche et droite de t entre lesquels v est recherchée.

```
Fonction DICHO_RECUR(t, g, d, v)
    ▷ Renvoie un indice de présence de v dans le tableau t si v est dans t,
      Nil sinon. Méthode par dichotomie récursive. ◁
    si d < g alors                                  # Cas de base : tableau vide
        renvoyer Nil
    sinon
        m ← (g + d)//2                              # Division entière de g + d par 2
        si t[m] > v alors
            renvoyer DICHO_RECUR(t, g, m − 1, v)
        sinon si t[m] = v alors
            renvoyer m
        sinon
            renvoyer DICHO_RECUR(t, m + 1, d, v)
```

Rq.1 : Dans le programme principal, la recherche dichotomique récursive est réalisée par l'appel (dans une affectation, ou une instruction d'affichage) de DICHO_RECUR$(t, 0, len(t) - 1, v)$.

Rq. 2 : cet algorithme fonctionne correctement même si v n'est pas comprise entre la valeur minimale et la maximale de t. On peut toutefois améliorer son efficacité en évitant une recherche inutile à l'aide d'un test réalisé avant l'appel de la fonction DICHO_RECUR.

Ex. 2 Coder l'algorithme de dichotomie récursif en Python. Le tester sur des tableaux.

Ex. 3 À partir de quelle taille du tableau t l'appel de DICHO_RECUR(t, 0, len(t)-1, v) peut-il provoquer une erreur de type `RecursionError` ? Cette taille de tableau est-elle envisageable à l'heure actuelle ? (Rappel : par défaut, l'interpéteur Python lève ce type d'erreur lorsqu'une fonction est appelée récusivement plus de 1000 fois.)

Sommaire

Le tri fusion

- On se propose de trier une liste de nombres présents dans un tableau (une liste en Python).
- On a vu en Première deux algorithmes de tri (dont vous devez connaître le nom et le principe).
- On va ici appliquer le principe Diviser pour régner au tri, ce qui conduira à l'algorithme dit de *tri fusion*.
- Voici d'abord (page suivante) une illustration de cet algorithme sur un exemple avec un tableau de 10 nombres.

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
0	4	8	3	9	1	2	5	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
0	4	8	3	9	1	2	5	6	7
0	3	4	8	9	1	2	5	6	7

Table 1 – Tri fusion d'une liste de 10 valeurs.

0	1	2	3	4	5	6	7	8	9
4	2	9	6	0	1	3	7	8	5
4	9	0	3	8	2	6	1	7	5
4	0	8	9	3	2	1	5	6	7
4	8	0	9	3	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
4	8	0	3	9	2	5	1	6	7
0	4	8	3	9	1	2	5	6	7
0	3	4	8	9	1	2	5	6	7
0	1	2	3	4	5	6	7	8	9

En appliquant le principe Diviser pour régner, c'est à dire en appliquant récursivement les opérations de division/résolution/recombinaison, on obtient l'algorithme suivant de tri fusion écrit en pseudo-code, où l est la liste de nombres à trier.

```
1: Fonction TRI_FUSION(l)
2:     ▷ Renvoie la liste de nombres l triée par ordre croissant à l'aide de
         l'algorithme récursif du tri fusion.                                ◁
3:     si l est vide ou ne contient qu'un élément alors
                                                  # Cas de base : liste triée
4:         renvoyer l
5:     sinon
6:         (l1, l2) ← decoupe(l)
                  # Découpe de l en deux sous-listes l1 et l2 égales (à un près)
7:         l1 ←TRI_FUSION(l1)
8:         l2 ←TRI_FUSION(l2)
9:         renvoyer FUSION(l1, l2)
```

- Dans l'algorithme ci-dessus, on suppose disposer, ligne 6, d'une fonction DECOUPE(l) qui renvoie un couple constitué de deux listes l_1 et l_2 ayant le même nombre d'éléments (à un près), dont la réunion forme l'ensemble des éléments de l.
- Ces deux listes ayant été triées (lignes 7 et 8), on les réassemble à l'aide d'une fonction FUSION(l_1, l_2) qui renvoie la liste triée associée à l_1 et l_2.
- Pour la fonction DECOUPE, une solution est de remplir alternativement l_1 et l_2 en parcourant toute la liste l. On donne page suivante, écrit en pseudo-code, un algorithme de cette fonction.

Fonction DECOUPE(l)

▷ *Renvoie un couple de deux listes (l_1, l_2) remplies avec les éléments de la liste l et ayant (à un près) le même nombre d'éléments.* ◁

Initialiser l_1 et l_2 avec la liste vide
pour tout $e \in l$ **faire**
    ajouter e à l_1
    échanger l_1 et l_2
renvoyer (l_1, l_2)

Pour la fonction FUSION, un algorithme itératif possible est donné page suivante (il est possible d'écrire une version récursive).

Fonction FUSION(l_1, l_2)

▷ *Renvoie la liste triée contenant les éléments (nombres) de l_1 et de l_2, supposées triées.* ◁

Initialiser l avec la liste vide
tant que l_1 et l_2 ne sont pas vides **faire**
    $maxi \leftarrow$ plus grand élément de l_1 et l_2
    ajouter $maxi$ au début de l
    enlever $maxi$ de l_1 ou l_2, selon sa provenance (mais pas des 2 !)
si l_1 n'est pas vide **alors**
    concaténer l_1 au début de l
si l_2 n'est pas vide **alors**
    concaténer l_2 au début de l
renvoyer l

Ex. 4 À l'aide des algorithmes précédents, écrire en Python une fonction `tri_fusion(l)` qui renvoie la liste `l` triée par l'algorithme de tri fusion.

Remarques/rappels : en Python

- il est possible d'ajouter un élément `e` à la fin d'une liste `l` à l'aide de `l.append(e)` ;
- on peut enlever le dernier élément de `l` à l'aide de `l.pop()` ;
- pour insérer insérer un élément `e` au début de `l` (à l'indice 0), on peut utiliser `l.insert(0, e)` ;
- la concaténation des listes `l1` et `l2` dans cet ordre est obtenue avec `l1 + l2`.

Ex. 5 Réécrire l'algorithme du tri fusion pour qu'il trie une liste chaînée. On utilisera à cet effet la classe `Cellule` vue dans le chapitre sur les listes chaînées.

Ex. 6 Voici (page suivante) l'illustration sur un tableau de 10 nombres de deux tris qui figurent au programme d'algorithmique de Première.

1. Rappeler le nom de ces deux tris.
2. Associer chaque tri à la figure qui lui correspond dans les deux exemples donnés.
3. On donne (page suivante) les algorithmes en Python de deux fonctions de tri qui implémentent les deux tris des questions précédentes.
   1. Quel est le nom du `tri1` ? du `tri2` ?
   2. Implémenter et tester ces fonctions.

4 2 9 6 0 1 3 7 8 5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5
0	1	2	3	4 |	9	6	7	8	5

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5
0	1	2	3	4 |	9	6	7	8	5
0	1	2	3	4	5 |	6	7	8	9

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5
0	1	2	3	4 |	9	6	7	8	5
0	1	2	3	4	5 |	6	7	8	9
0	1	2	3	4	5	6 |	7	8	9

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5
0	1	2	3	4 |	9	6	7	8	5
0	1	2	3	4	5 |	6	7	8	9
0	1	2	3	4	5	6 |	7	8	9
0	1	2	3	4	5	6	7 |	8	9

4	2	9	6	0	1	3	7	8	5
0 |	2	9	6	4	1	3	7	8	5
0	1 |	9	6	4	2	3	7	8	5
0	1	2 |	6	4	9	3	7	8	5
0	1	2	3 |	4	9	6	7	8	5
0	1	2	3	4 |	9	6	7	8	5
0	1	2	3	4	5 |	6	7	8	9
0	1	2	3	4	5	6 |	7	8	9
0	1	2	3	4	5	6	7 |	8	9
0	1	2	3	4	5	6	7	8 |	9

4	2	9	6	0	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

0	1	2	4	6	9	3	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

0	1	2	4	6	9	3	7	8	5

0	1	2	3	4	6	9	7	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

0	1	2	4	6	9	3	7	8	5

0	1	2	3	4	6	9	7	8	5

0	1	2	3	4	6	7	9	8	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

0	1	2	4	6	9	3	7	8	5

0	1	2	3	4	6	9	7	8	5

0	1	2	3	4	6	7	9	8	5

0	1	2	3	4	6	7	8	9	5

4	2	9	6	0	1	3	7	8	5

0	2	9	6	4	1	3	7	8	5

0	1	9	6	4	2	3	7	8	5

0	1	2	6	4	9	3	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	9	6	7	8	5

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

4	2	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	9	6	0	1	3	7	8	5

2	4	6	9	0	1	3	7	8	5

0	2	4	6	9	1	3	7	8	5

0	1	2	4	6	9	3	7	8	5

0	1	2	3	4	6	9	7	8	5

0	1	2	3	4	6	7	9	8	5

0	1	2	3	4	6	7	8	9	5

0	1	2	3	4	5	6	7	8	9

```python
def tri1(t):
    for i in range(1, len(t)):
        j = i
        v = t[i]
        while j>0 and t[j-1] > v:
            t[j] = t[j-1]
            j = j - 1
        t[j] = v
```

```python
def tri2(t):
    for i in range(len(t)):
        m = i
        for j in range(i+1, len(t)):
            if t[j] < t[m]:
                m = j
        t[i], t[m] = t[m], t[i]
```

Ex. 7 Pour mesurer la complexité temporelle d'un algorithme on peut en Python utiliser la fonction `perf_counter()` du module `time` qui permet de mesurer le temps d'exécution (en s) d'un bloc d'instructions :

```
from time import *

debut = perf_counter()
# bloc dont on veut
# mesurer le temps
# d'exécution
fin = perf_counter()
temps = fin - debut  # temps d'exécution
```

On se propose d'utiliser `perf_counter` pour comparer la complexité des algorithmes de tri évoqués à l'exercice précédent et de l'algorithme de tri fusion. Il s'agit de mesurer le temps t (en s) d'exécution de ces algorithmes sur un même tableau en fonction de la taille N de celui-ci.

1. Écrire en Python une nouvelle fonction associée à chaque fonction de tri qui :
   1. fait une *copie* du tableau t qui leur est passé en argument (de sorte que les appels successifs des 3 tris ne modifieront pas ce dernier) ;
   2. réalise le tri de cette copie ;
   3. renvoie uniquement le temps d'exécution du tri.

   Par exemple, pour le tri fusion, on écrira une fonction `tri_fusion_complexite(t)` dont le code est donné ci-dessous (s'en inspirer pour écrire les deux autres).

```
def tri_fusion_complexite(t):
    """
    Renvoie le temps d'exécution (en s) du tri fusion réalisé sur
    le tableau t (non modifié) à l'aide de la fonction tri_fusion
    """
    tab = t[:]  # tab: copie de t
    debut = perf_counter()
    tri_fusion(tab)
    fin = perf_counter
    return fin - debut
```

2. À l'aide de la fonction `randint` du module `random` on peut générer un tableau `tab` aléatoire de `N` nombres tirés au hasard (avec une loi uniforme), par exemple dans l'intervalle de 0 à 99 (inclus) :

```
tab = [randint(0,99] for i in range(N)]
```

Utiliser cette possibilité et les fonctions écrites à la question précédentes pour calculer et afficher le temps d'exécution des 3 tris pour des valeurs de `N` successivement égales à $1000, 2000, 4000, 8000, 16000$. Rappeler l'ordre de grandeur (en fonction de N) de la complexité des deux algorithmes de tri étudiés en Première. Si on multiplie par 2 la taille du tableau à trier, par quel facteur doit être multiplié le temps d'exécution du tri ? Cette règle est-elle vérifiée par vos mesures ?

3. On souhaite afficher à l'aide de la bibliothèque `matplotlib` les courbes donnant pour chaque algorithme de tri le temps d'exécution du tri (en s) en fonction de la taille N du tableau, ceci pour N prenant les valeurs indiquées à la question précédente. Utiliser la fonction `plot` pour faire apparaître ces courbes. Vous devez obtenir un graphique semblable à celui donné page suivante. Quel est l'algorithme qui semble le plus performant ?

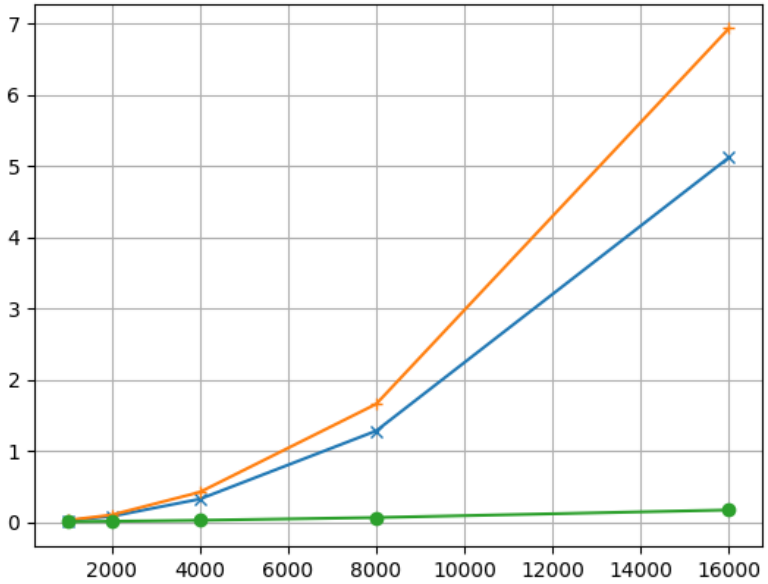


Figure 1 – Temps d'exécution des algorithmes de tri en fonction de la taille des tableaux.

4. On démontre que la complexité de l'algorithme de fusion est en $N \log_2(N)$. Si on double N, par quel facteur est multiplié le temps d'exécution ? Vérifier ce résultat à l'aide de mesures. On pourra faire varier N jusqu'à 64000 en n'exécutant que le tri fusion.

Sommaire

Rotation d'une image d'un quart de tour

- Dans ce paragraphe on montre comment il est possible d'utiliser le principe diviser pour régner pour écrire un algorithme réalisant la rotation d'une image bitmap d'un quart de tour dans le sens direct.
- La bibliothèque `PIL` permet de manipuler une image en Python, plus précisément à l'aide de son module `Image` comme le montrent les lignes de code ci-dessous :

```
from PIL import Image

im = Image.open("Image.png")
largeur, hauteur = im.size
px = im.load()
```

- Ci-dessus, on charge l'image contenue dans le fichier dont le nom est «`Image.png`», on obtient sa taille (variables `largeur` et `hauteur`). La variable `px` est la matrice des pixels constituant l'image.

- Pour `0 <= x <= largeur` et `0 <= y <= hauteur`, `px[x, y]` est la couleur du pixel qui se trouve à la colonne i et à la ligne j (les indices de colonne augmentant de gauche à droite et ceux des lignes de haut en bas). Dans le cas d'une image en noir et blanc, `px[x, y]` est l'intensité du pixel sur échelle de 0 à 255, 0 correspondant au noir et 255 au blanc (dans le format .png).
- On peut affecter une couleur (ou une intensité de gris) à un pixel avec une instruction de la forme `px[x, y]=c` où c est une couleur (ou une intensité de gris).
- On suppose que l'image est carrée et que sa taille est une puissance de 2.
- L'idée du principe Diviser pour régner appliqué à la rotation d'une image est la suivante :
  1. diviser l'image en quatre sous-images ;
  2. faire tourner chaque sous-image de 90 degrés dans le sens direct (appel récursif) ;
  3. déplacer chaque sous-image tournée vers sa position finale.
- Ce principe est illustré page suivante.

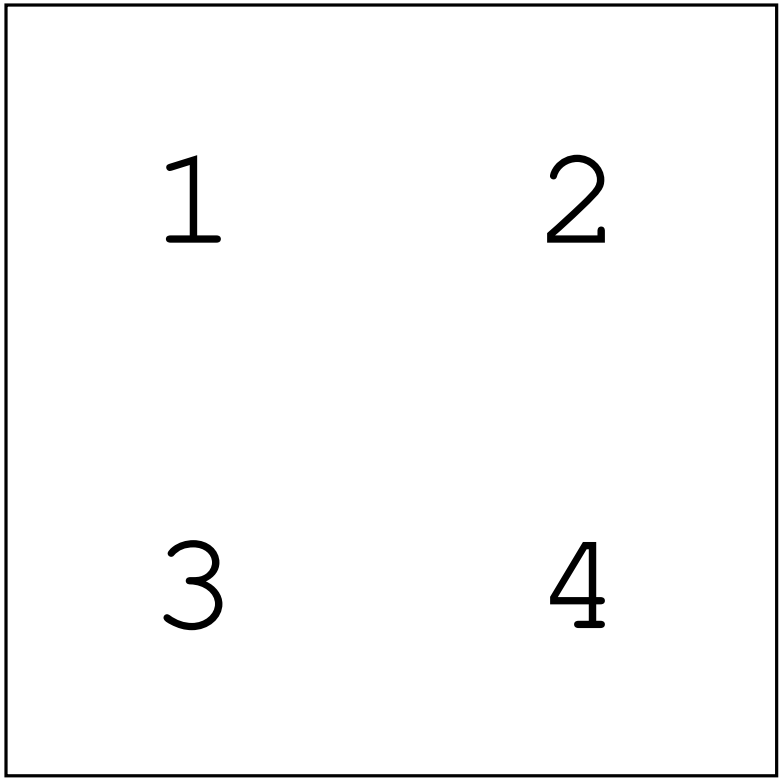


Image initiale

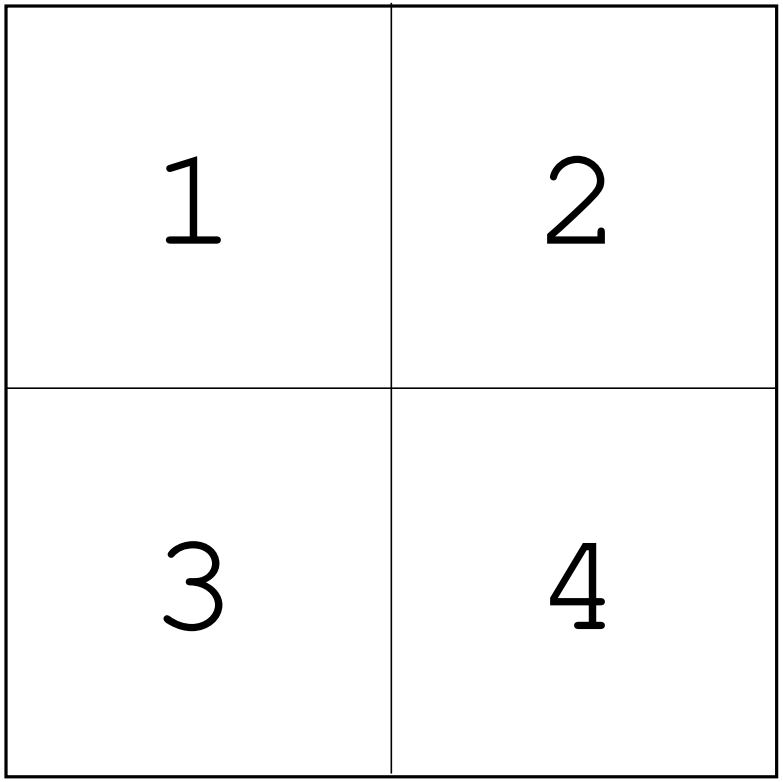
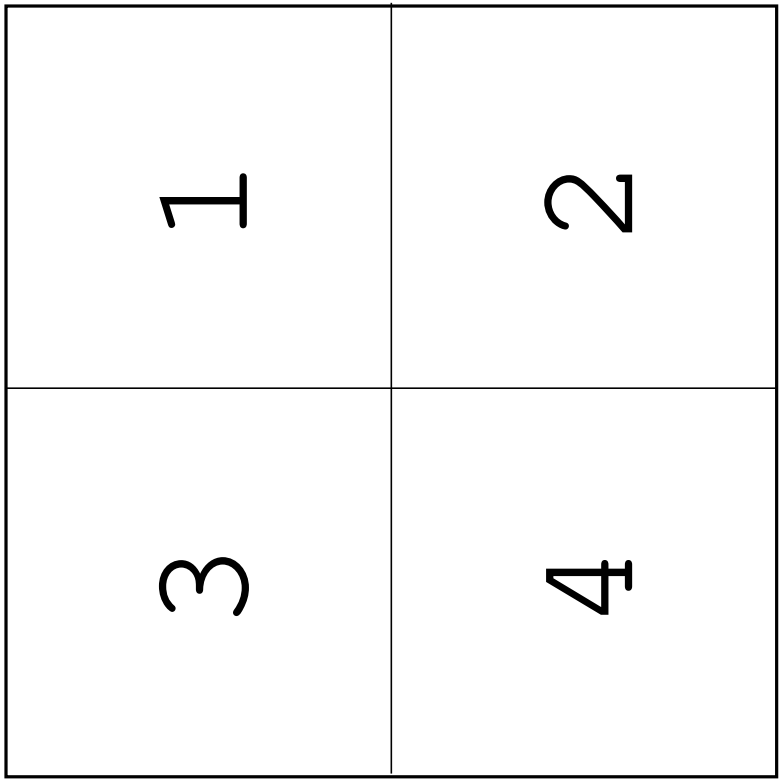
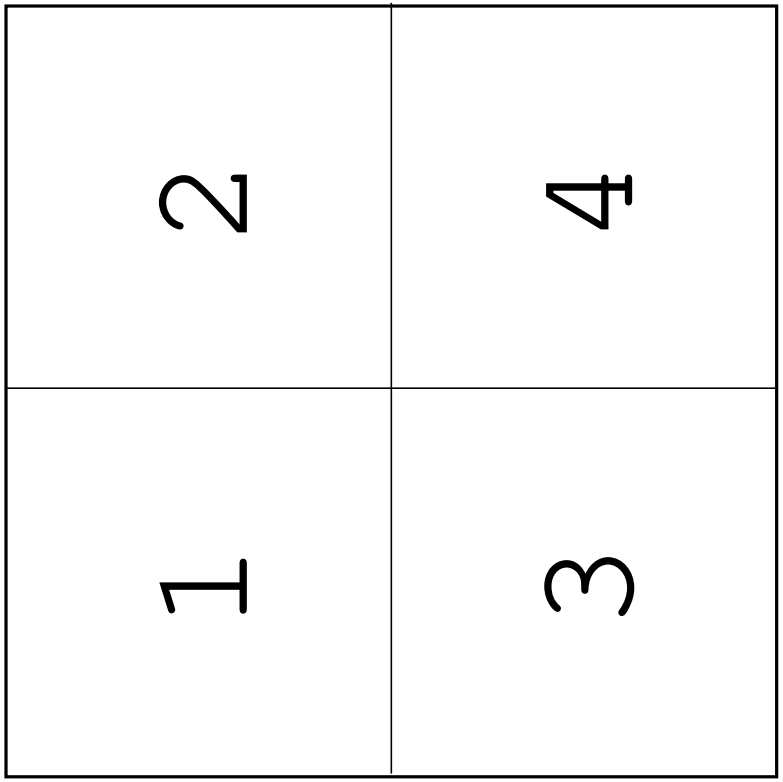


Image découpée



Rotation des sous-images



Permutation (par translation)
des sous-images

Ex. 8 Écrire d'abord en Python le code d'une fonction `rotation_aux(px, x, y, t)` qui effectue récursivement la rotation de la portion carrée de l'image comprise entre les pixels de coordonnées `(x, y)` et `(x + t, y + t)`. Cette fonction ne renvoie rien.

Écrire ensuite le code d'une fonction `rotation(px, taille)` qui effectue la rotation de 90 degrés de l'image toute entière, sa dimension étant donnée par le paramètre `taille`. Une fois la rotation effectuée, on pourra sauvegarder le résultat dans un autre fichier avec la commande `im.save("Rotation.png")`.

Vous pourrez utiliser les images `A.png` et et `lena.png` téléchargeables sur le site du cours.

Rq : pour réaliser l'étape 3 ci-dessus, vous avez deux possibilités :

- faire une copie d'une des sous-images tournées avant de les déplacer, de façon à ne pas écraser une des sous-images ;
- utiliser une affectation multiple à quatre variables, ce qui est plus compact et élégant.

Si vous optez pour la première solution, il est possible de recopier la sous-image choisie dans un tableau à deux dimensions, à l'aide de deux boucles `for` imbriquées.

Sommaire

Exercices

Ex. 9 Dans la recherche dichotomique récursive (utilisant le principe diviser pour régner), donner la séquence d'appels à la fonction `dicho_recur` engendrés par l'appel de `dicho_recur(t, 0, len(t)-1, v)` pour :

1. `t=[0, 1, 1, 2, 3, 5, 8, 13, 21]` et `v=13`
2. `t=[0, 1, 1, 2, 3, 5, 8, 13, 21]` et `v=12`

Ex. 10 Dans le tri fusion, quelles sont les deux listes renvoyées par l'appel de `decoupe(l)` lorsque `l=[8, 1, 3, 0, 1, 2, 5]`. L'ordre des éléments est-il préservé ? Est-ce important ?

Ex. 11 Le problème des tours de Hanoï est un jeu célèbre constitué de trois tiges verticales sur lesquelles sont empilés des disques de diamètres différents. Il s'agit de déplacer tous les disques de la première tige (dessin de gauche) vers la dernière tige (dessin de droite) en respectant deux contraintes : d'une part, on ne peut déplacer qu'un seul disque à la fois ; d'autre part, on ne peut pas poser un disque sur un disque de diamètre plus petit. Sur l'exemple ci-dessus, avec quatre disques, il ne faut pas moins de 15 déplacements pour y parvenir.

1. Écrire une fonction `hanoi(n)` qui affiche la solution du problème des tours de Hanoï pour `n` disques, sous la forme de déplacements élémentaires désignant les trois tiges par les entiers 1, 2 et 3, de la manière suivante :

   ```
   déplace de 1 vers 3
   déplace de 1 vers 2
   ...
   ```

   Indication : commencer par une fonction récursive `deplace(a, b, c, k)` qui résout le problème plus général du déplacement de `k` disques de la tige `a` vers la tige `b` en se servant de la tige `c` comme stockage intermédiaire.

2. Dans le problème des tours de Hanoï, combien faut-il de déplacements élémentaires au total pour déplacer les `n` disques de la tour de départ à la tour d'arrivée ?

3. Utiliser `tkinter` pour réaliser une version graphique montrant la solution du jeu des tours de Hanoï pour `n` tours.