

Chapitre 3 : Algorithmes de tri



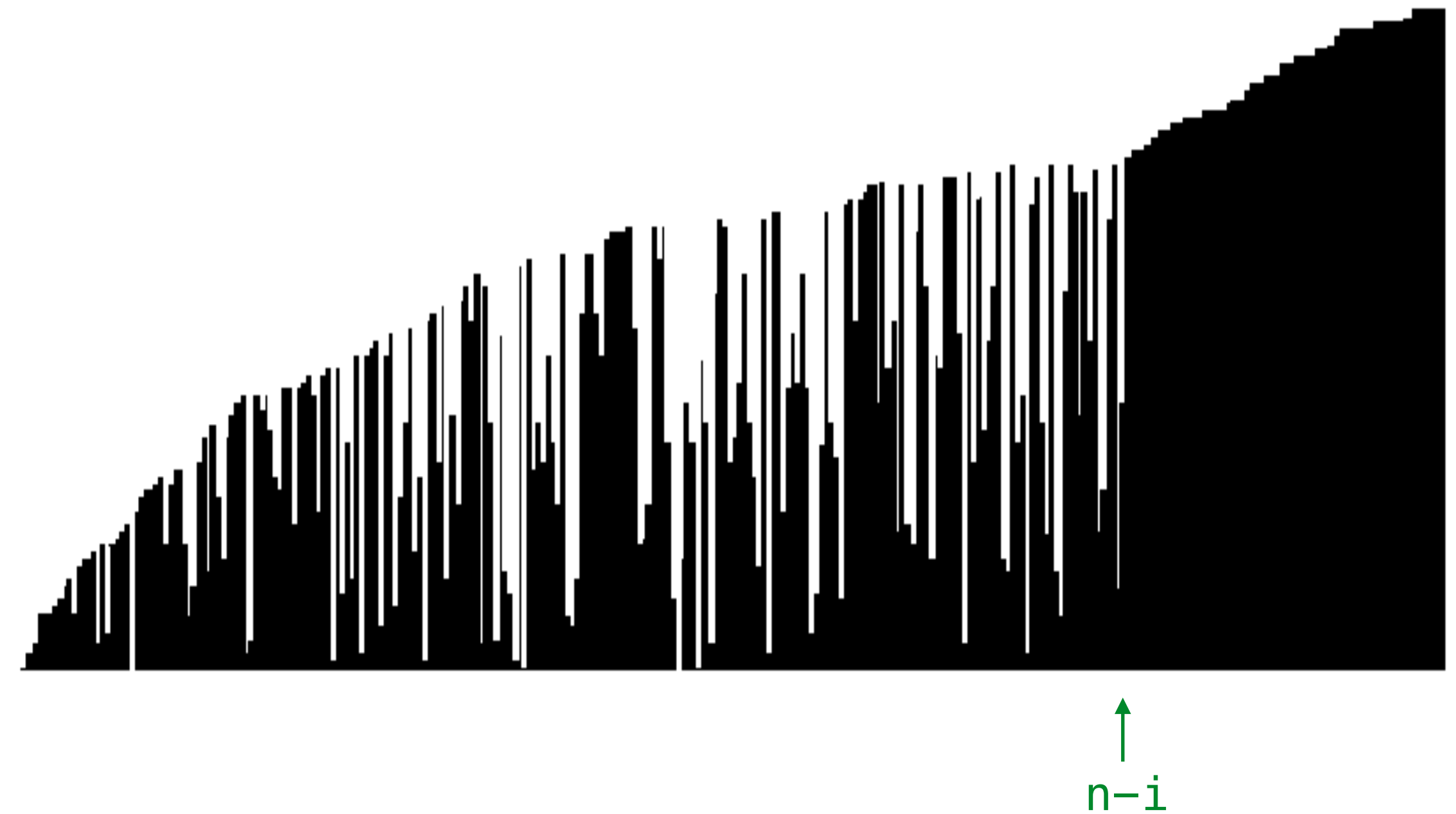
3.1. Tris simples (Rappels)



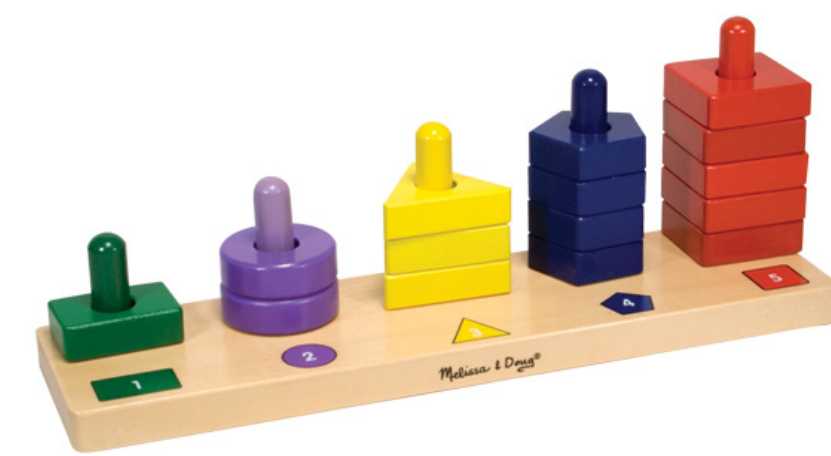
Tri à bulles



```
fonction BubbleSort( $A, n$ )  
    (tableau A de n éléments)  
    pour  $i$  de 1 à  $n-1$  boucler  
        pour  $j$  de 1 à  $n-i$  boucler  
            si  $A(j+1) < A(j)$ , alors  
                permuter  $A(j)$  et  $A(j+1)$   
            fin si  
        fin pour  $j$   
    fin pour  $i$ 
```

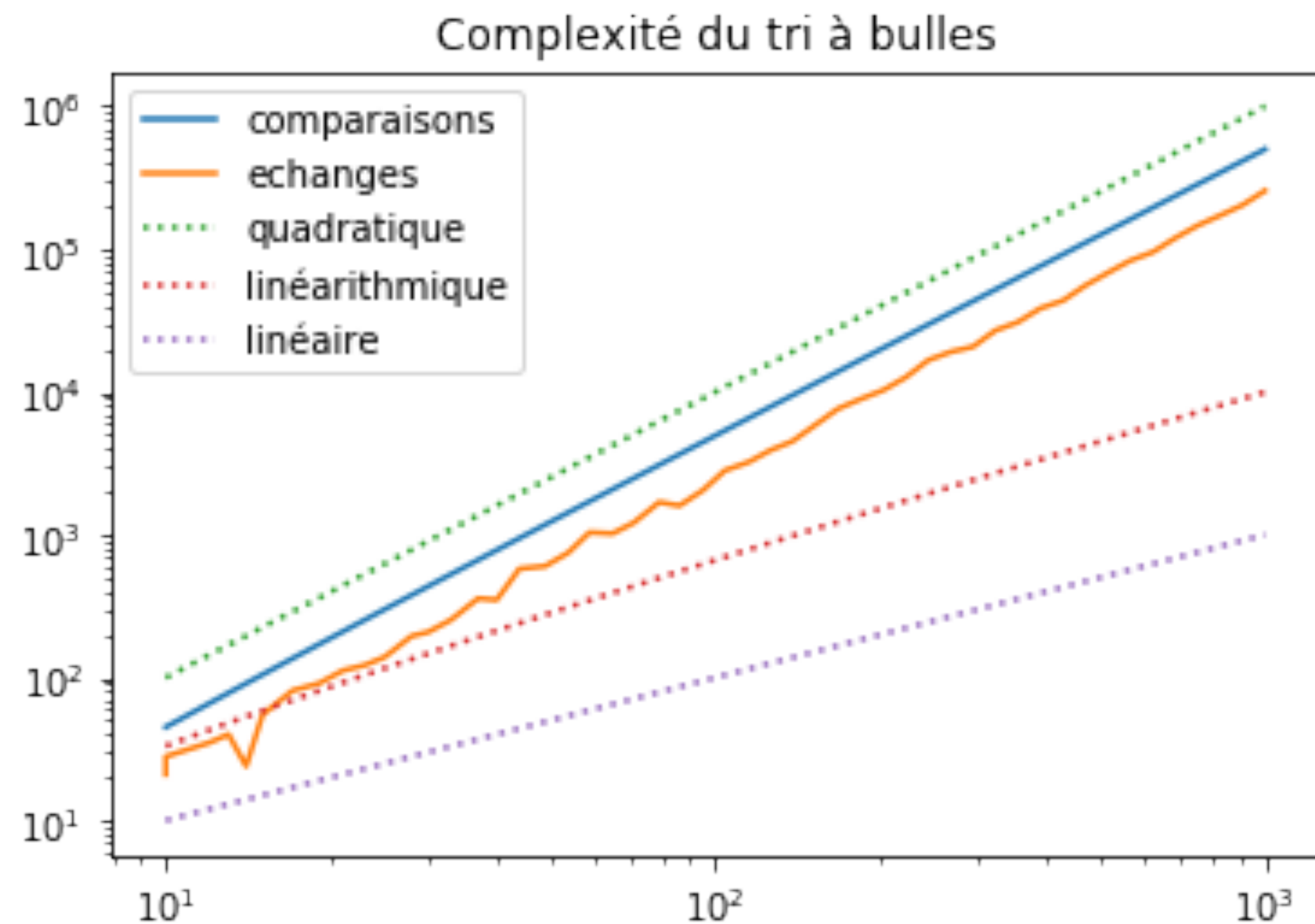


Tri à bulles

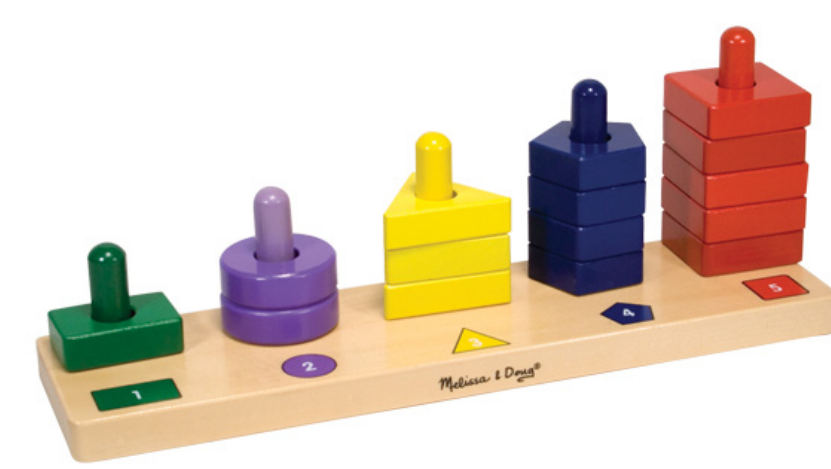


pour i de 1 à $n-1$ boucler
pour j de 1 à $n-i$ boucler

- $O(n^2)$ comparaisons
- De 0 à $O(n^2)$ échanges selon l'ordre des éléments à trier



N	Comp .	Ech .
10	45	21
19	171	91
37	666	358
71	2485	1212
138	9453	4502
268	35778	19041
517	133386	67990
1000	499500	256151



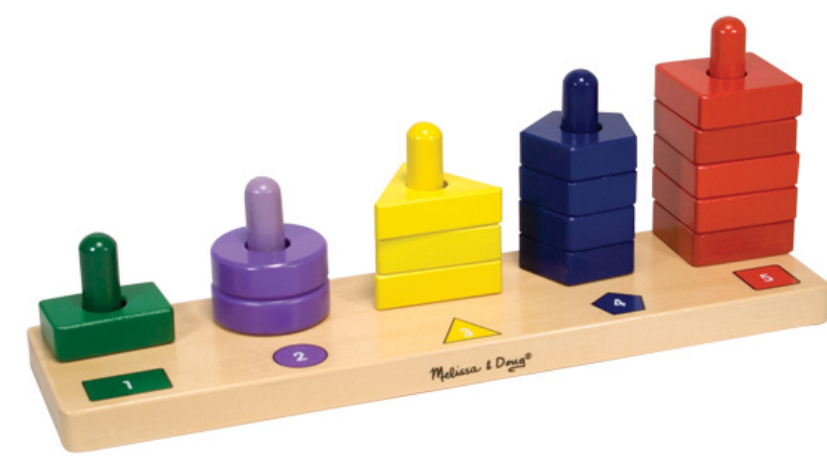
- Que se passe-t-il quand 2 éléments du tableau sont équivalents du point de vue de critère de tri ?

Ordre alphabétique des prénoms

Caron Alain (27 ans)
Aubert Alexandre (43 ans)
Bonnet Anne (27 ans)
Aubert Beatrice (22 ans)
Bonnet Benoit (21 ans)
Caron Brigitte (22 ans)
Aubert Carole (27 ans)
Caron Catherine (4 ans)
Bonnet Christine (22 ans)
Aubert Denis (33 ans)

Ordre alphabétique des noms

Aubert Alexandre (43 ans)
Aubert Beatrice (22 ans)
Aubert Carole (27 ans)
Aubert Denis (33 ans)
Bonnet Anne (27 ans)
Bonnet Benoit (21 ans)
Bonnet Christine (22 ans)
Caron Alain (27 ans)
Caron Brigitte (22 ans)
Caron Catherine (4 ans)



- Personnes dont on connaît nom, prénom, âge, ...

```
struct Personne { string nom; string prenom; unsigned age; };
```

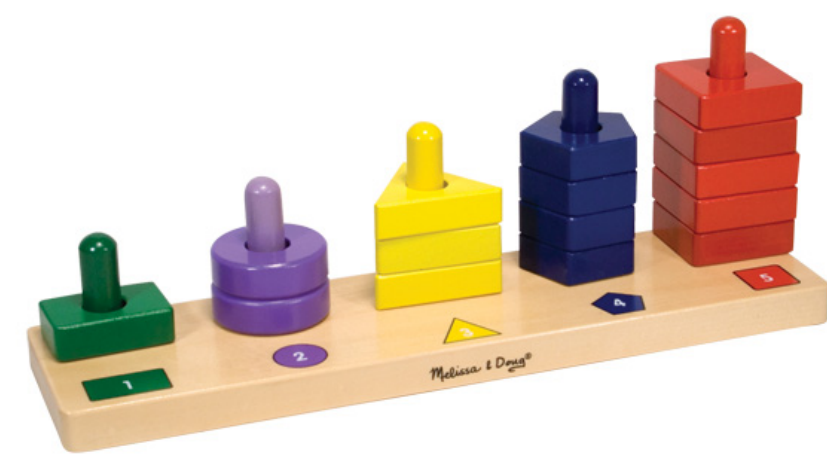
```
ostream& operator<<(ostream& o, Personne const& p) {  
    return o << p.prenom << " " << p.nom << " (" << p.age << ")";  
}
```

- Triables selon différents critères

```
bool comparer_noms (const Personne& p, const Personne& q) {  
    return p.nom < q.nom;  
}
```

```
bool comparer_prenoms (const Personne& p, const Personne& q) {  
    return p.prenom < q.prenom;  
}
```

Tri à bulles générique



- Fonction de comparaison reçue en paramètre
- L'appel à cette fonction remplace operator<

```
template<class T, class Compare>
void bubbleSort(vector<T>& v, Compare compare) {

    for(size_t i = 0; i < v.size() - 1; ++i) {
        for(size_t j = 1; j < v.size() - i; ++j) {
            if( compare(v[j], v[j-1]) )
                swap(v[j], v[j-1]);
        }
    }
}
```

Code appellant



```
int main() {
    vector<Personne> gens =
        { {"Aubert", "Beatrice", 22},
          {"Caron", "Alain", 27},
          {"Bonnet", "Christine", 22},
          {"Bonnet", "Anne", 27},
          {"Aubert", "Alexandre", 43},
          {"Caron", "Catherine", 4},
          {"Bonnet", "Benoit", 21},
          {"Aubert", "Denis", 33},
          {"Aubert", "Carole", 27},
          {"Caron", "Brigitte", 22}};

    bubbleSort(gens, comparer_prenoms );
    cout << gens << endl;

    bubbleSort(gens, comparer_noms );
    cout << gens << endl;
}
```

Caron Alain (27 ans)
Aubert Alexandre (43 ans)
Bonnet Anne (27 ans)
Aubert Beatrice (22 ans)
Bonnet Benoit (21 ans)
Caron Brigitte (22 ans)
Aubert Carole (27 ans)

Caron Cath
Bonnet Chr
Aubert Den

Aubert Alexandre (43 ans)
Aubert Beatrice (22 ans)
Aubert Carole (27 ans)
Aubert Denis (33 ans)
Bonnet Anne (27 ans)
Bonnet Benoit (21 ans)
Bonnet Christine (22 ans)
Caron Alain (27 ans)
Caron Brigitte (22 ans)
Caron Catherine (4 ans)

Attention !

< ou <= ?



```
bool comparer_noms_lt (const Personne& p,
                      const Personne& q) {
    return p.nom < q.nom;
}
```

```
bool comparer_noms_le (const Personne& p,
                      const Personne& q) {
    return p.nom <= q.nom;
}
```

p.nom < q.nom;

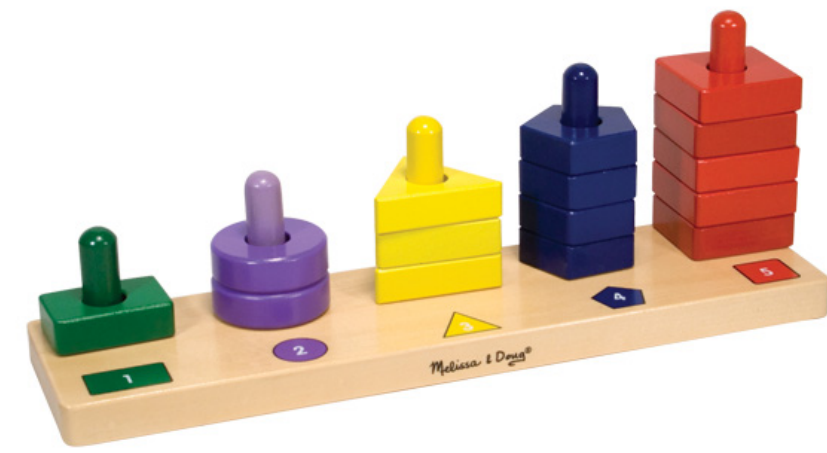
Aubert Alexandre (43 ans)
Aubert Beatrice (22 ans)
Aubert Carole (27 ans)
Aubert Denis (33 ans)
Bonnet Anne (27 ans)
Bonnet Benoit (21 ans)
Bonnet Christine (22 ans)
Caron Alain (27 ans)
Caron Brigitte (22 ans)
Caron Catherine (4 ans)

p.nom <= q.nom;

Aubert Denis (33 ans)
Aubert Beatrice (22 ans)
Aubert Alexandre (43 ans)
Aubert Carole (27 ans)
Bonnet Christine (22 ans)
Bonnet Benoit (21 ans)
Bonnet Anne (27 ans)
Caron Catherine (4 ans)
Caron Brigitte (22 ans)
Caron Alain (27 ans)

- Le tri à bulle n'est **stable** que s'il **n'échange pas les éléments égaux**

Tri par sélection



```
fonction SelectionSort(A,n)
```

```
  pour i de 1 à n-1 boucler
```

```
    imin ← i
```

```
    pour j de i+1 à n boucler
```

```
      si  $A(j) < A(imin)$ , alors
```

```
        imin ← j
```

```
      fin si
```

```
    fin pour j
```

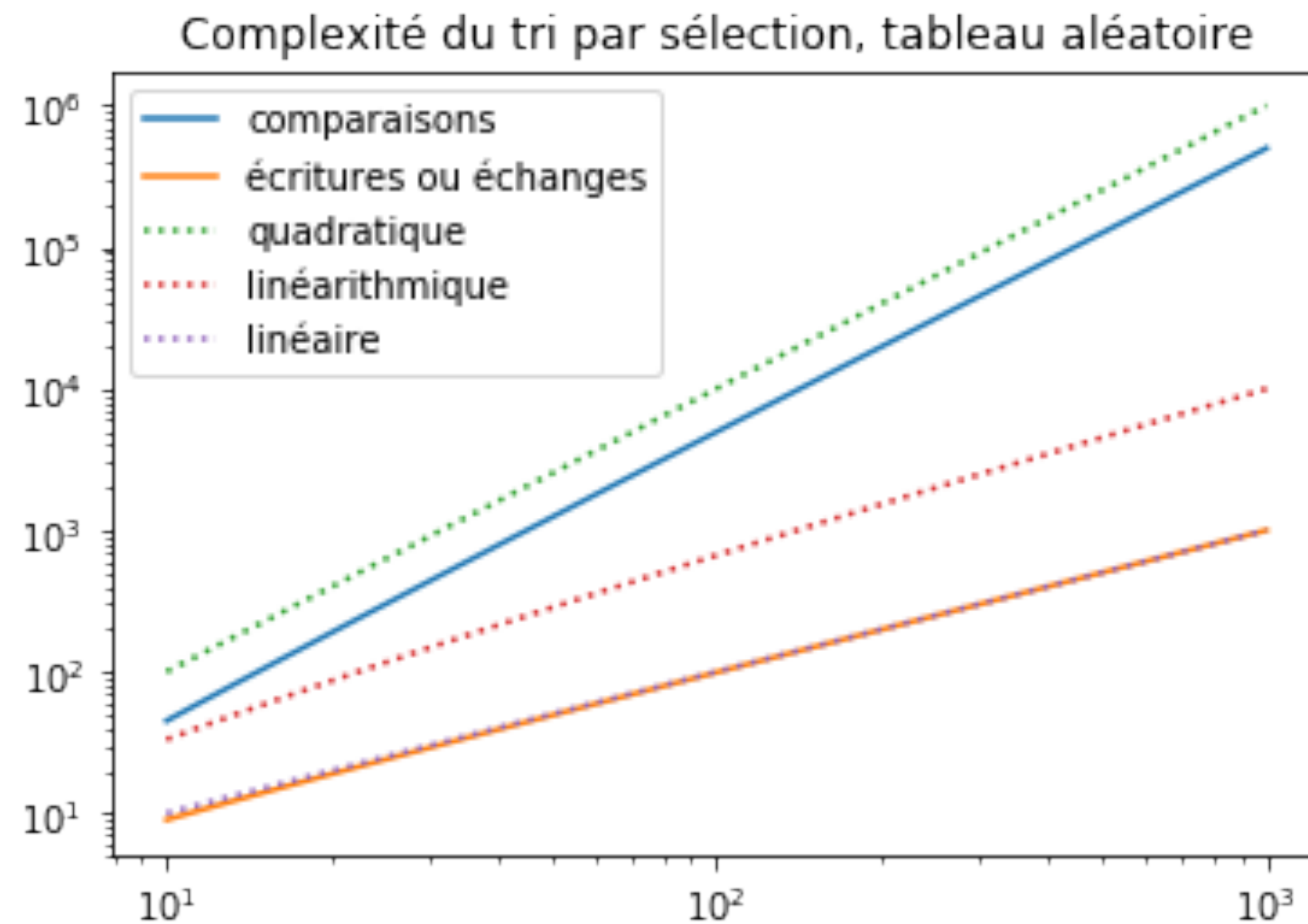
```
    permuter  $A(i)$  et  $A(imin)$ 
```

```
  fin pour i
```

Tri par sélection



- $O(n^2)$ comparaisons, $O(n)$ échanges, quel que soit l'ordre d'entrée

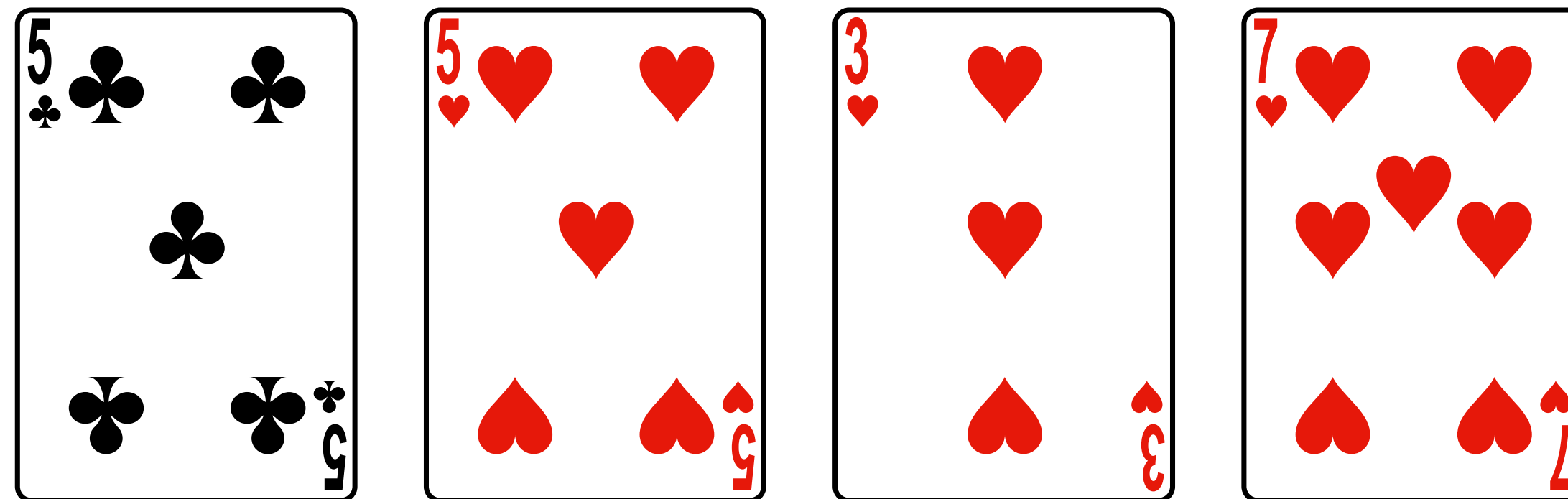


N	Comp.	Ecr.
10	45	9
19	171	18
37	666	36
71	2485	70
138	9453	137
268	35778	267
517	133386	516
1000	499500	999

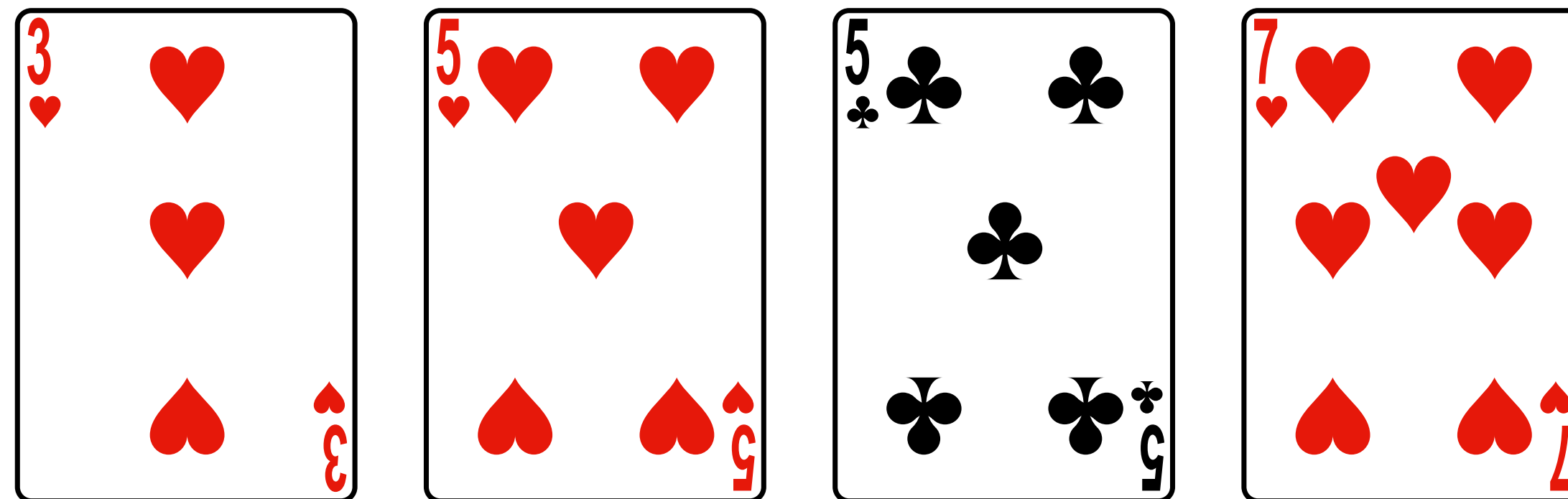
Tri par sélection non stable



- Cartes triées par couleur que l'on veut maintenant trier par valeur

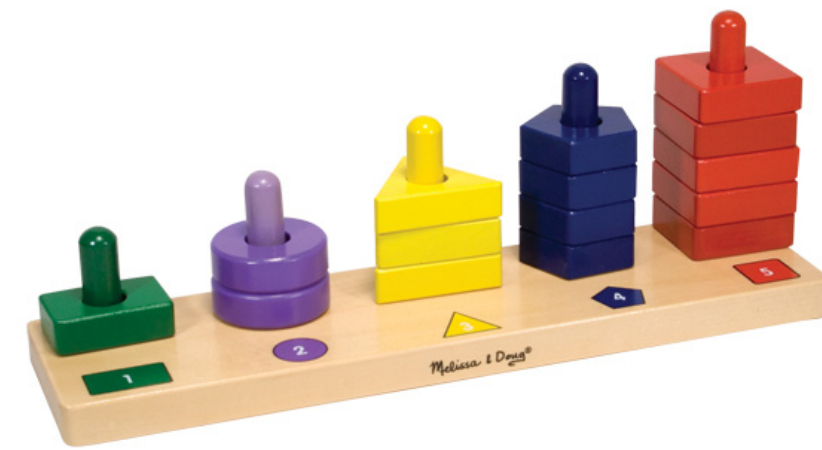


- 1ère étape, on trouve le minimum 3♥ et on échange avec 5♣



- Résultat final. Les « 5 » ne sont plus triés par couleur

Tri par insertion



```
fonction InsertionSort(A, n)
```

```
  pour i de 2 à n boucler
```

```
    tmp ← A(i)
```

```
    j ← i
```

```
    tant que j−1 ≥ 1 et A(j−1) > tmp boucler
```

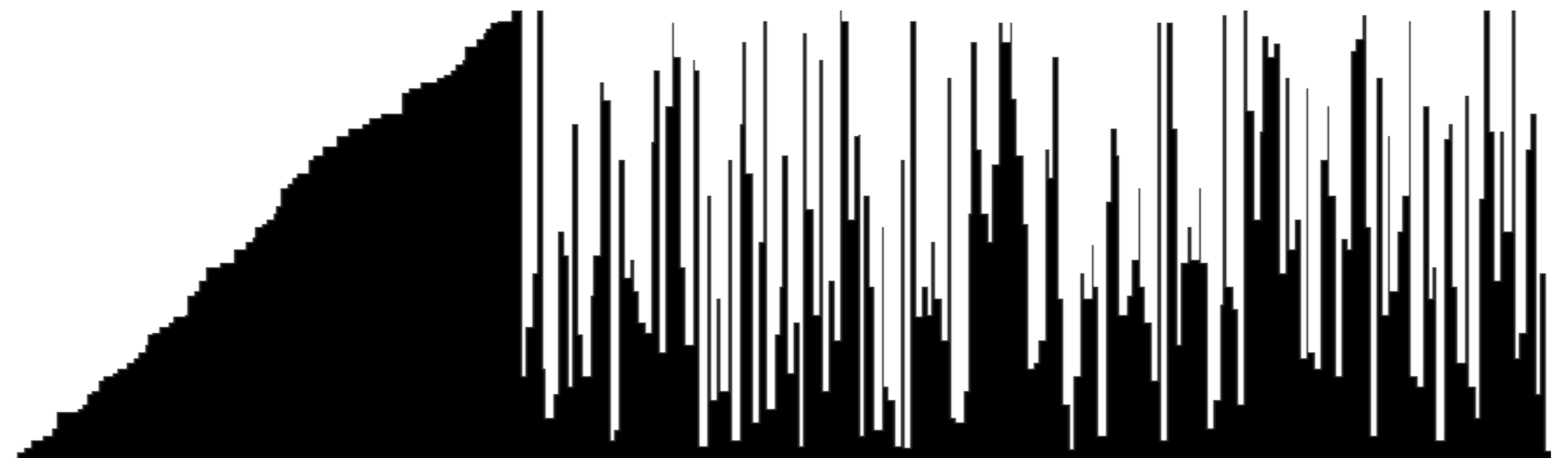
```
      A(j) ← A(j−1)
```

```
      décrémenter j de 1
```

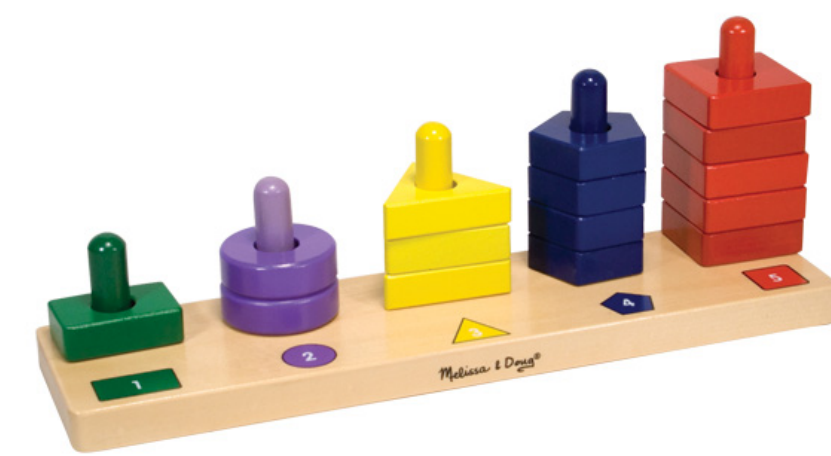
```
    fin tant que
```

```
    A(j) ← tmp
```

```
  fin pour i
```



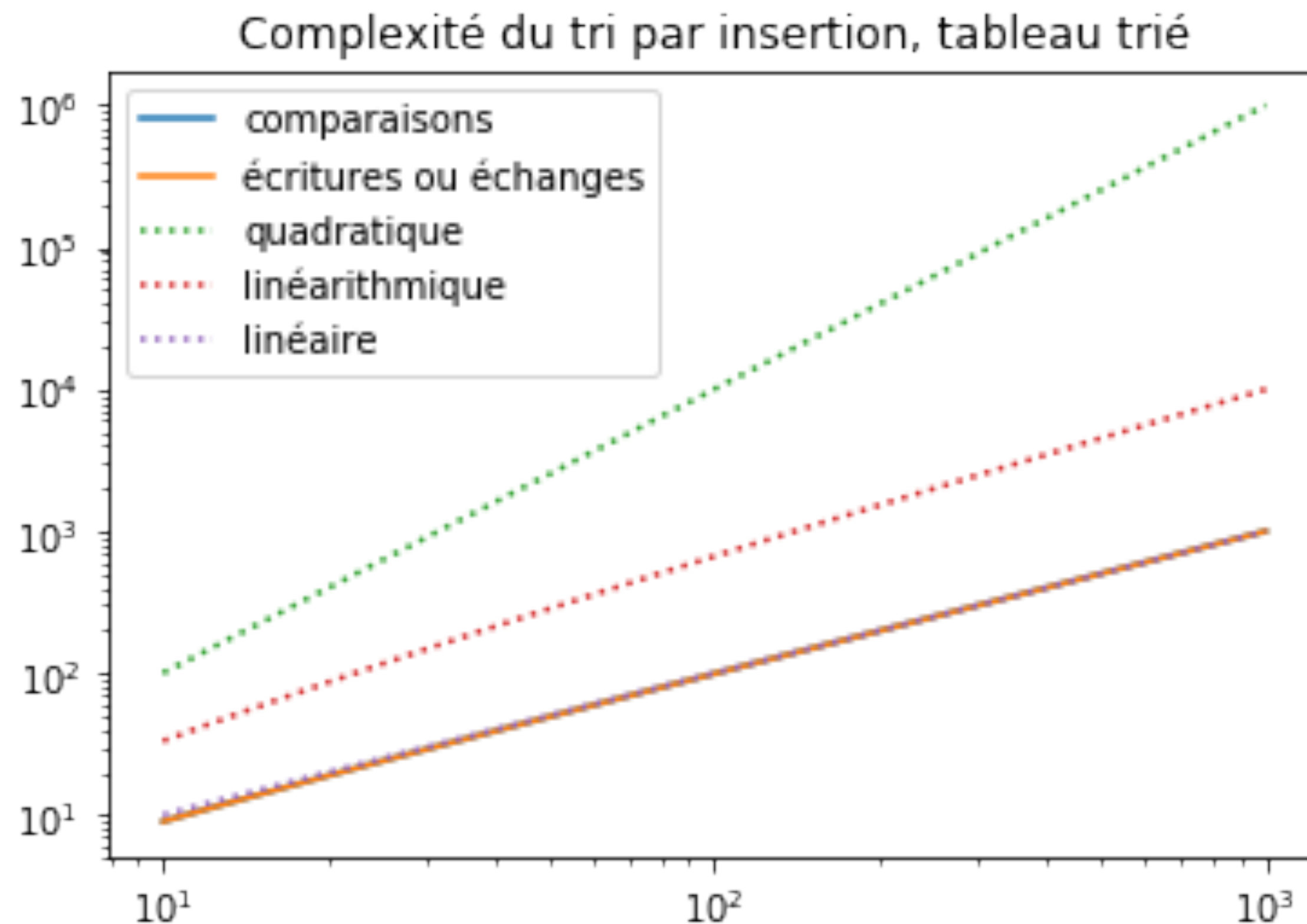
Insertion - Meilleur cas



- Entrée déjà triée



- Complexité linéaire $O(n)$



N	Comp.	Ecr.
10	9	9
19	18	18
37	36	36
71	70	70
138	137	137
268	267	267
517	516	516
1000	999	999

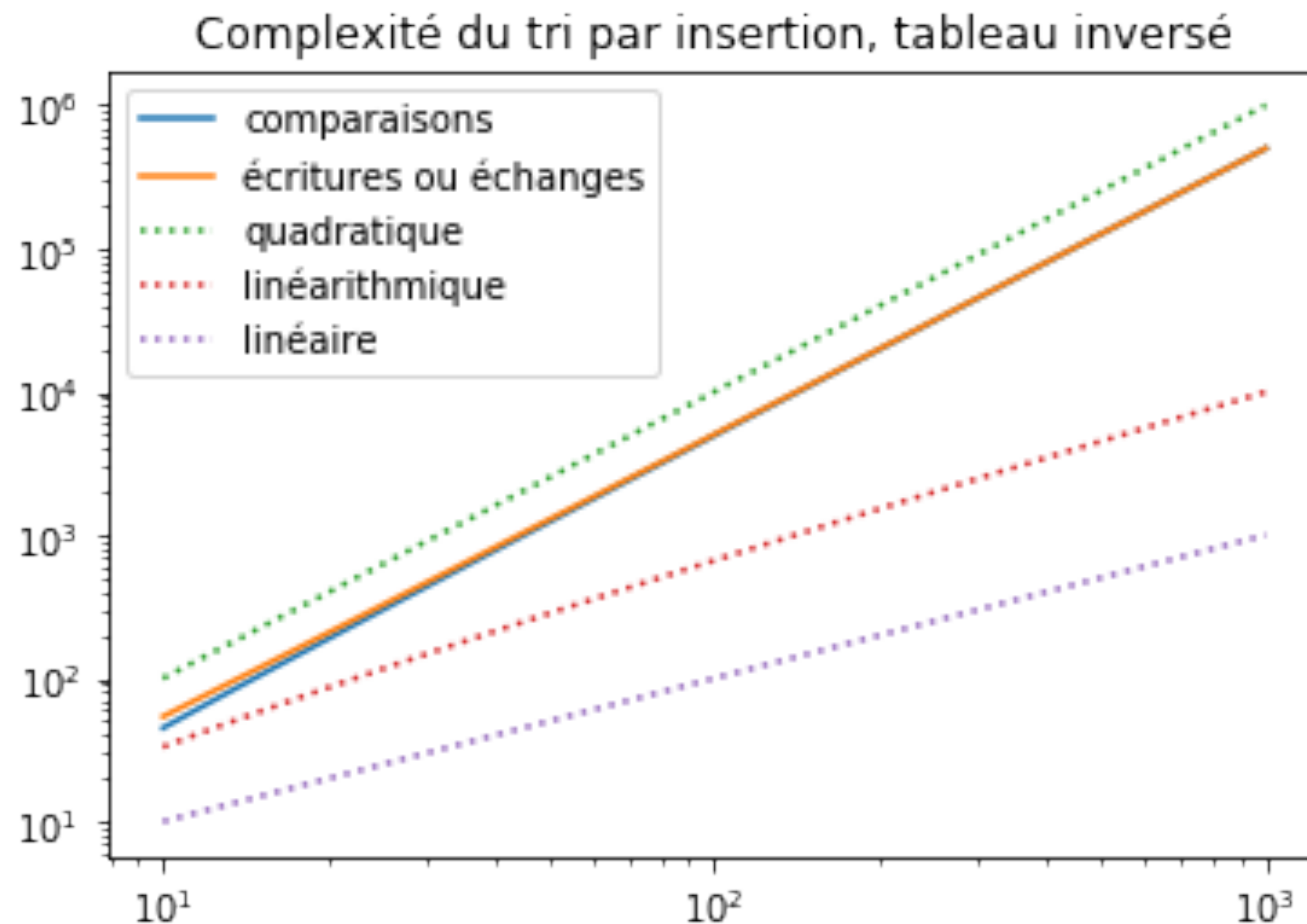
Insertion - pire cas



- Entrée triée à l'envers



- Complexité quadratique $O(n^2)$

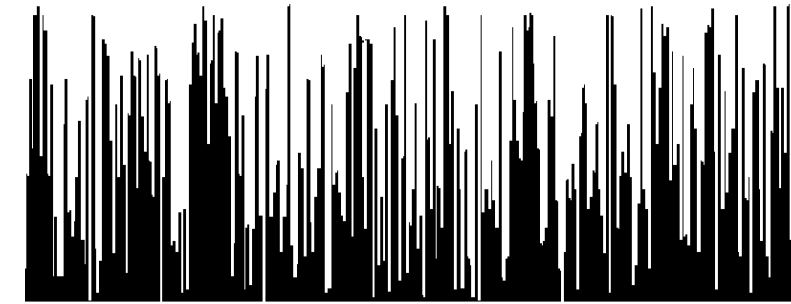


N	Comp.	Ecr.
10	45	54
19	171	189
37	666	702
71	2485	2555
138	9453	9590
268	35778	36045
517	133386	133902
1000	499500	500499

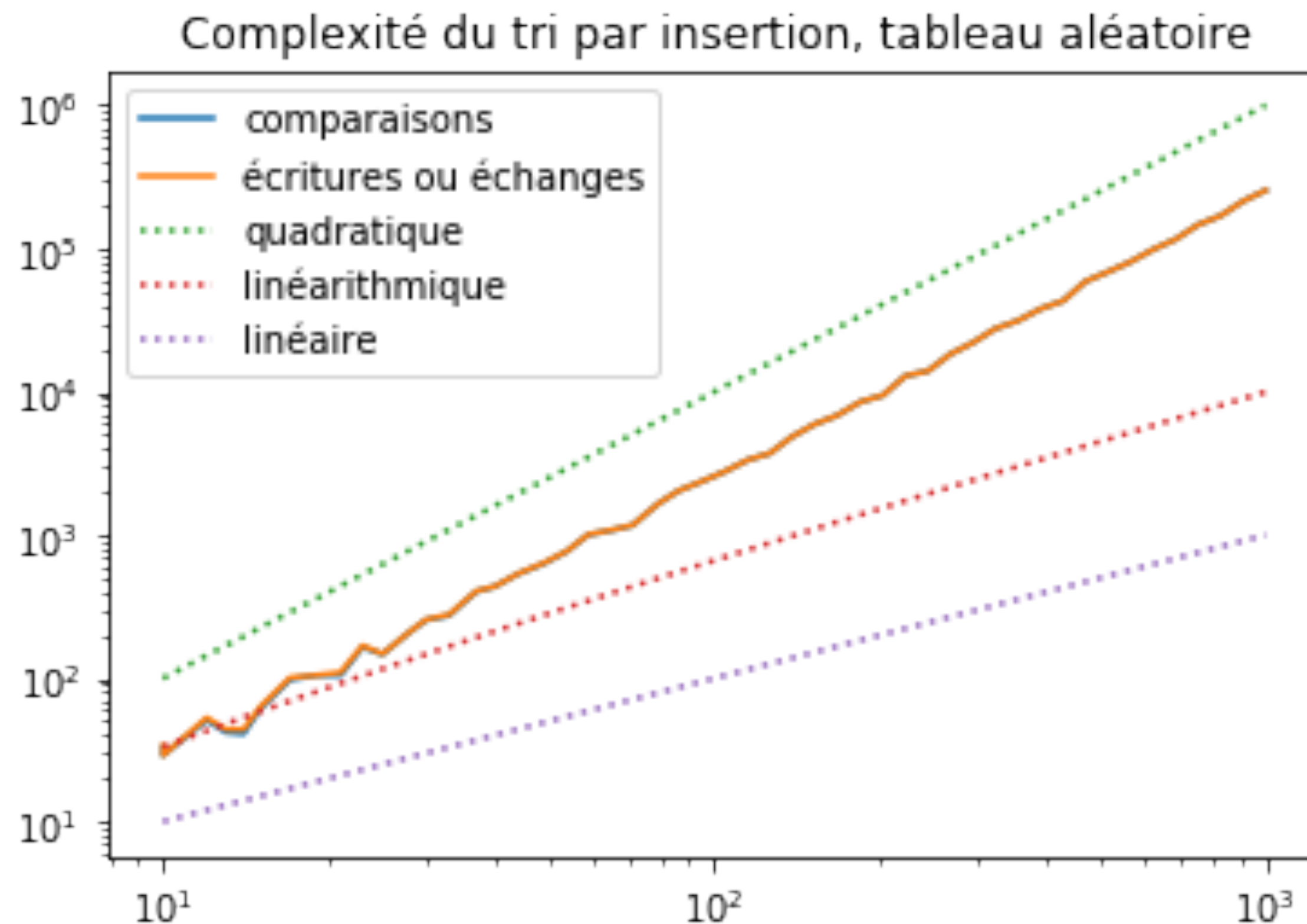
Complexité - cas moyen - $O(n^2)$



- Entrée aléatoire

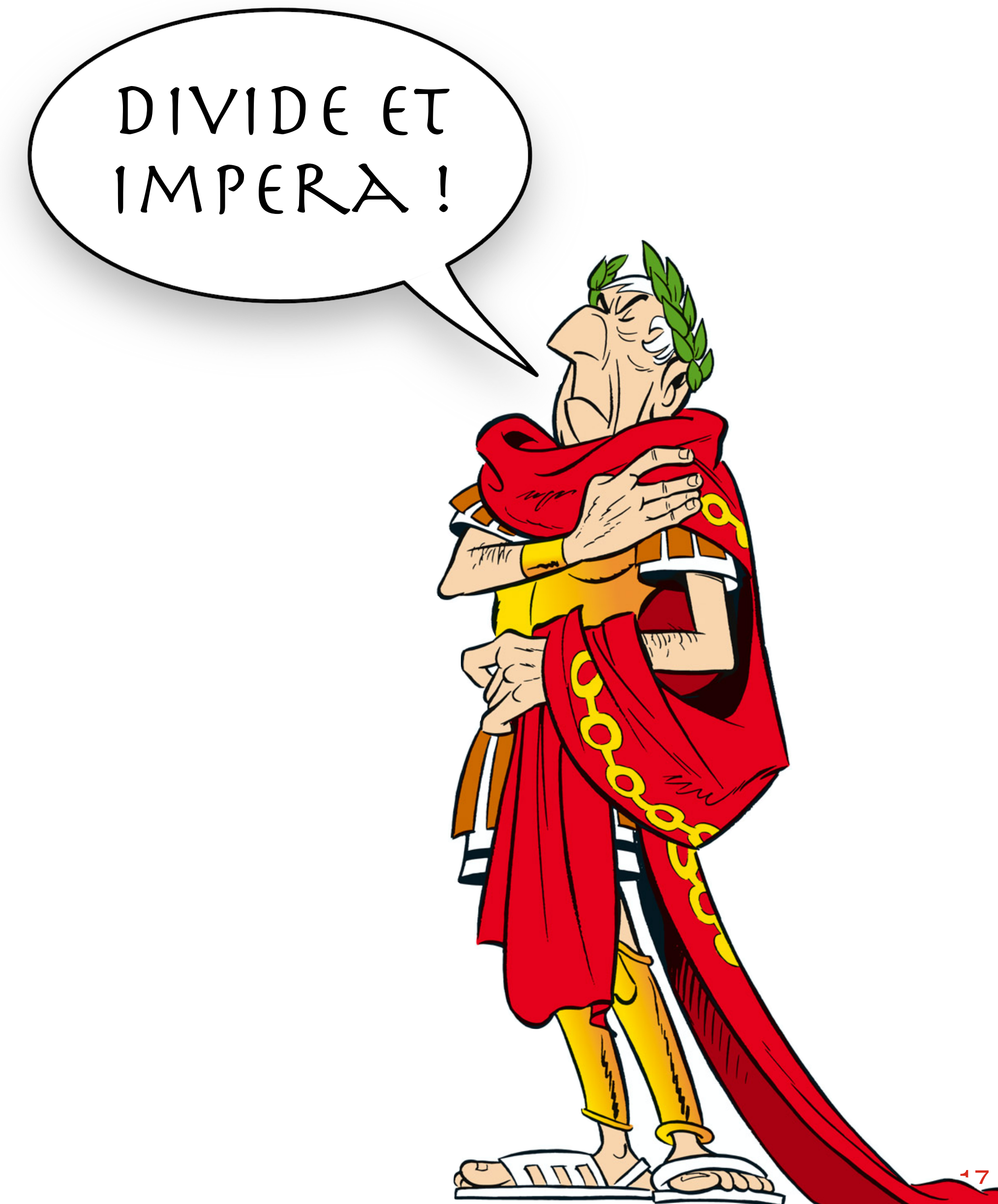


- Complexité quadratique $O(n^2)$



N	Comp.	Ecr.
10	33	34
19	105	105
37	403	406
71	1167	1171
138	4828	4831
268	18327	18333
517	68577	68582
1000	254774	254779

3.2. Tri par fusion



Tri par fusion

Approche récursive

- Cas trivial :

Comment trier un tableau de 1 élément ?

- Cas général :

Comment trier un tableau, en sachant trier des tableaux plus petits ?

DIVIDE ET
IMPERA !



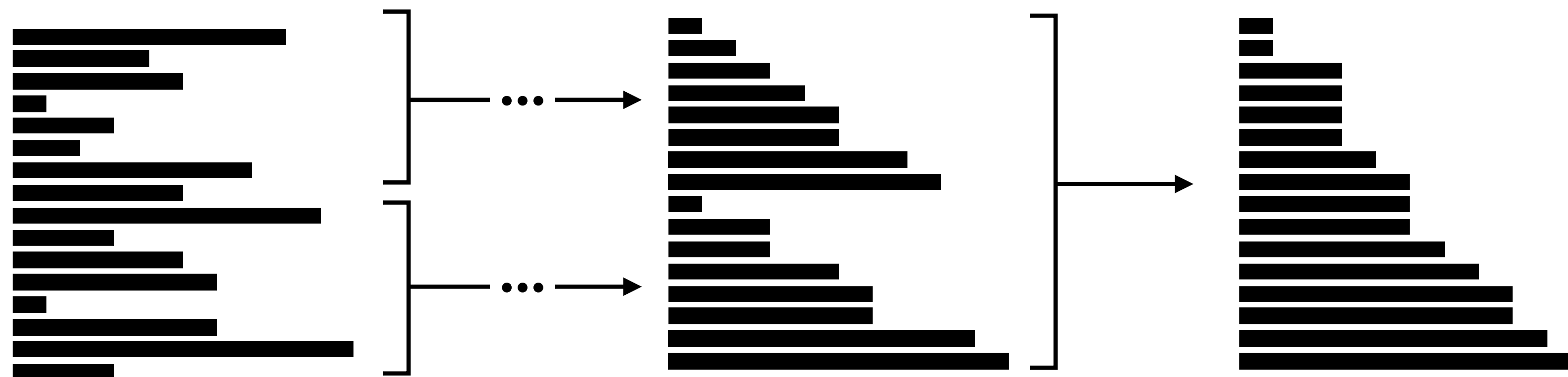
Tri par fusion



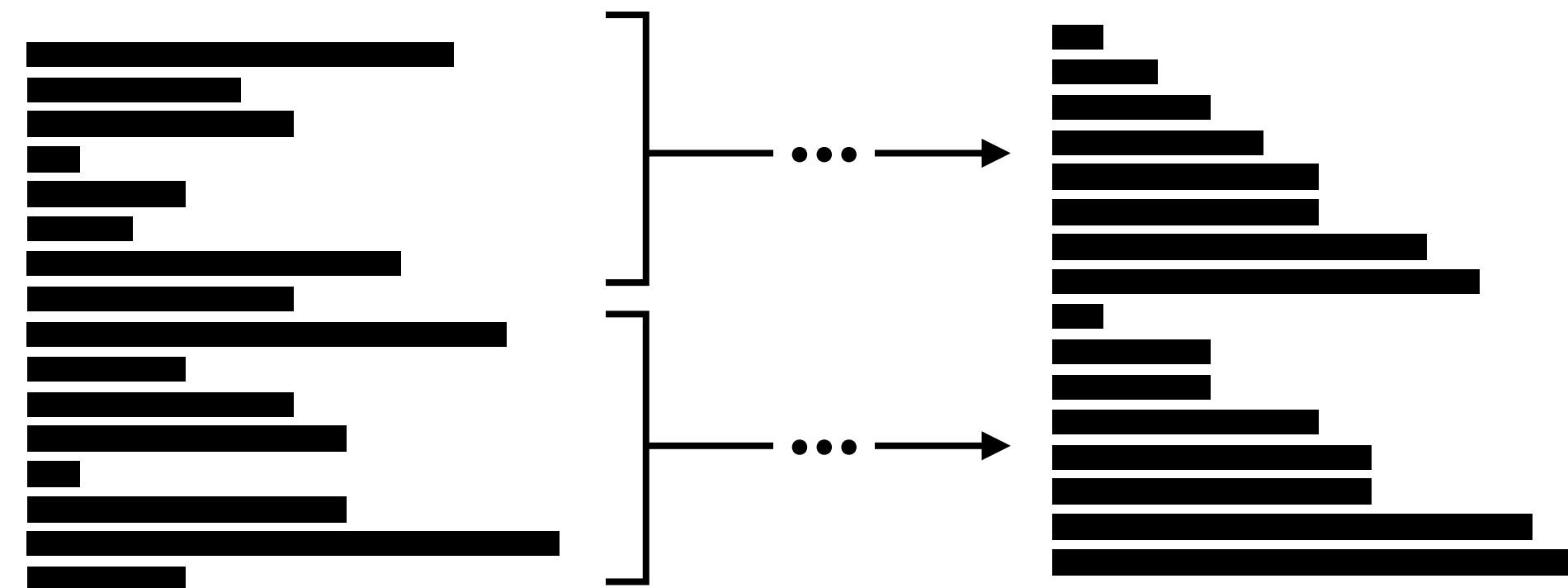
Comment trier un tableau, en sachant trier des tableaux plus petits ?

Si je sais trier des tableaux de taille $n/2$, comment trier un tableau de taille n ?

- Trier chaque moitié du tableau (récursivement)
- Fusionner les deux moitiés du tableau

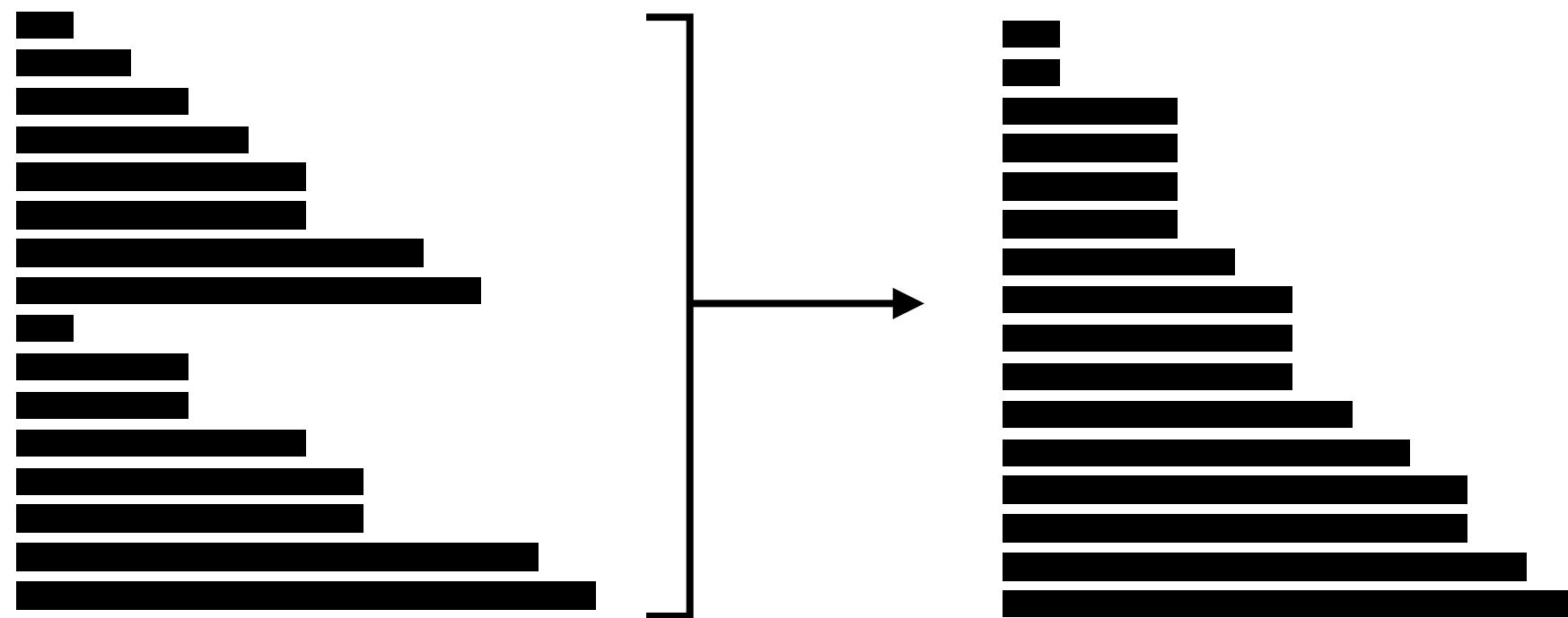


Trier chaque moitié du tableau ?

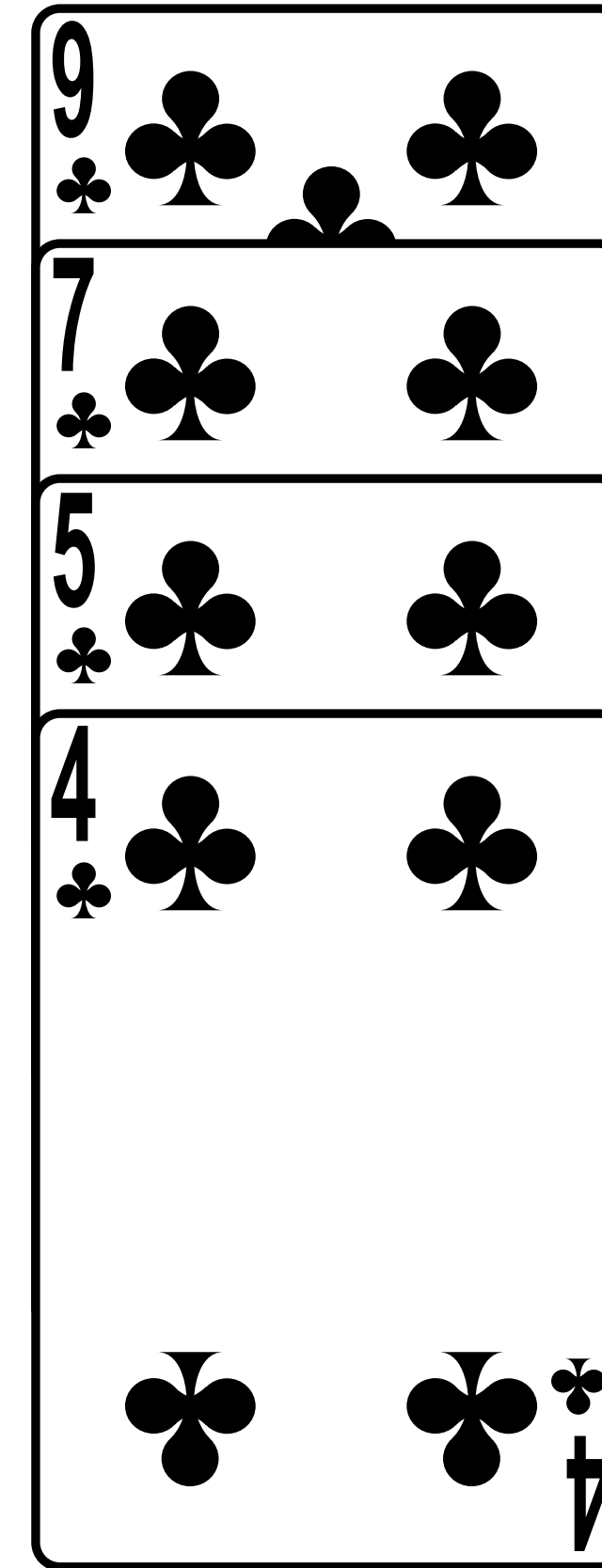
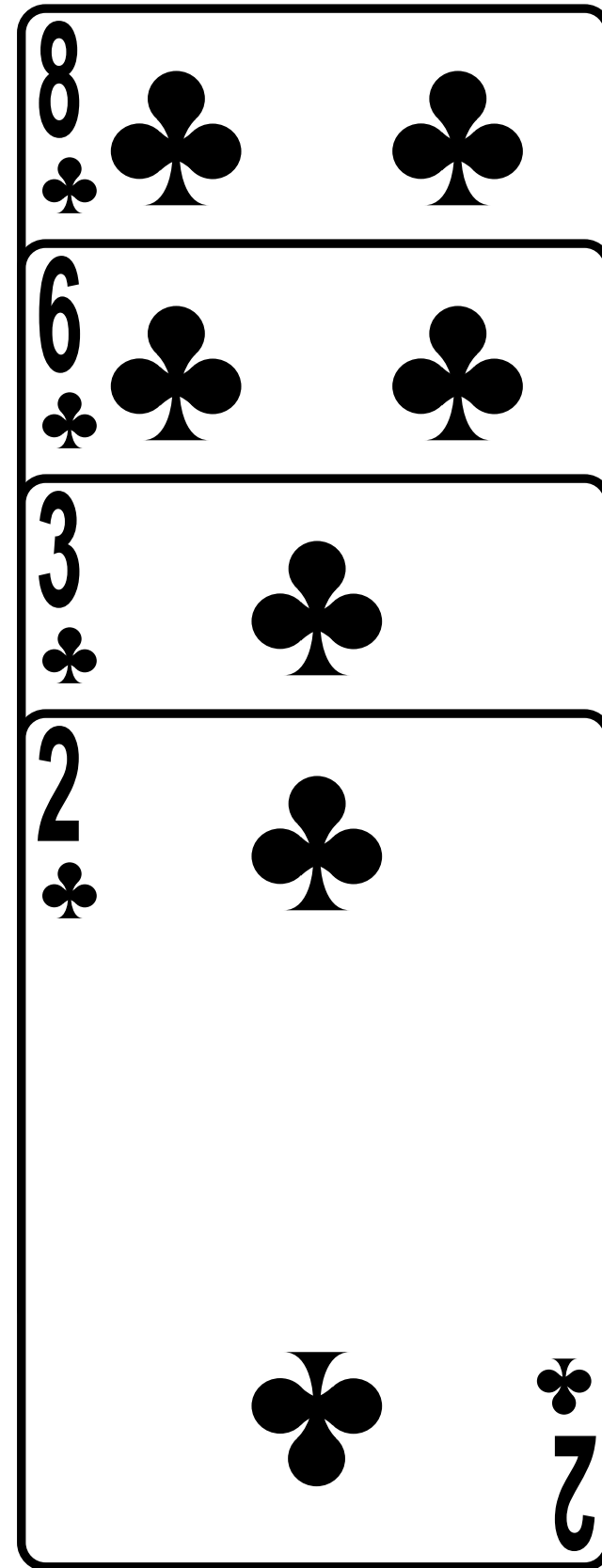


Facile, deux appels récur­sifs !

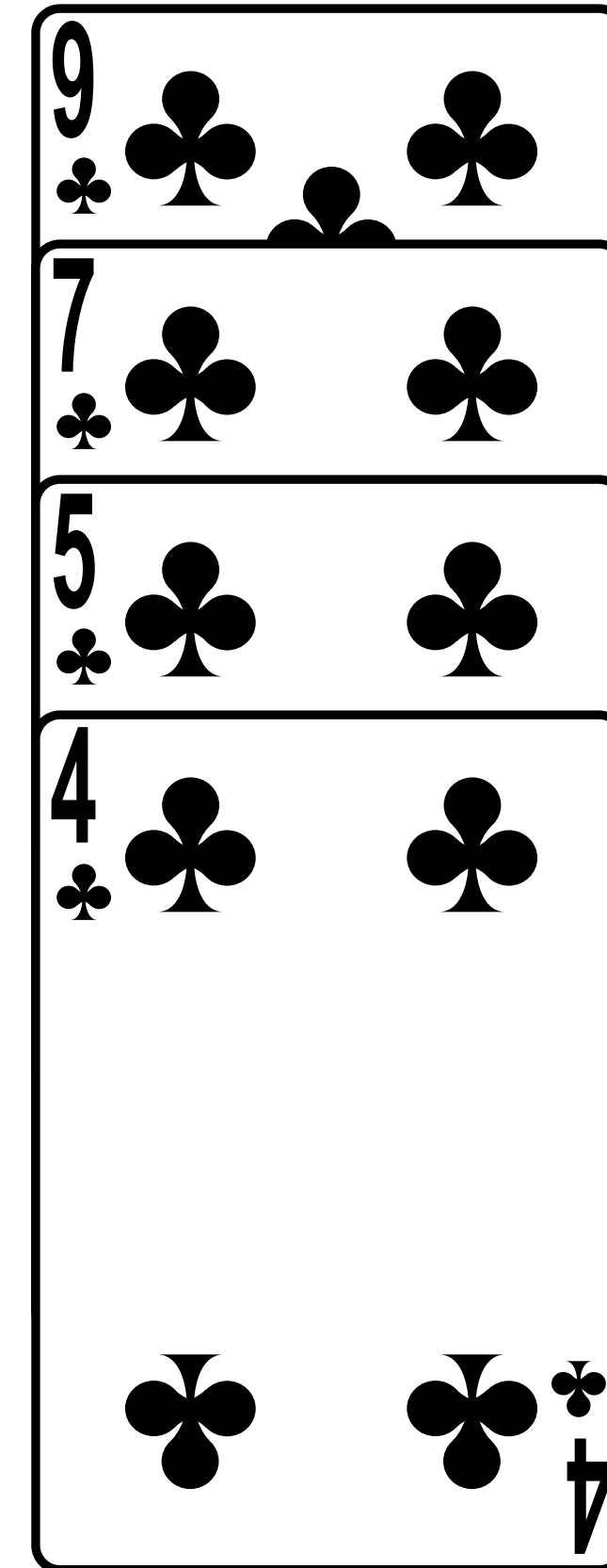
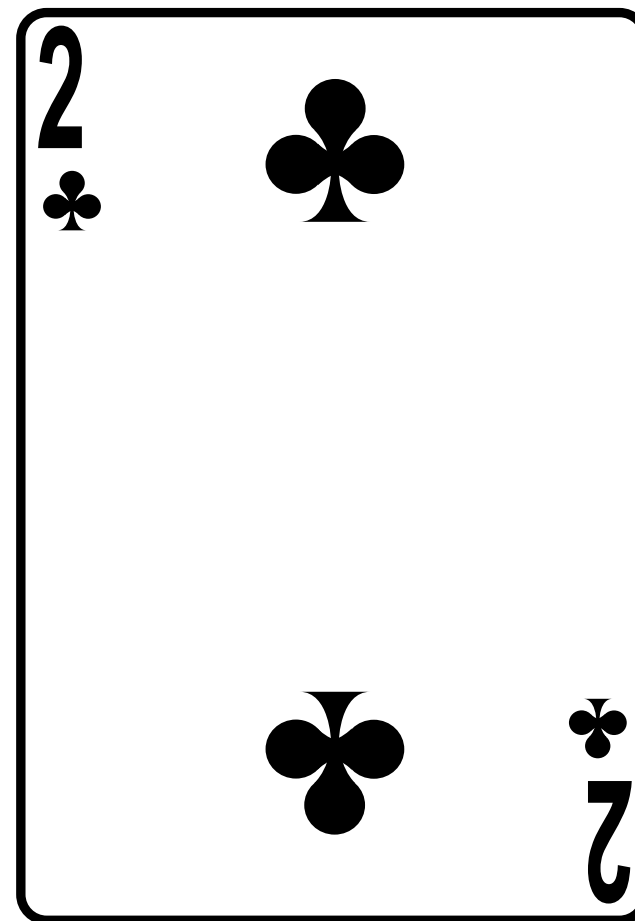
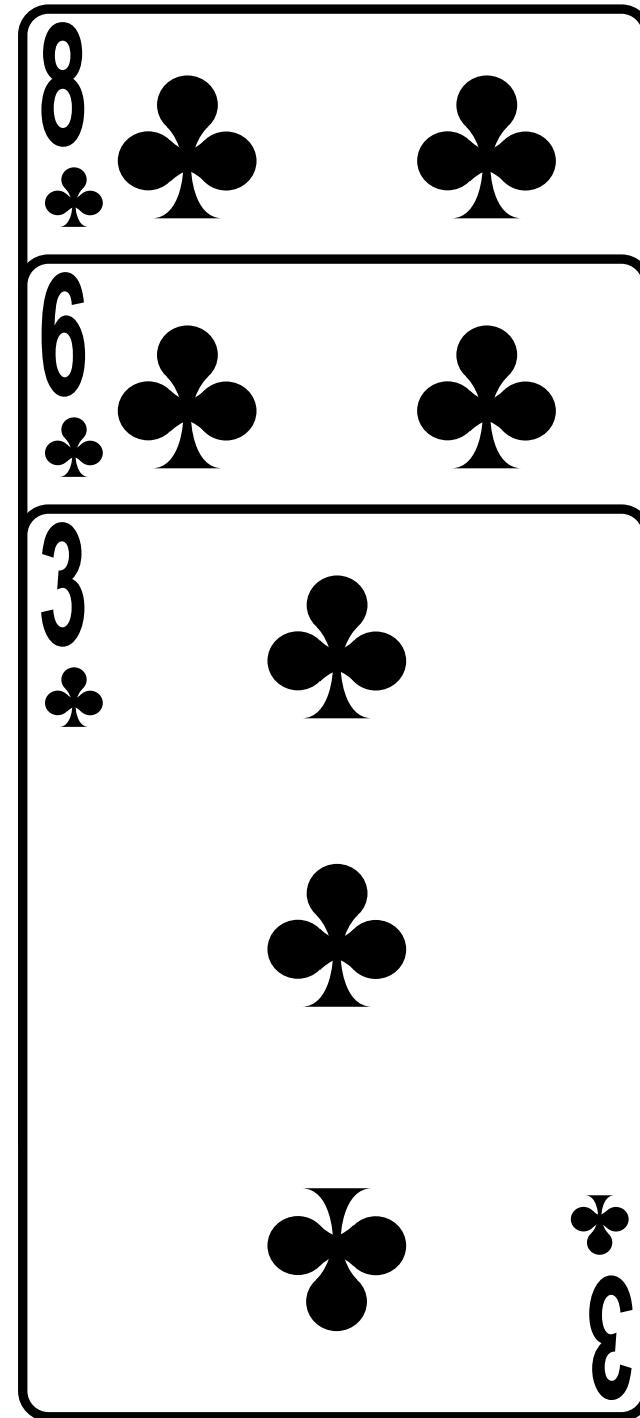
Fusionner les deux moitiés ?



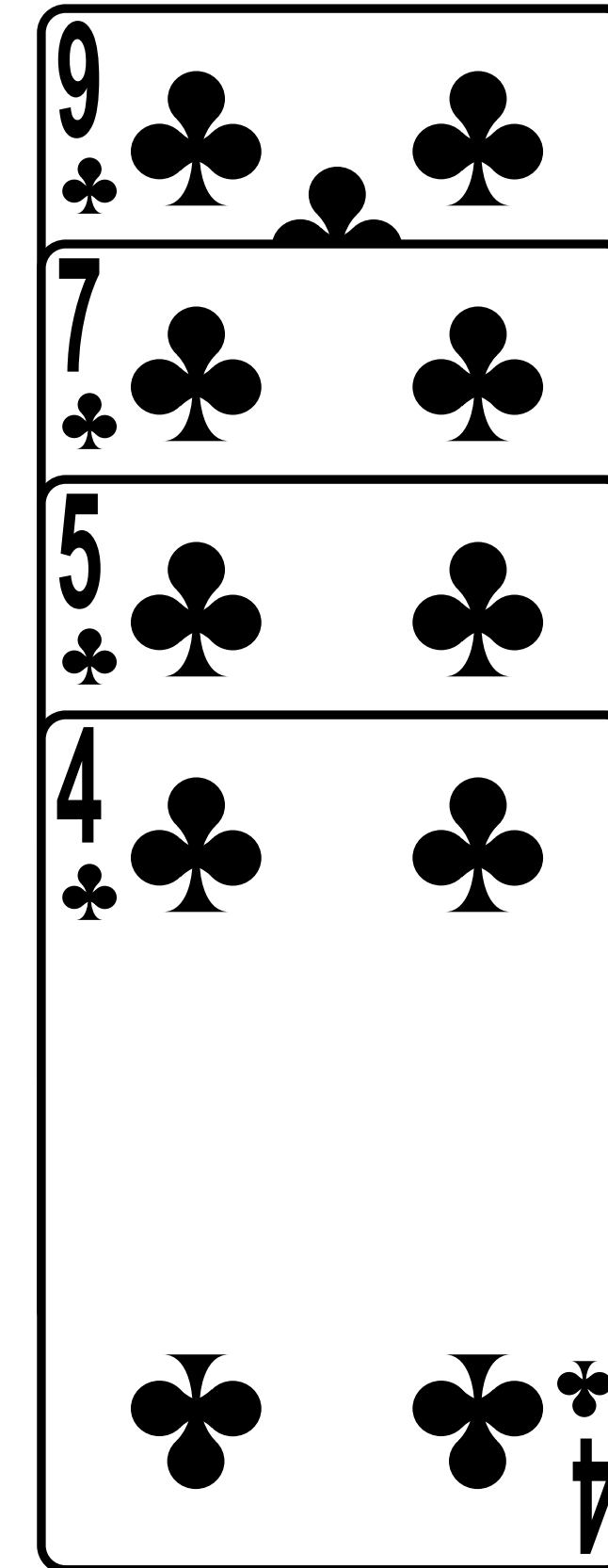
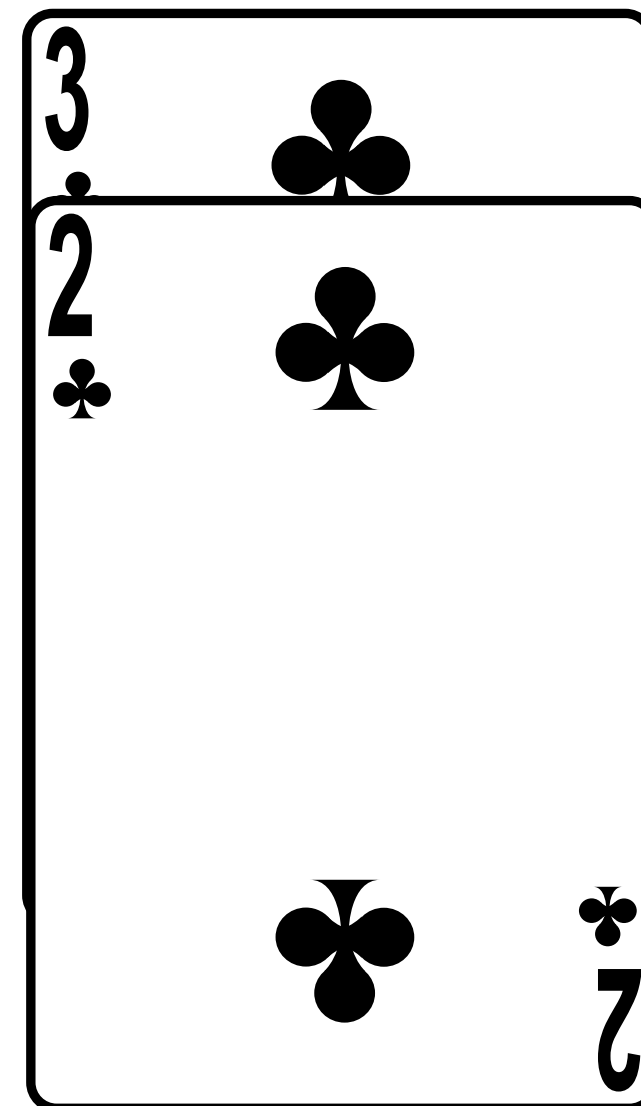
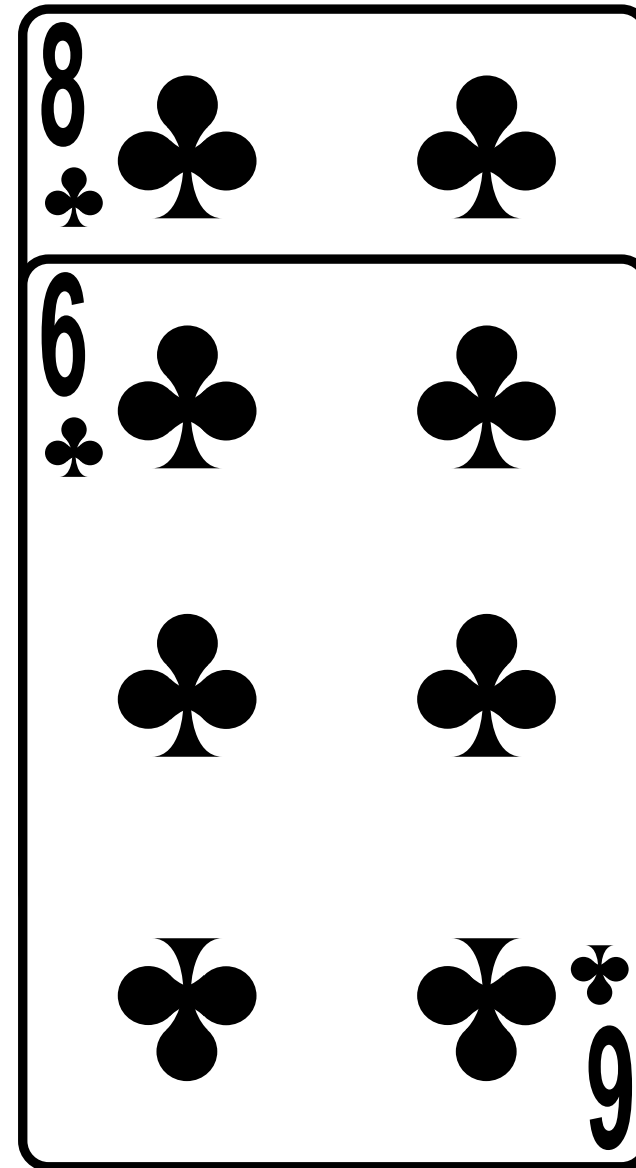
Fusionner deux tableaux triés



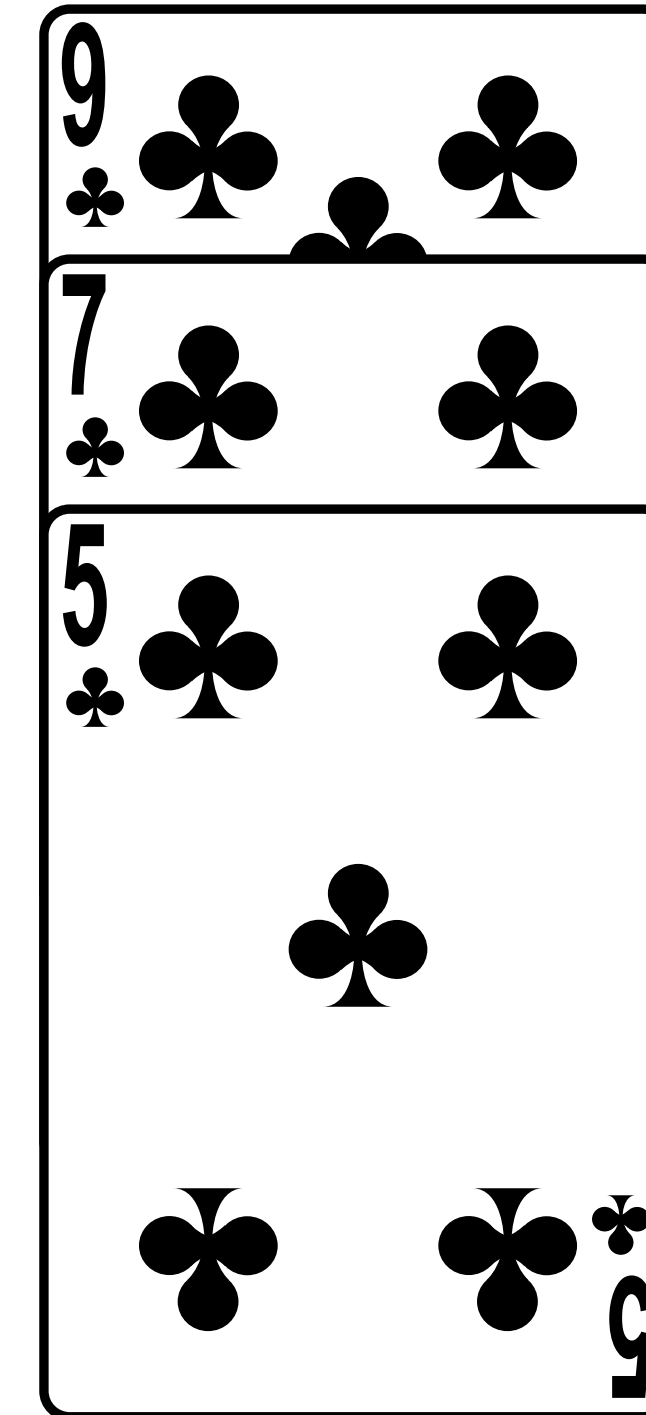
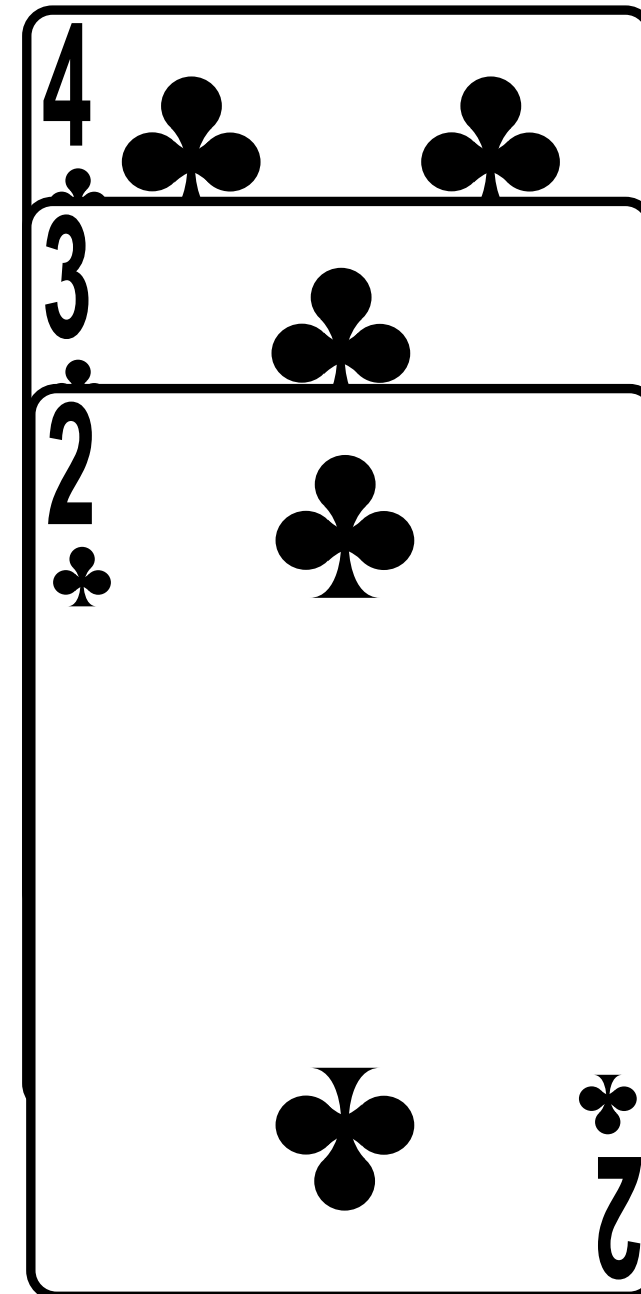
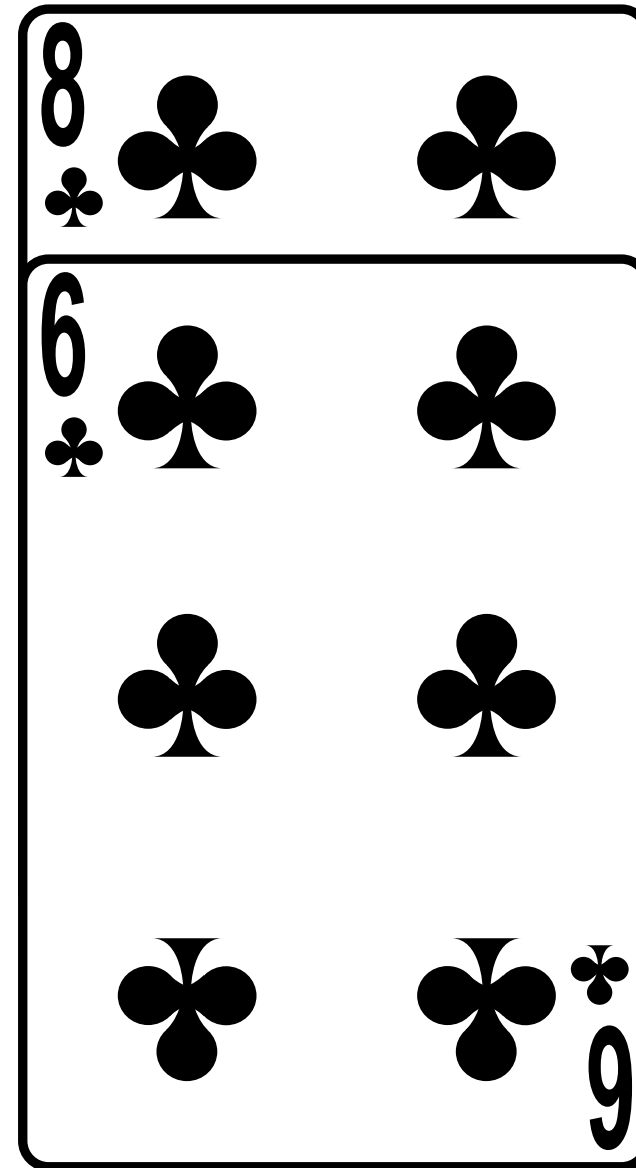
Fusionner deux tableaux triés



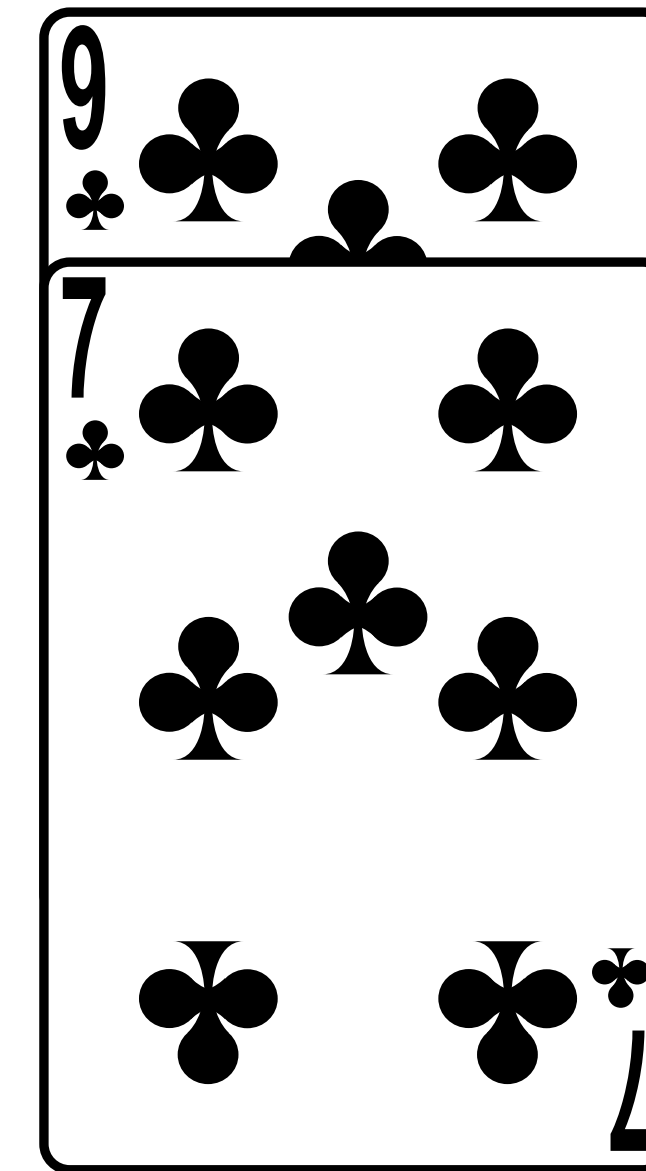
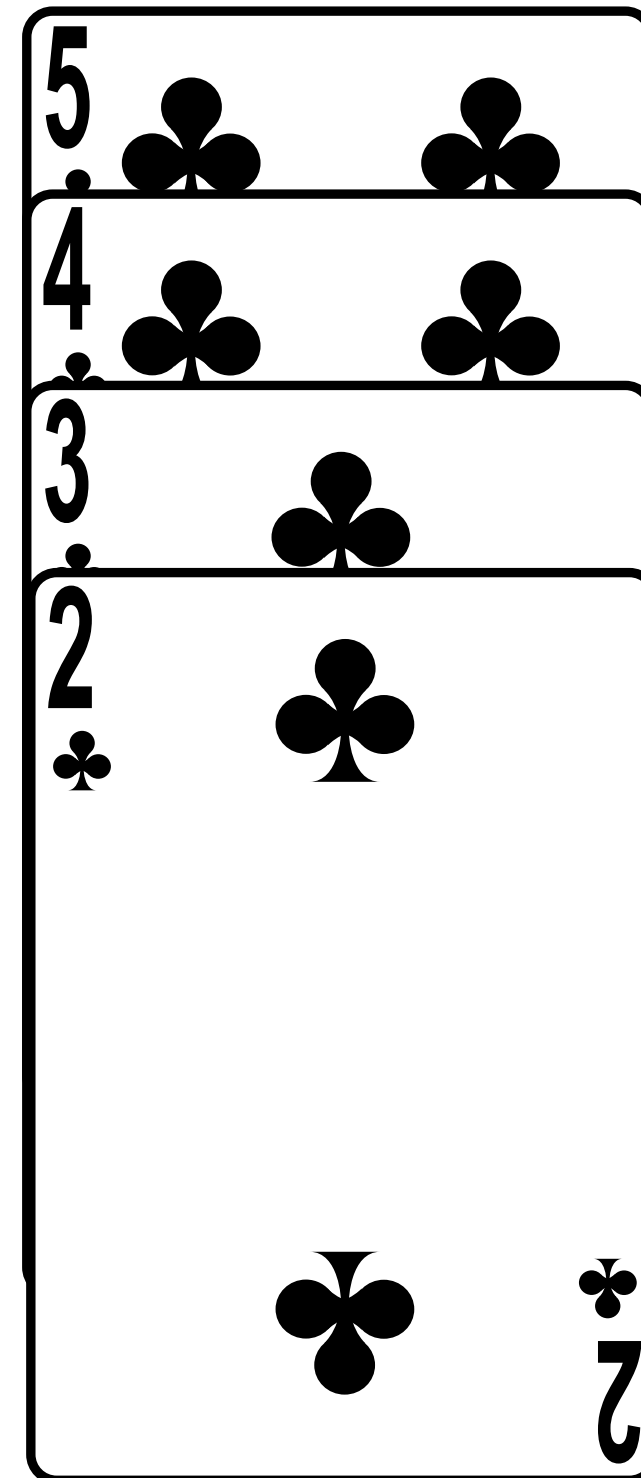
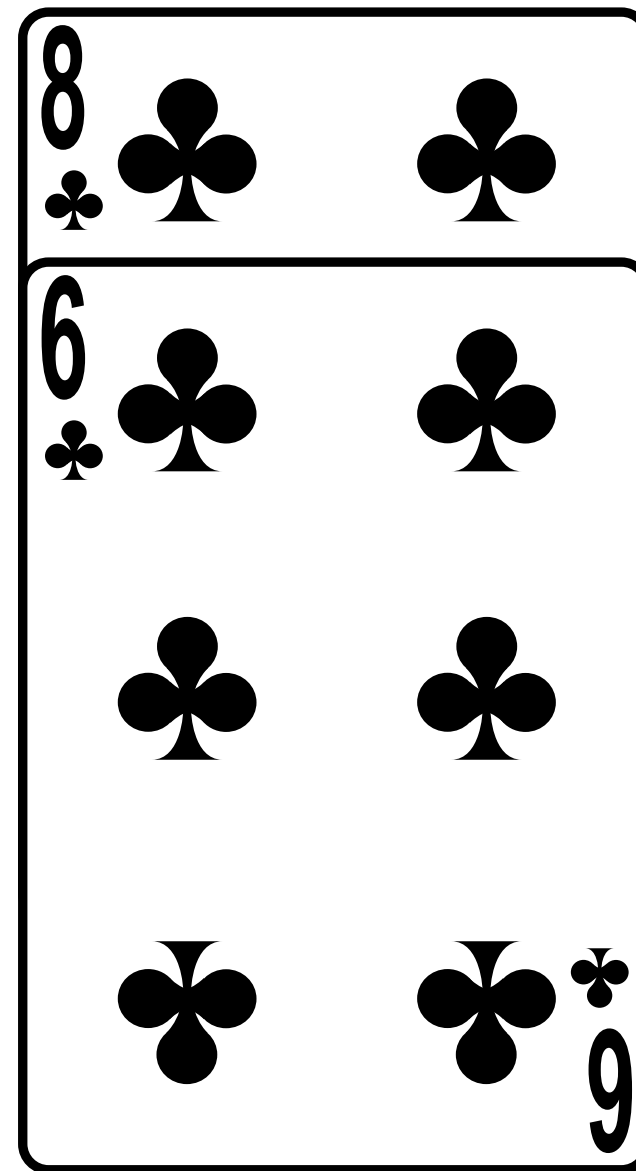
Fusionner deux tableaux triés



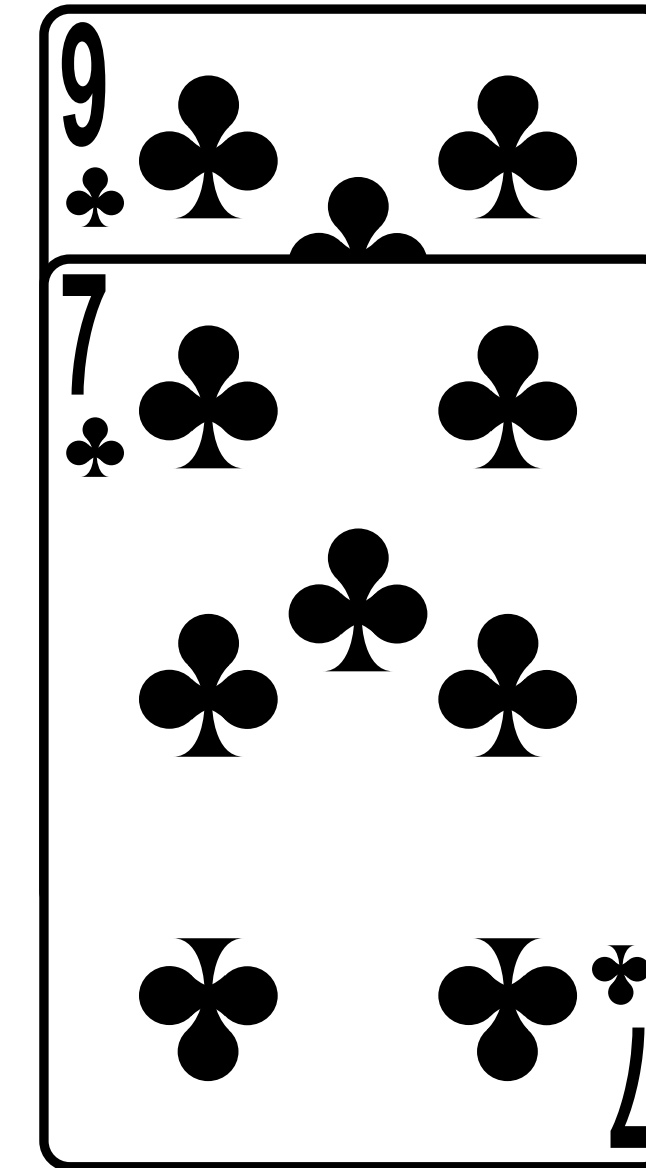
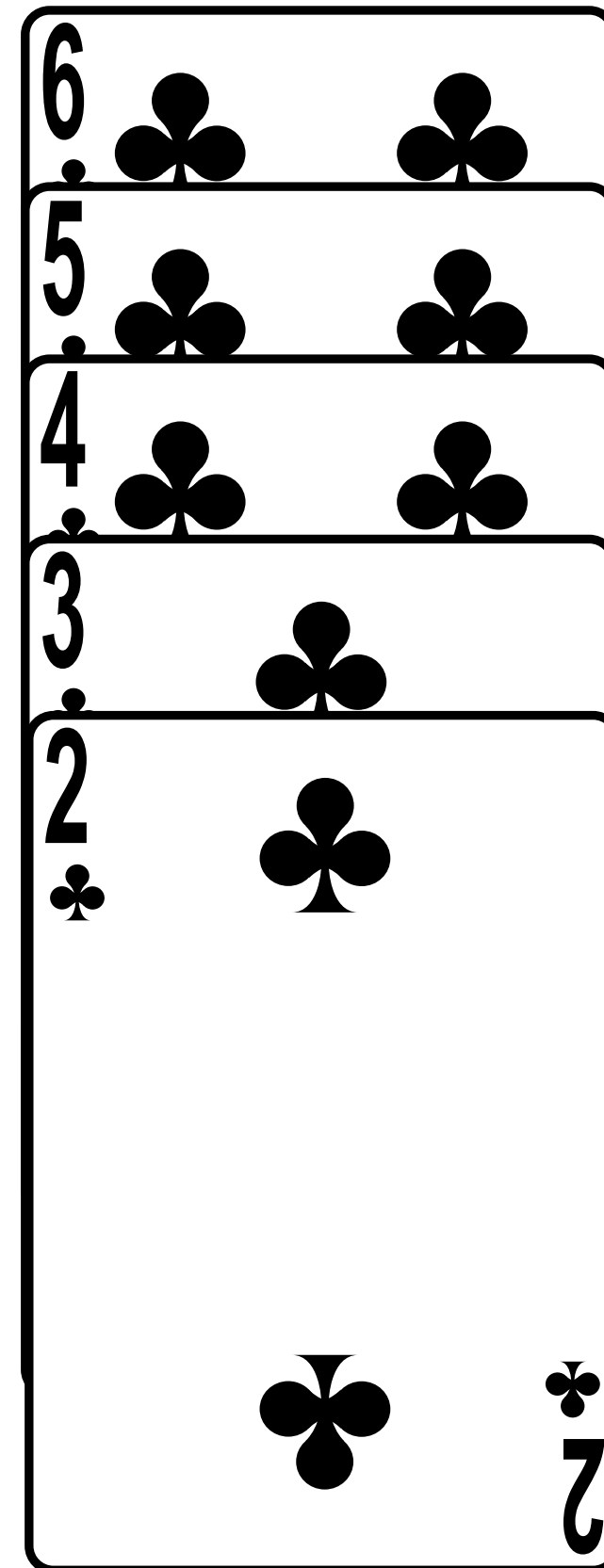
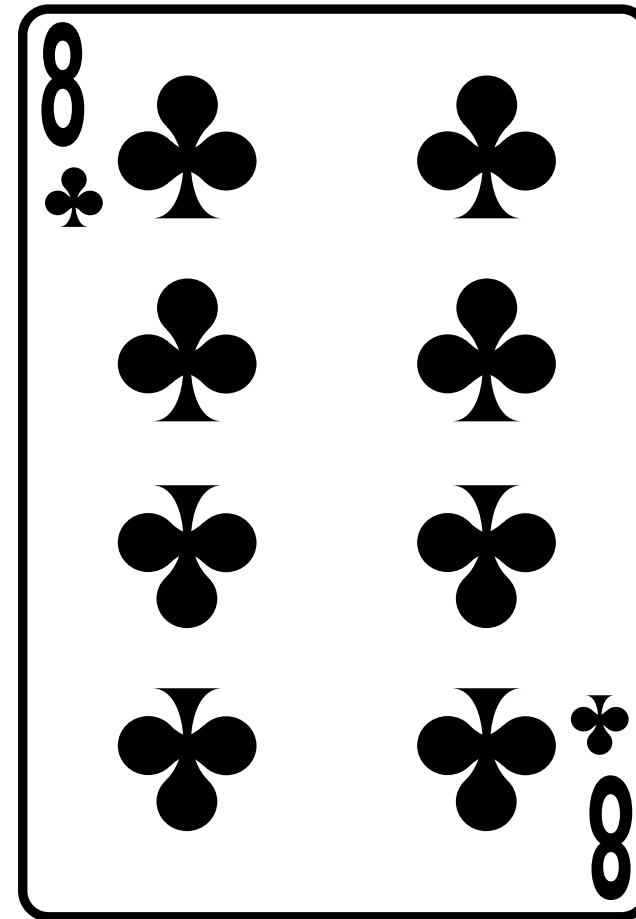
Fusionner deux tableaux triés



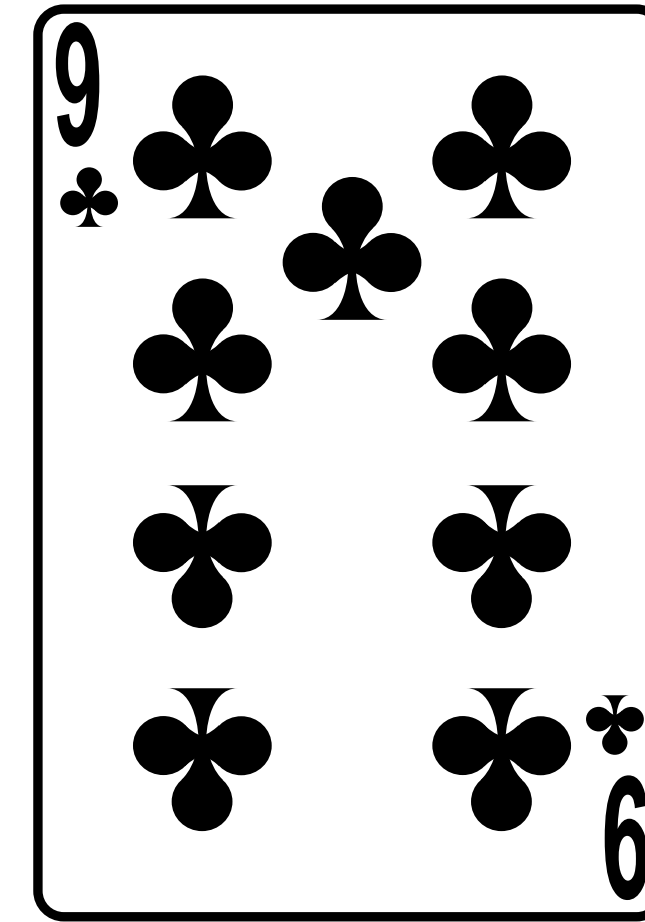
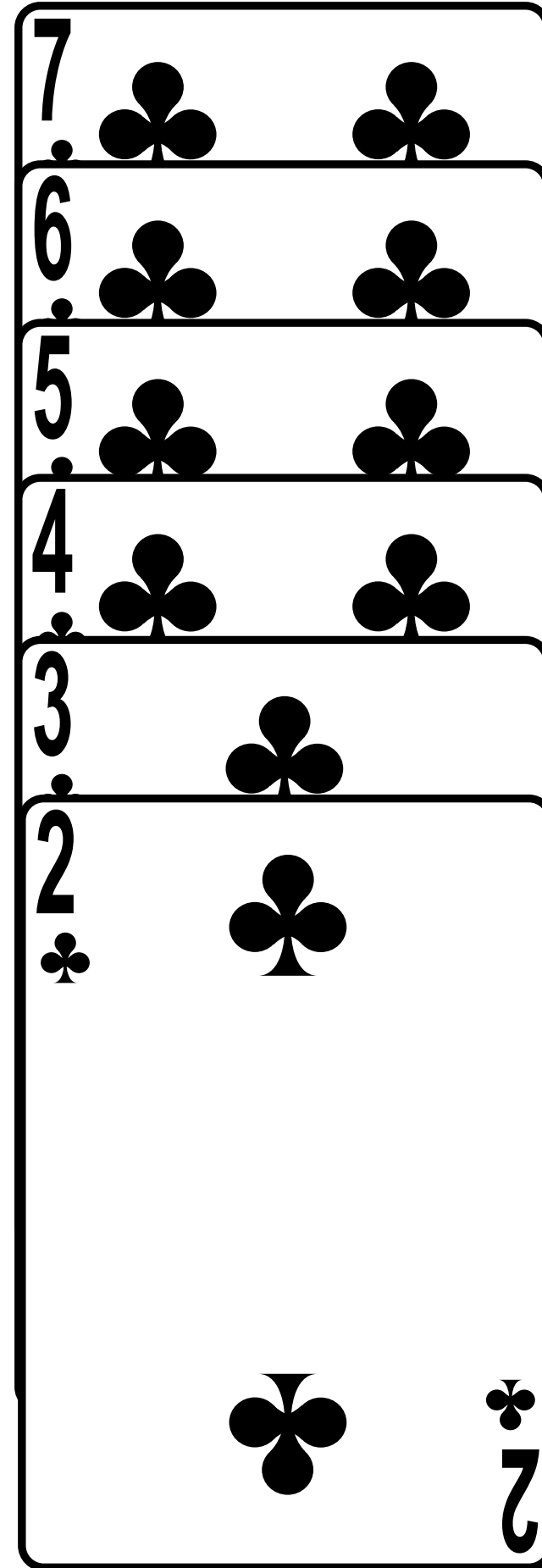
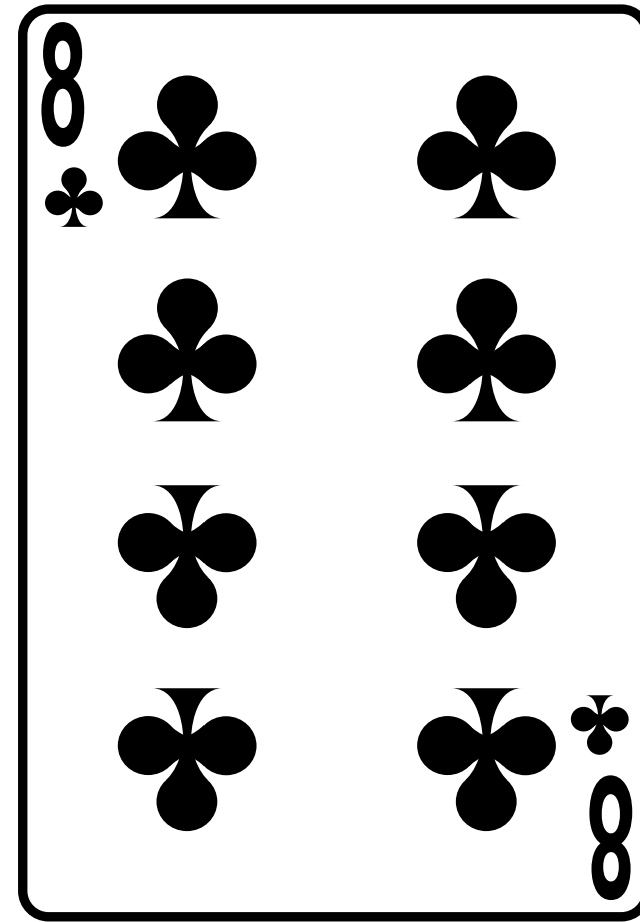
Fusionner deux tableaux triés



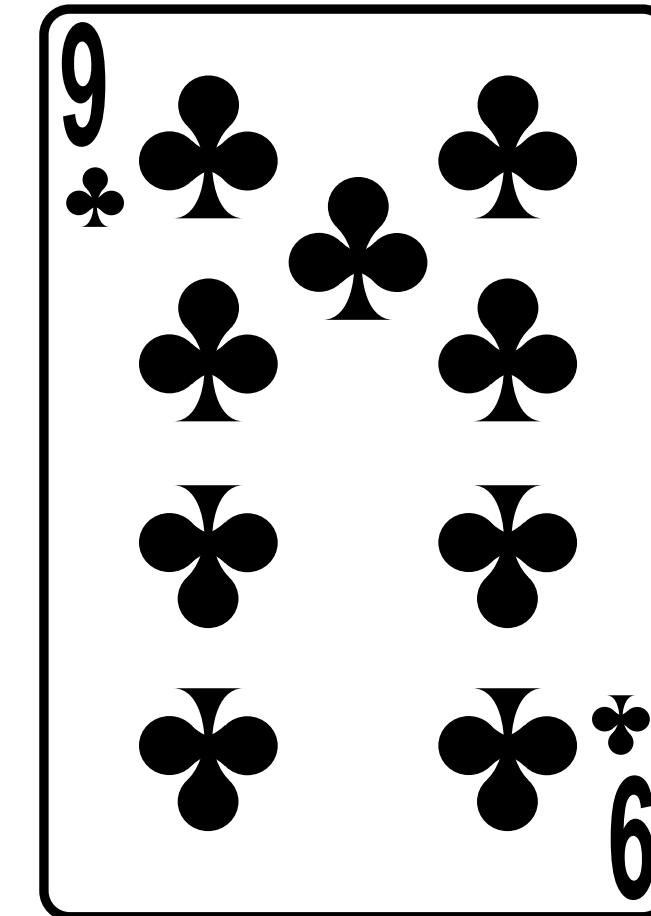
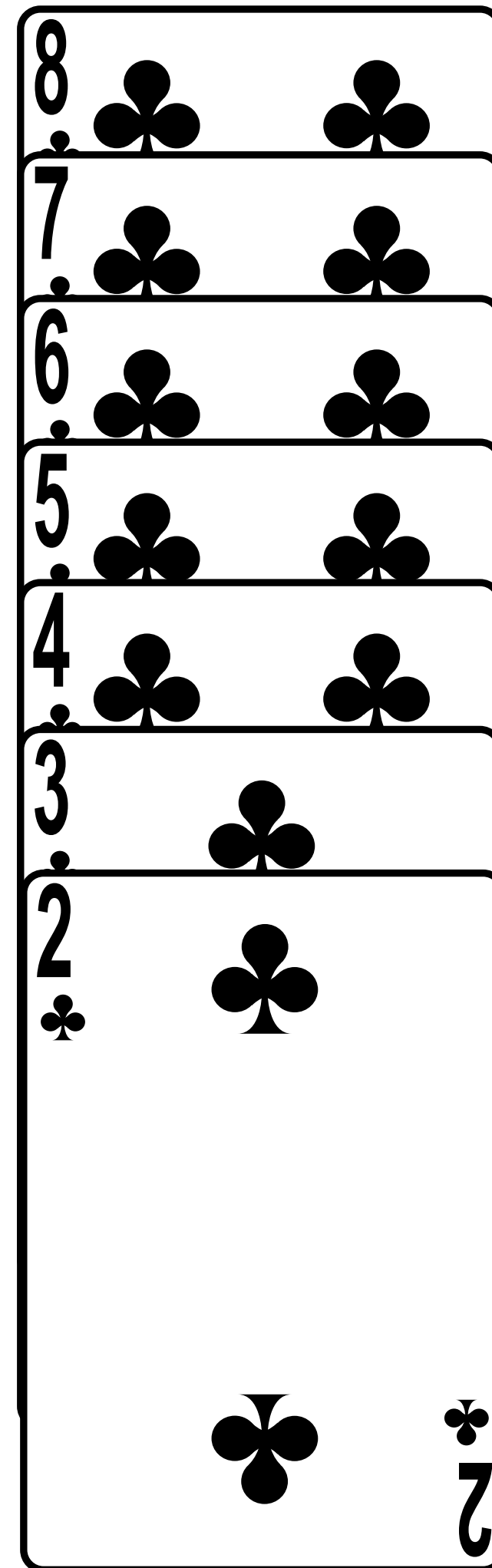
Fusionner deux tableaux triés



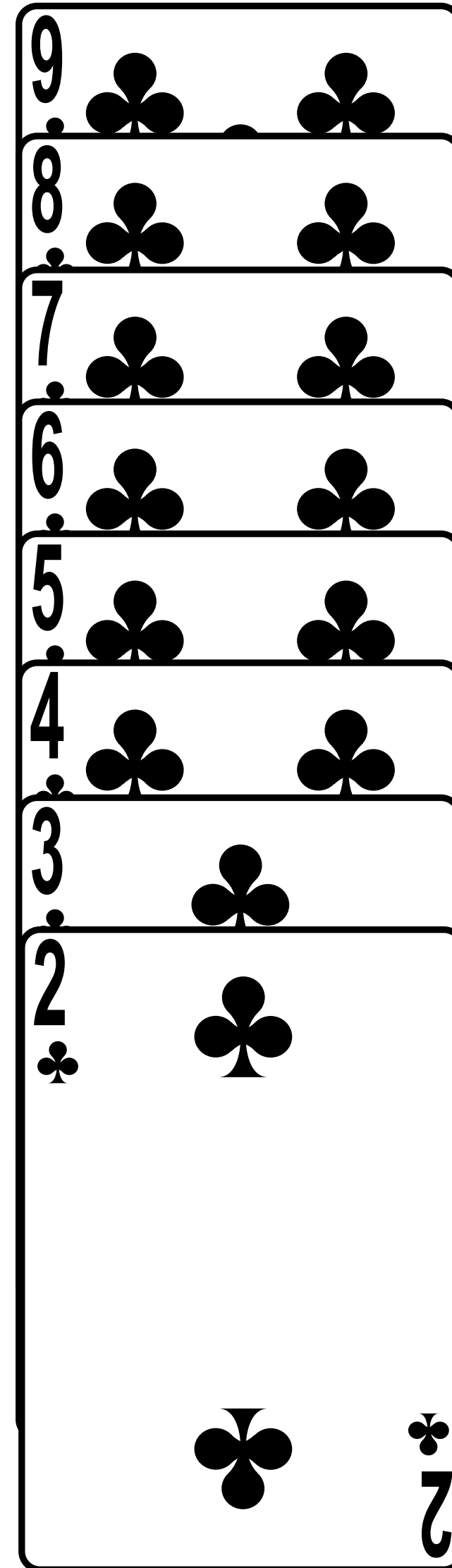
Fusionner deux tableaux triés



Fusionner deux tableaux triés



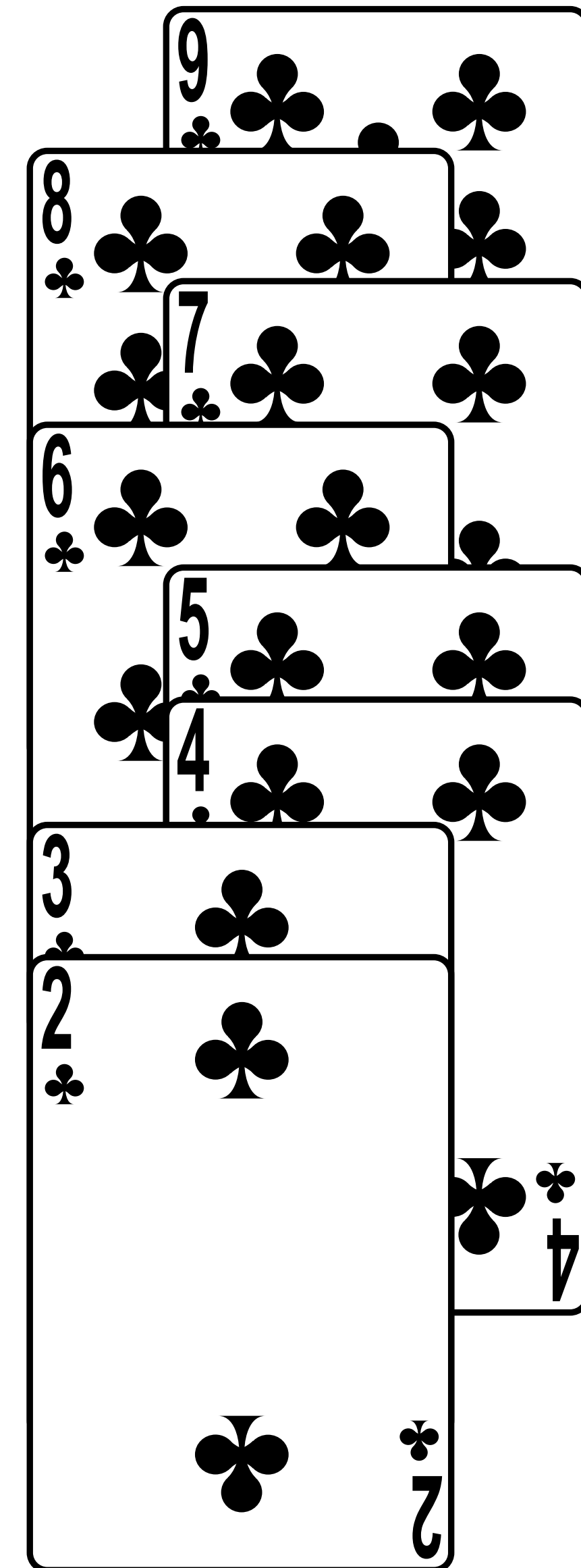
Fusionner deux tableaux triés



Fusionner deux tableaux triés



- **Entrée** : 2 tableaux triés par ordre croissant
- **Sortie** : 1 tableau fusionné trié par ordre croissant
- **Algorithme** :
 - Comparer les premiers (et donc plus petits) éléments des 2 tableaux
 - Déplacer le plus petit des 2 dans le tableau fusionné
 - Répéter jusqu'à ce que les 2 tableaux soient vides



Algorithme de fusion



Entrée :

- Tableau A
- les éléments p à q du tableau A sont triés
- les éléments $q+1$ à r du tableau A sont triés

Sortie :

- Tableau A modifié
- les éléments p à r du tableau A sont triés.

fonction Fusionner(A, p, q, r)

$L \leftarrow$ copie du tableau A de p à q

$R \leftarrow$ copie du tableau A de $q+1$ à r

$i \leftarrow 1, j \leftarrow 1$

pour k de p à r **boucler**

si $j \geq r - q$ **ou** $(i \leq q - p$ **et** $L(i) \leq R(j))$, **alors**

$A(k) \leftarrow L(i)$

incrémenter i

sinon

$A(k) \leftarrow R(j)$

incrémenter j

fin si

fin pour

Vérification de bornes

*On prend de L si
 R est épuisé, **ou**
 L ne l'est pas **et**
 $L(i)$ est le plus petit.*

Algorithme de fusion



Entrée :

- Tableau A
- les éléments p à q du tableau A sont triés
- les éléments $q+1$ à r du tableau A sont triés

Sortie :

- Tableau A modifié
- les éléments p à r du tableau A sont triés.

fonction Fusionner(A, p, q, r)

$L \leftarrow$ copie du tableau A de p à q

$R \leftarrow$ copie du tableau A de $q+1$ à r

insérer une sentinelle ∞ en fin de L et R

Évite de vérifier les bornes.

$i \leftarrow 1, j \leftarrow 1$

pour k de p à r **boucler**

si $L(i) \leq R(j)$, **alors**

$A(k) \leftarrow L(i)$

incrémenter i

sinon

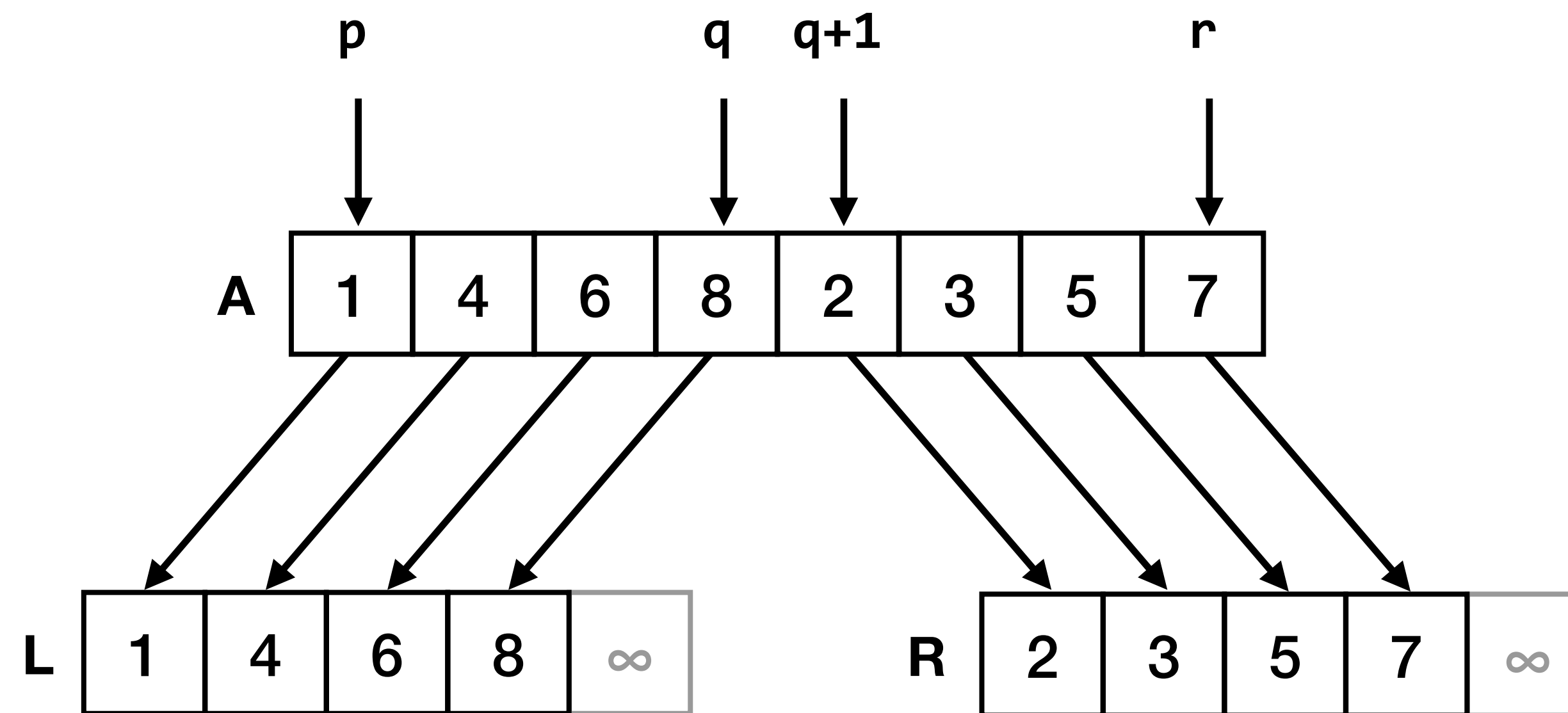
$A(k) \leftarrow R(j)$

incrémenter j

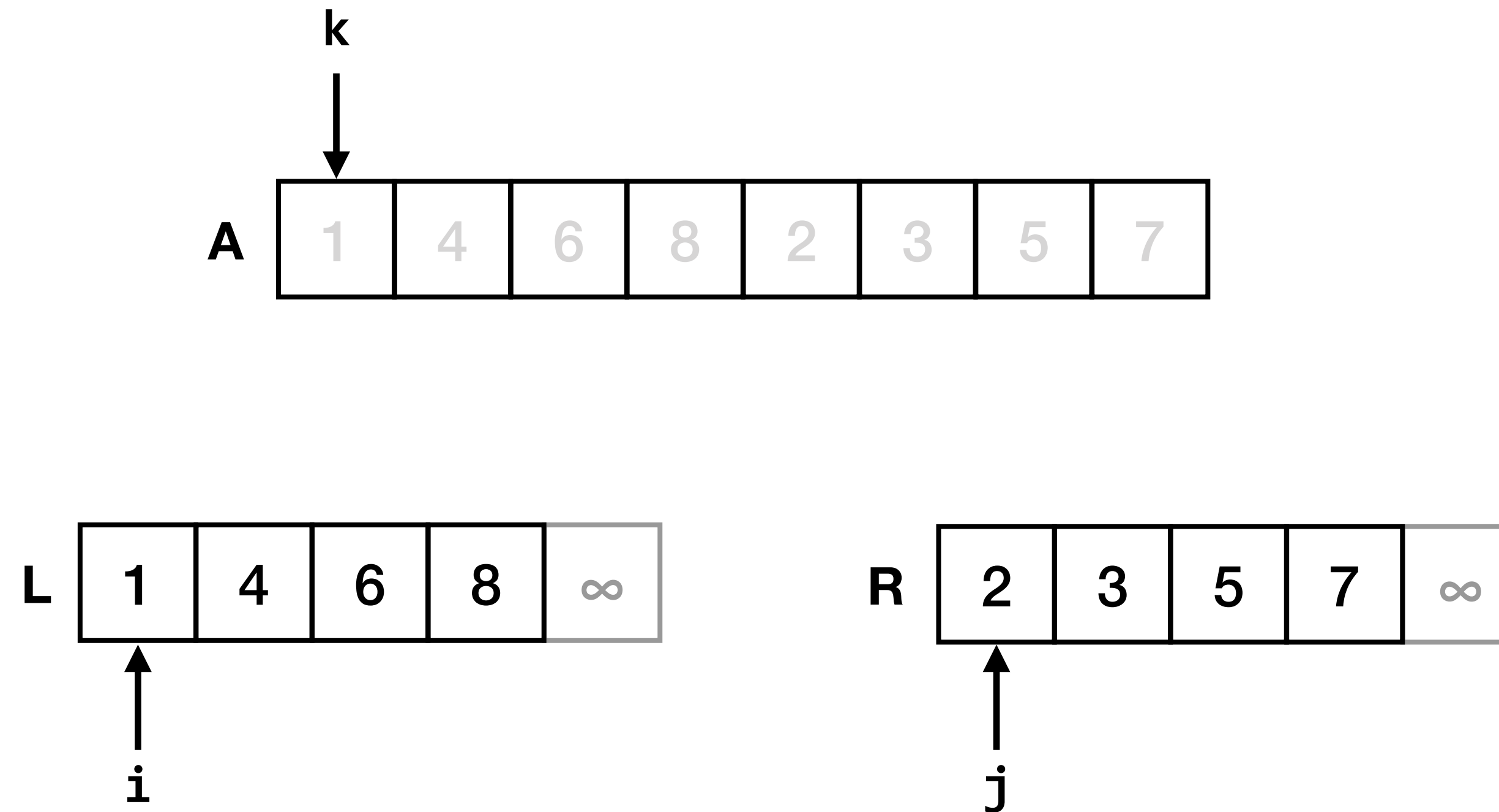
fin si

fin pour

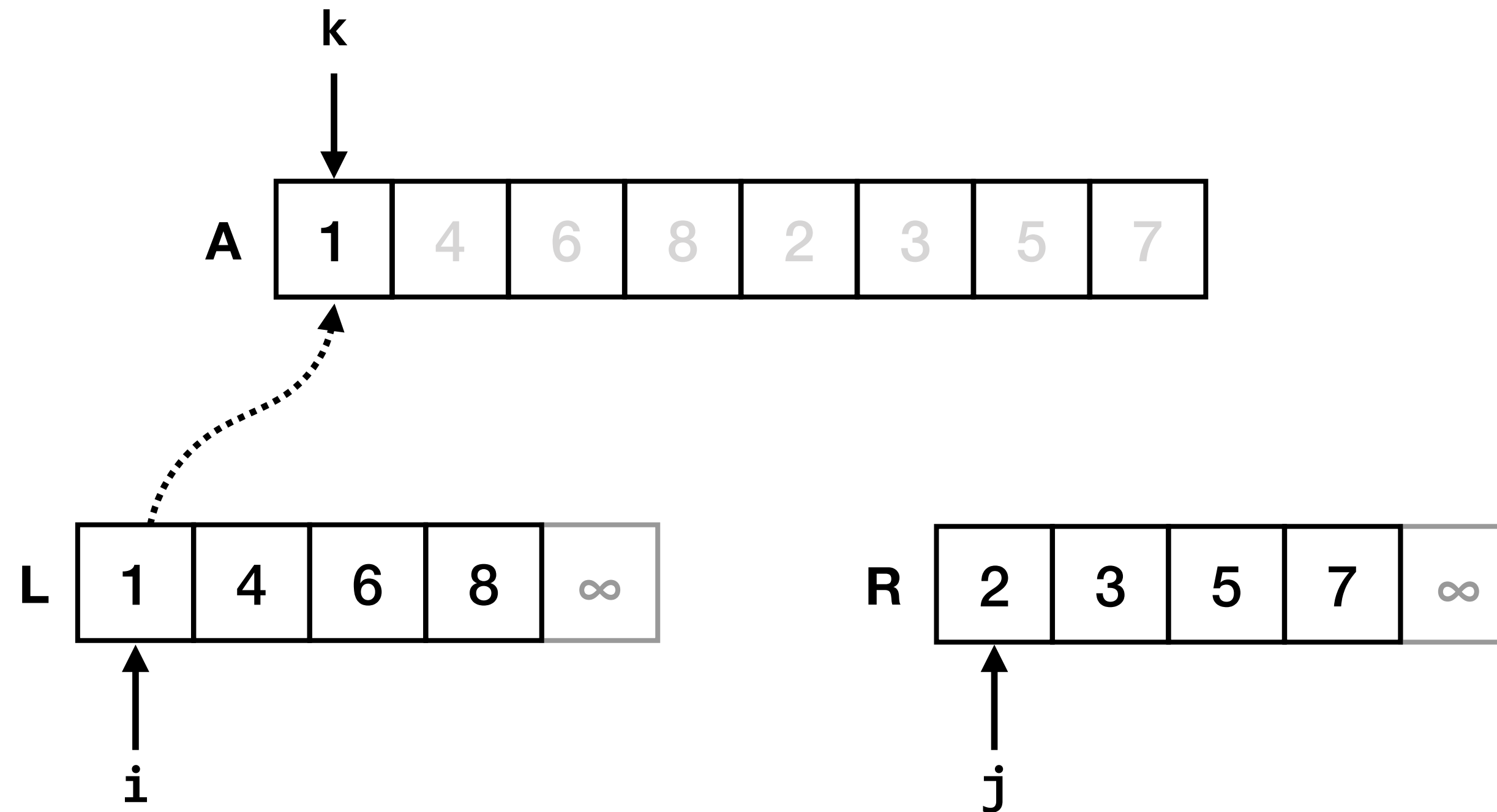
Algorithme de fusion



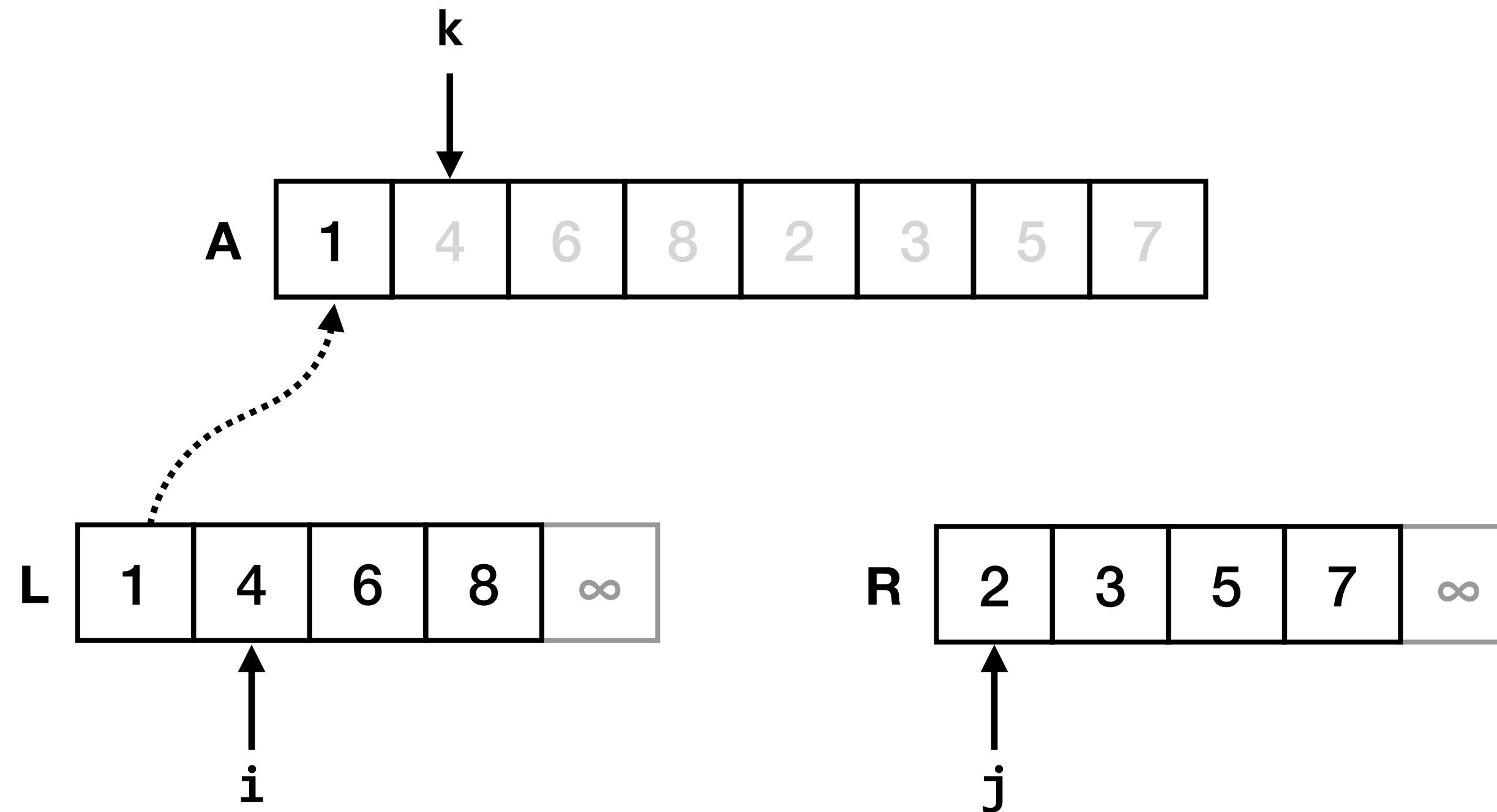
Algorithme de fusion



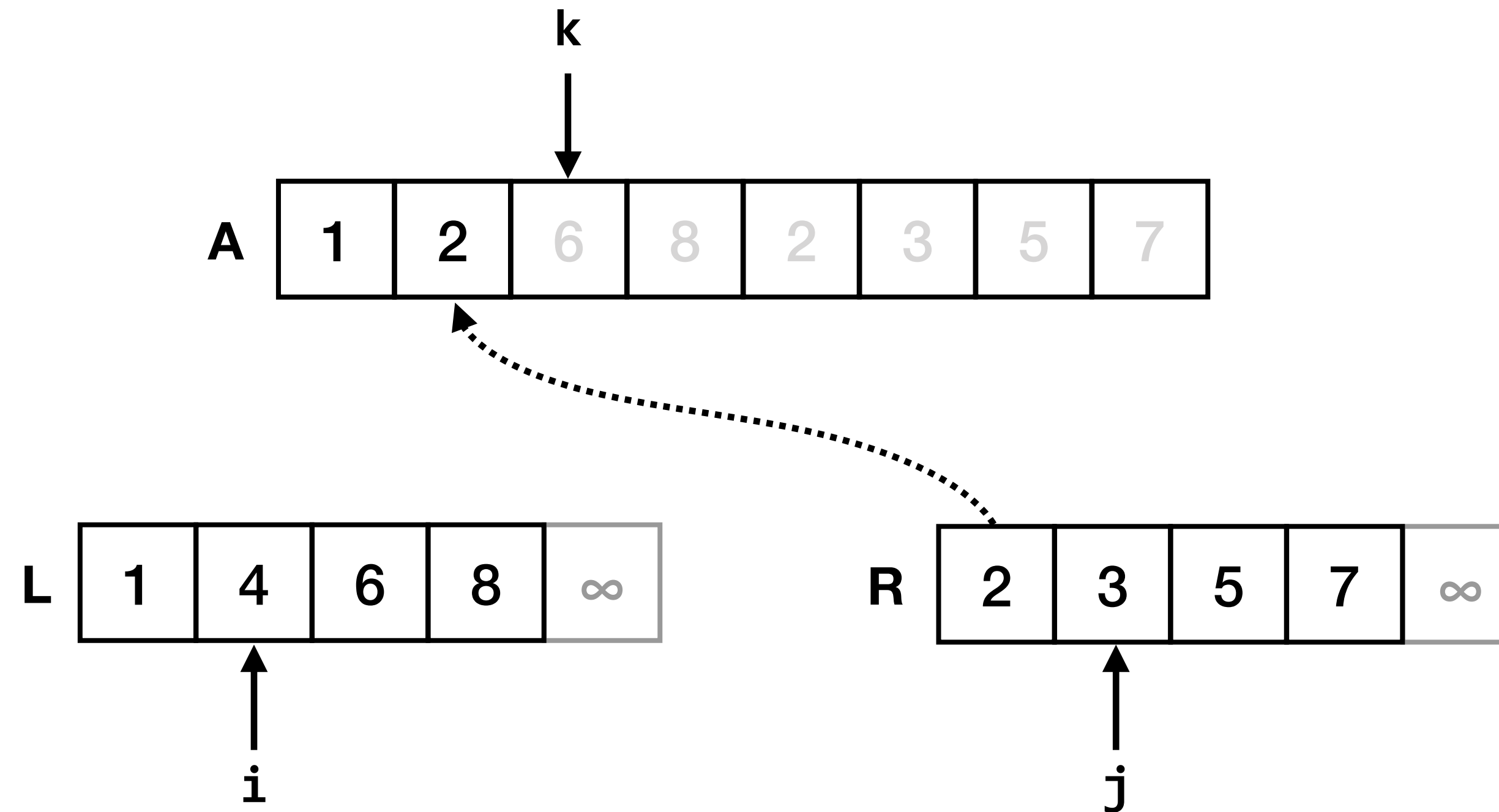
Algorithme de fusion



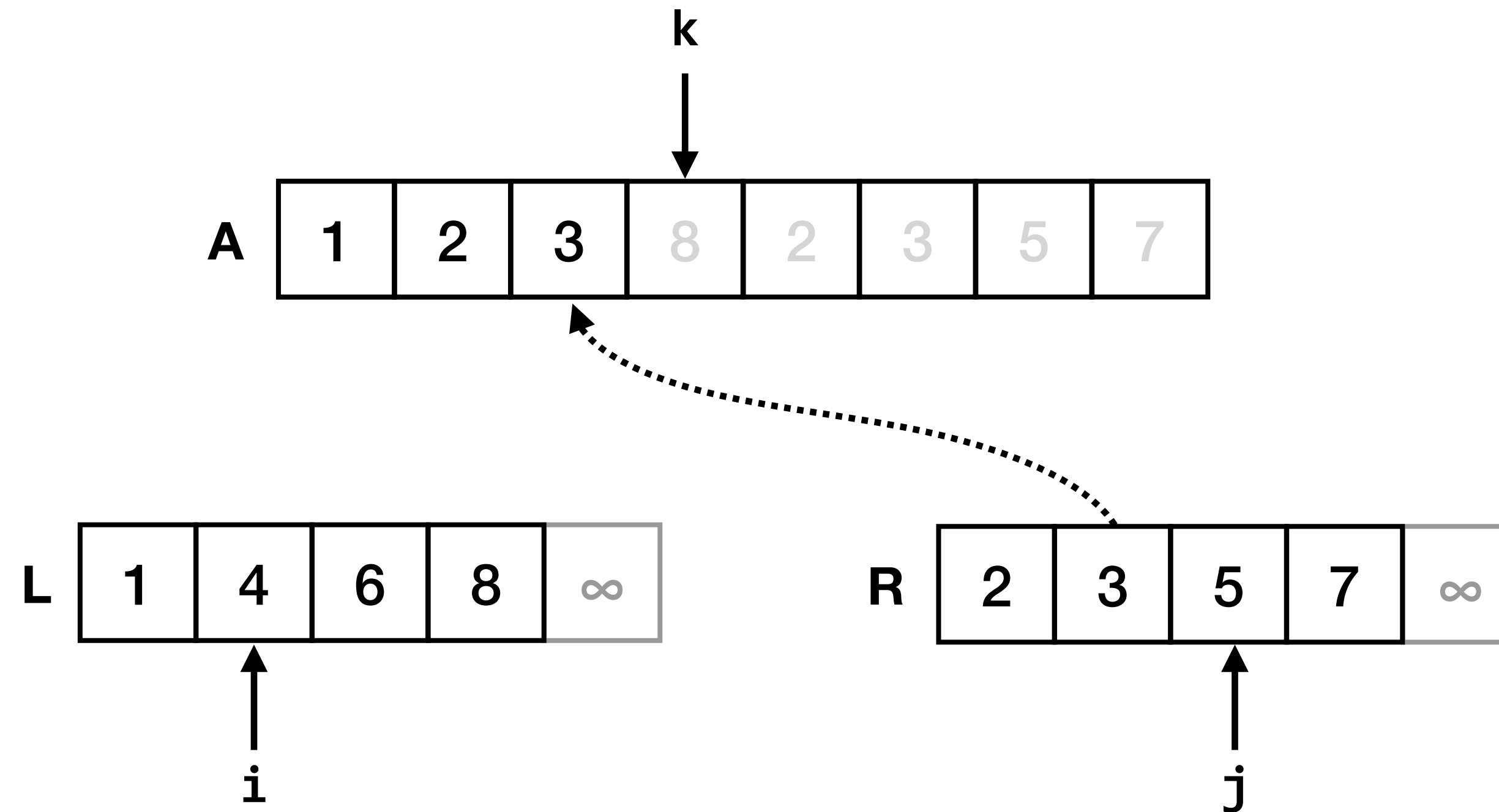
Algorithme de fusion



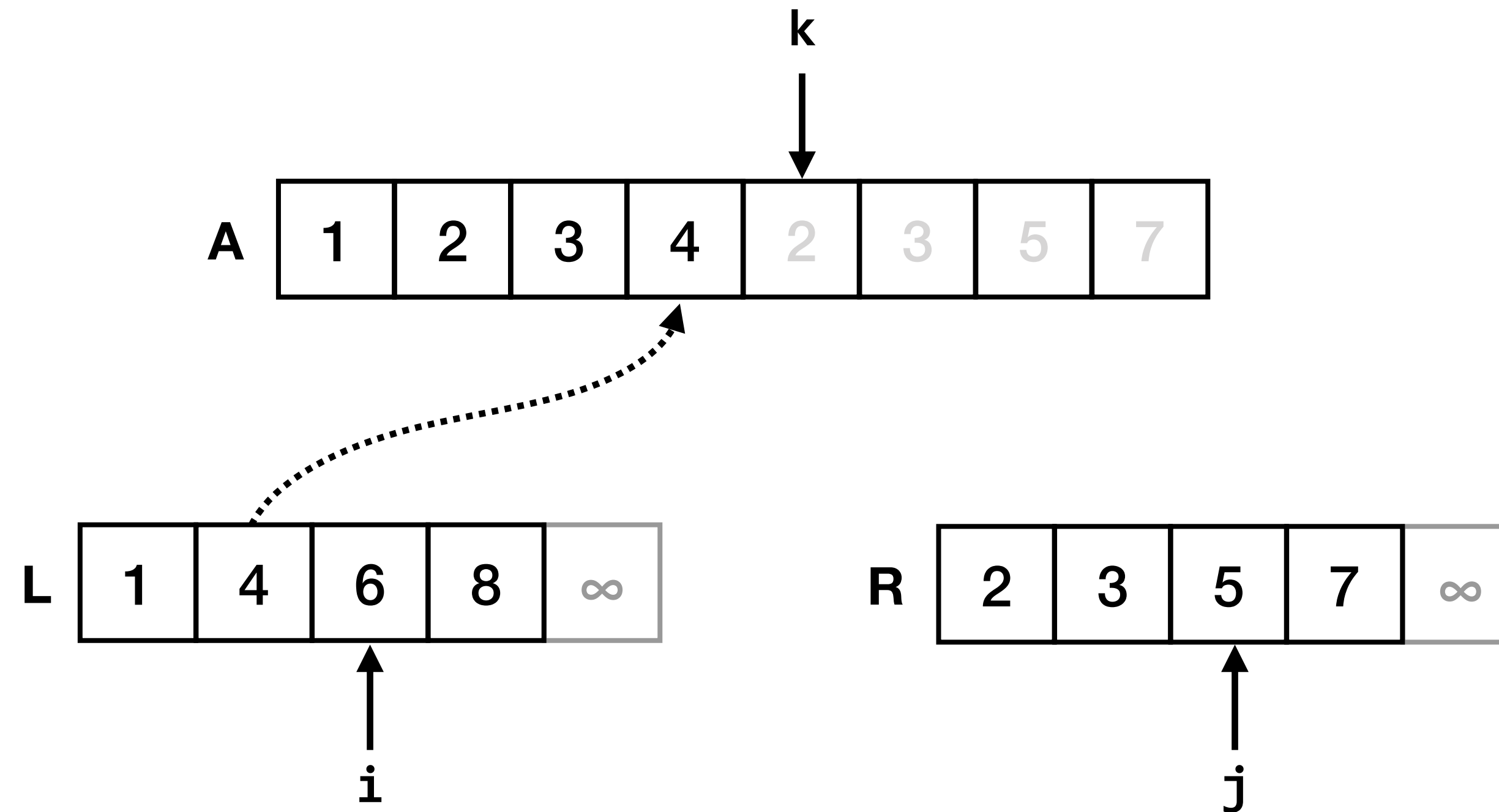
Algorithme de fusion



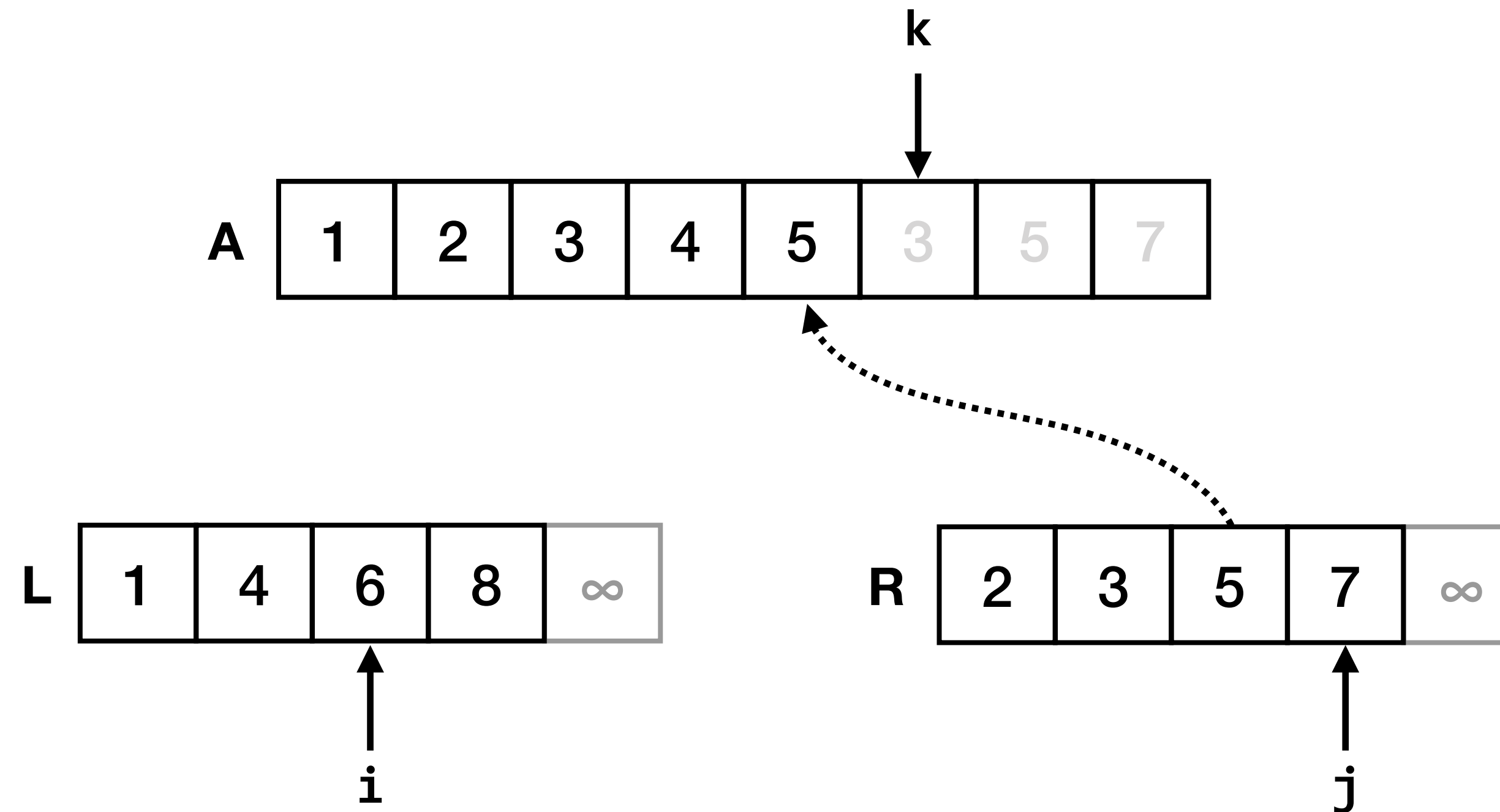
Algorithme de fusion



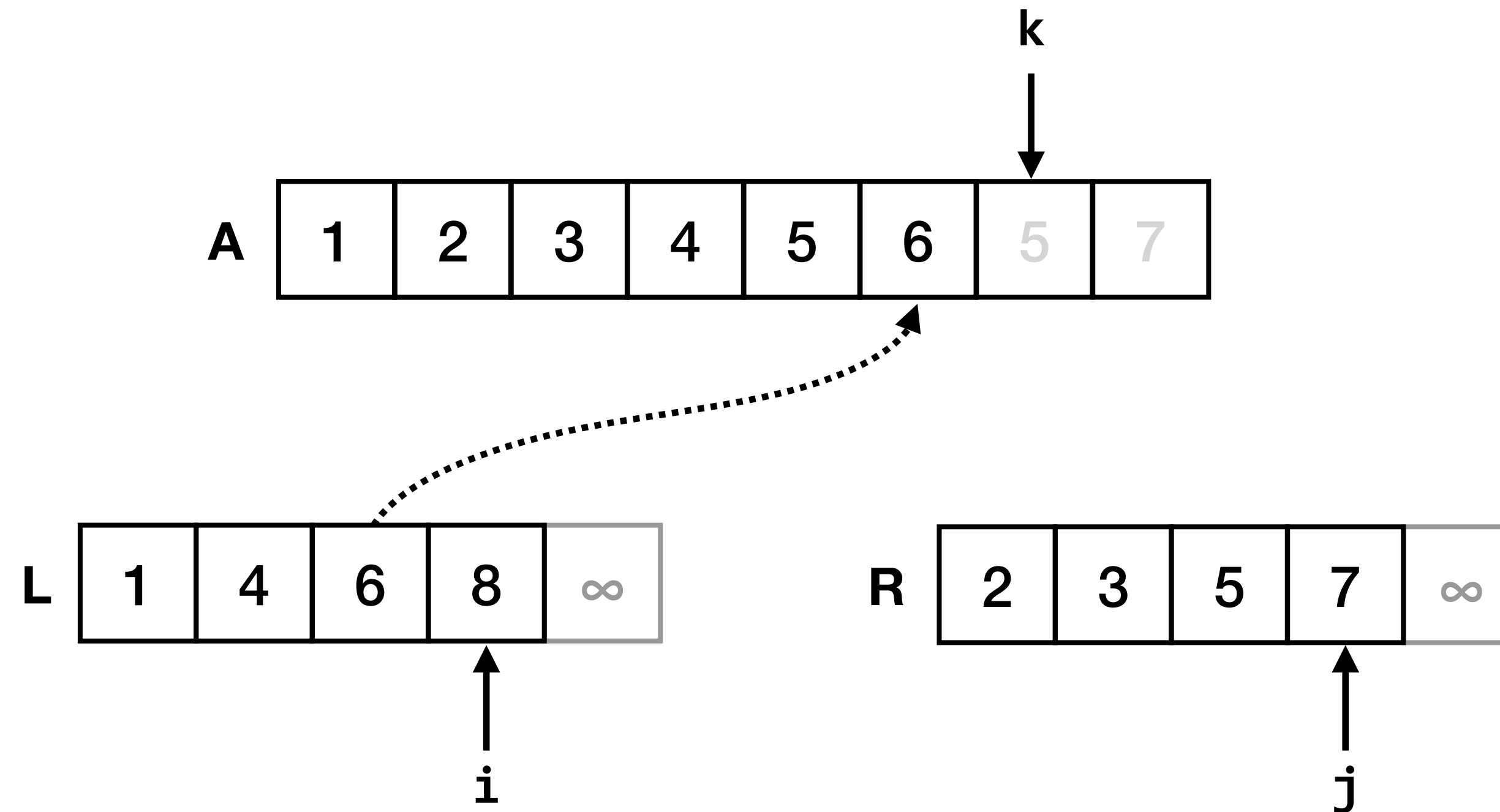
Algorithme de fusion



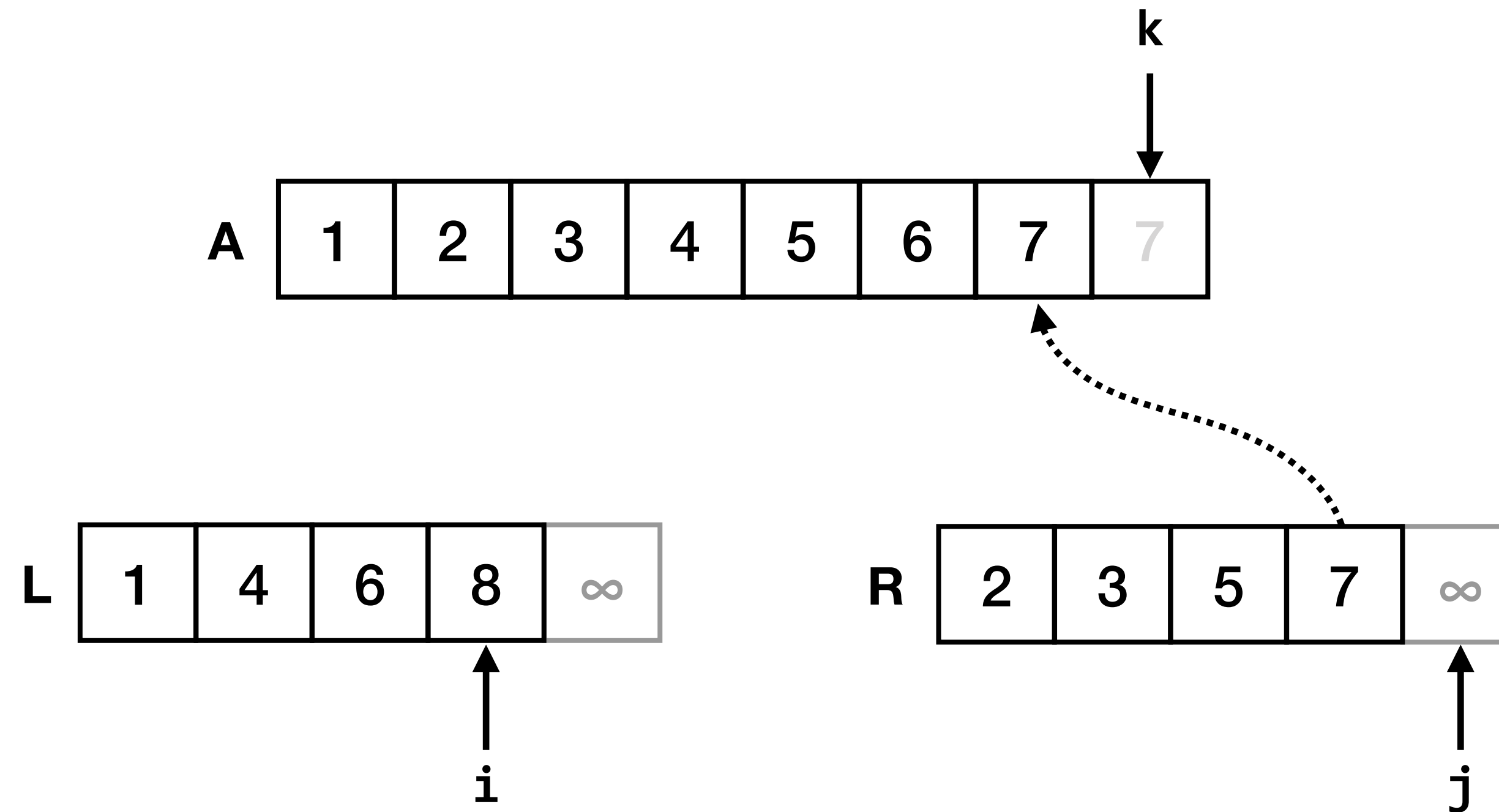
Algorithme de fusion



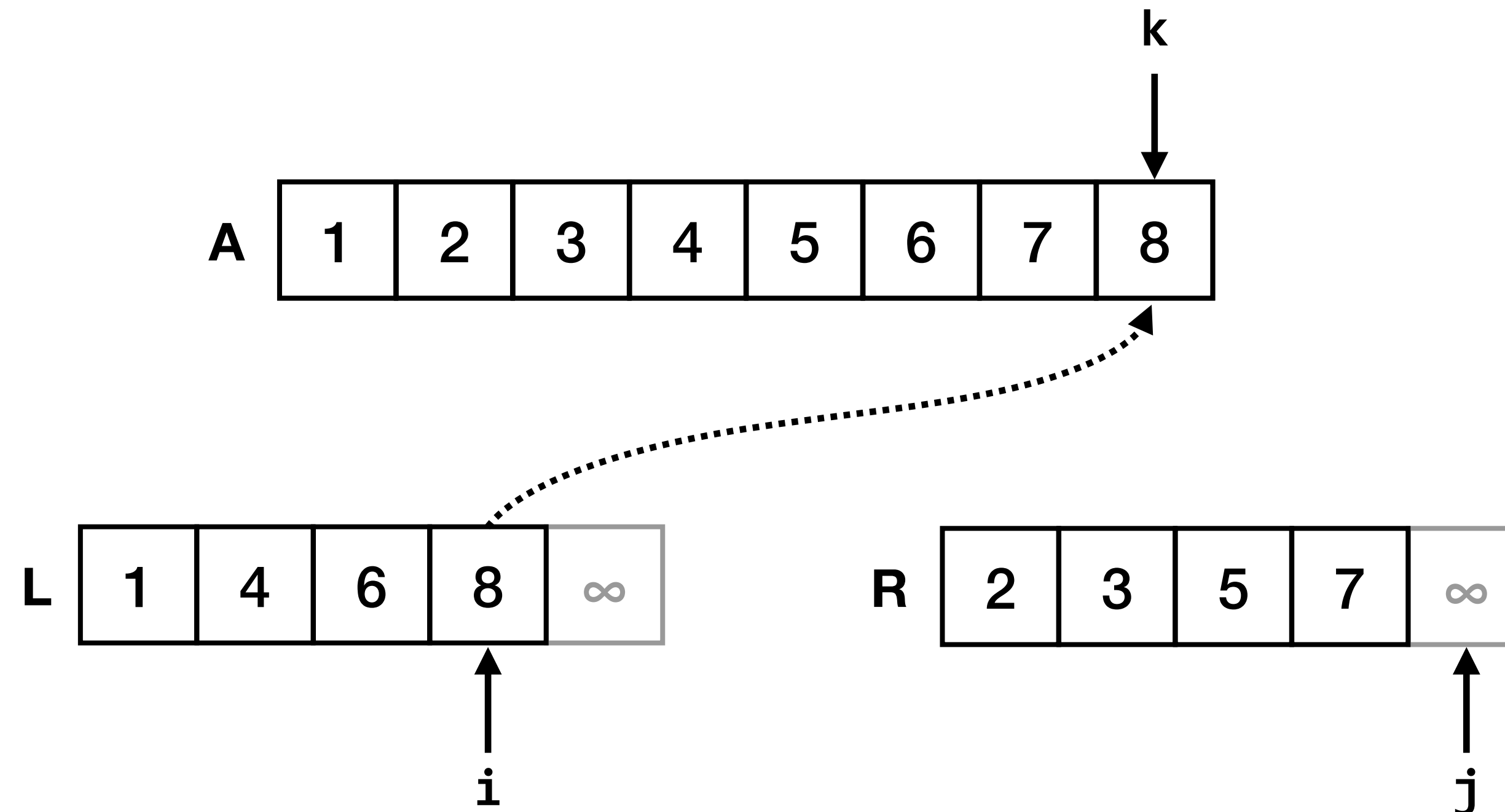
Algorithme de fusion



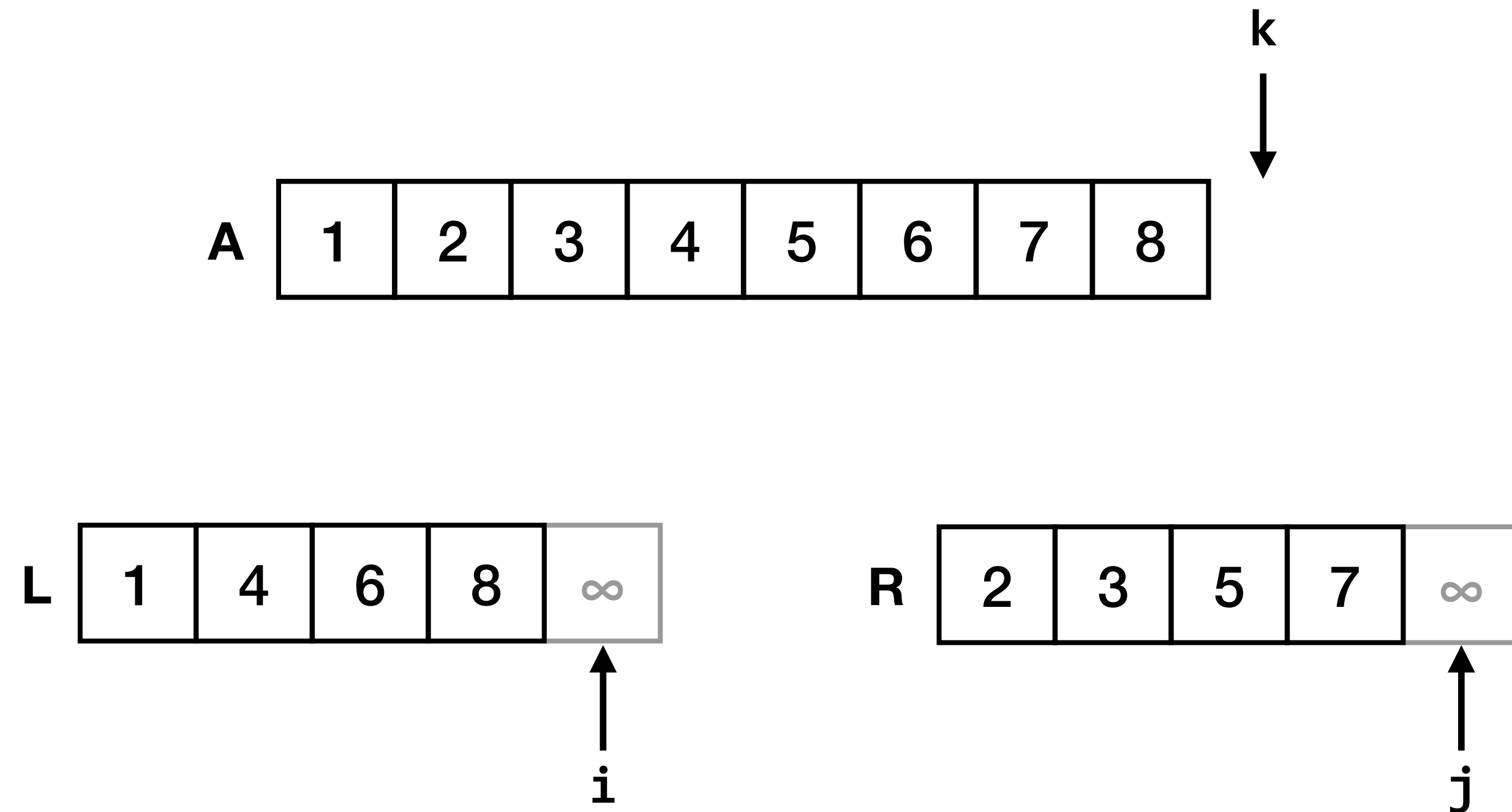
Algorithme de fusion



Algorithme de fusion



Algorithme de fusion



Récursion

Maintenant que l'on sait fusionner des tableaux

- **Entrée** : un tableau non trié
- **Sortie** : un tableau trié
- **Algorithme** :
 - Cas trivial : tableau de 0 ou 1 élément
 - Cas général : tableau de plusieurs éléments
 - Diviser le tableau en 2 parts les plus égales possibles
 - *Trier (récursivement)* les deux sous-tableaux
 - Fusionner les deux sous-tableaux triés



Fonction réursive et appel à fusion



Entrée

- Tableau *A*
- L'intervalle [*lo*, *hi*] des indices à trier

Sortie

- Tableau *A* trié des indices *lo* à *hi*

Pour trier tout le tableau *A* de taille *n*, on appelle *TriFusion*(*A*, 1, *n*)

```
fonction TriFusion(A, lo, hi)
```

```
  si hi <= lo, alors  
    retourner  
  fin si
```

```
  mid ← lo + (hi - lo) / 2
```

```
  TriFusion(A, lo, mid)  
  TriFusion(A, mid + 1, hi)
```

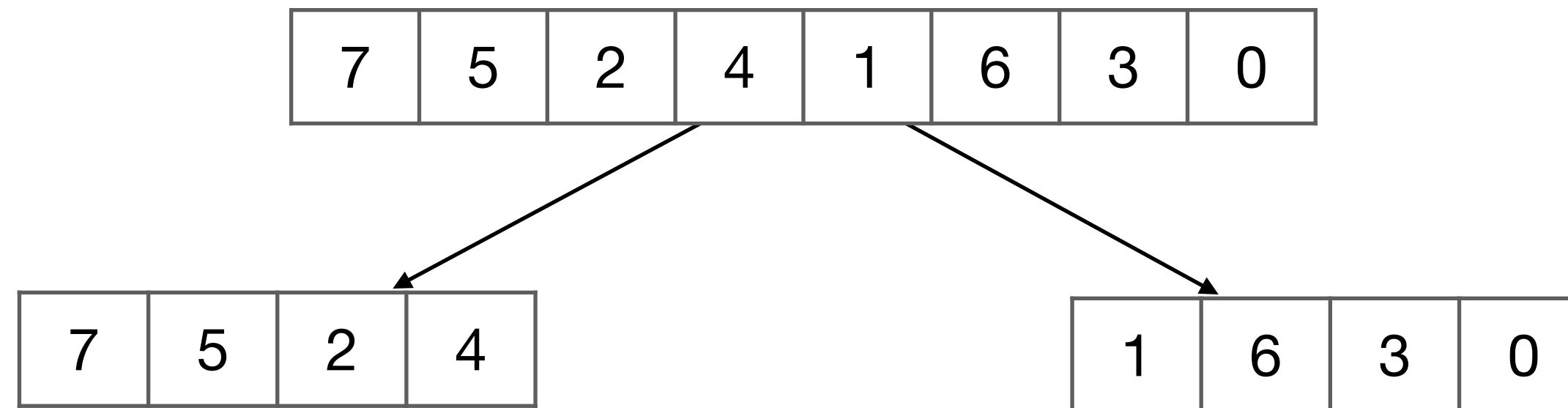
```
  Fusionner(A, lo, mid, hi)
```

Visualisation

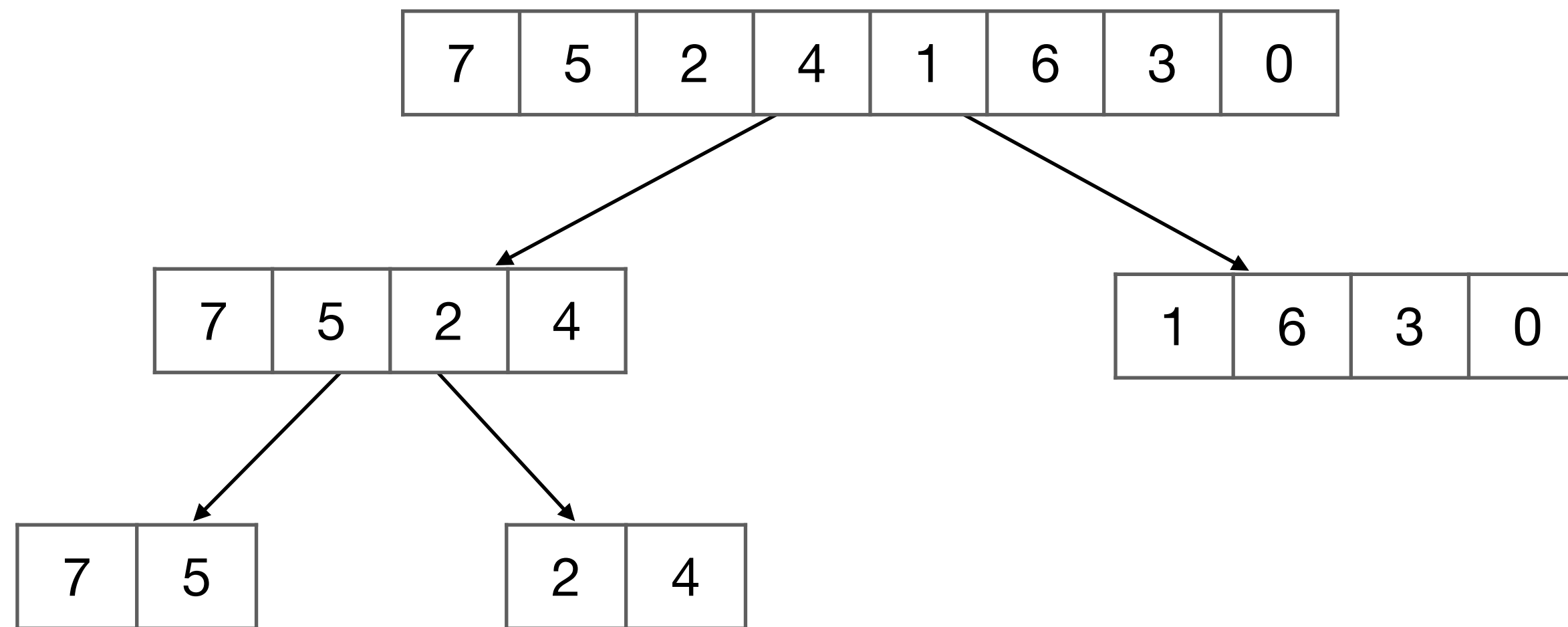


7	5	2	4	1	6	3	0
---	---	---	---	---	---	---	---

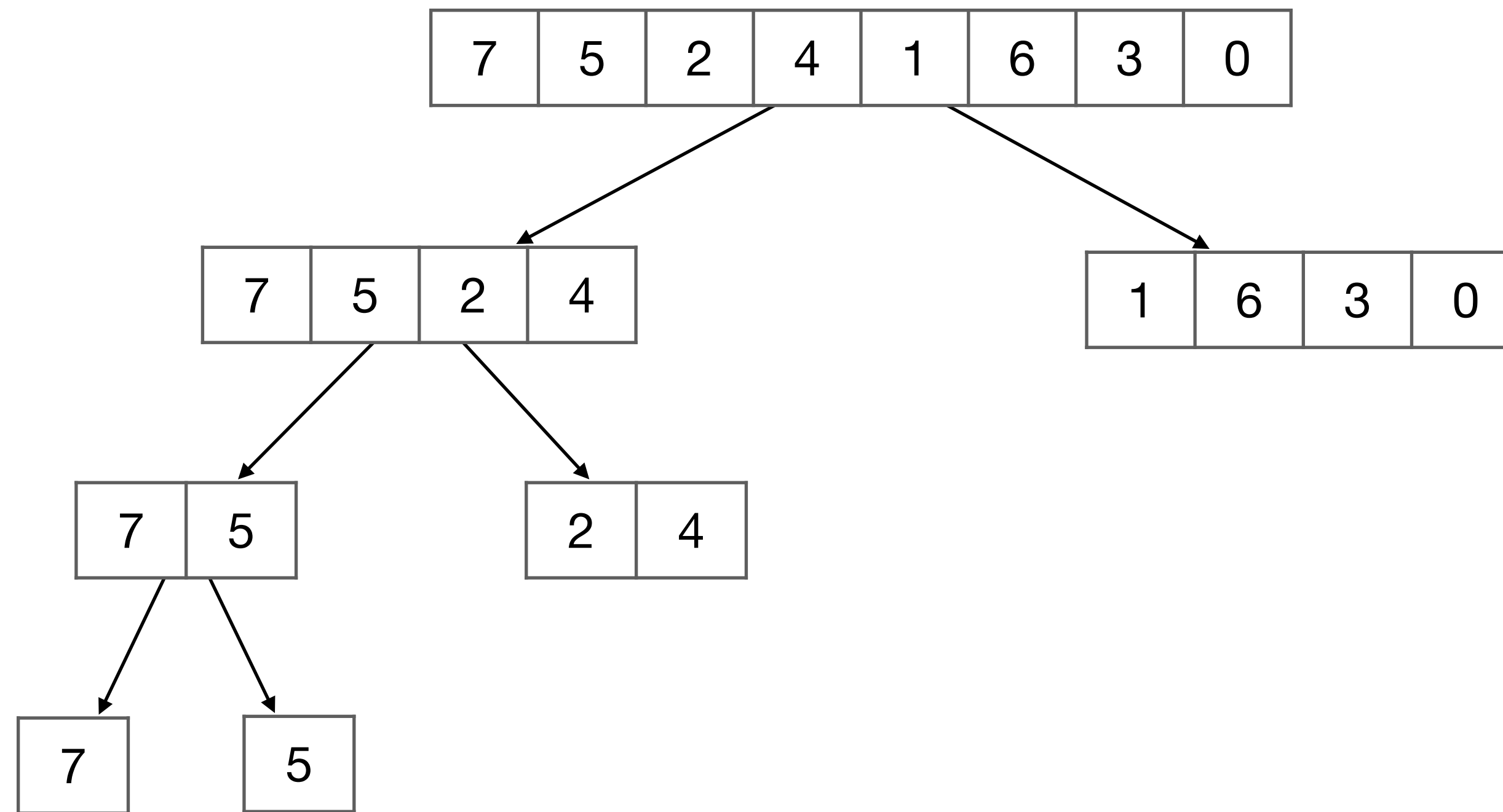
Visualisation



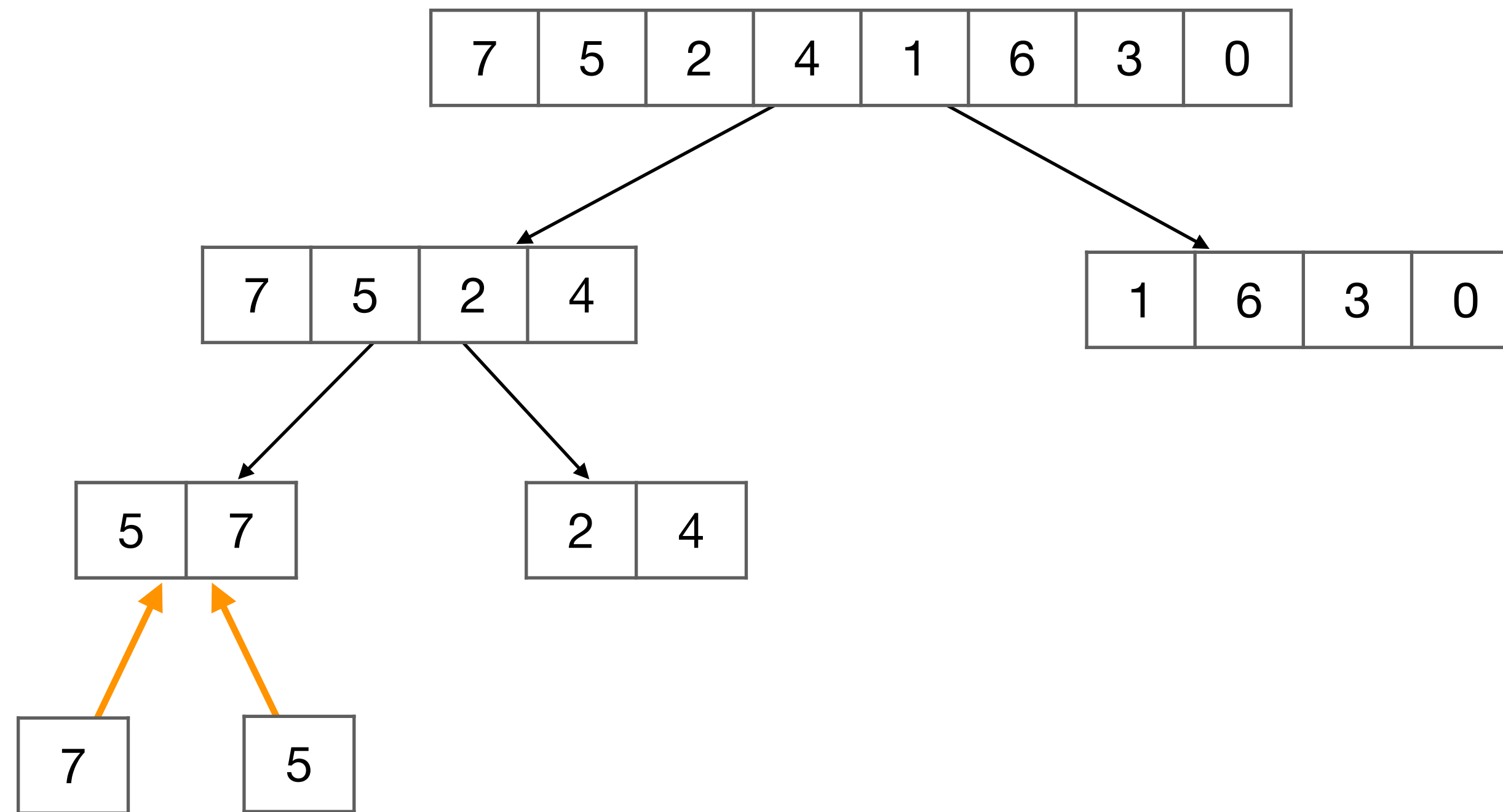
Visualisation



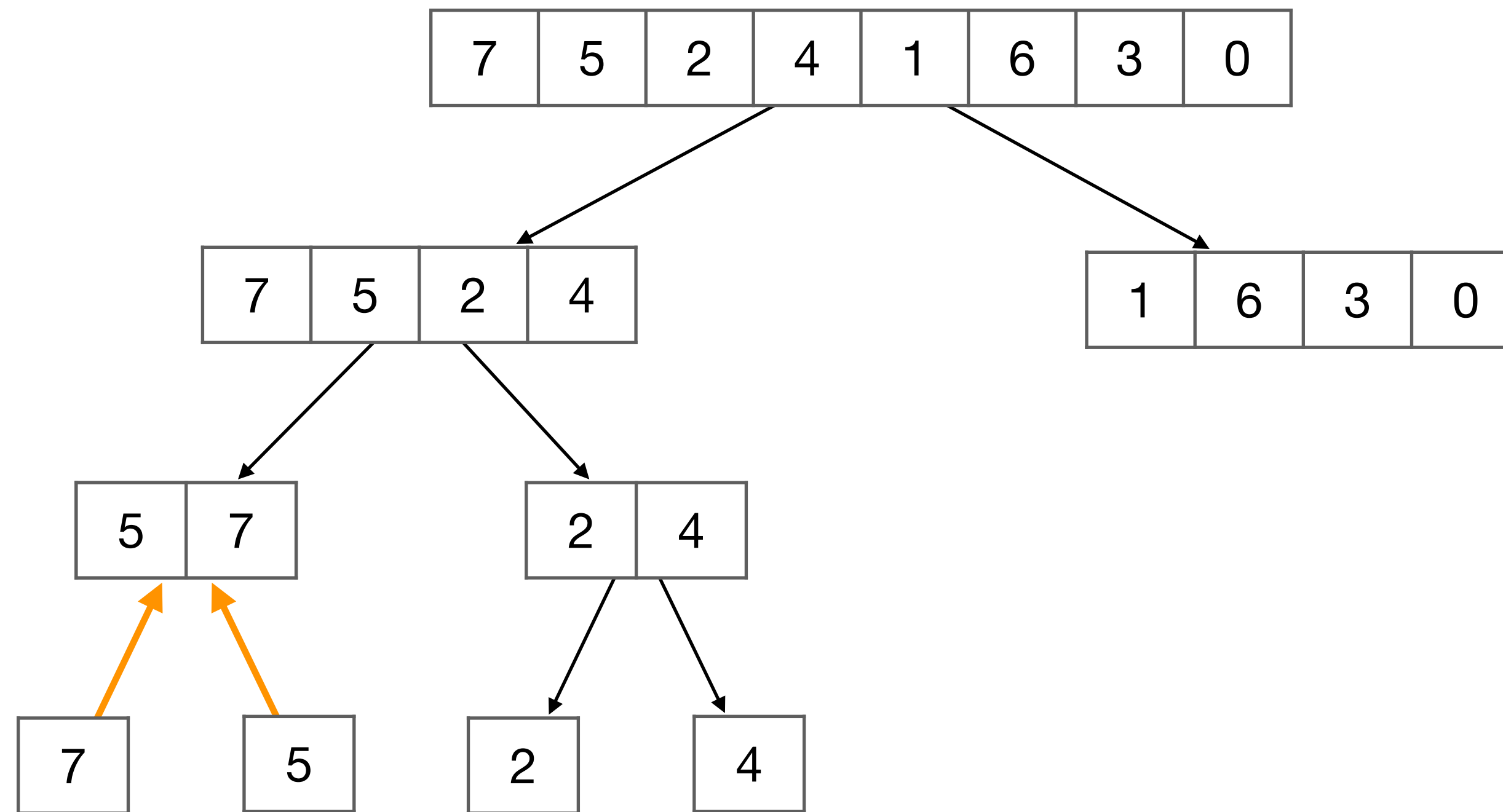
Visualisation



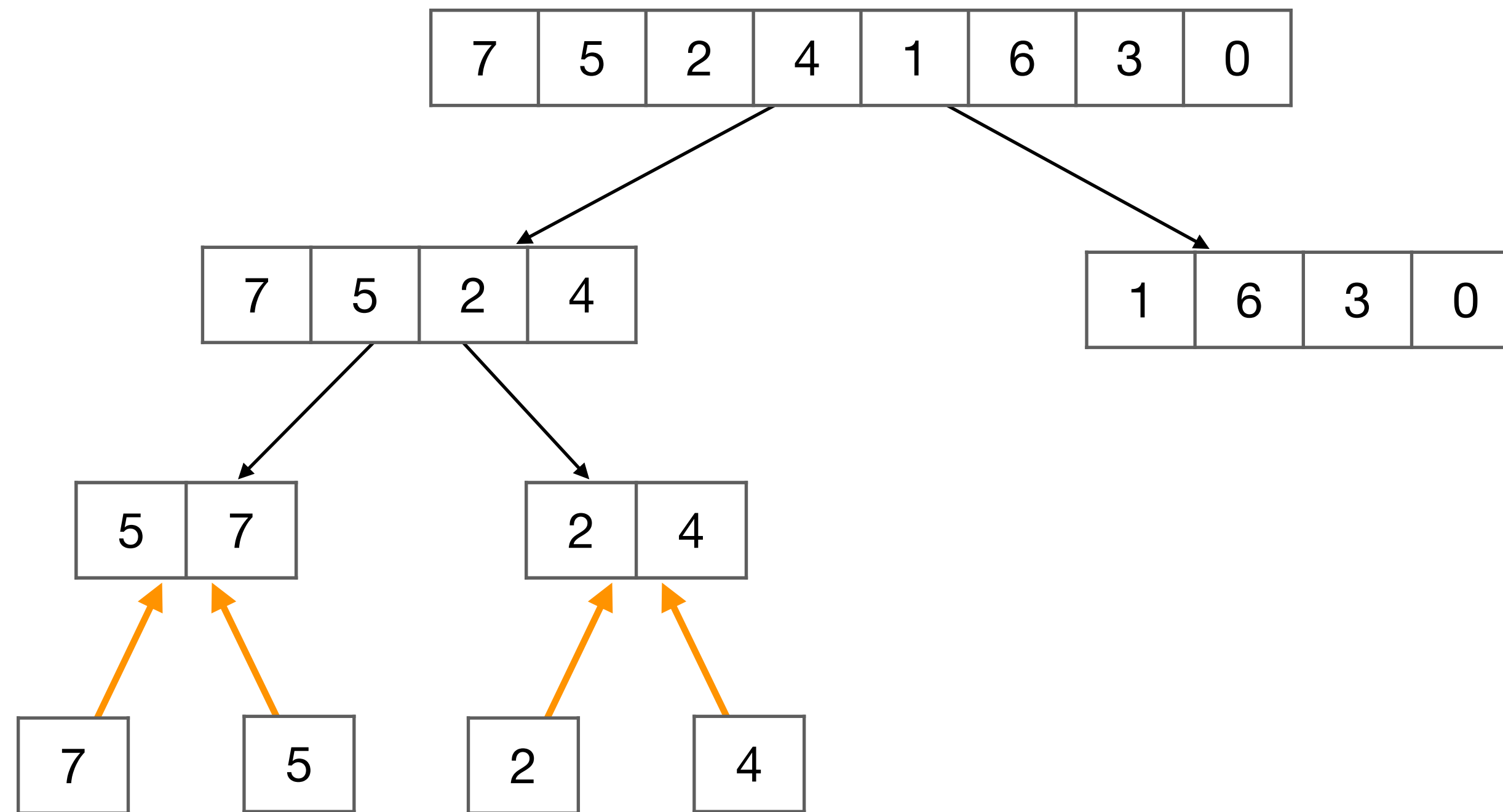
Visualisation



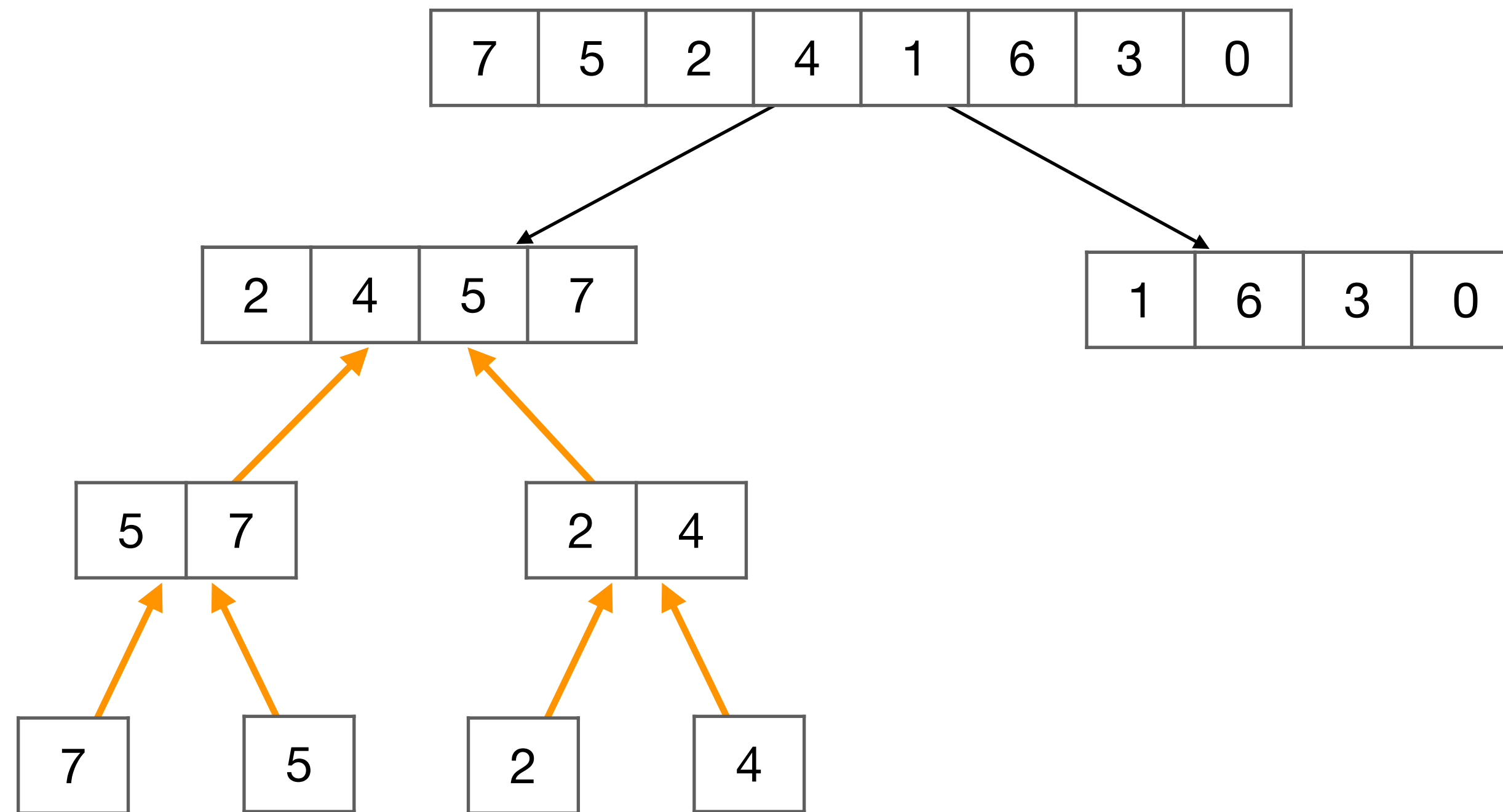
Visualisation



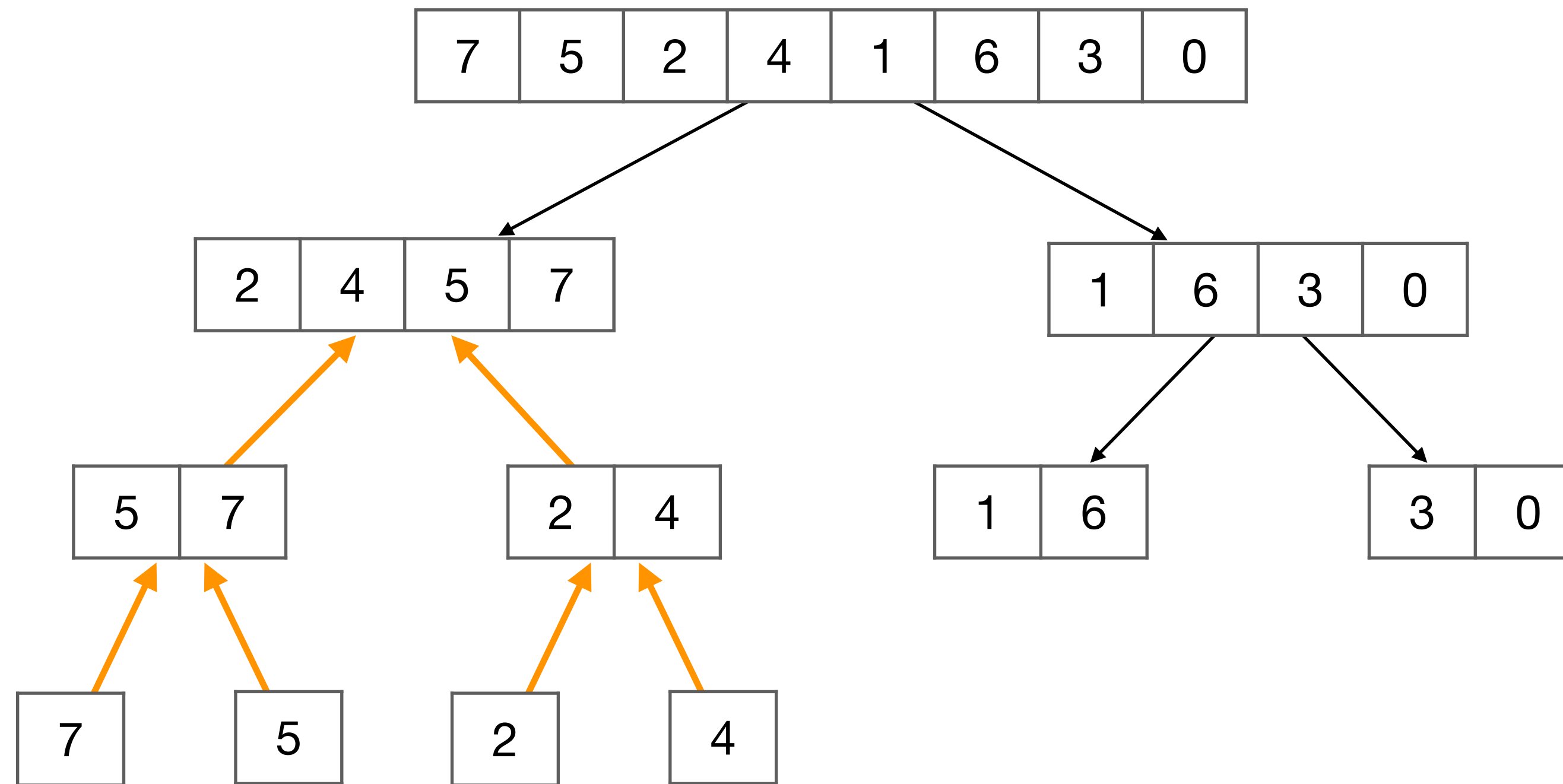
Visualisation



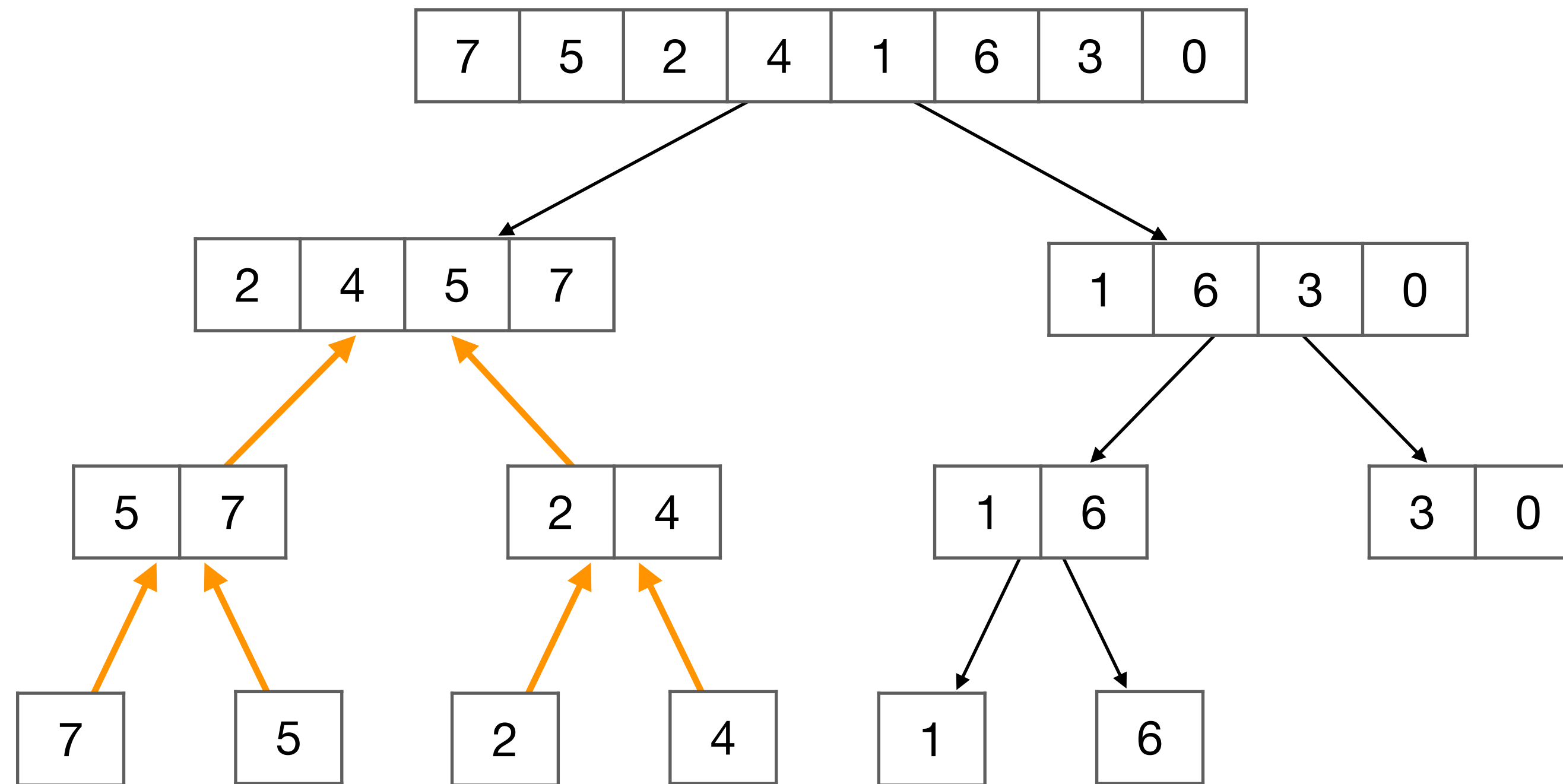
Visualisation



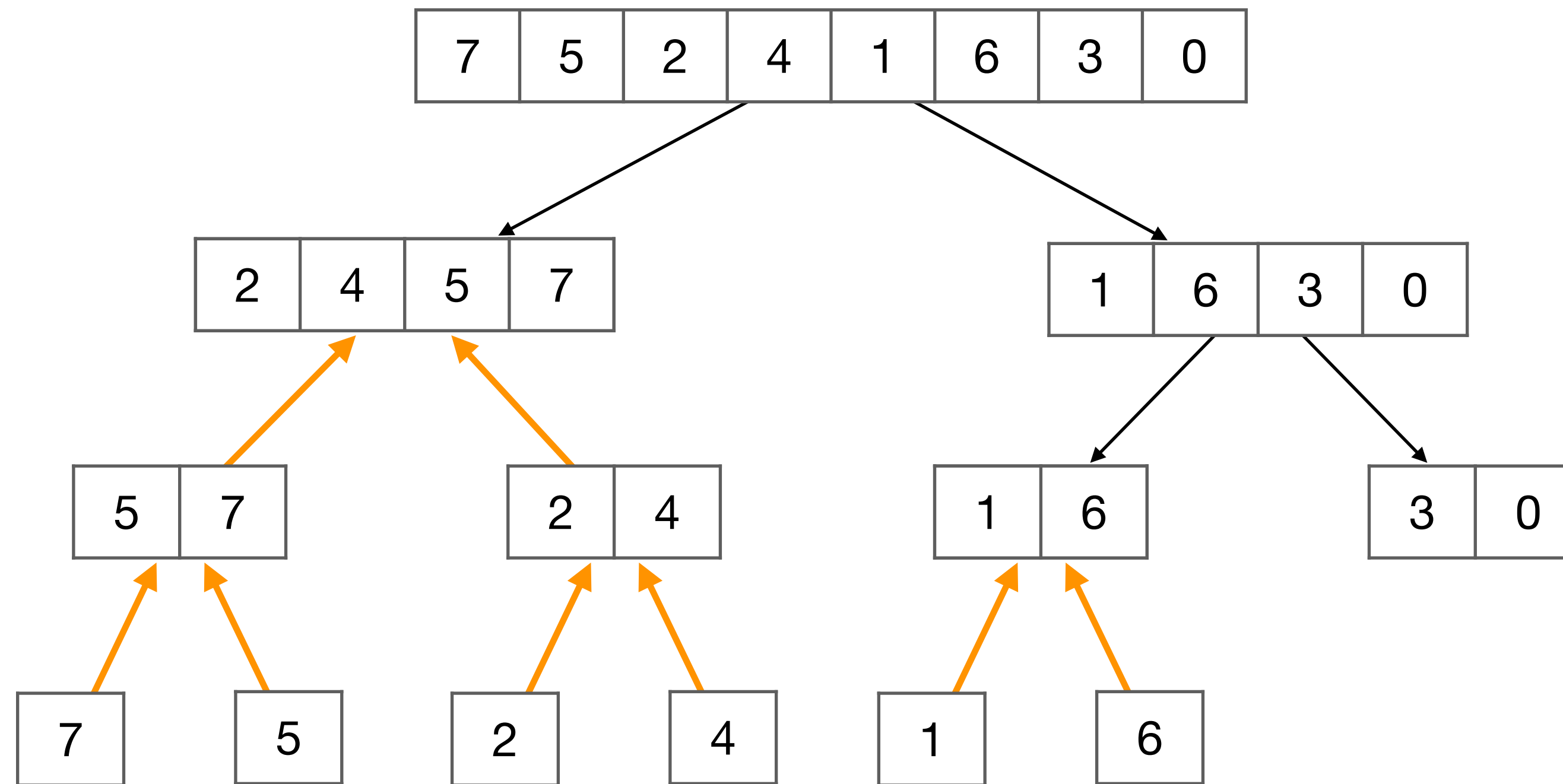
Visualisation



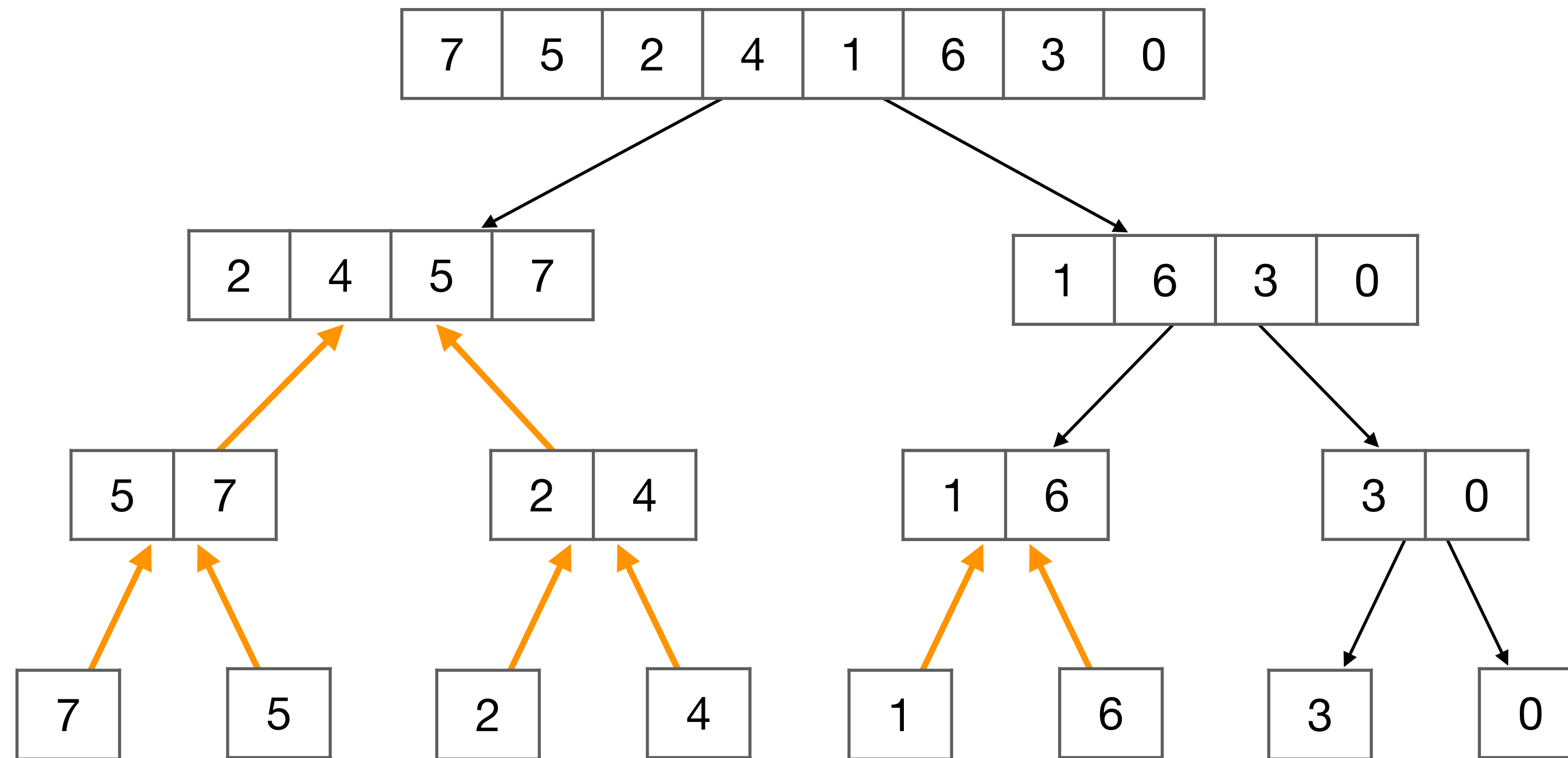
Visualisation



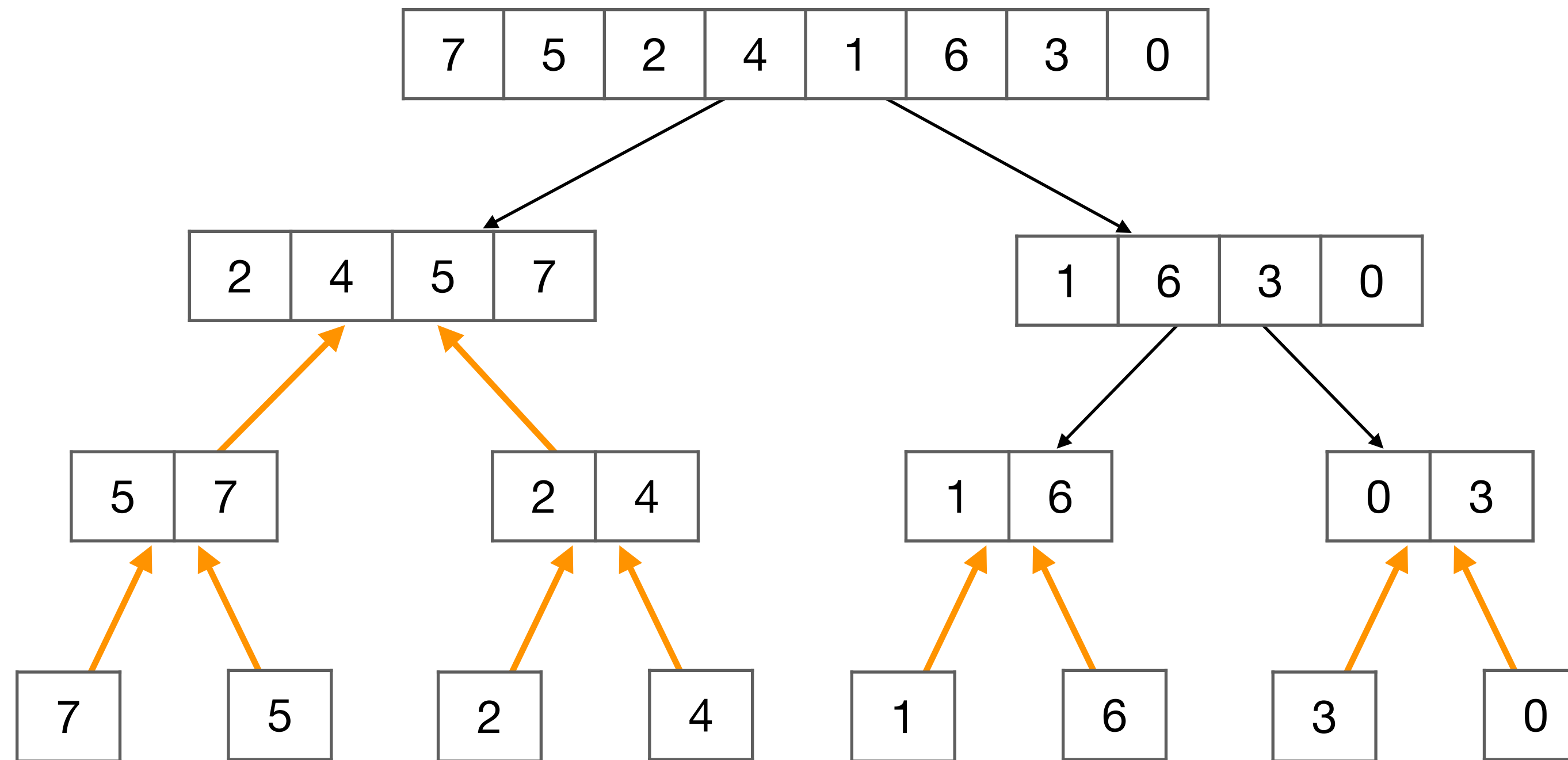
Visualisation



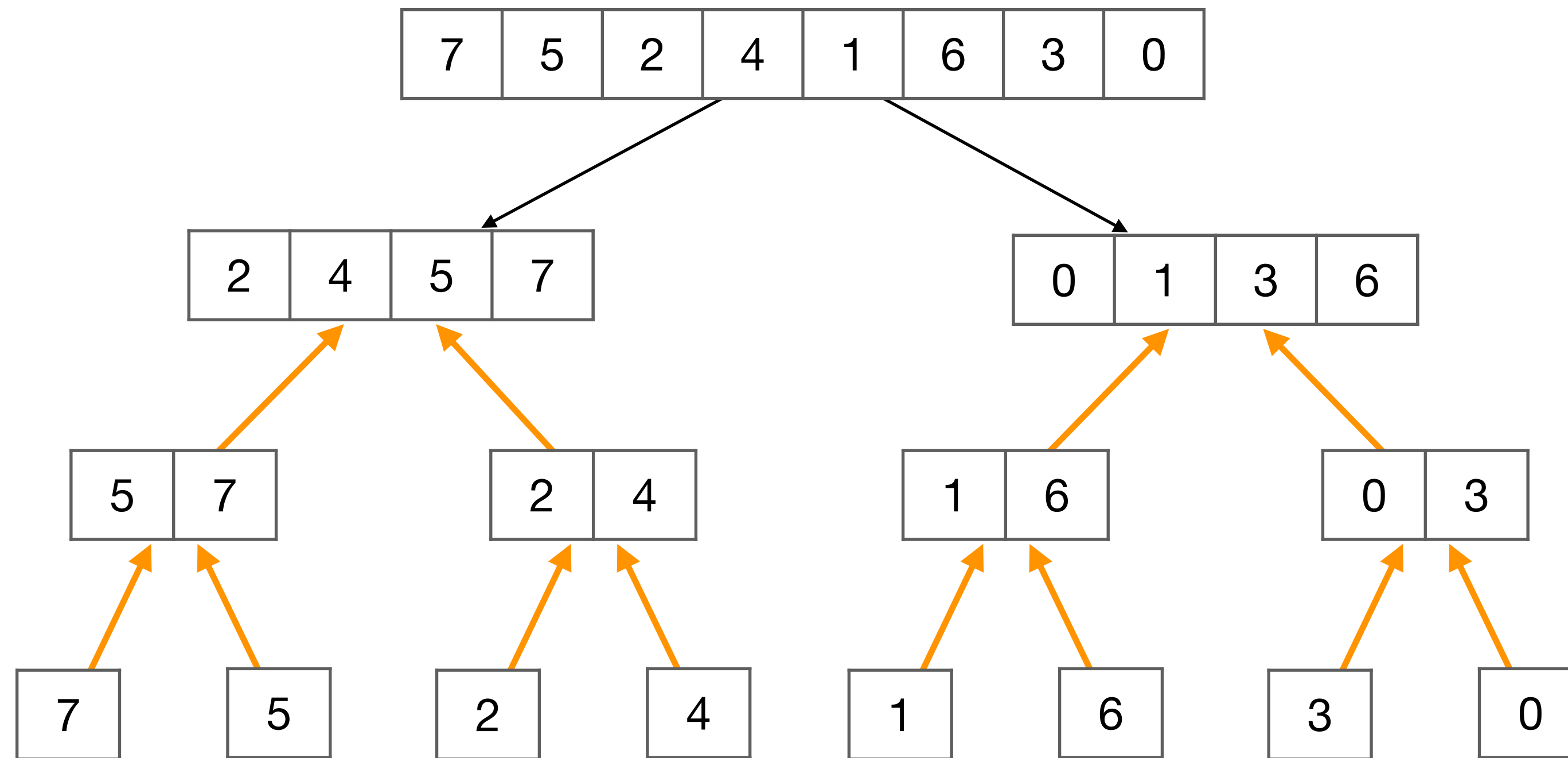
Visualisation



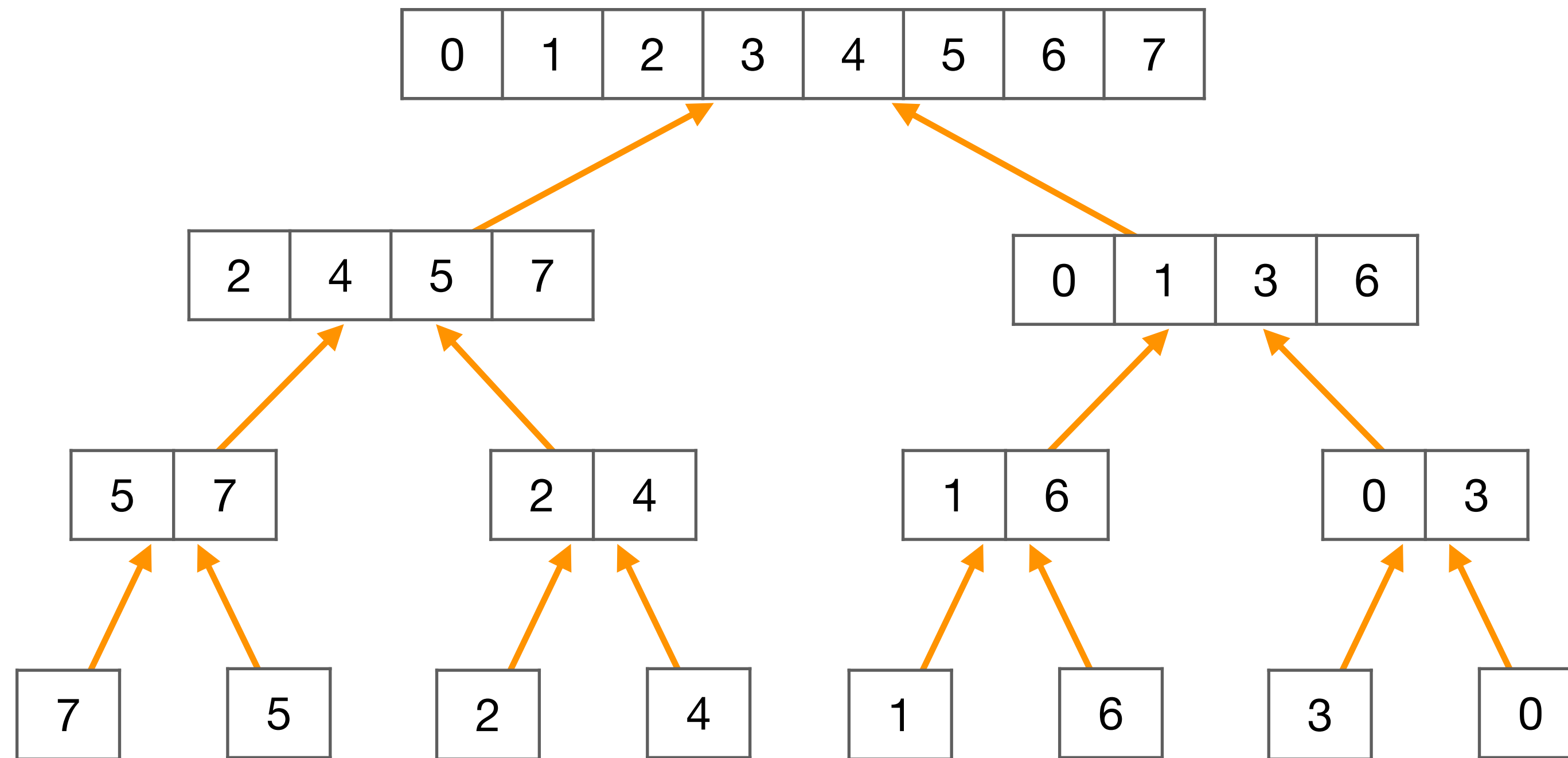
Visualisation



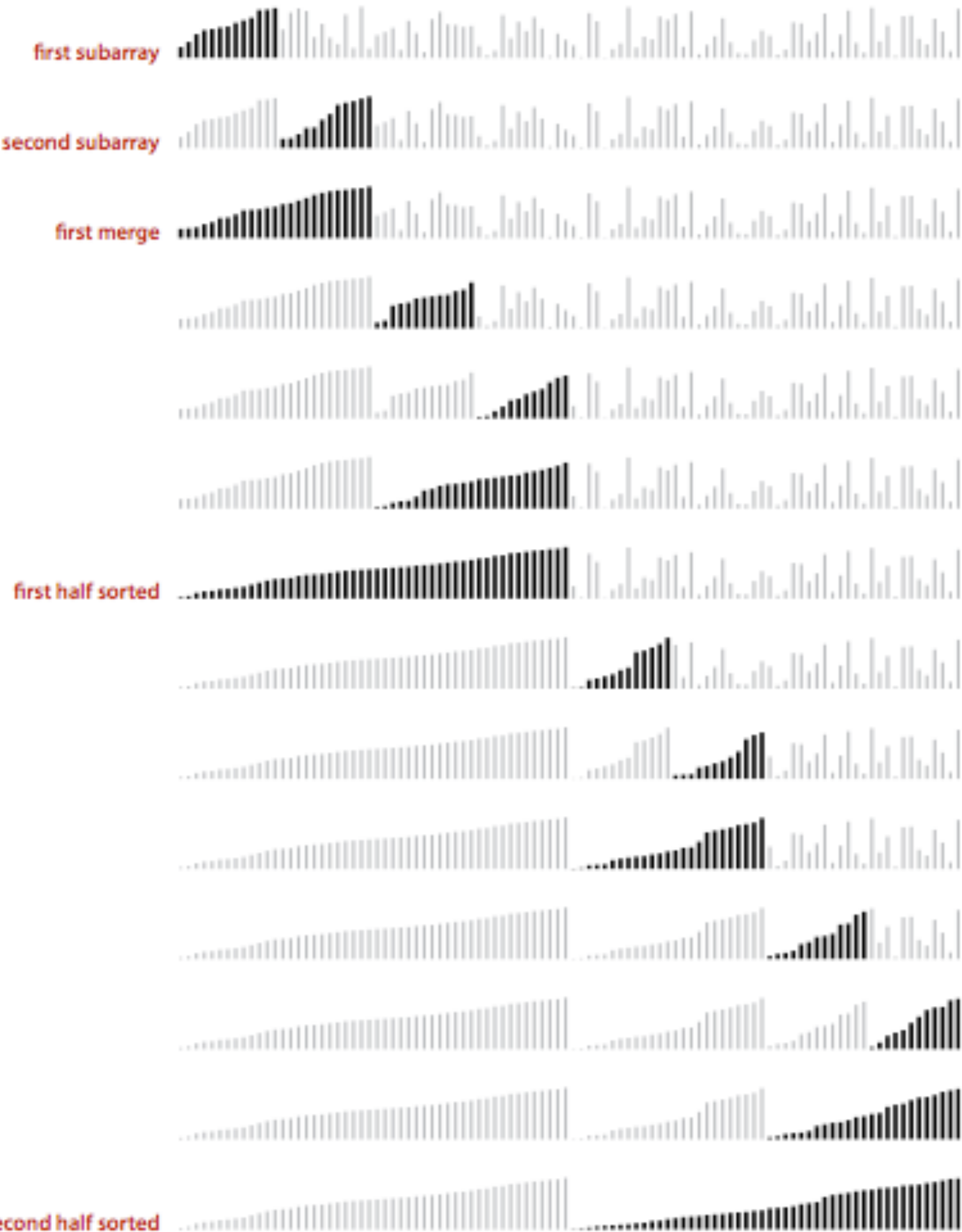
Visualisation



Visualisation



Visualisation



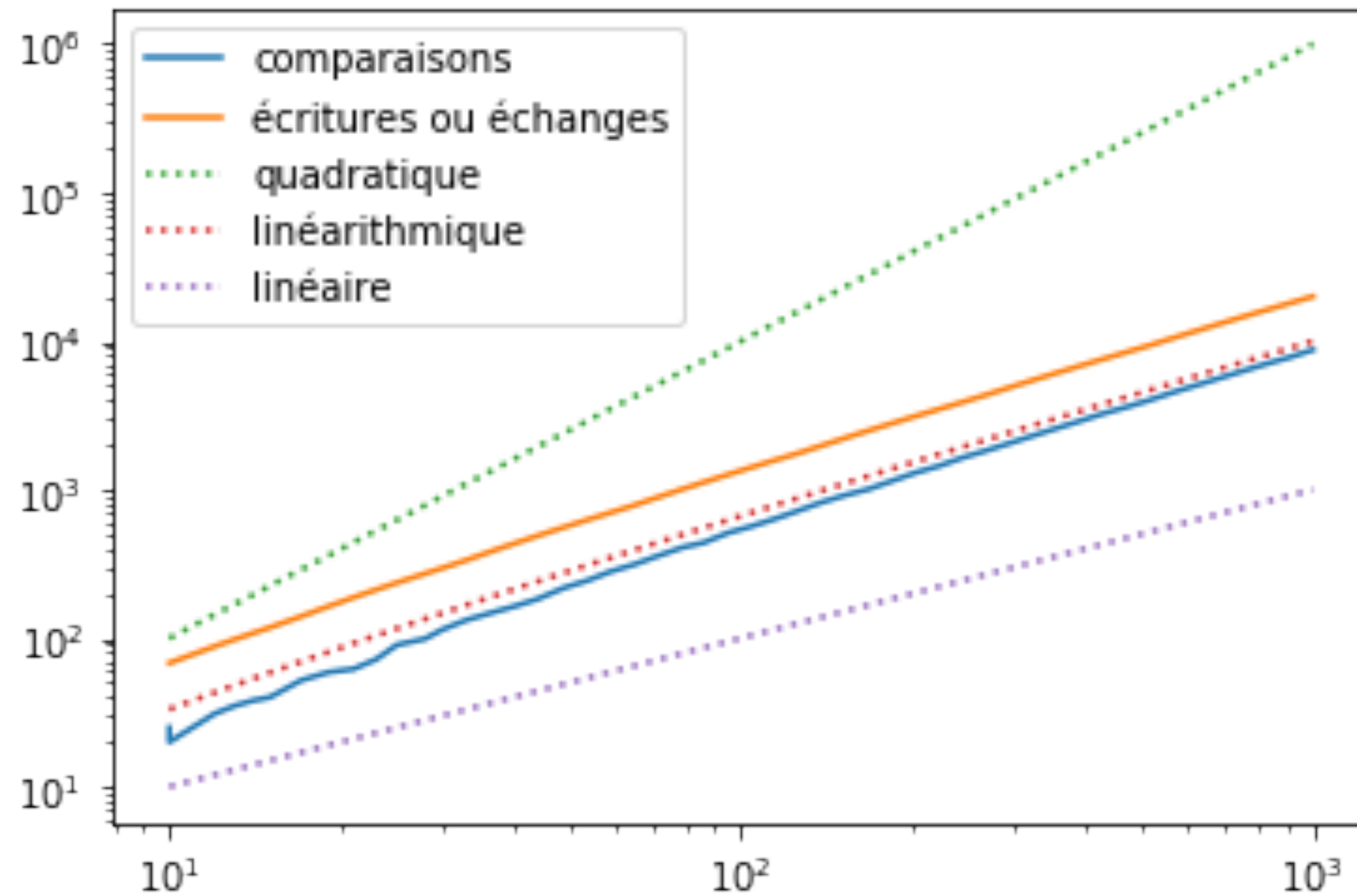
Visualisation des étapes successives du tri par fusion

(On ne visualise pas les tris de sous-tableaux de moins d'une dizaine d'éléments)

Complexité linéarithmique



Complexité du tri fusion, tableau aléatoire



N	Comp.	Ecr.
10	25	68
19	59	164
37	150	390
71	354	880
138	818	1972
268	1826	4336
517	3992	9326
1000	8731	19952

Complexité linéarithmique



$$C(1) = 0$$

appels récurrents

$$C(n) = \overset{\text{fusion}}{n} + 2 \cdot C\left(\frac{n}{2}\right)$$

Complexité linéarithmique



$$C(1) = 0$$

appels récurrents

$$C(n) = \text{fusion } n + 2 \cdot C\left(\frac{n}{2}\right)$$

$$C(n) = n + 2 \cdot \frac{n}{2} + 4 \cdot C\left(\frac{n}{4}\right)$$

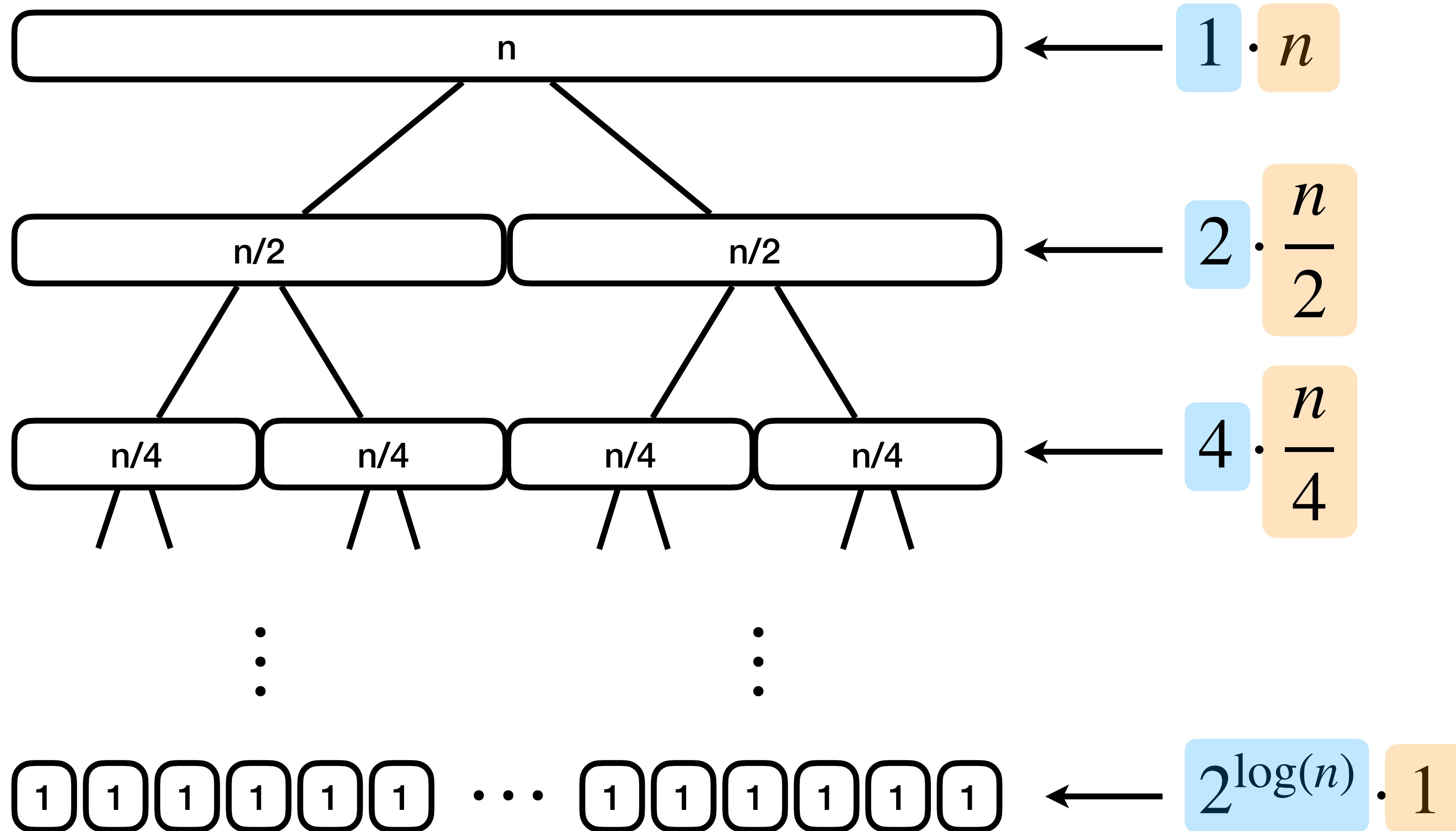
$$C(n) = n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + 8 \cdot C\left(\frac{n}{8}\right)$$

$$C(n) = n \cdot \log_2 n$$

Complexité linéarithmique



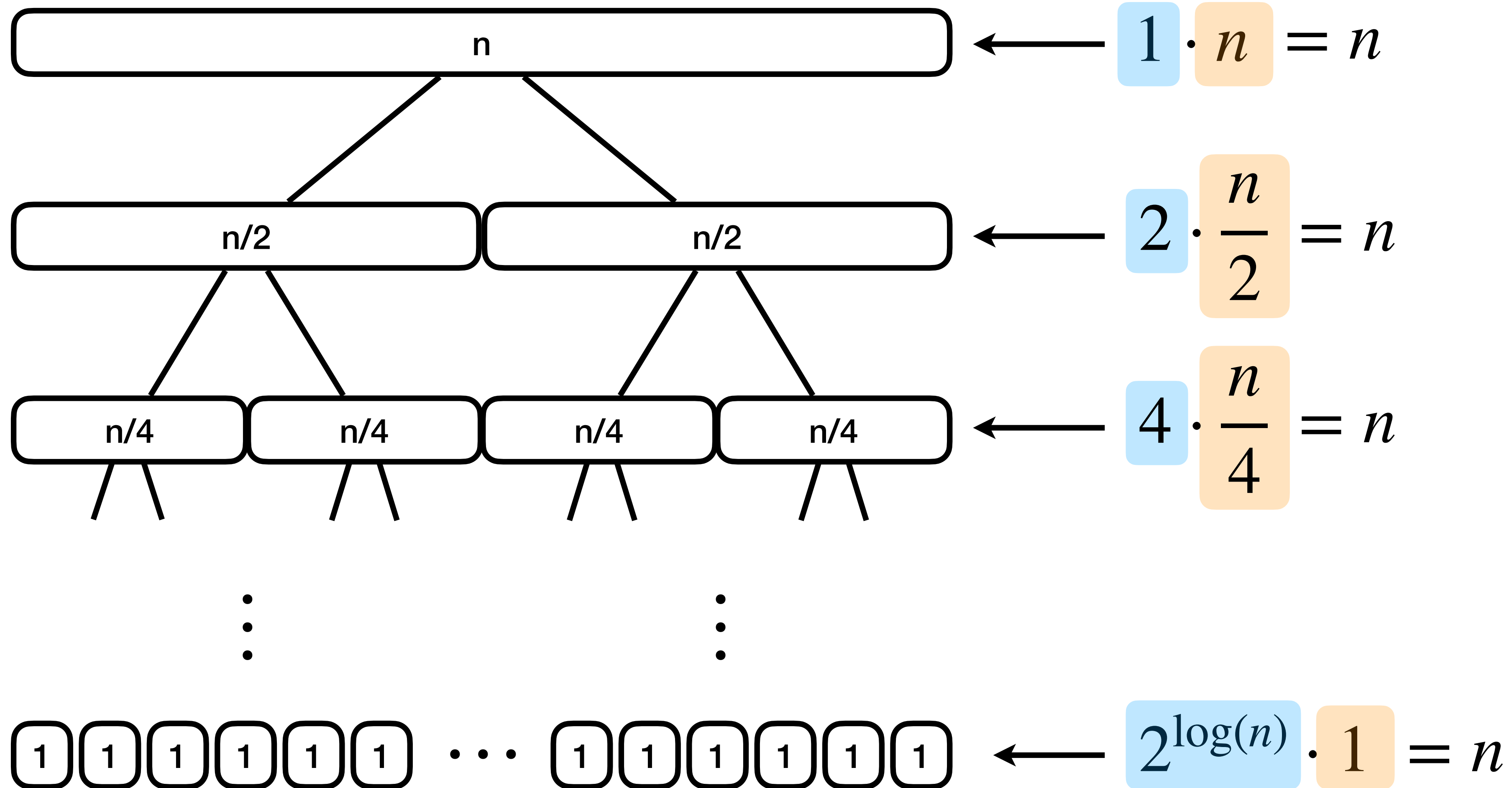
appels récurrents *fusion*



Complexité linéarithmique



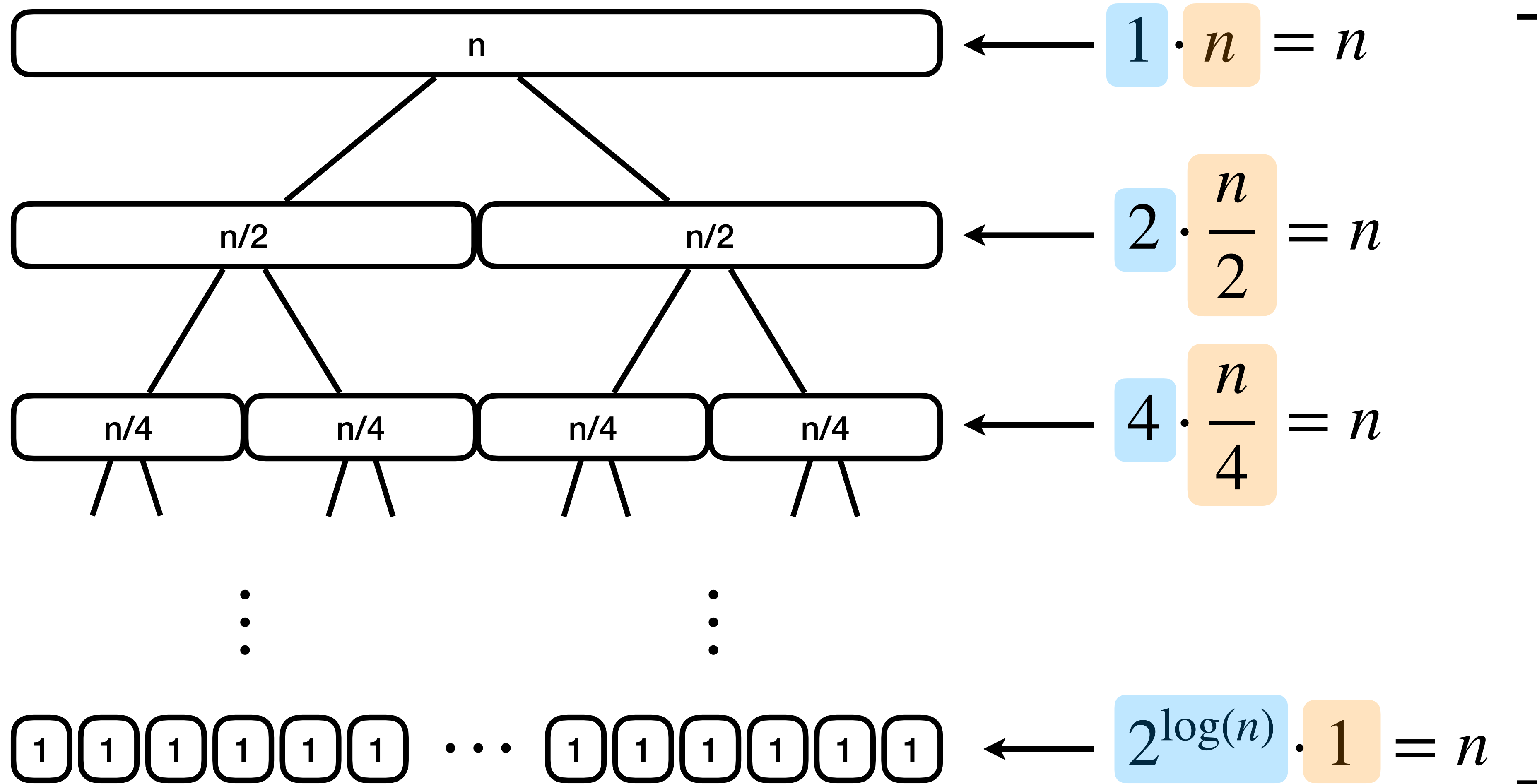
appels récurrents *fusion*



Complexité linéarithmique



appels récurrents *fusion*



Cout par étage
 $\downarrow$
 $O(n \log(n))$
 $\uparrow$
Nombre d'étages

Autres caractéristiques



Complexité spatiale (mémoire)

- Besoin d'un tableau auxiliaire de même taille que l'original

Stabilité

- Si $L(i)$ et $R(j)$ sont égaux
- On choisit $L(i)$
- Ce qui maintient l'ordre relatif

```
si  $L(i) \leq R(j)$ , alors  
     $A(k) \leftarrow L(i)$   
    incrémenter  $i$   
sinon  
     $A(k) \leftarrow R(j)$   
    incrémenter  $j$   
fin si
```

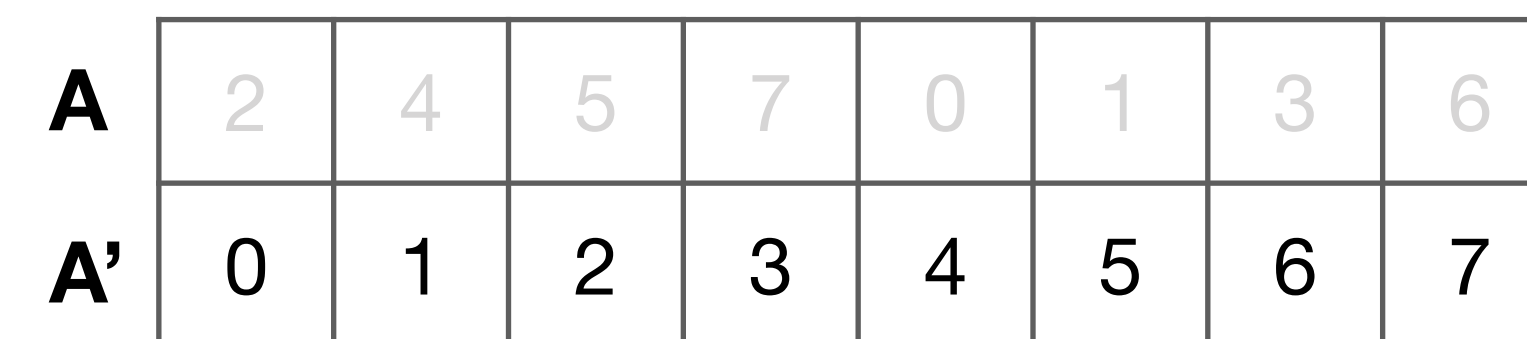
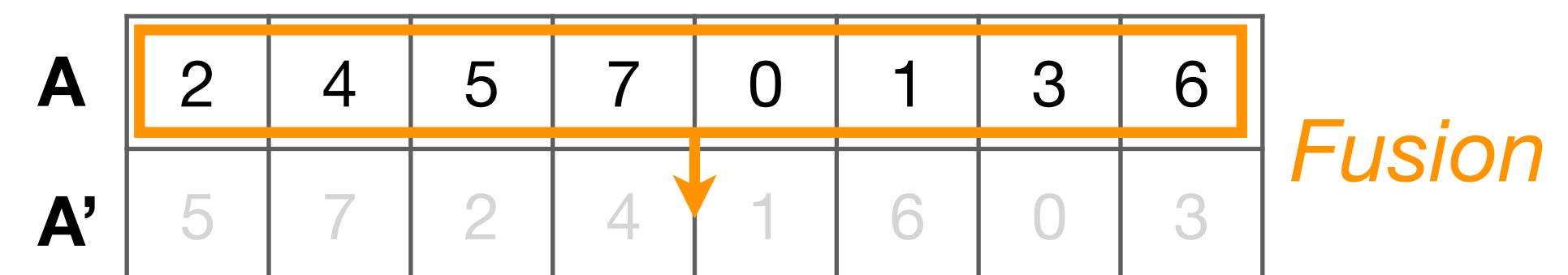
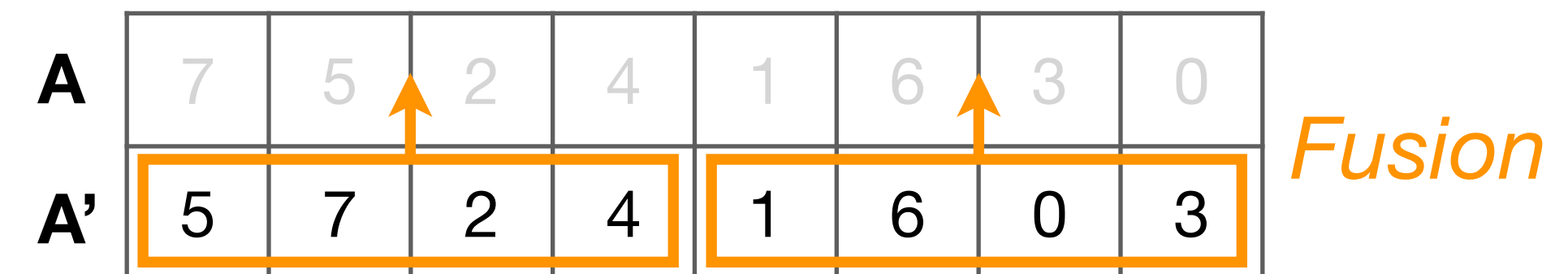
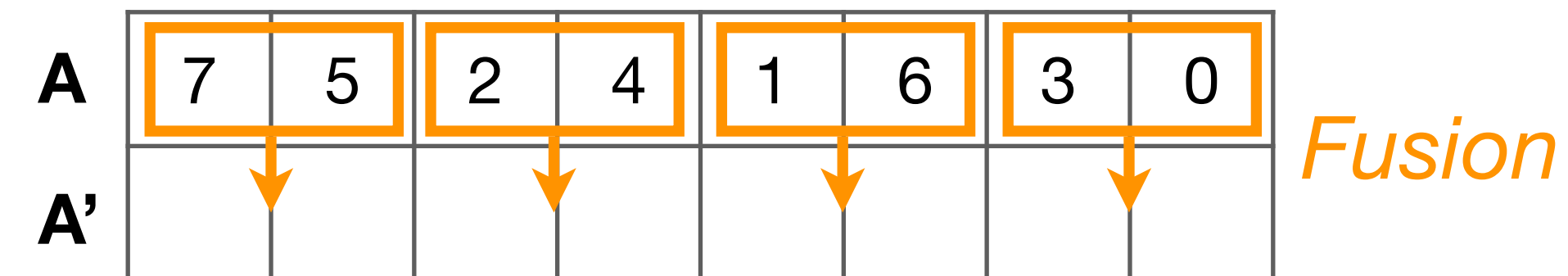
Minimiser le nombre d'écritures



Fusionner copie 2x, de **A** à **LR**, puis de **LR** à **A**

Pour ne faire qu'une copie :

- Allouer une seule fois un tableau **A'** de même taille que **A**
- Utiliser **A** comme si c'était **LR**
- Copier de **LR** à **A'** pour fusionner
- Utiliser **A'** comme le nouveau **A**.



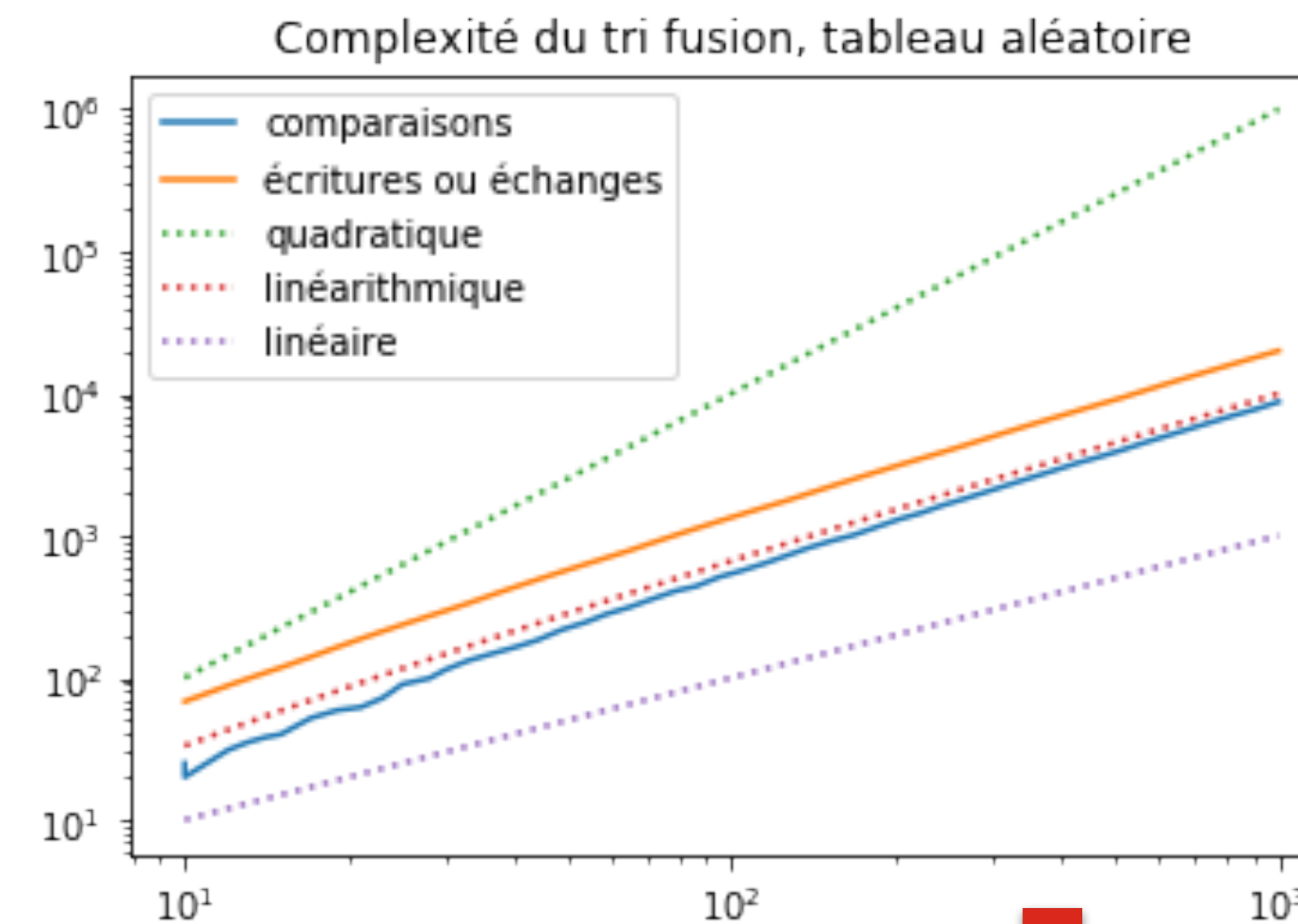
Minimiser le nombre d'écritures



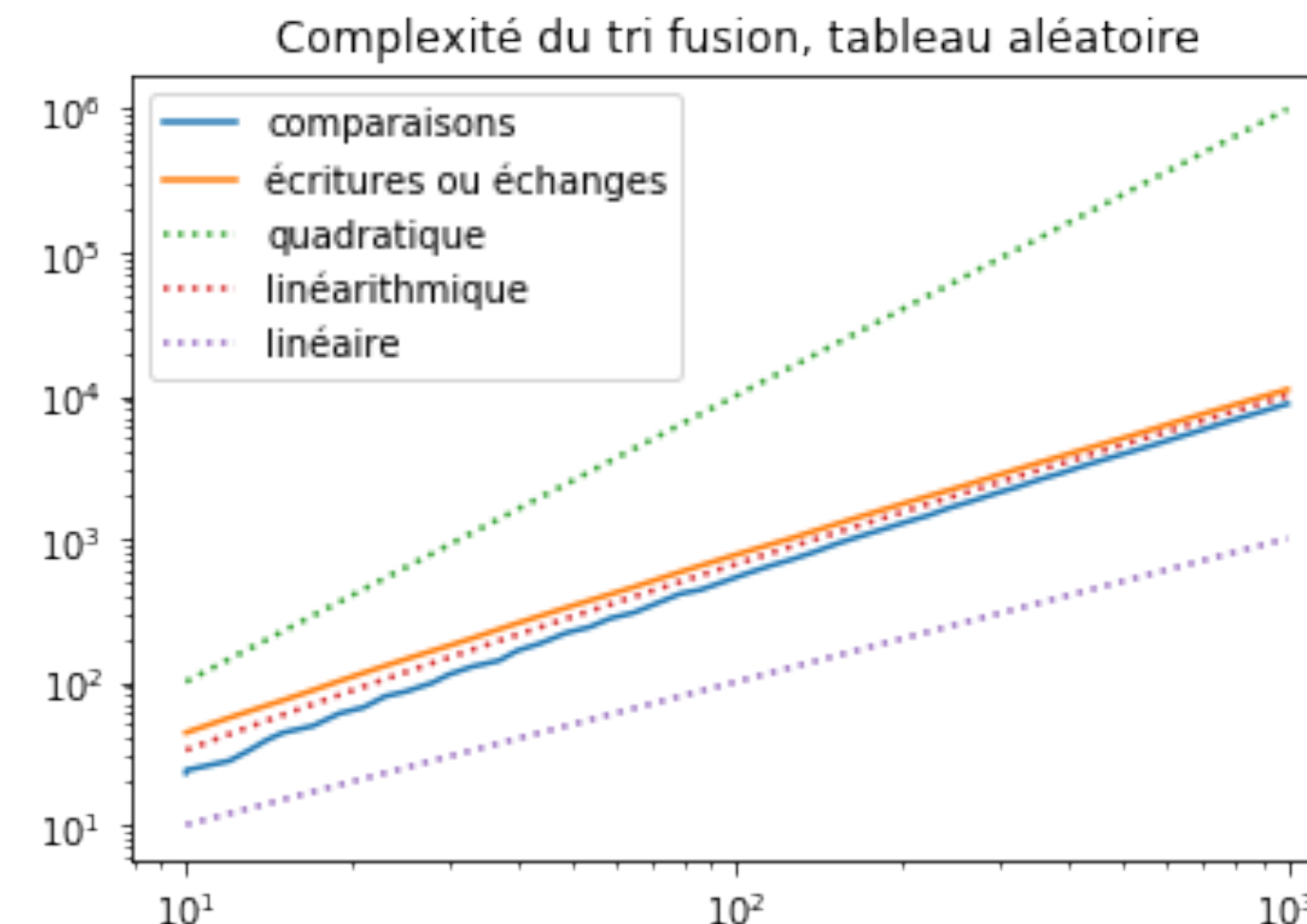
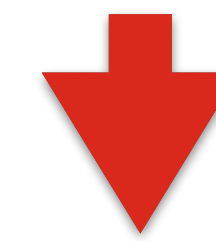
Fusionner copie 2x, de **A** à **LR**, puis de **LR** à **A**

Pour ne faire qu'une copie :

- Allouer une seule fois un tableau **A'** de même taille que **A**
- Utiliser **A** comme si c'était **LR**
- Copier de **LR** à **A'** pour fusionner
- Utiliser **A'** comme le nouveau **A**.



N	Comp.	Ecr.
10	25	68
19	59	164
37	150	390
71	354	880
138	818	1972
268	1826	4336
517	3992	9326
1000	8731	19952



N	Comp.	Ecr.
10	23	44
19	60	101
37	140	232
71	348	511
138	802	1124
268	1830	2436
517	4013	5180
1000	8721	10976

Autres optimisations possibles



- La récursion est chère pour les petits tableaux.
- Insertion sort est alors plus rapide.
- Des méthodes hybrides (ex. TimSort) mélangent tris par fusion et par insertion
- Une mise en oeuvre itérative (bottom up) est également possible
- Une mise en oeuvre en place est possible, mais au prix d'une complexité en temps en $O(n \log^2(n))$

HE^{VD} IG Demo

Noir - concernés par
l'appel à TriFusion

Bleu - fusionnés

Gris - pas concernés
par la procédure en
cours

Point (.) - appel à
TriFusion sur un
tableau de taille 1

```
TriFusion(1,12)
TriFusion(1,6)
TriFusion(1,3)
TriFusion(1,2) ...
Fusionner(1,1,2)
Fusionner(1,2,3)
TriFusion(4,6)
TriFusion(4,5) ...
Fusionner(4,4,5)
Fusionner(4,5,6)
Fusionner(1,3,6)
TriFusion(7,12)
TriFusion(7,9)
TriFusion(7,8) ...
Fusionner(7,7,8)
Fusionner(7,8,9)
TriFusion(10,12)
TriFusion(10,11) ...
Fusionner(11,11,12)
Fusionner(10,11,12)
Fusionner(7,9,12)
Fusionner(1,6,12)
```

```
E X E M P L E D E T R I
E X E M P L E D E T R I
E X E M P L E D E T R I
E X E M P L E D E T R I
E X E M P L E D E T R I
E E X M P L E D E T R I
E E X M P L E D E T R I
E E X M P L E D E T R I
E E X L M P E D E T R I
E E L M P X E D E T R I
E E L M P X E D E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
E E L M P X D E E T R I
D E E E E I L M P R T X
```

