Tris optimaux et tris linéaires

Yann Strozecki
yann.strozecki@uvsq.fr

Octobre 2016

Tous les tris ne sont pas comparables

Les algorithmes que nous avons vus jusqu'ici reposent uniquement sur des comparaisons des éléments d'entrée et sont de complexité $O(n^2)$.

Nous allons donner dans ce cours :

- des tris de complexité optimale $O(n \log n)$
- un algorithme de tri de complexité linéaire mais qui nécessite certaines hypothèses
- un modèle de calcul pour les tris différent

Tri Fusion

- Machine à trier des cartes perforées en 1938 ;
- 1er algo de tri fusion écrit par Von Neumann pour l'EDVAC en 1945 ;
- Basé sur le paradigme

«Diviser pour régner»

Diviser Pour Régner

- Séparer le problème en plusieurs sous-problèmes similaires au problème initial.
- 3 étapes :
  - **Diviser** le problème en un certain nombre de sous-problèmes
  - **Régner** sur les sous-problèmes en les résolvant
  - **Combiner** les solutions des sous-problèmes en une solution unique au problème initial.

Tri Fusion (2)

3 étapes :

- **Diviser** le tableau de n éléments à trier en 2 sous-tableaux de $\frac{n}{2}$ éléments.
- **Régner** :
  - Tout tableau de longueur 1 est trié.
  - Trier les 2 sous-tableaux récursivement à l'aide du Tri Fusion
- **Combiner** : Action Principale : la Fusion
  - Fusionner les 2 sous-tableaux triés pour produire un tableau trié.

Tri Fusion (3)

Algorithme 1 Tri Fusion

TriFusion(T : tableau d'entiers, g : entier, d : entier)

▷ g ***et*** d ***sont les indices entre lesquels on veut trier le tableau. On suppose*** $g \leq d$.

Début

si $g < d$ **faire**

$m \leftarrow \lfloor \frac{g+d}{2} \rfloor$

TriFusion(T,g,m)

TriFusion(T,m+1,d)

Fusion(T,g,m,d)

fin si

Fin

Tri Fusion (4)

Algorithme de fusion vu en td.

Noter les entiers g et d qui servent à délimiter un sous-tableau. Technique très fréquente pour gérer intelligemment la mémoire. Ainsi on ne créé pas de nouveaux tableaux quand on divise un tableau en morceaux.

Tri Fusion (5)

Complexité pour n éléments

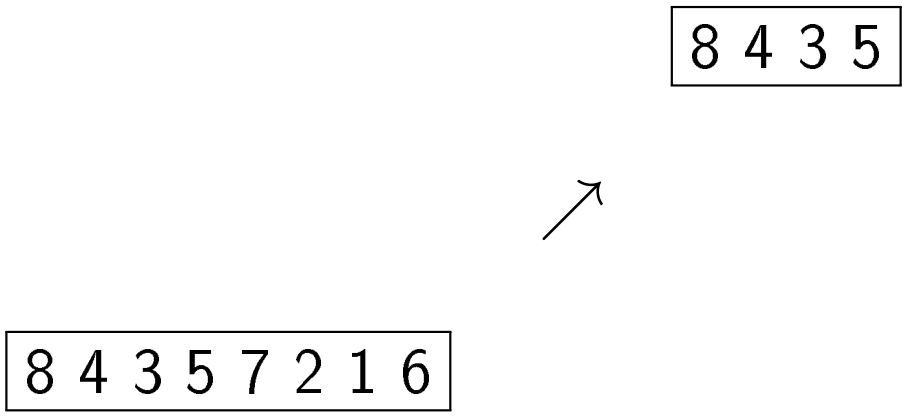
- Intuitivement il faut résoudre :

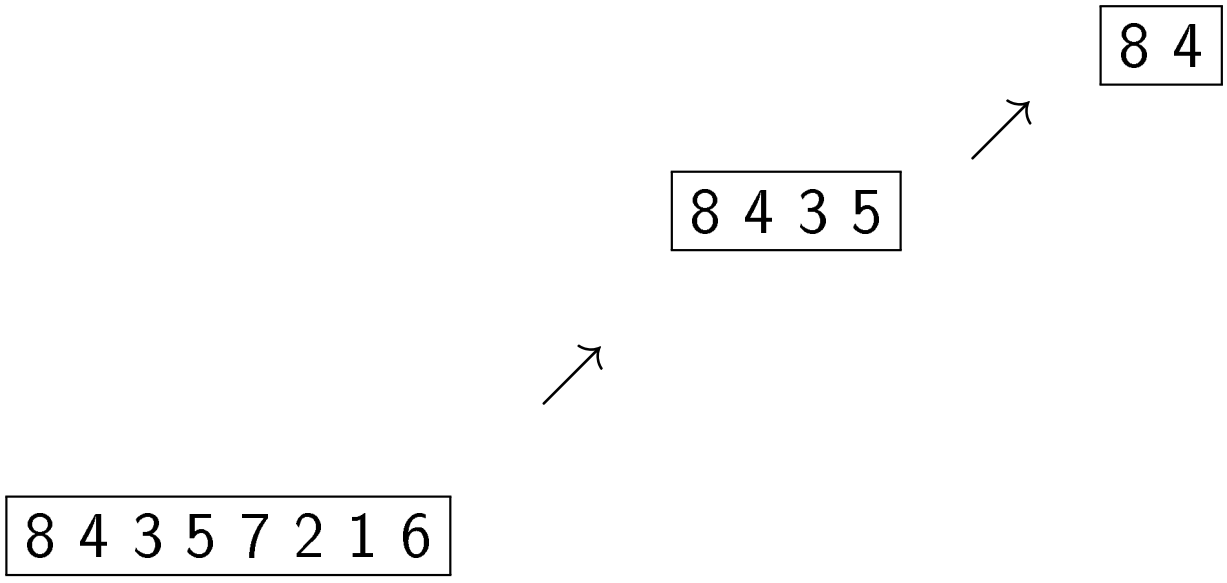
$$Tri(n) = 2 \times Tri(\frac{n}{2}) + O(n)$$

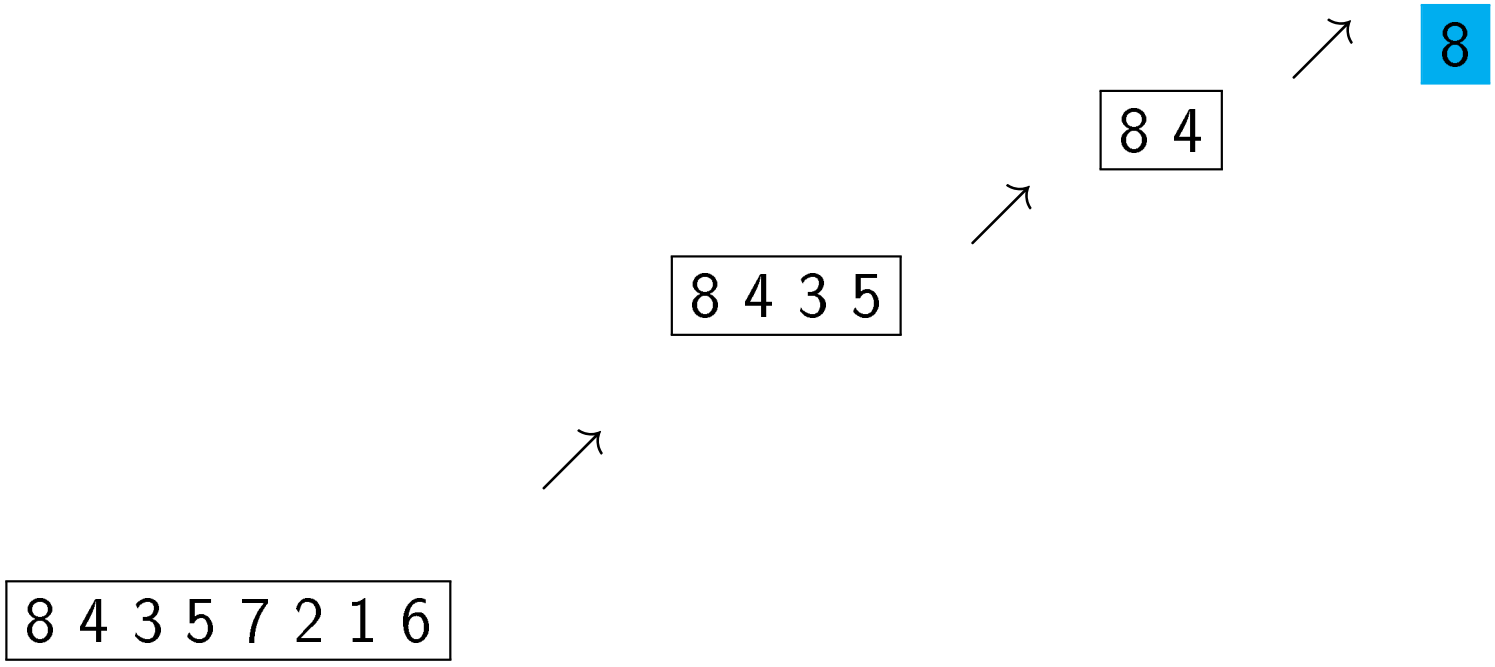
- $O(n)$: complexité de la fusion

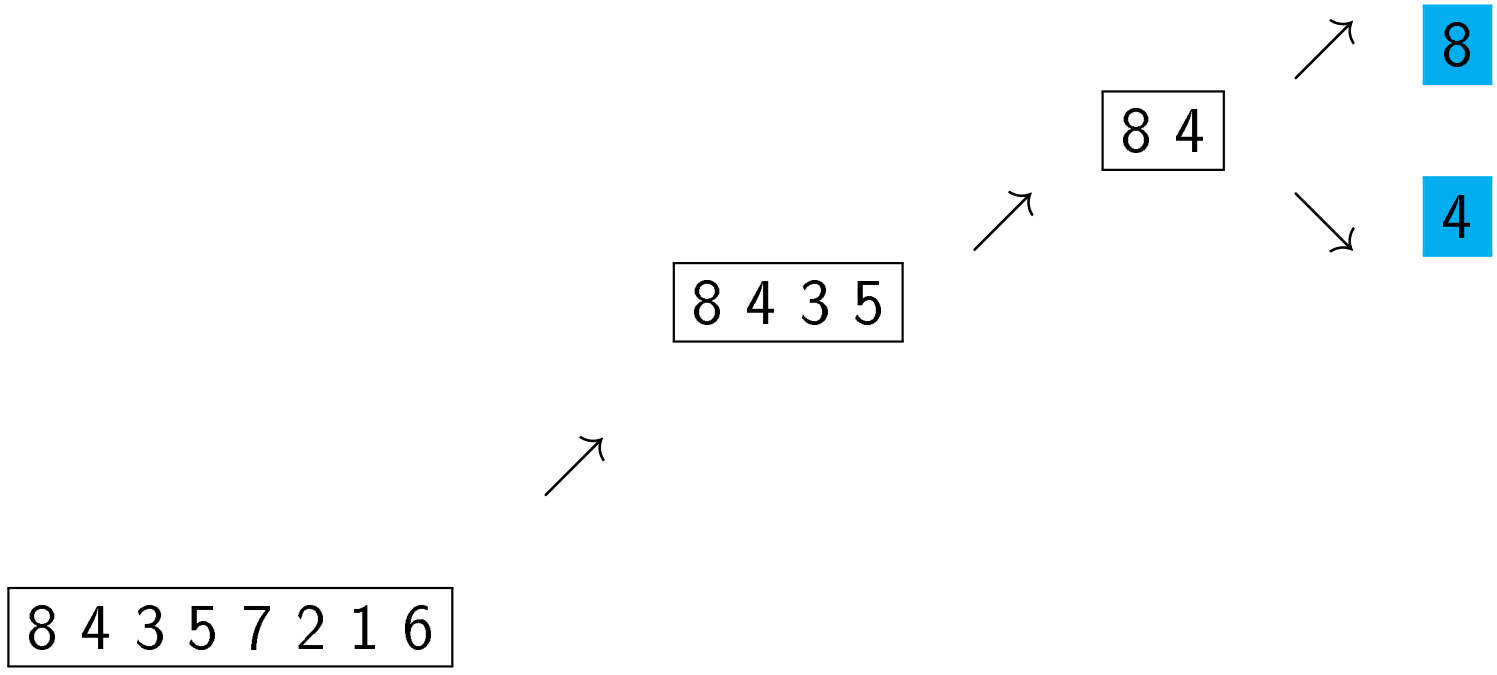
La complexité du tri fusion est en $O(n \log_2 n)$.
Preuve au tableau.

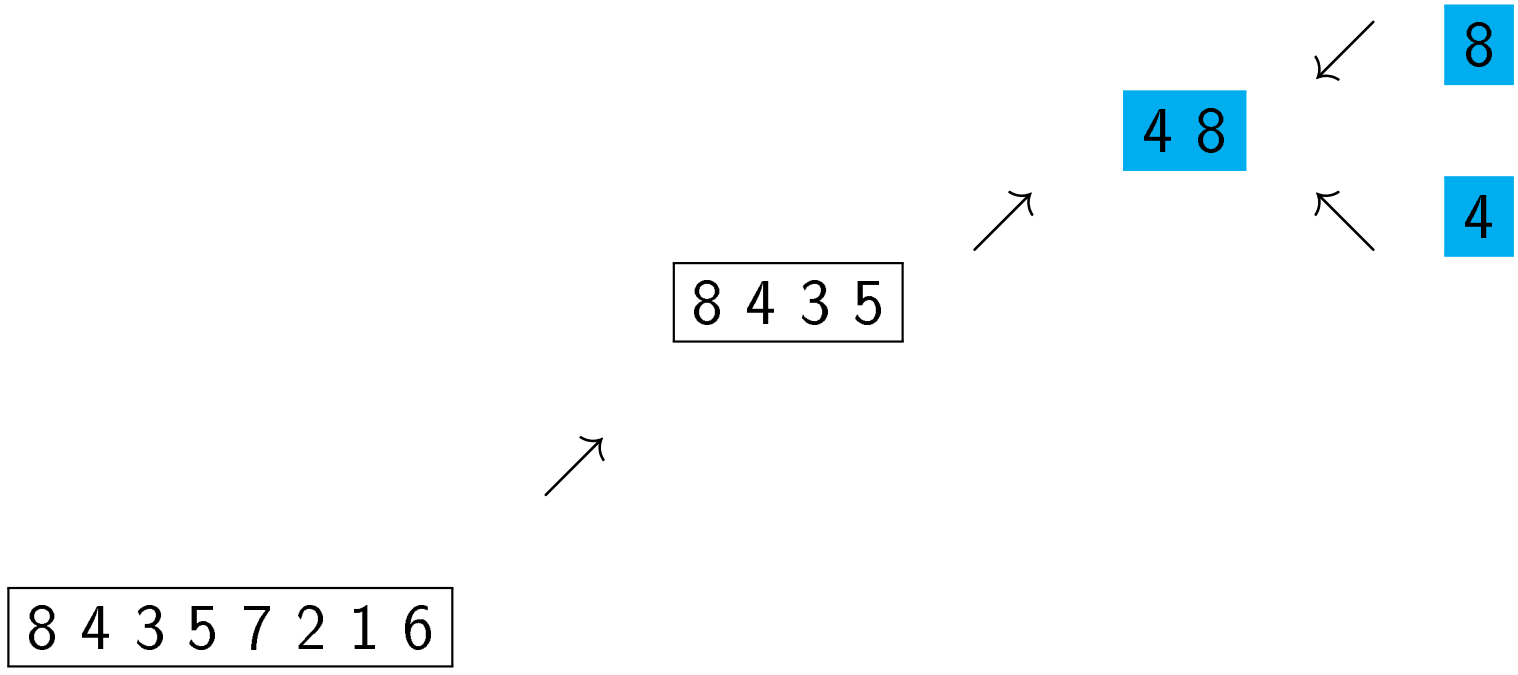
8 4 3 5 7 2 1 6

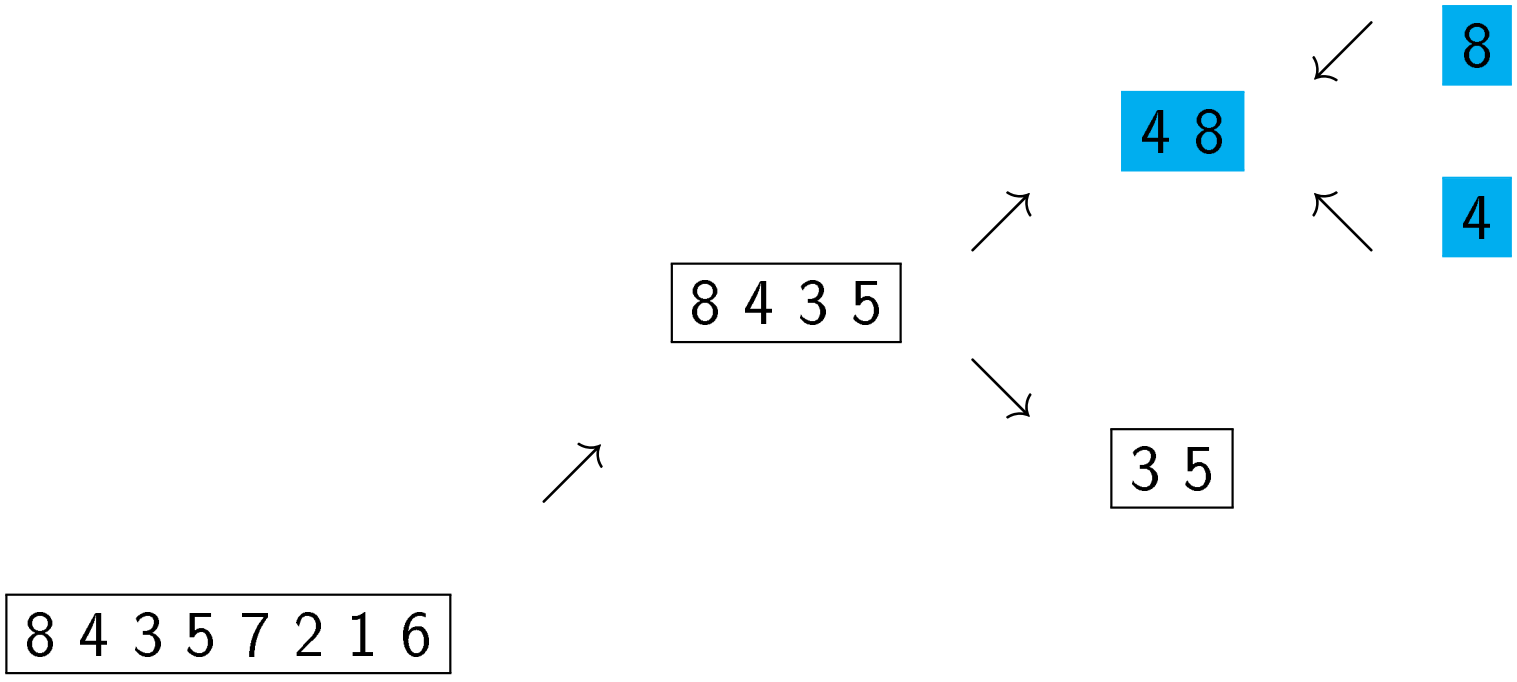

8 4 3 5
8 4 3 5 7 2 1 6

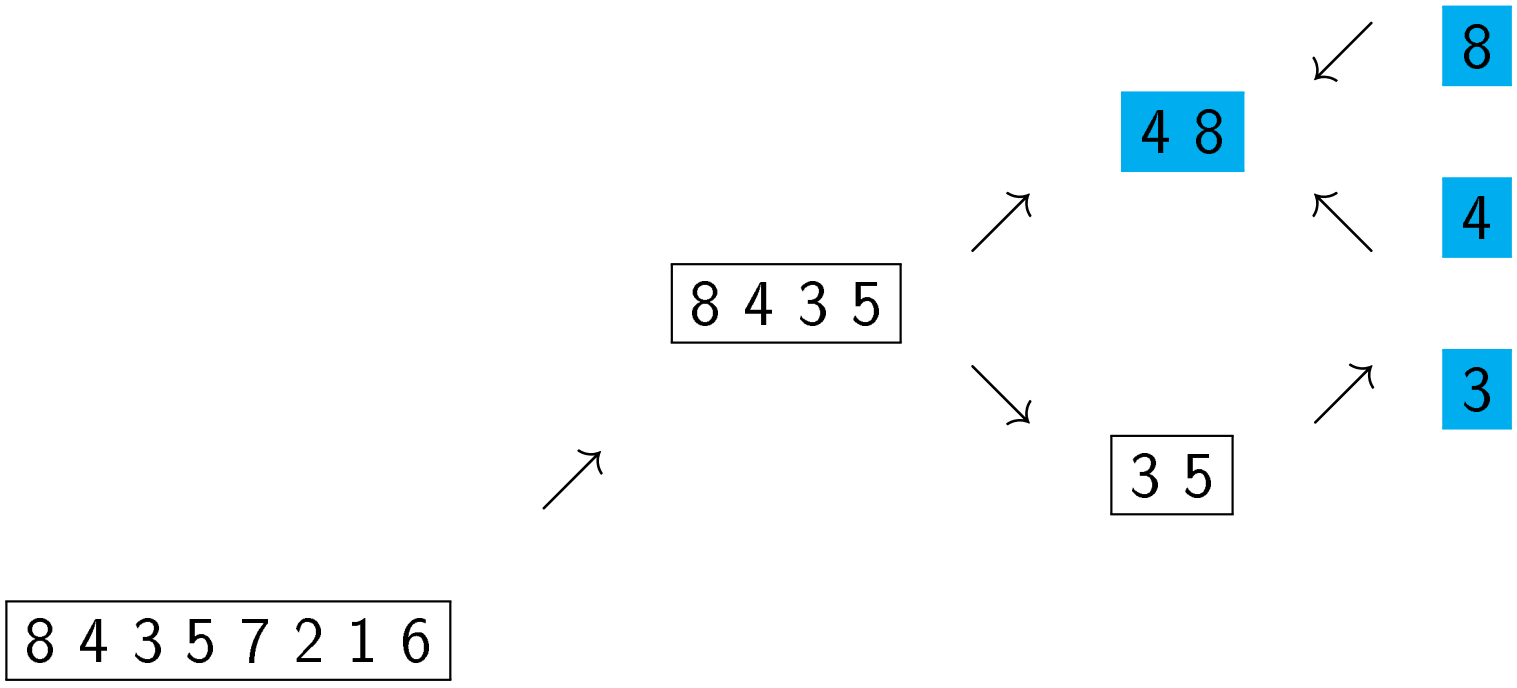

8 4
8 4 3 5
8 4 3 5 7 2 1 6

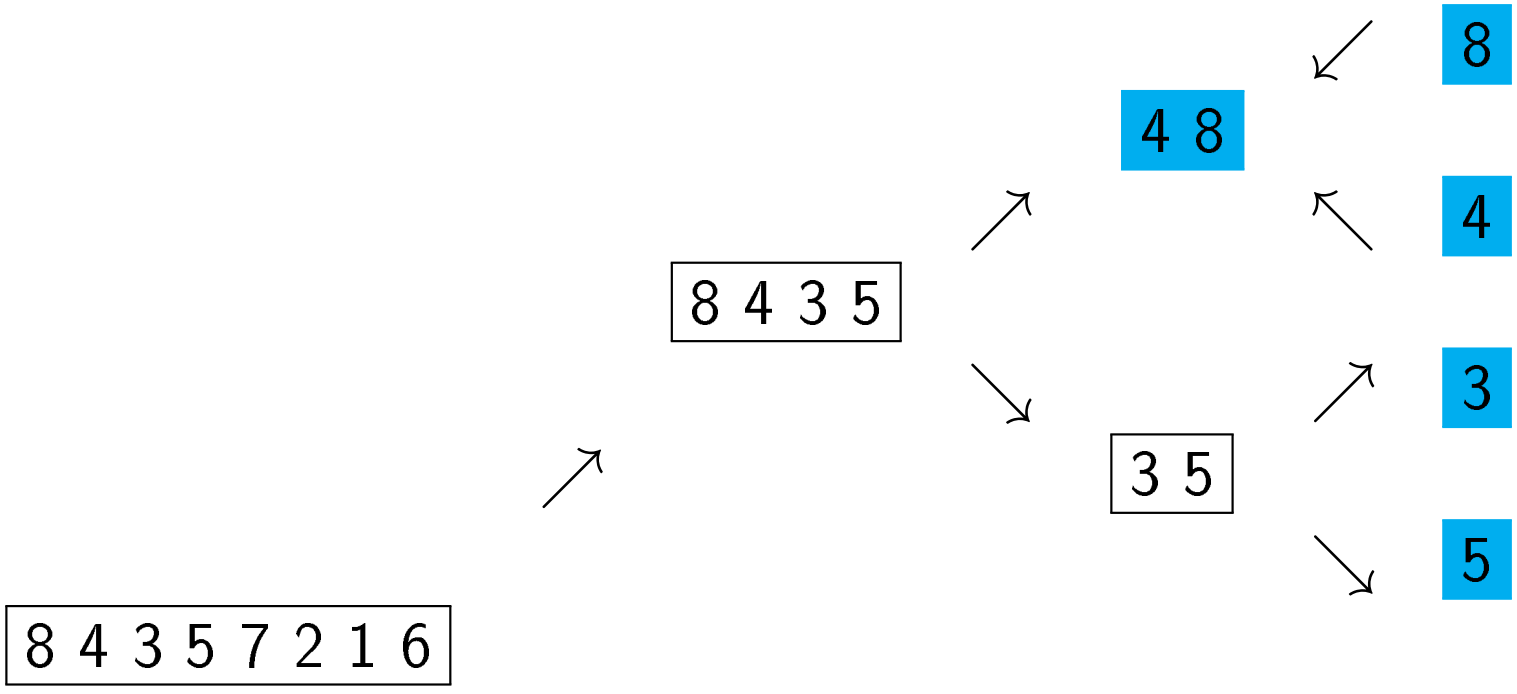

8
8 4
8 4 3 5
8 4 3 5 7 2 1 6

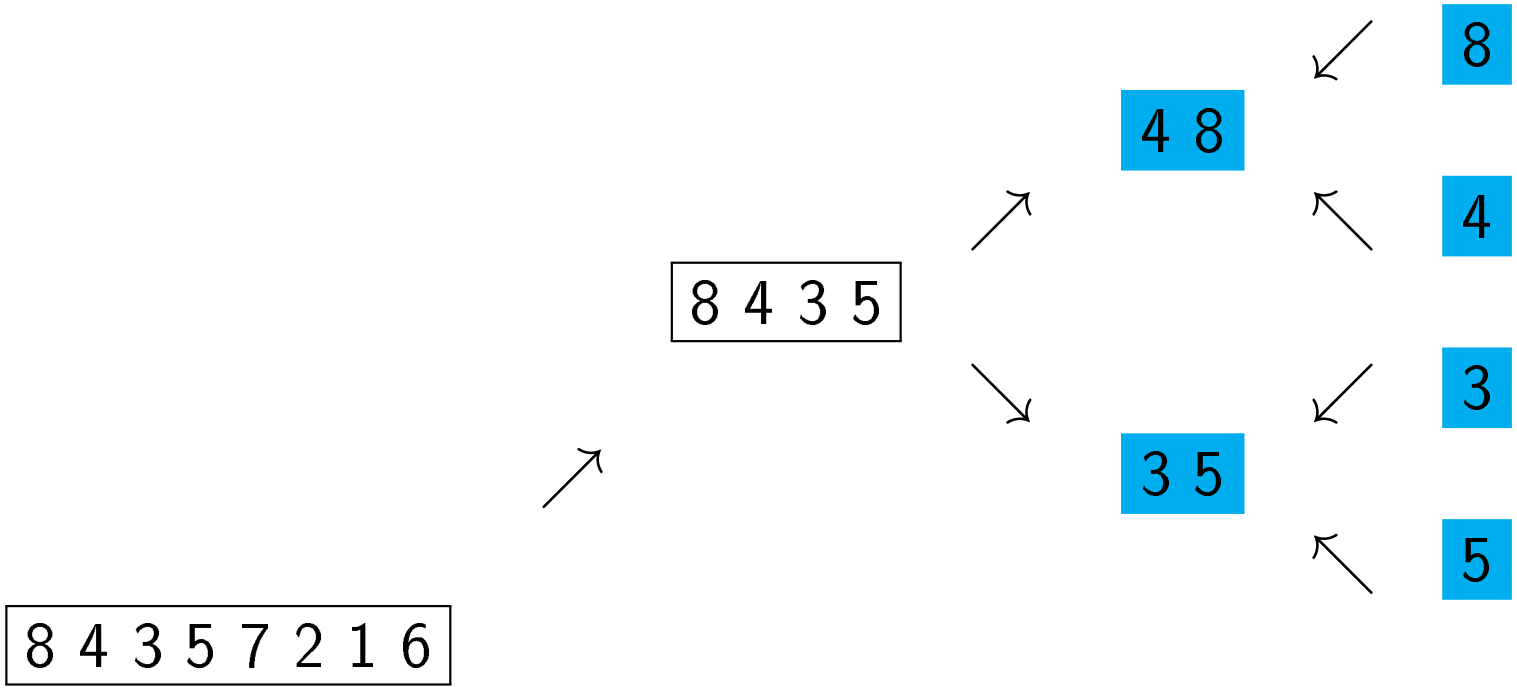

8 4 3 5 7 2 1 6
8 4 3 5
8 4
8
4

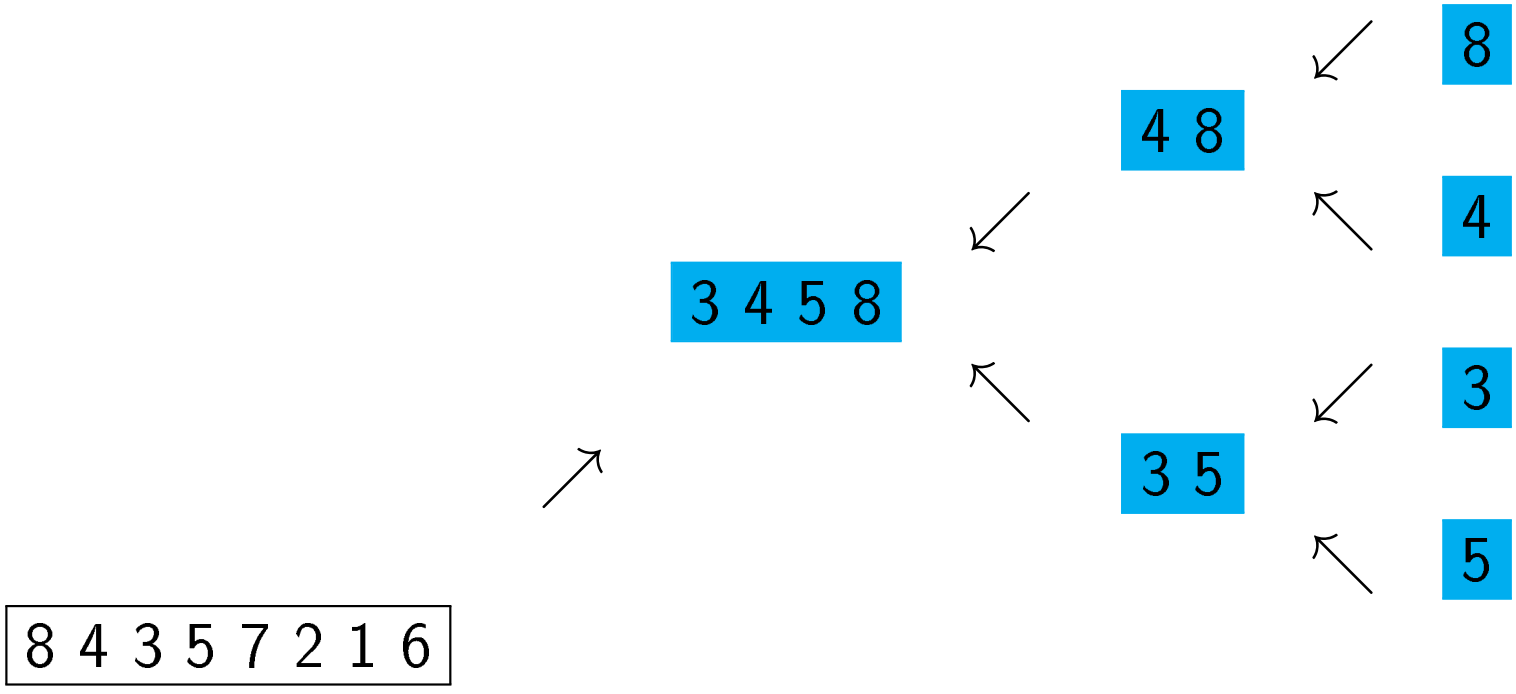

8
4 8
4
8 4 3 5
8 4 3 5 7 2 1 6

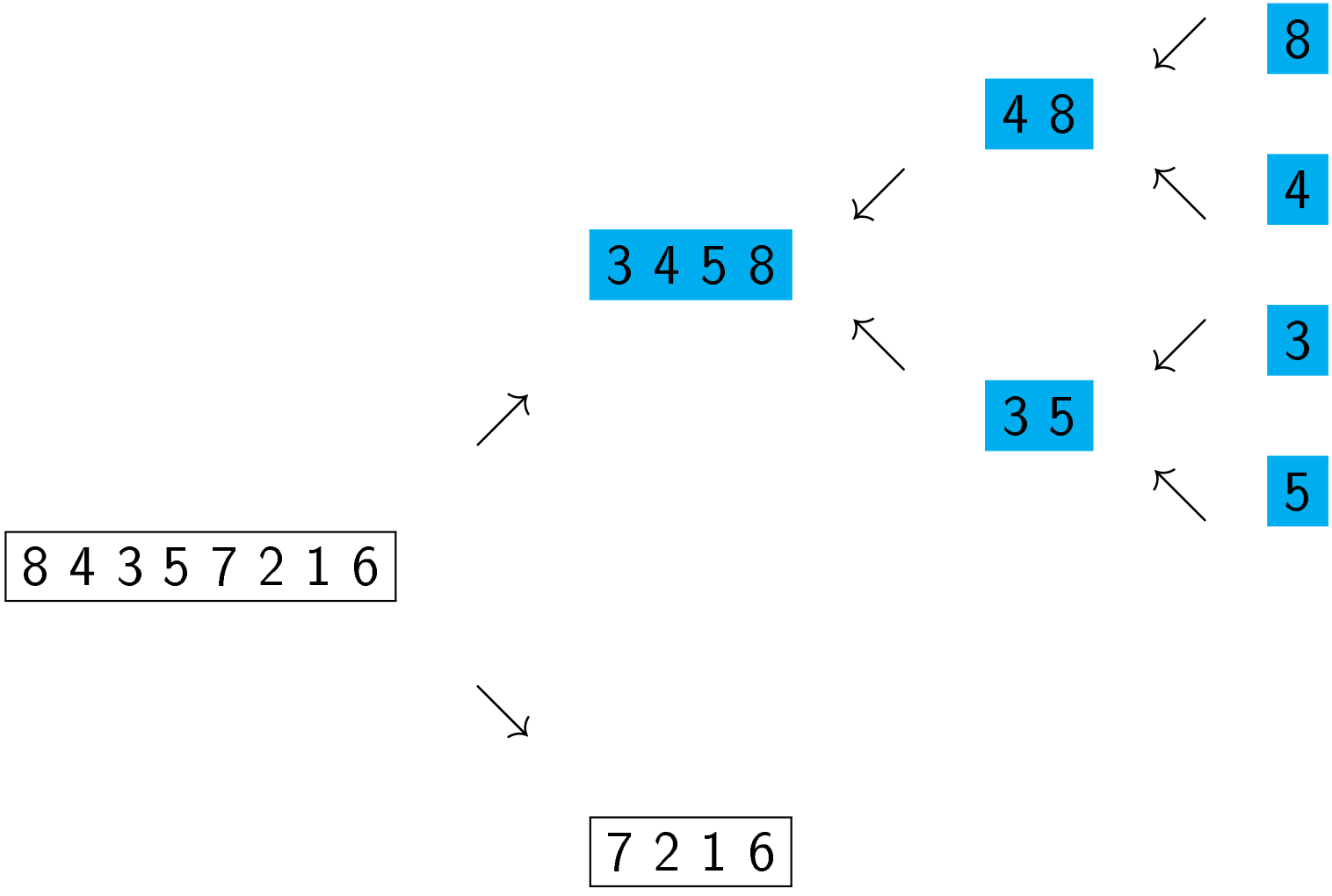

8
4 8
4
8 4 3 5
3 5
8 4 3 5 7 2 1 6

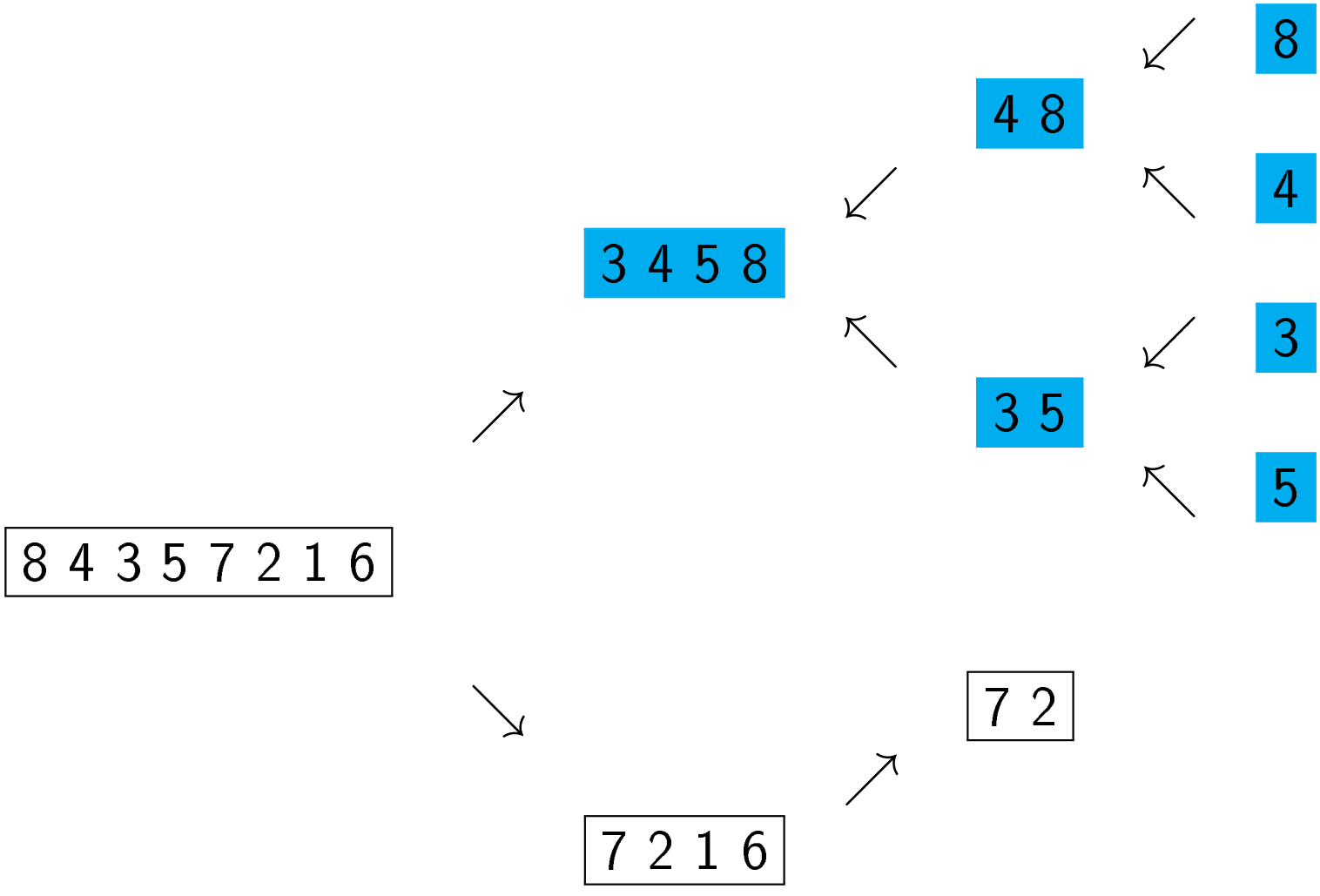

8 4 3 5 7 2 1 6
8 4 3 5
4 8
3 5
8
4
3

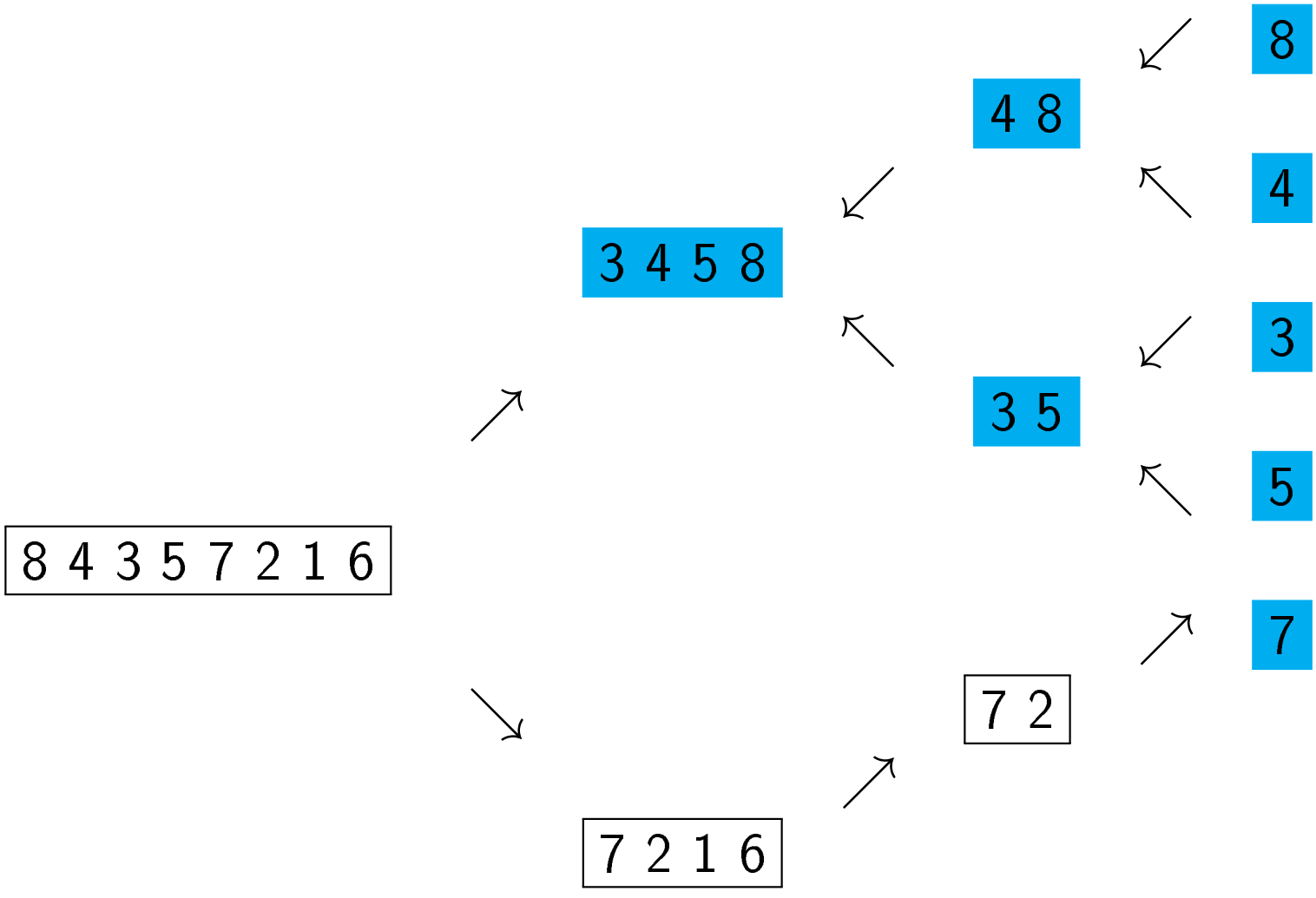

8 4 3 5 7 2 1 6
8 4 3 5
4 8
3 5
8
4
3
5

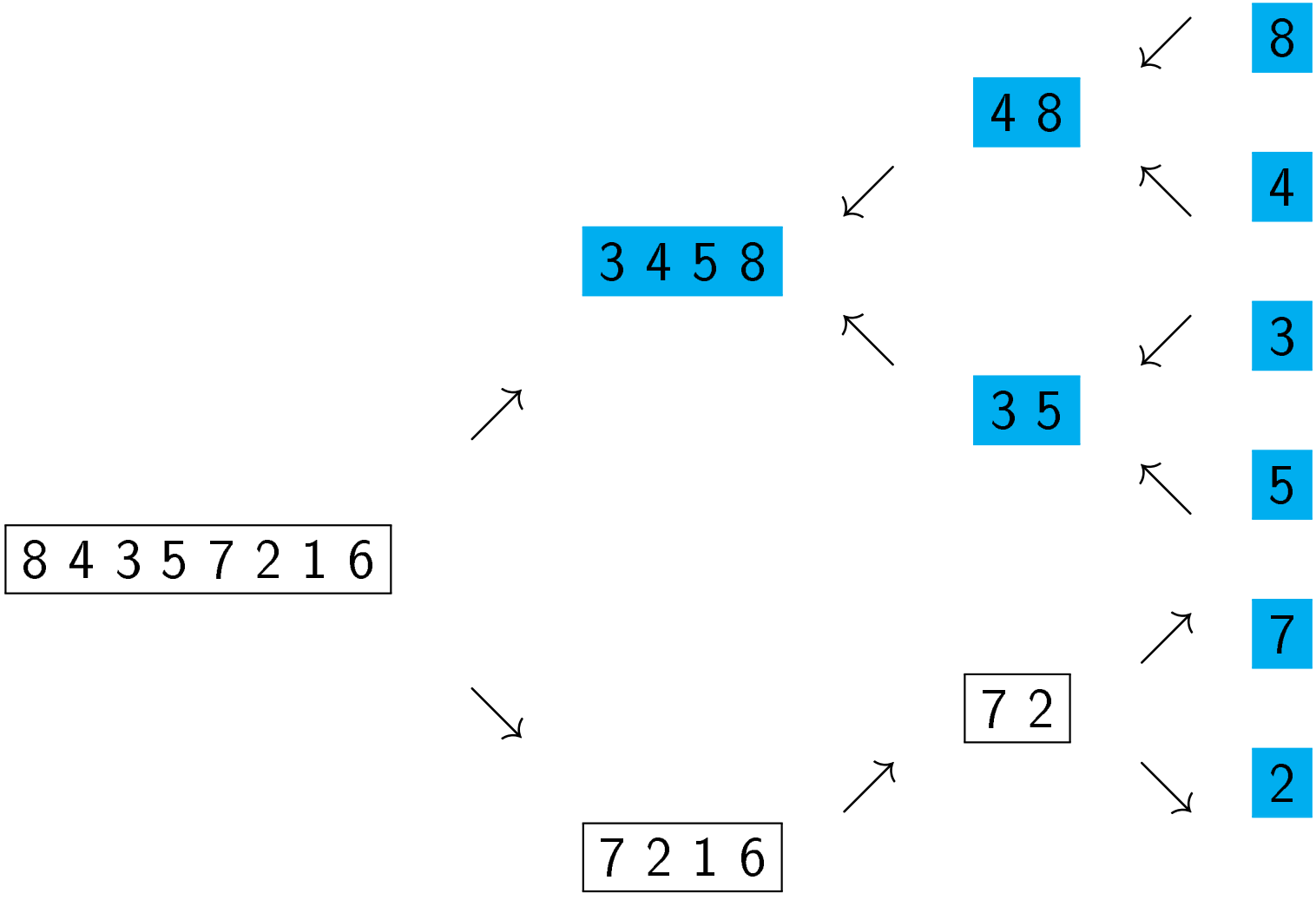

8 4 3 5 7 2 1 6
8 4 3 5
4 8
3 5
8
4
3
5

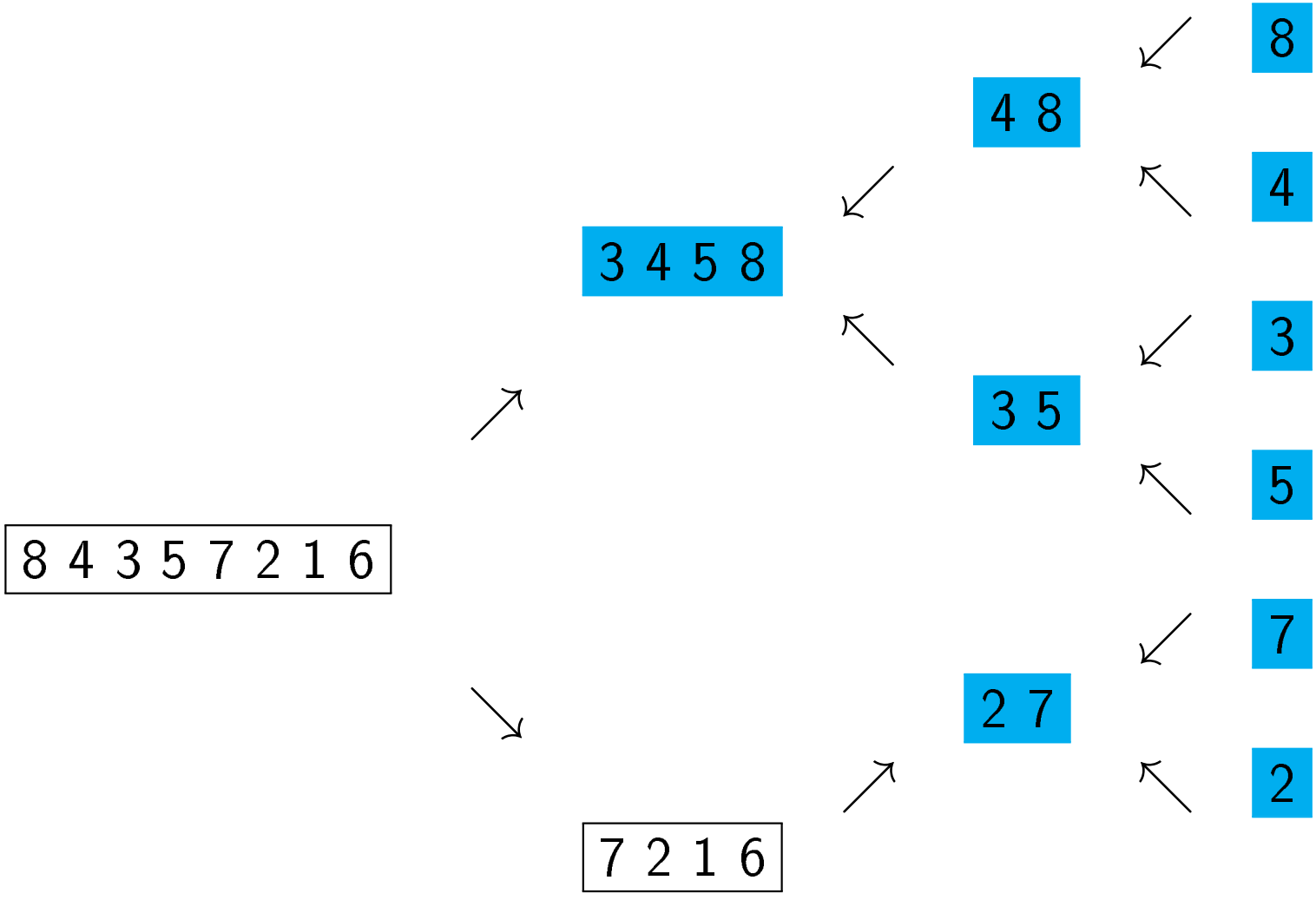

8
4 8
4
3 4 5 8
3
3 5
5
8 4 3 5 7 2 1 6

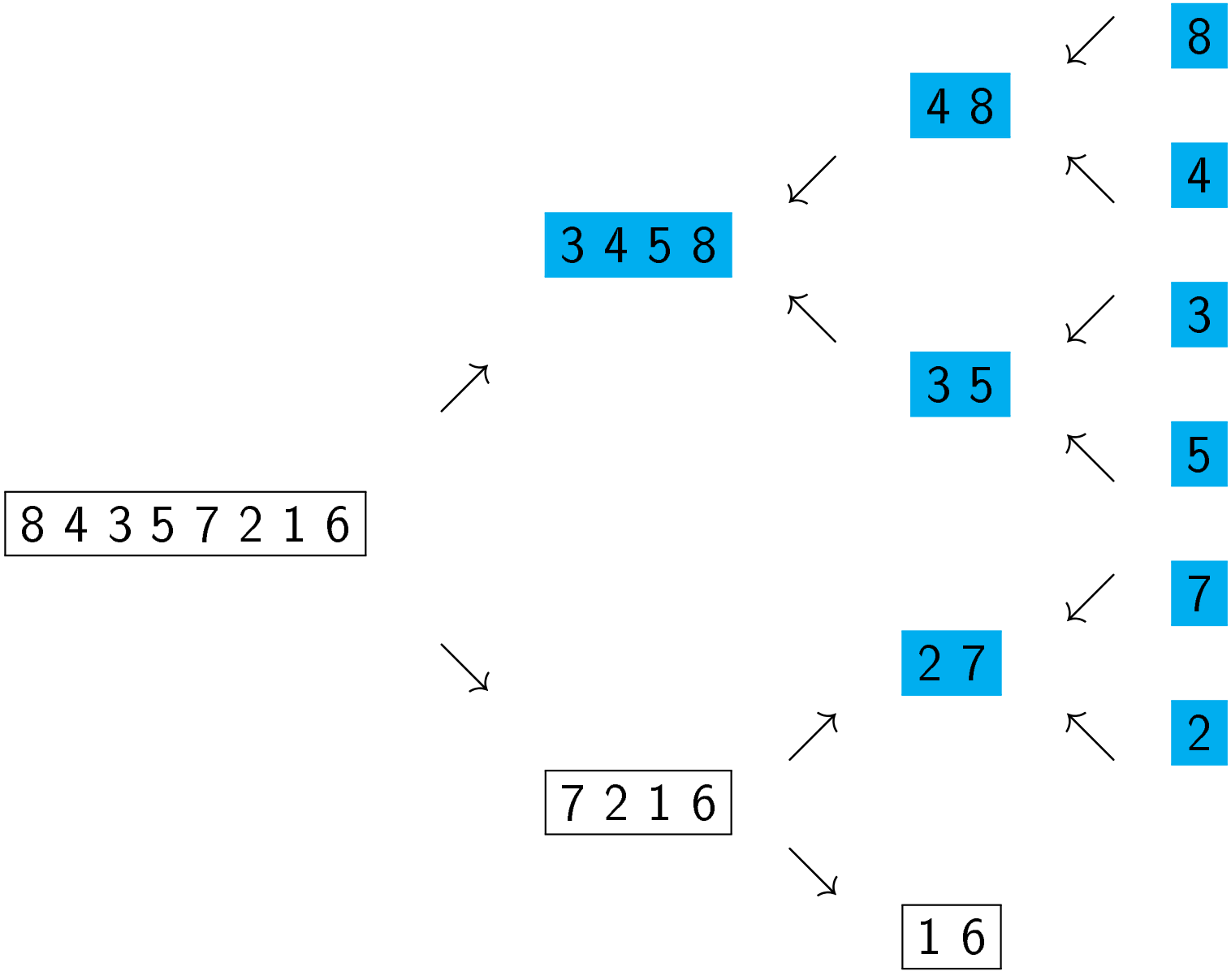

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
8
4
3
5

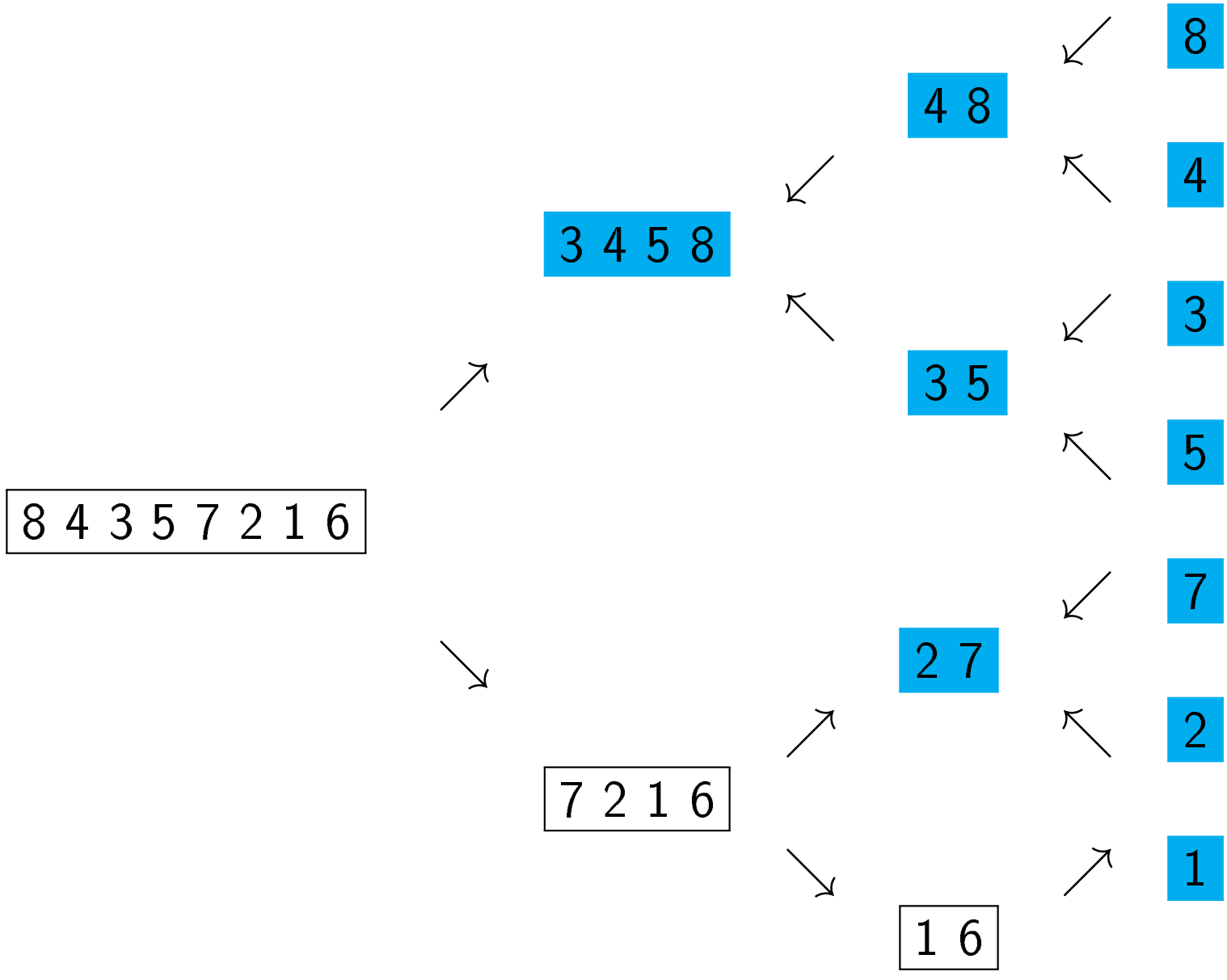

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
7 2
8
4
3
5

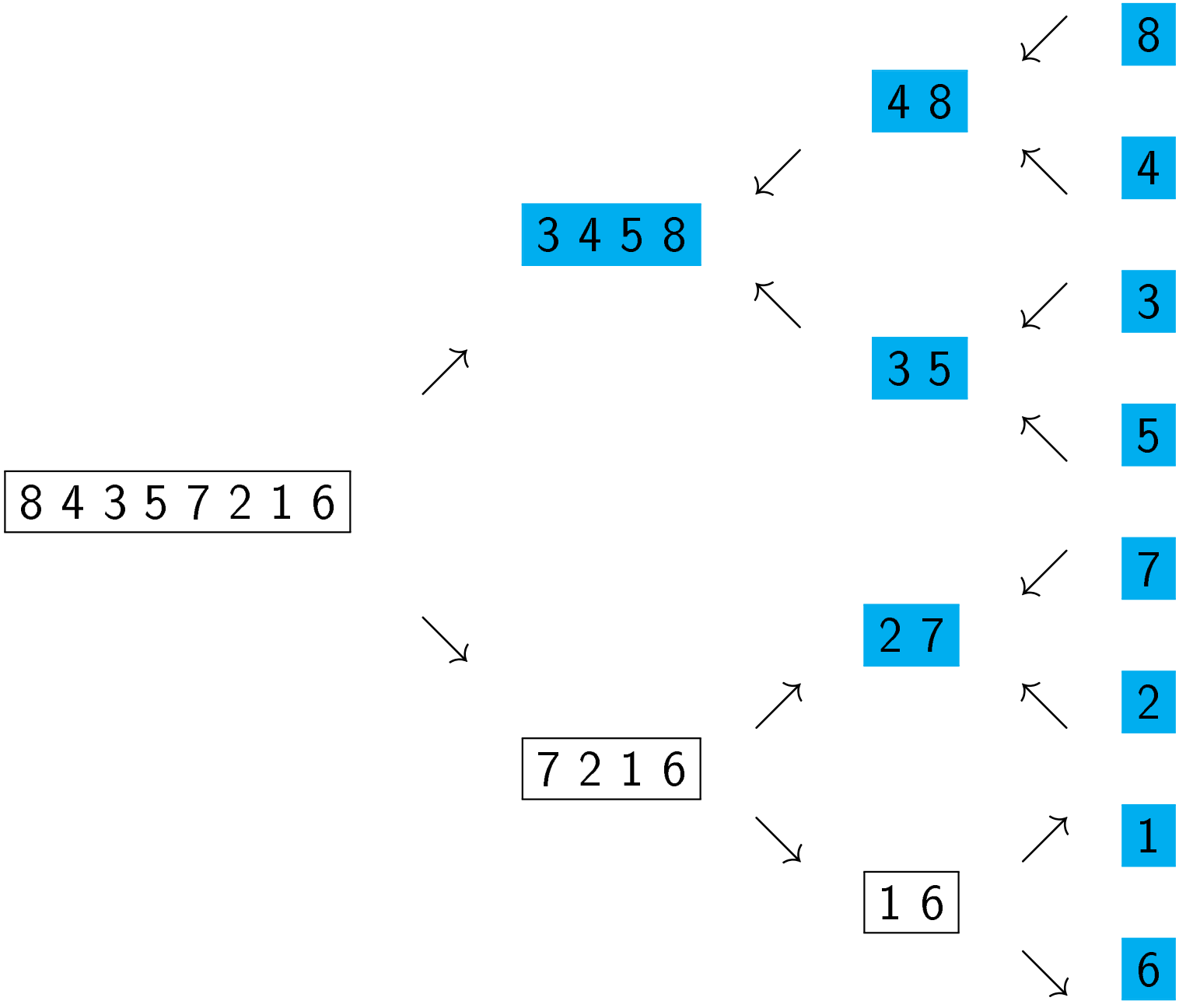

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
7 2
8
4
3
5
7

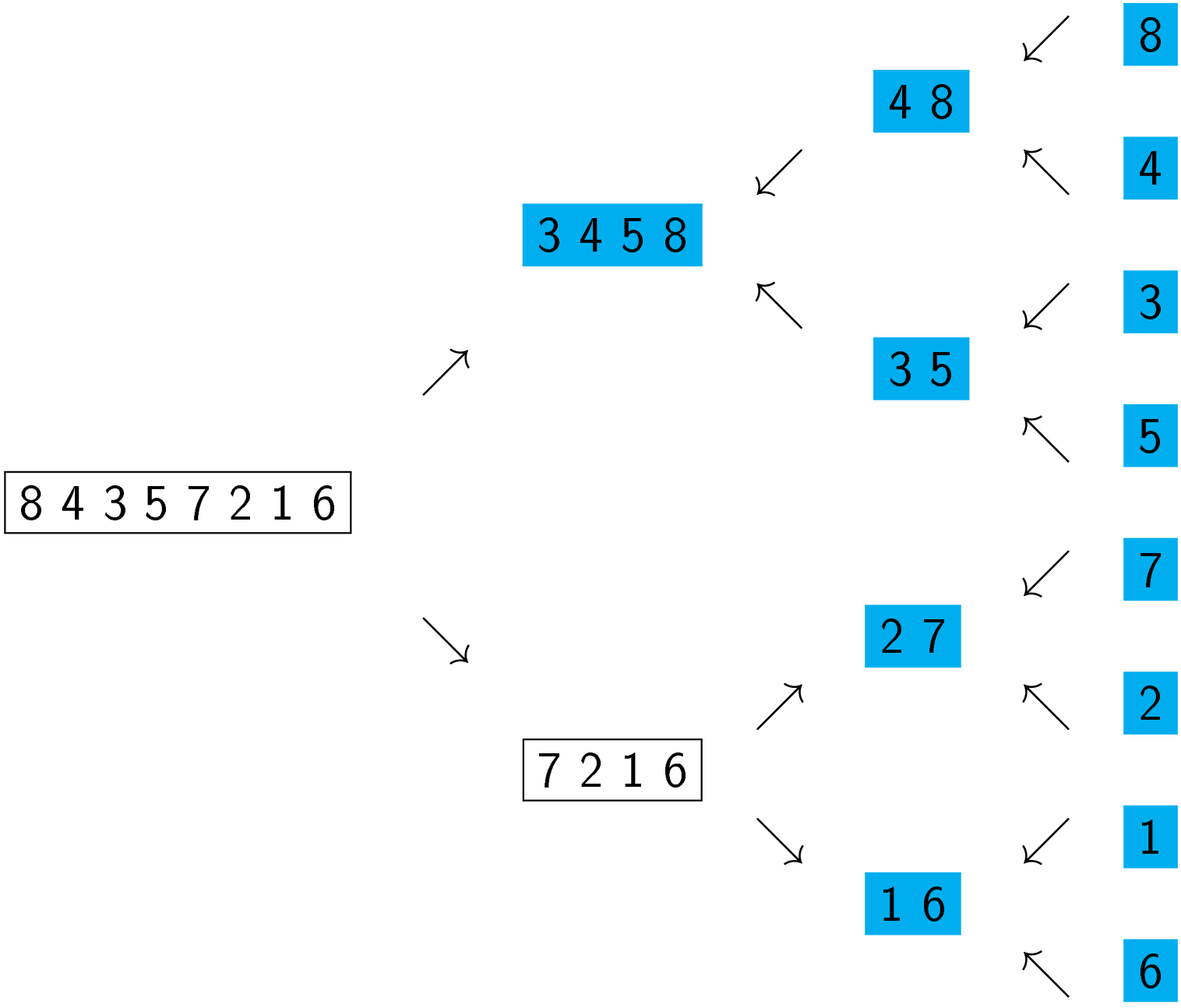

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
7 2
8
4
3
5
7
2

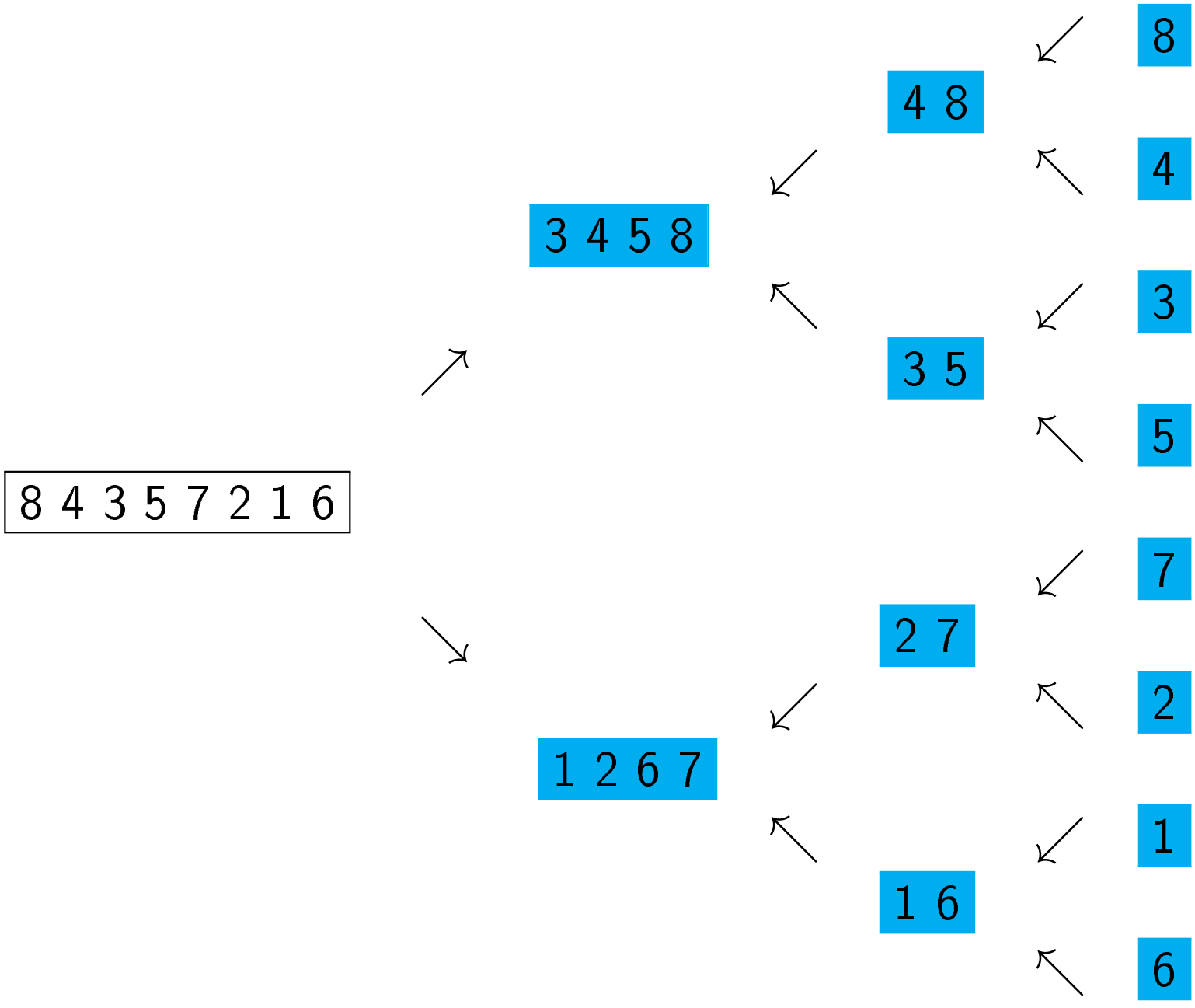

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
2 7
8
4
3
5
7
2

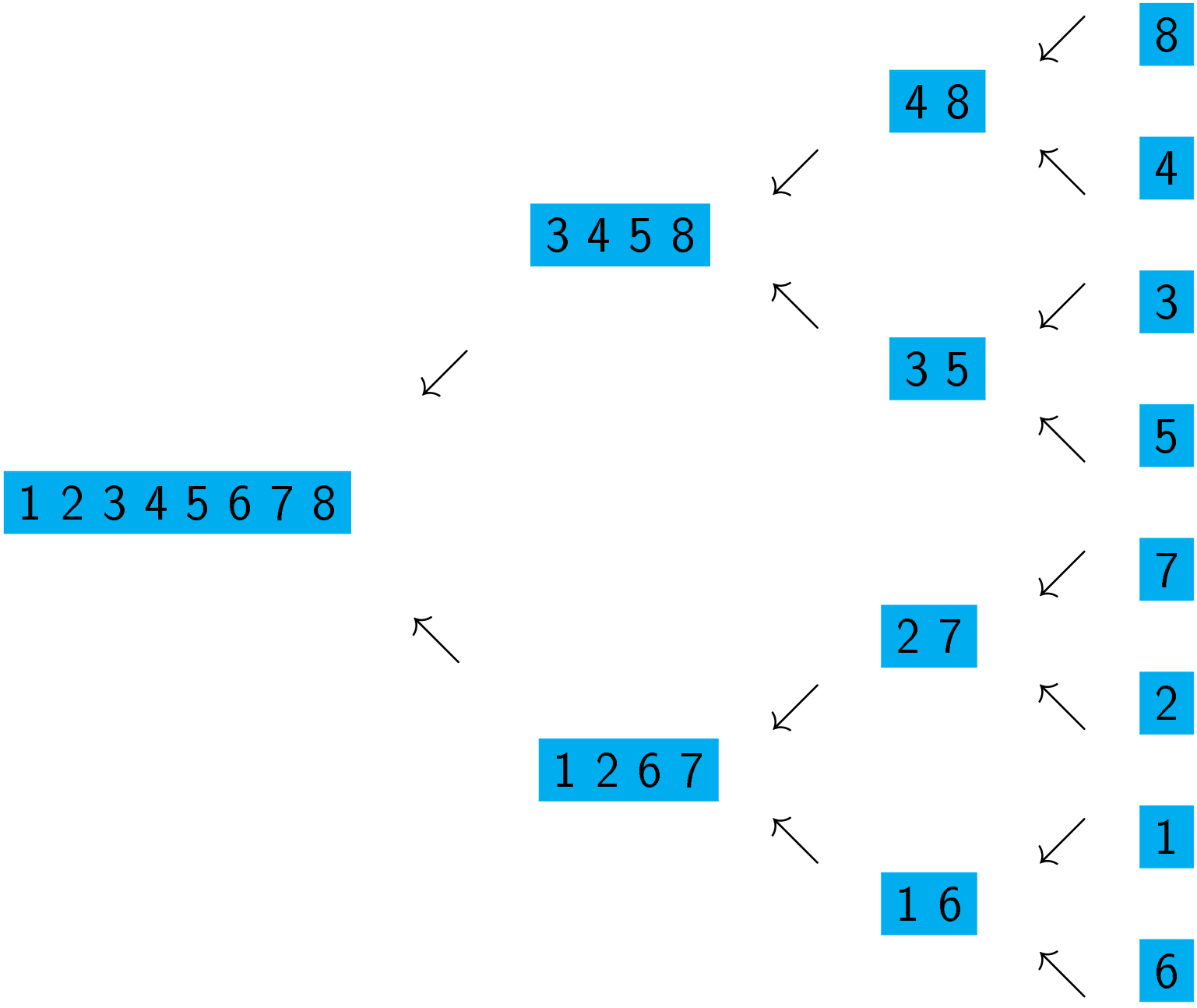

8 4 3 5 7 2 1 6
3 4 5 8
7 2 1 6
4 8
3 5
2 7
1 6
8
4
3
5
7
2


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


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


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


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


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

Tri Rapide

- Proposé par Hoare en 1962.
- Basé sur le paradigme «diviser pour régner» :
  - **Diviser** : le Tableau $T[g \dots d]$ est réorganisé puis divisé en 2 sous-tableaux non vides $T[g \dots p]$ et $T[p+1 \dots d]$. Chaque élément de $T[g \dots p]$ est inférieur à chaque élément de $T[p+1 \dots d]$.
  - **Régner** : Les 2 sous-tableaux sont triés grâce à la récursivité.
  - **Combiner** : Rien à faire.

Tri Rapide (2)

Algorithme 2 Tri Rapide

TriRapide(T : tableau d'entiers, g : entier, d : entier)

▷ g ***et*** d ***sont les indices entre lesquels on veut trier le tableau. On suppose*** $g \leq d$.

```
Début
    si g < d  faire
        p ← partitionner(T,g,d)
        TriRapide(T,g,p)
        TriRapide(T,p+1,d)

    fin si
Fin
```

Tri Rapide (3)

Algorithme 3 Partitionner

Partitionner(T : tableau d'entiers, g : entier, d : entier)

▷ *g et d sont les indices entre lesquels on veut trier le tableau. On suppose $g \leq d$.*

▷ ***Variables locales : i,j,pivot : entiers***

```
Début
i ← g+1 ; j ← d ; pivot ← T[g] ;
tant que (i < j) faire
    tant que (T[i] < pivot) faire i ← i + 1 fin tant que
    tant que (T[j] > pivot) faire j ← j - 1 fin tant que
    si (i < j) faire
        T[i] ↔ T[j]
        i ← i + 1
        j ← j - 1

    fin si
fin tant que
retourner j
Fin
```

Tri Rapide (4)

Complexité pour n éléments

- Partitionner n éléments coûte $O(n)$.
- Temps d'exécution dépend de l'équilibre ou non du partitionnement :
  - S'il est équilibré : **aussi rapide que le tri fusion**
  - S'il est déséquilibré : **aussi lent que le tri par insertion**

Tri Rapide (5)

Partitionnement dans le pire cas

- 2 sous-tableaux de 1 élément et $n-1$ éléments.
- Trier un élément coûte $O(1)$
- Supposons que ce partitionnement déséquilibré intervienne à chaque étape.
- Résolution de :

$$Tri(n) = Tri(n-1) + O(n)$$

Tri Rapide (6)

Partitionnement dans le pire cas

$$\begin{aligned} Tri(n) &= Tri(n-1) + cn \\ &= \sum_{k=1}^{n} ck \\ &= O(\sum_{k=1}^{n} k) \\ &= O(n^2) \end{aligned}$$

- Ce partitionnement apparaît quand le tableau est trié.
- Dans ce cas-là le tri par insertion est linéaire

Tri Rapide (7)

Partitionnement dans le meilleur cas

- Le partitionnement est équilibré
- Il faut résoudre :

$$Tri(n) = 2 \times Tri(\frac{n}{2}) + O(n)$$

- Solution : $Tri(n) = O(n \log_2 n)$

Tri Rapide (8)

Complexité en moyenne.

- Modèle : tous les ordres possibles sur les éléments du tableau sont équiprobables
- Il y a $n!$ tels ordres.

Tri Rapide (8)

Complexité en moyenne.

- Modèle : tous les ordres possibles sur les éléments du tableau sont équiprobables
- Il y a $n!$ tels ordres.
- La position du pivot dans le tableau trié est équiprobable.

Tri Rapide (8)

Complexité en moyenne.

- Modèle : tous les ordres possibles sur les éléments du tableau sont équiprobables
- Il y a $n!$ tels ordres.
- La position du pivot dans le tableau trié est équiprobable.
- Équation : $C_m(n) = \sum_{i=1}^{n-1} \frac{1}{n}(C_m(i) + C_m(n-i) + n)$

Tri Rapide (8)

Complexité en moyenne.

- Modèle : tous les ordres possibles sur les éléments du tableau sont équiprobables
- Il y a $n!$ tels ordres.
- La position du pivot dans le tableau trié est équiprobable.
- Équation : $C_m(n) = \sum_{i=1}^{n-1} \frac{1}{n}(C_m(i) + C_m(n-i) + n)$

Synthèse

Algorithme	Pire cas	En moyenne	Meilleur cas
Insertion	$O(n^2)$	$O(n^2)$	$O(n)$
Permutation	$O(n^2)$	$O(n^2)$	$O(n^2)$
Fusion	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Rapide	$O(n^2)$	$O(n \log_2 n)$	$O(n \log_2 n)$

Tri par dénombrement

Cet algorithme s'appuie sur une supposition *forte* :
les éléments à trier sont des entiers positifs inférieurs à t.

Théorème

La complexité du tri par dénombrement et l'espace qu'il utilise sont en $O(n+t)$. Quand t est fixé, la complexité est donc linéaire.

Première étape : dénombrer

Le principe est de compter combien de fois chaque élément entre 0 et $t-1$ apparaît et noter cela dans un nouveau tableau.

Algorithme 4 Calcul des fréquences

Frequence(A : tableau d'entiers, $TailleMax$: entier, t : entier) : tableau d'entiers

▷ ***Variables Locales***

i : **entier**

B : **tableau d'entiers de taille** t

Début
pour i **de 0 à t-1 faire**
    $B[i] = 0$
fin pour
pour i **de 1 à TailleMax faire**
    $B[A[i]]++$
fin pour
retourner B
Fin

Deuxième étape : créer le tableau trié

On écrit les éléments dans l'ordre en lisant leur fréquence.

Algorithme 5 Tri par dénombrement

Frequence(A **: tableau d'entiers,** $TailleMax$ **: entier,** t **: entier) : tableau d'entiers**

▷ ***Variables Locales***

i, j **: entiers initialisées à** 0

B **: tableau d'entiers de taille** t

Début
tant que $j < t$ **faire**
  si $B[j] > 0$ **faire**
    $A[i] = j$
    $B[j]--$
    $i++$
  sinon
    $j++$
  fin si
fin tant que
Fin

Propriétés utiles

On s'intéresse souvent à deux propriétés des algorithmes de tri :

- Un tri est stable si plusieurs éléments qui sont égaux pour l'ordre sont dans le même ordre avant et après le tri.
- Un tri est en place si on utilise pas de tableau additionnel pour trier.

Utilisation du tri stable : on peut trier des éléments à plusieurs coordonnées selon plusieurs de leurs coordonnées, comme un jeu de carte ou une feuille de tableur.

Propriétés utiles

On s'intéresse souvent à deux propriétés des algorithmes de tri :

- Un tri est stable si plusieurs éléments qui sont égaux pour l'ordre sont dans le même ordre avant et après le tri.
- Un tri est en place si on utilise pas de tableau additionnel pour trier.

Utilisation du tri stable : on peut trier des éléments à plusieurs coordonnées selon plusieurs de leurs coordonnées, comme un jeu de carte ou une feuille de tableur.

La zoologie

Il y a toute une zoologie des algorithmes de tri, lire la fiche wikipedia en anglais si vous êtes curieux.

Un animal exotique, le Bogosort.
Tant que le tableau n'est pas en ordre, permuter aléatoirement le tableau.

La zoologie

Il y a toute une zoologie des algorithmes de tri, lire la fiche wikipedia en anglais si vous êtes curieux.

Un animal exotique, le Bogosort.
Tant que le tableau n'est pas en ordre, permuter aléatoirement le tableau.
Complexité ?

La zoologie

Il y a toute une zoologie des algorithmes de tri, lire la fiche wikipedia en anglais si vous êtes curieux.

Un animal exotique, le Bogosort.
Tant que le tableau n'est pas en ordre, permuter aléatoirement le tableau.
Complexité ?

Pour finir en musique : le chant des tris.

La zoologie

Il y a toute une zoologie des algorithmes de tri, lire la fiche wikipedia en anglais si vous êtes curieux.

Un animal exotique, le Bogosort.
Tant que le tableau n'est pas en ordre, permuter aléatoirement le tableau.
Complexité ?

Pour finir en musique : le chant des tris.

Réseaux de tri

Un autre moyen de représenter un algorithme de tri que l'arbre de comparaison, qui correspond à plusieurs processeurs en parallèle ou au design d'un processeur spécialisé.

- n entrées qui correspondent aux n valeurs du tableau
- des portes qui comparent deux entrées et les ressortent dans l'ordre croissant

Voir plusieurs exemples dans le Cormen.

Réseaux de tri (2)

Notions de complexité différentes :

- la profondeur du circuit = le temps parallèle
- la taille ou nombre de portes = le temps séquentiel

Théorème

Il existe un réseau de tri (AKS) de profondeur $O(\log n)$ et de taille $O(n \log n)$.