Introduction à l'informatique
Algorithmes et structures linéaires 2

Portail René Descartes
Luminy

Université d'Aix-Marseille

Plan du cours

Plan du cours

Parcours d'un tableau

- Objectif : réaliser un traitement sur l'ensemble des éléments d'un tableau
  ⟶ parcours séquentiel

```
Procédure parcours(d tab[1...n] : tab_t)
    i : entier ;
Début
    i := 1 ;
    Tant que i ⩽ n faire
        traiter(tab[i]) ;        # p.ex. écrire
        i := i + 1 ;
    Fin faire
Fin
```

D♢	7♢	D♠	7♠	V♠	7♡	R♣

```
parcours(tab[1... 7]);
```

```
D♢
7♢
D♠
7♠
V♠
7♡
R♣
```

Parcours d'un tableau

- Objectif : réaliser un traitement sur l'ensemble des éléments d'un tableau
  ⟶ parcours séquentiel

Procédure parcours(**d** tab[1 . . . n] : **tab_t**)
    i : **entier** ;
Début
    $i := 1$;
    Tant que $i \leqslant n$ **faire**
        traiter(tab[i]) ; # p.ex. **écrire**
        $i := i + 1$;
    Fin faire
Fin

D♢	7♢	D♠	7♠	V♠	7♡	R♣

```
parcours(tab[1... 7]);
```

```
D♢
7♢
D♠
7♠
V♠
7♡
R♣
```

- Terminaison ? ok
- Correction ? ok
- Complexité ? $3n + 1$ opérations $\rightsquigarrow O(n)$

Recherche dans un tableau non trié

- Objectif : répondre vrai ou faux selon qu'un élément donné appartient ou non à un tableau
  $\longrightarrow$ recherche séquentielle

Recherche dans un tableau non trié

- Objectif : répondre vrai ou faux selon qu'un élément donné appartient ou non à un tableau
  $\longrightarrow$ recherche séquentielle

Fonction r(**d** tab$[1 \ldots n]$: **tab_t**, **d** e : **t**) : **booléen**
    i : **entier** ;
Début
    $i := 1$;
    Tant que $(i < n)$ **et** $(t[i] \neq e)$ **faire**
        $i := i + 1$;
    Fin faire
    retour tab$[i] = e$;
Fin

Procédure rech(**d** tab$[1 \ldots n]$: **tab_t**, **d** elem : **t**)
Début
    écrire(r(tab$[1 \ldots n]$, elem)) ;
Fin

4	17	64	25	1	38	51

```
rech(tab[1... 7], 32);
          faux

rech(tab[1... 7], 64);
          vrai
```

Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?

Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

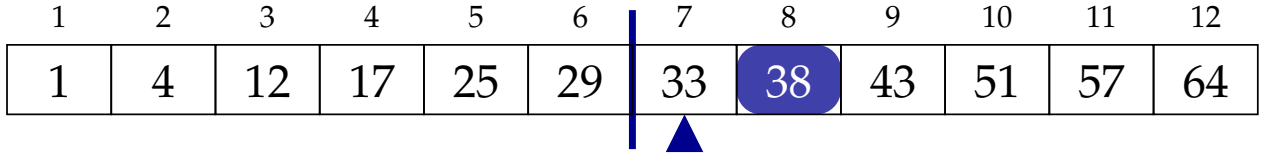
- Recherche séquentielle ? $\rightsquigarrow$ complexité en $O(n)$, peut mieux faire...

Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

- Recherche séquentielle ? $\rightsquigarrow$ complexité en $O(n)$, peut mieux faire...

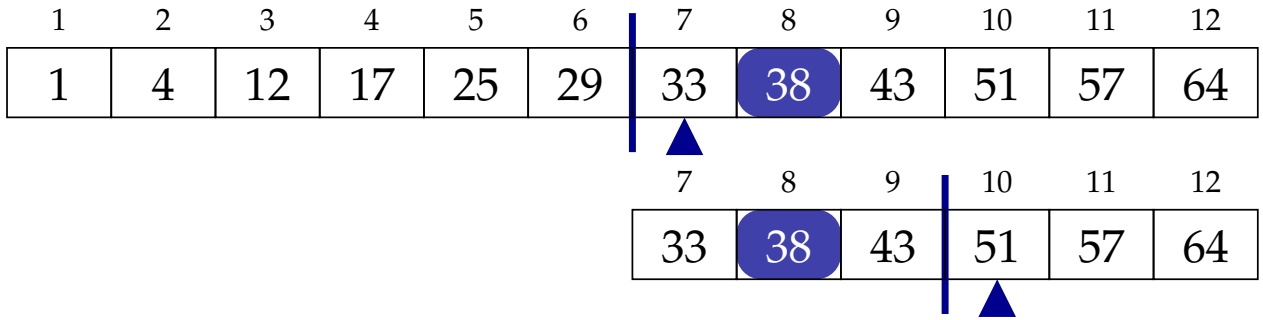


Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

- Recherche séquentielle ? $\rightsquigarrow$ complexité en $O(n)$, peut mieux faire...

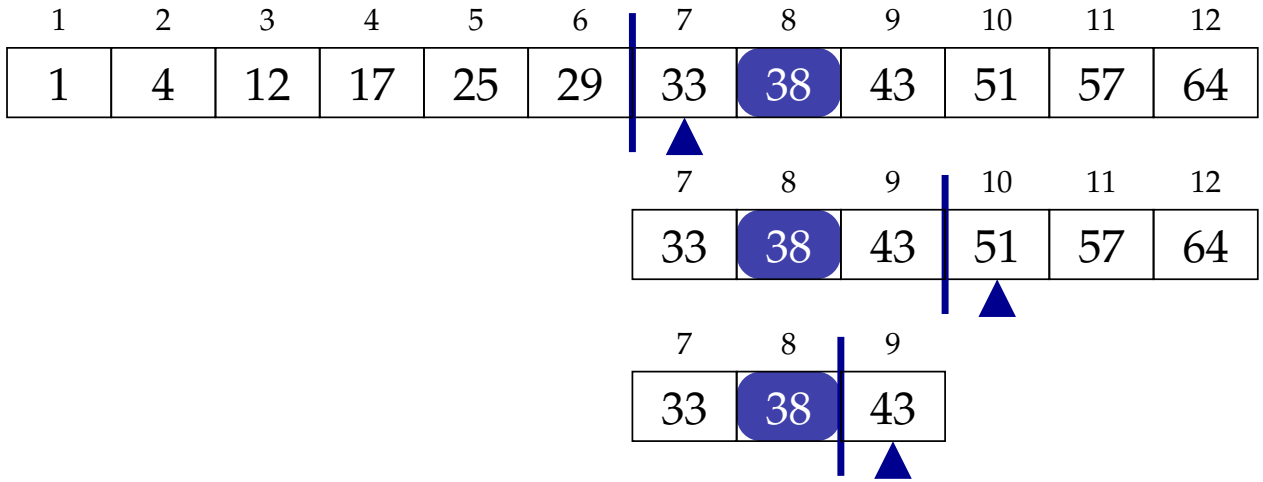


Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

- Recherche séquentielle ? $\rightsquigarrow$ complexité en $O(n)$, peut mieux faire...

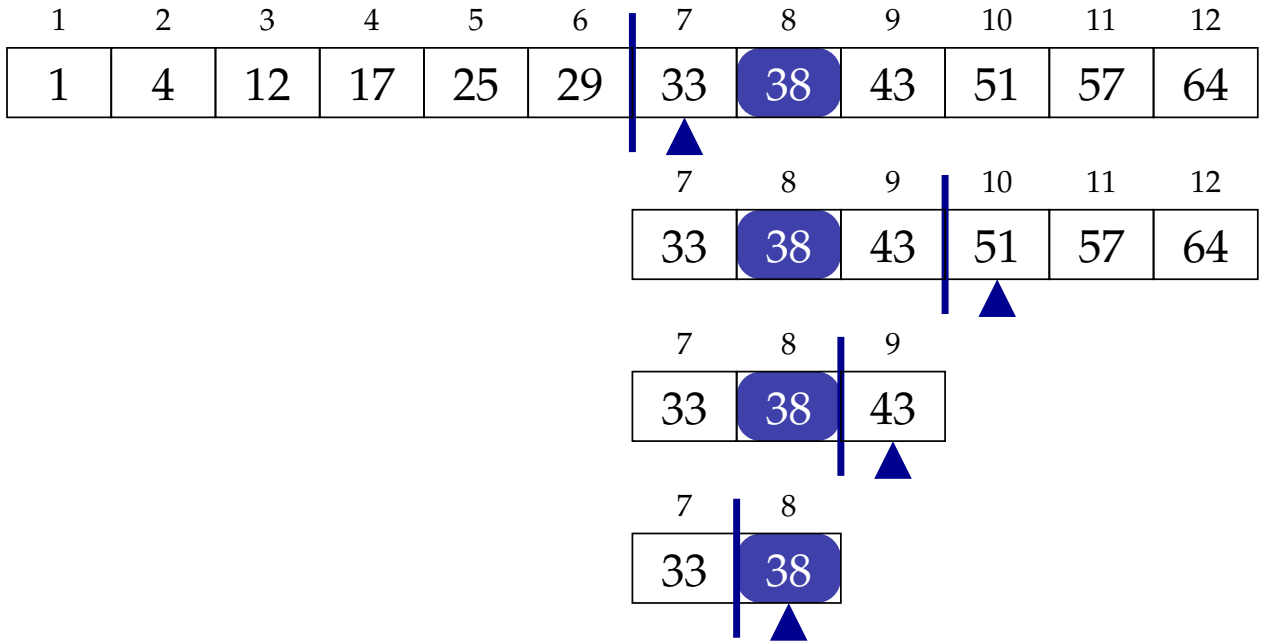


Recherche dans un tableau trié

- Question : comment faire dans un tableau trié, i.e. dont les éléments sont ordonnés ?
- Exemple : recherche de 38 dans le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
1	4	12	17	25	29	33	38	43	51	57	64

- Recherche séquentielle ? $\rightsquigarrow$ complexité en $O(n)$, peut mieux faire...



Recherche dans un tableau trié

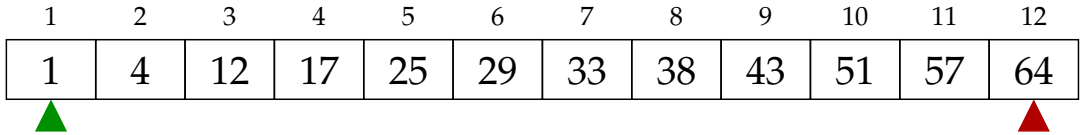
```
Fonction rechdich(d tab[1...n] : tab_t, d e : t) : booléen
    m, inf, sup : entier ;
    trouvé : booléen ;
Début
    trouvé := faux ;
    inf := 1 ;
    sup := n ;
    Tant que (inf ⩽ sup) et (trouvé = faux) faire
        m := ⌈(inf + sup)/2⌉ ;
        Si (t[m] = e) alors
            trouvé := vrai ;
        Sinon si (t[m] < e) alors
            inf := m + 1 ;
        Sinon
            sup := m − 1 ;
        Fin si
    Fin faire
    retour trouvé ;
Fin
```

Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent

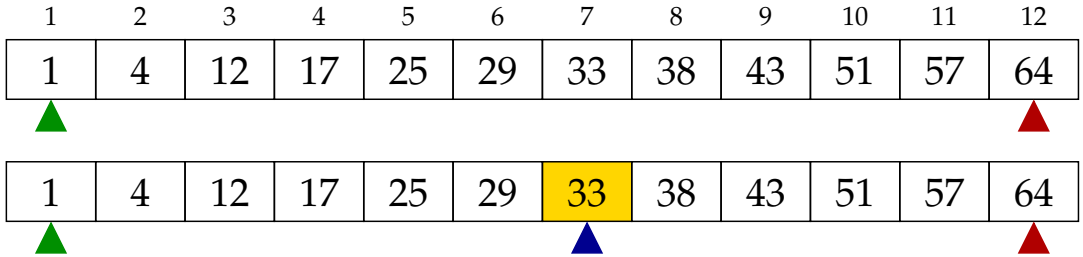
Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



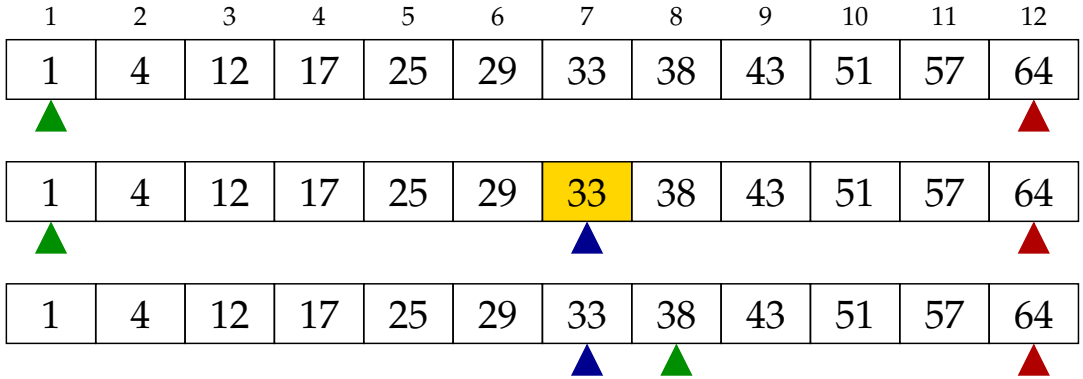
Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



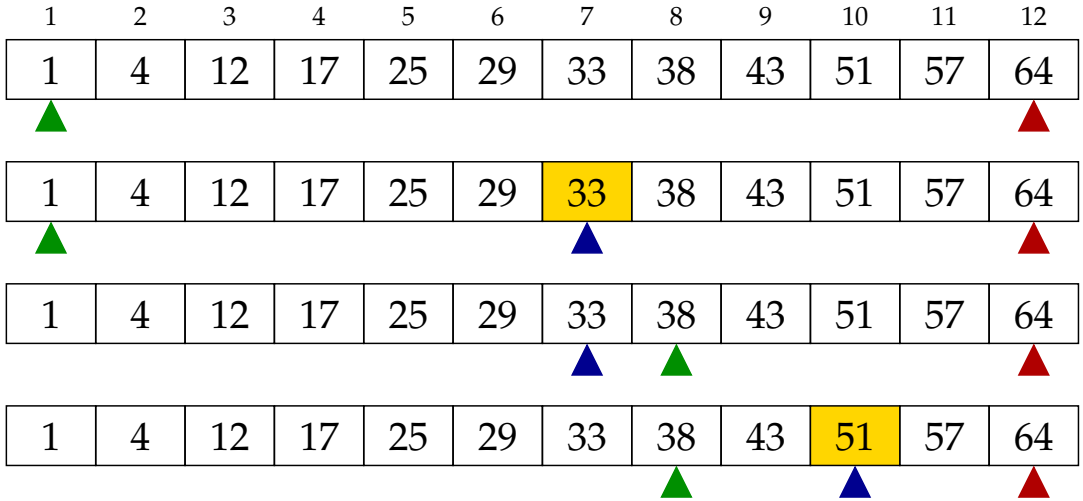
Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



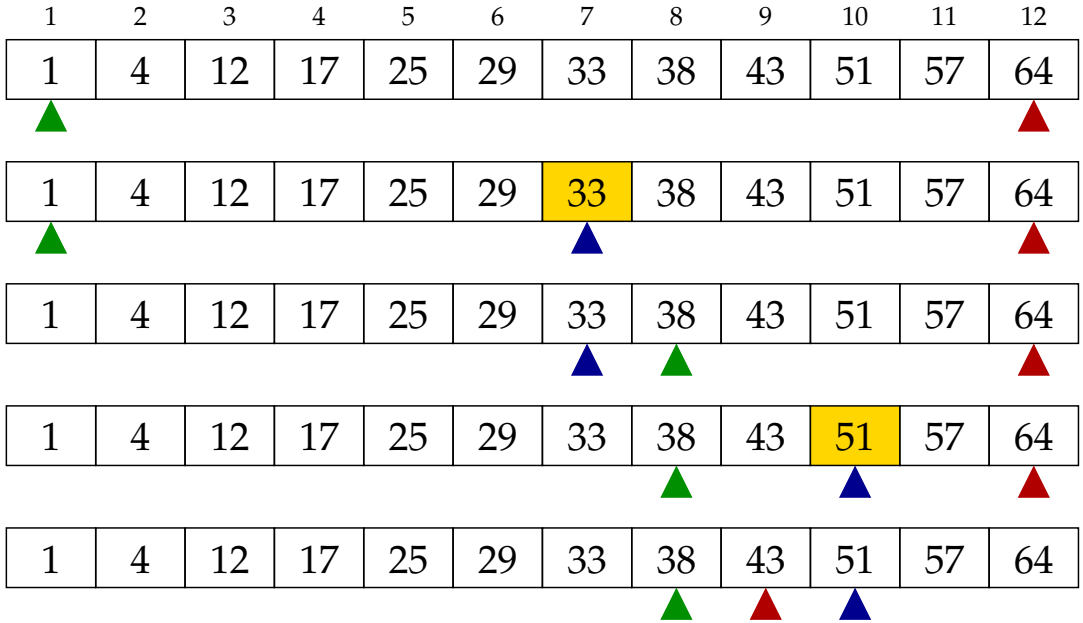
Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



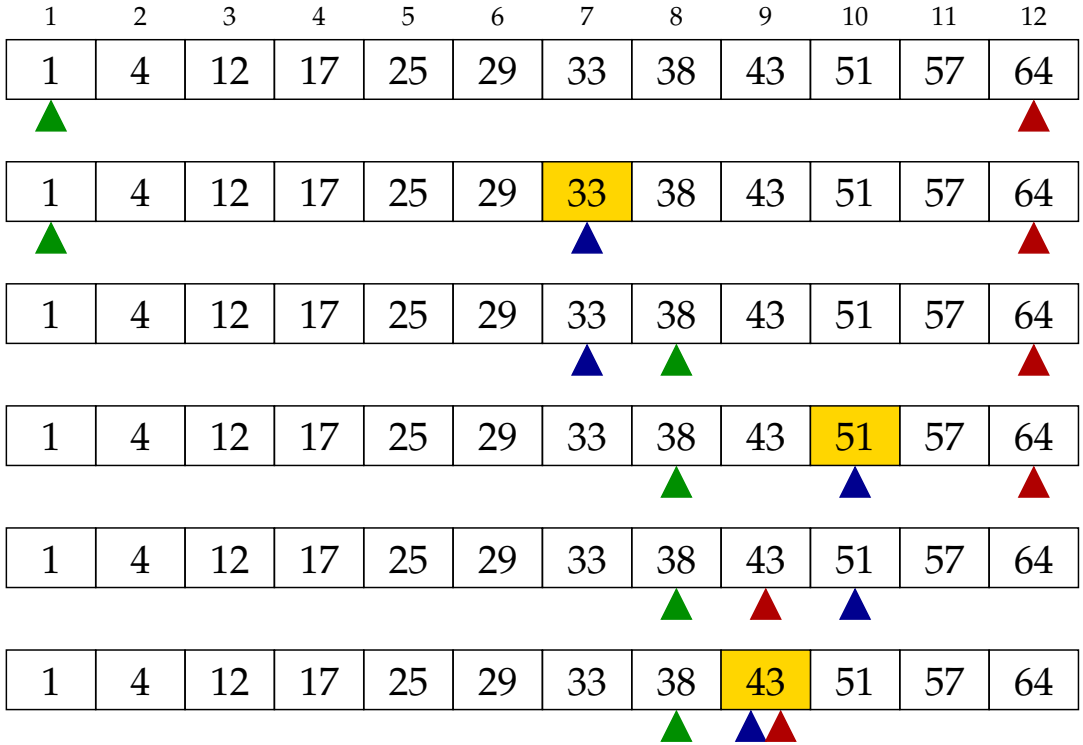
Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



Recherche dans un tableau trié

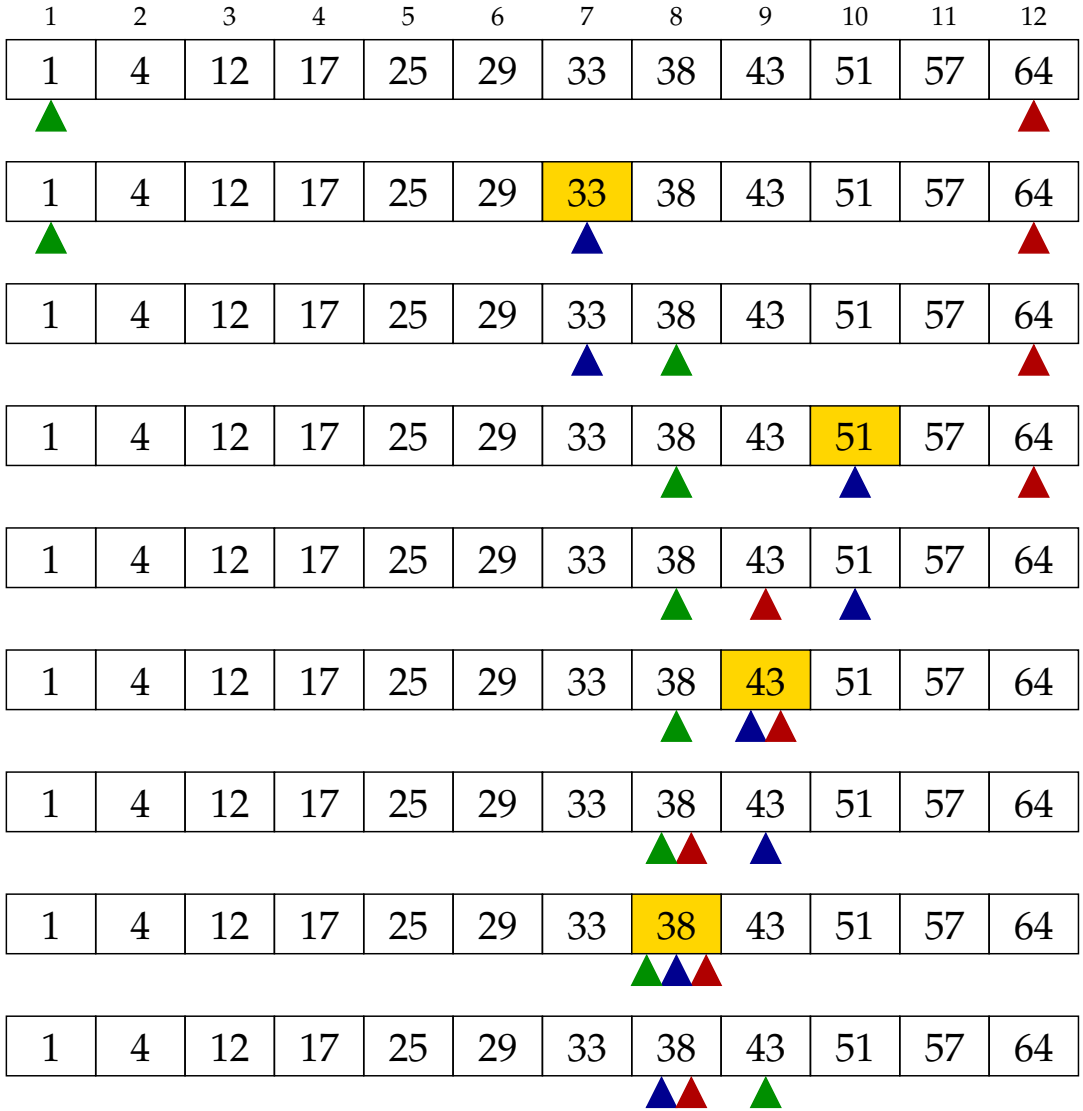
- Recherche de 41 dans le tableau précédent

Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent

Recherche dans un tableau trié

- Recherche de 41 dans le tableau précédent



Plan du cours

Insertion dans un tableau non trié

- Pas de critère d'insertion $\rightsquigarrow$ insertion en fin

Procédure insertion_fin(**dr** tab[tab$[1 \ldots n]$: **tab_t**, **d** e : **t**)
Début
    $n := n + 1$; Mise à jour du nombre d'éléments
    tab$[n] := e$;
Fin

- Si critère.s tel.s que "insertion à la $k$ème position"

Comment ?

Insertion dans un tableau trié

- Entrées :
  - un tableau trié tab$[1 \dots n]$
  - un élément e
- Sortie : un tableau trié tab$[1 \dots n+1]$
- Décomposition en 2 étapes :
  1. trouver la position de l'élément à insérer
  2. réaliser l'insertion de l'élément
- Exemple : insérer 12 dans $(1, 3, 4, 7, 10, 11, 14, 15, 16, 18)$

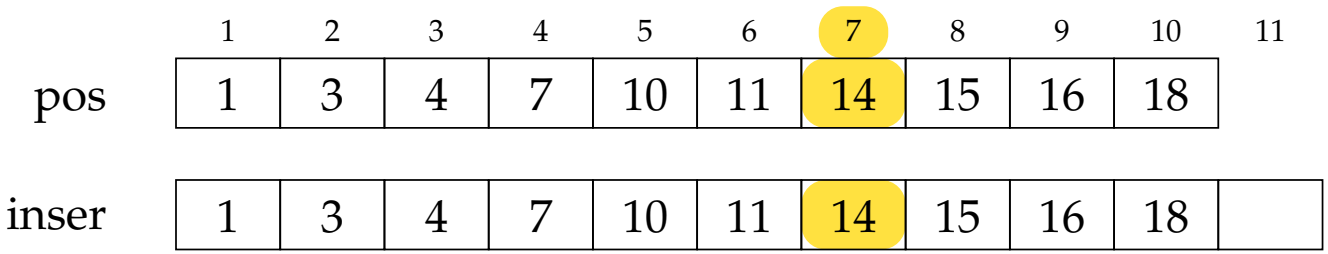
Insertion dans un tableau trié

- Entrées :
  - un tableau trié tab$[1 \dots n]$
  - un élément e
- Sortie : un tableau trié tab$[1 \dots n+1]$
- Décomposition en 2 étapes :
  1. trouver la position de l'élément à insérer
  2. réaliser l'insertion de l'élément
- Exemple : insérer 12 dans $(1,3,4,7,10,11,14,15,16,18)$

	1	2	3	4	5	6	7	8	9	10
pos	1	3	4	7	10	11	14	15	16	18

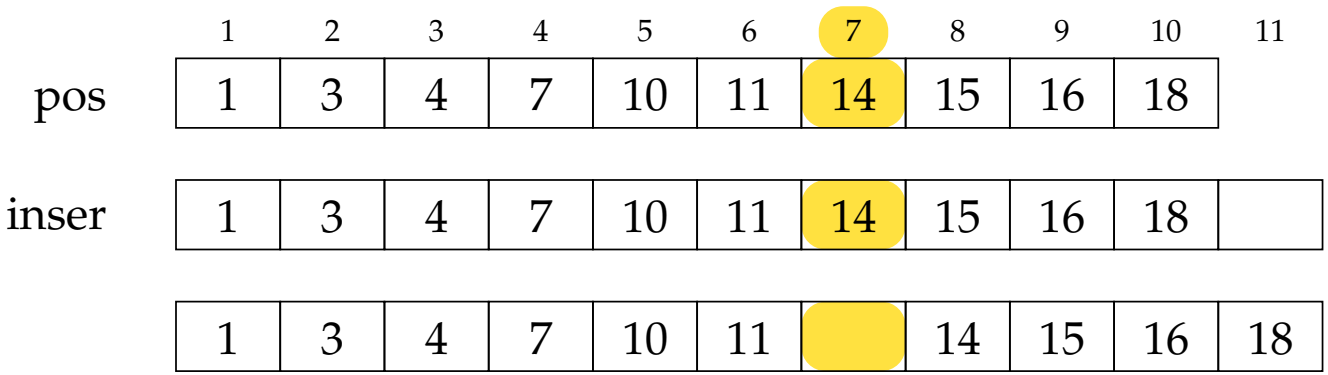
Insertion dans un tableau trié

- Entrées :
  - un tableau trié tab$[1 \dots n]$
  - un élément e
- Sortie : un tableau trié tab$[1 \dots n+1]$
- Décomposition en 2 étapes :
  1. trouver la position de l'élément à insérer
  2. réaliser l'insertion de l'élément
- Exemple : insérer 12 dans $(1,3,4,7,10,11,14,15,16,18)$



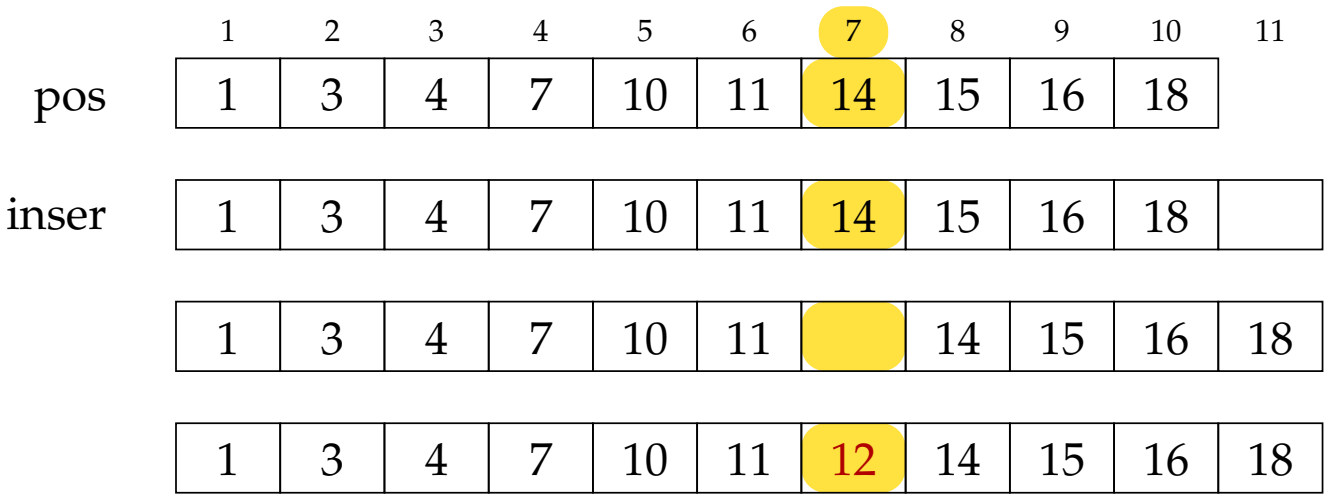
Insertion dans un tableau trié

- Entrées :
  - un tableau trié tab$[1 \dots n]$
  - un élément e
- Sortie : un tableau trié tab$[1 \dots n+1]$
- Décomposition en 2 étapes :
  1. trouver la position de l'élément à insérer
  2. réaliser l'insertion de l'élément
- Exemple : insérer 12 dans $(1,3,4,7,10,11,14,15,16,18)$



Insertion dans un tableau trié

- Entrées :
  - un tableau trié tab$[1 \dots n]$
  - un élément e
- Sortie : un tableau trié tab$[1 \dots n+1]$
- Décomposition en 2 étapes :
  1. trouver la position de l'élément à insérer
  2. réaliser l'insertion de l'élément
- Exemple : insérer 12 dans $(1,3,4,7,10,11,14,15,16,18)$



Insertion dans un tableau trié

- Algorithme principal :

```
Procédure insertion(dr tab[1...n] : tab_t, d e : t)
    p : entier ;
Début
    Si (n = 0) alors
        n := 1 ;
        tab[n] := e ;
    Sinon
        p := pos (tab[1...n], e) ;
        inser(tab[1...n], e, p) ;
    Fin si
Fin
```

Insertion dans un tableau trié

- Algorithme de recherche de la position la plus à droite $\rightsquigarrow$ nombre minimum de décalages

```
Fonction pos(d tab[1...n] : tab_t, d e : t) : entier
    p, m, inf, sup : entier ;
Début
    Si (t[n] ⩽ e) alors p := n + 1 ;
    Sinon
        inf := 1 ; sup := n ;
        Tant que (inf < sup) faire
            m := ⌈(inf + sup)/2⌉ ;
            Si (t[m] ⩽ e) alors inf := m + 1 ;
            Sinon sup := m ;
            Fin si
        Fin faire
        p := sup ;
    Fin si
    retour p ;
Fin
```

Insertion dans un tableau trié

- Algorithme d'insertion à la position p

```
Procédure inser(dr tab[1...n] : tab_t, d e : t, d p : entier)
    i : entier ;
Début
    i := n ;
    n := n + 1 ;
    Tant que (i ⩾ p) faire
        tab[i + 1] := tab[i] ;
    Fin faire
    tab[p] := e ;
Fin
```

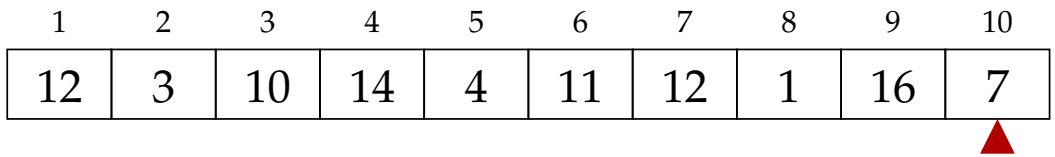
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau

1	2	3	4	5	6	7	8	9	10
12	3	10	14	4	11	12	1	16	7

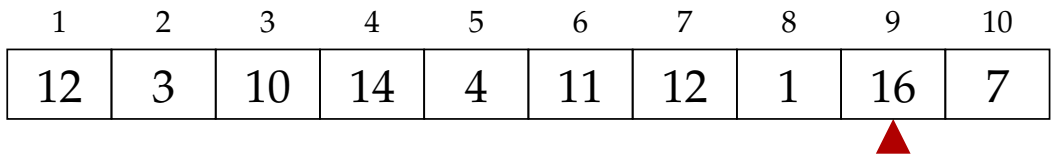
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



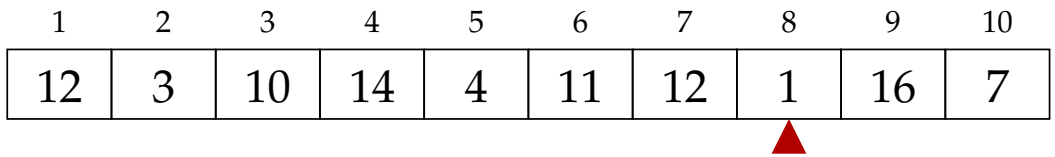
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



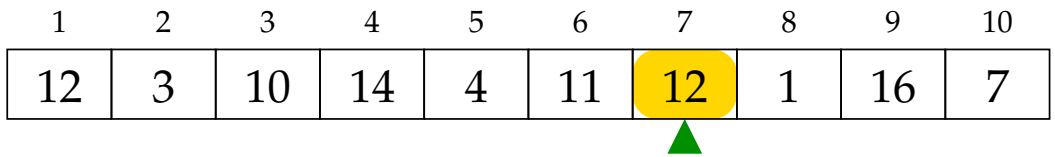
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



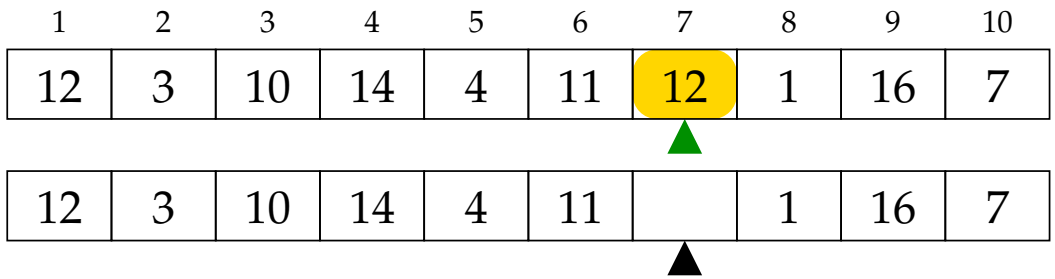
Suppression d'un tableau non trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



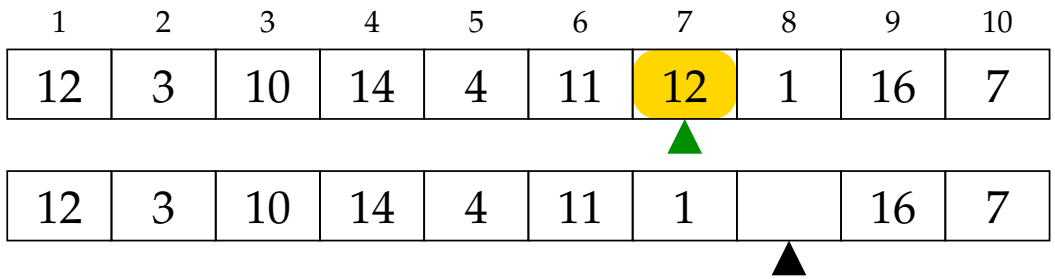
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



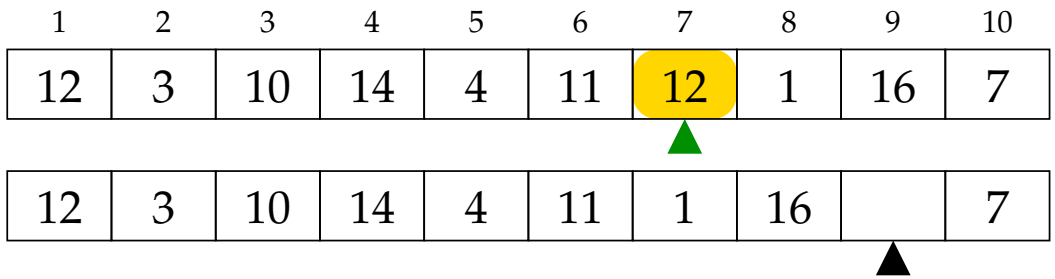
Suppression d'un tableau non trié

- Entrées :
  - un tableau $\text{tab}[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



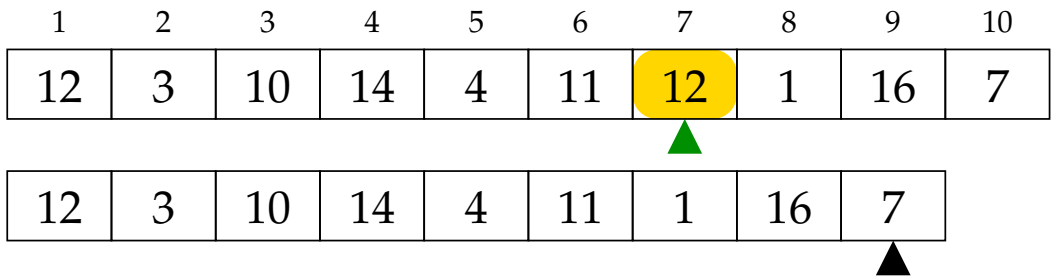
Suppression d'un tableau non trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



Suppression d'un tableau non trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher séquentiellement l'élément $\rightsquigarrow$ Quelle stratégie ?
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



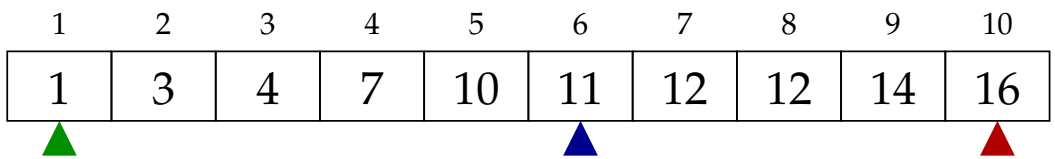
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau

1	2	3	4	5	6	7	8	9	10
1	3	4	7	10	11	12	12	14	16

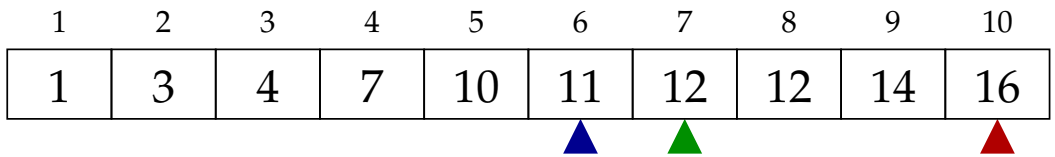
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



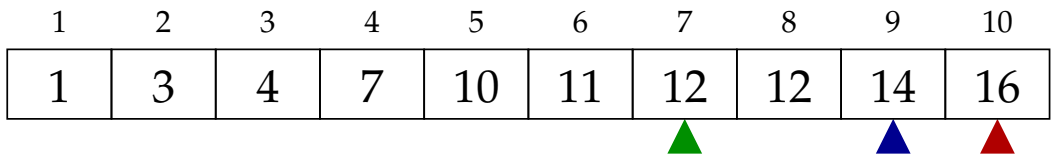
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



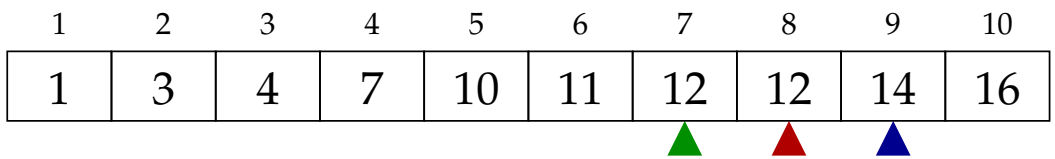
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



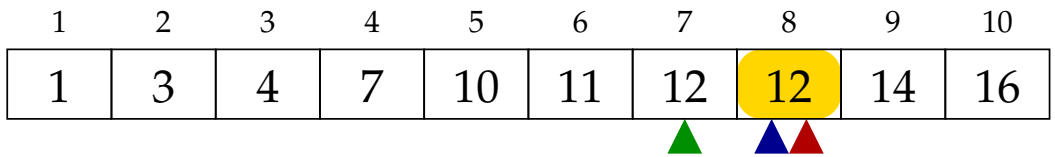
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



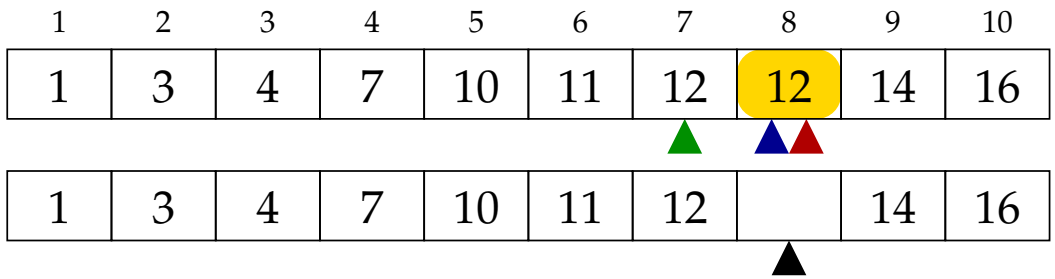
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



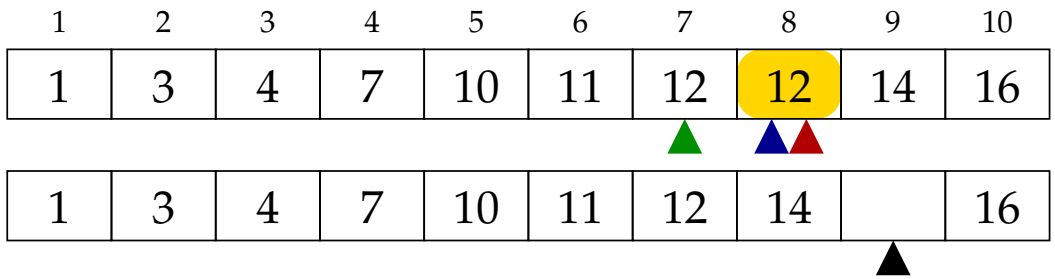
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



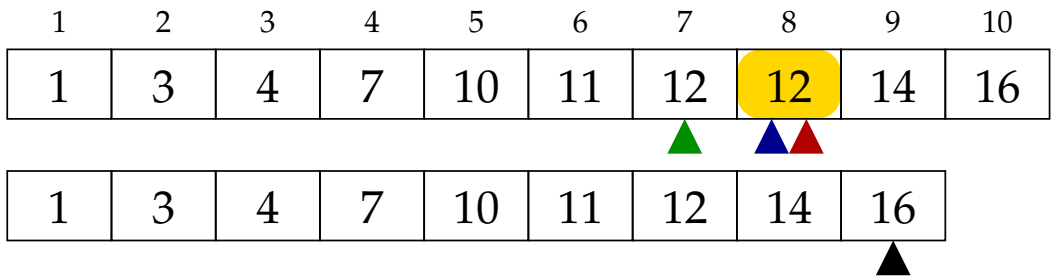
Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



Suppression d'un tableau trié

- Entrées :
  - un tableau tab$[1 \dots n]$
  - un élément e
- Sortie : $\begin{cases} \text{tab}[1 \dots n-1] = \text{tab}[1 \dots n] \backslash \{e\} & \text{si } e \in \text{tab} \\ \text{tab}[1 \dots n] & \text{sinon} \end{cases}$
- Idée générale :
  - rechercher par dichotomie l'élément
  - tasser le tableau sur la position de l'élément s'il a été trouvé
- Exemple : supprimer 12 du tableau



Plan du cours

Principes et objectifs du tri

- Pourquoi ?
  - réception d'informations par un algorithme $\rightsquigarrow$ organisation de celles-ci dans une structure de données
  - ensemble muni d'un ordre total $\rightsquigarrow$ intéressant de les classer / ordonner / trier selon cet ordre

- Une relation binaire R sur un ensemble E est une relation d'ordre si et seulement si elle est :
  - réflexive : $\forall i \in E,\ iRi$
  - anti-symétrique : $\forall i,j \in E,\ iRj \implies i \not R j$
  - transitive : $\forall i,j,k \in E,\ iRj$ et $jRk \implies iRk$

- Trier un tableau, c'est classer ses éléments selon un ordre total

- Plusieurs algorithmes, plus ou moins complexes, existent

Tri par remplacement

- Principe : construire un tableau $t'[1 \dots n]$ trié composé des mêmes éléments que $t[1 \dots n]$.
- $\forall i \in [2 \dots n],\ t'[i-1] \leqslant t'[i] \quad \Longrightarrow \quad t'[1] = \min(t[1 \dots n])$

- Exemple :

i	t	j (indice de min(t))	t (après remplacement)	t'
1	B**A**TEAUX	2	BXTEAUX	A
2	BXTE**A**UX	5	BXTEXUX	AA
3	**B**XTEXUX	1	XXTEXUX	AAB
4	XXT**E**XUX	4	XXTXXUX	AABE
5	XX**T**XXUX	3	XXXXXUX	AABET
6	XXXXX**U**X	6	XXXXXXX	AABETU
7	XXXXXXX	_		AABETUX

Tri par remplacement

Algorithme

```
Fonction i_min(d t[1...n] : tab_t) : entier
    i, j : entier ;
Début
    j := 1 ;
    i := 2 ;
    Tant que (i ⩽ n) faire
        Si (t[j] > t[i]) alors
            j := i ;
        Fin si
        i := i + 1;
    Fin faire
    retour j ;
Fin
```

```
Fonction maxi(d t[1...n] : tab_t) : t
    i : entier ;
    max : t ;
Début
    max := t[1] ;
    i := 2 ;
    Tant que (i ⩽ n) faire
        Si (t[i] > max) alors
            max := t[i] ;
        Fin si
        i := i + 1;
    Fin faire
    retour max ;
Fin
```

Tri par remplacement

Algorithme

Procédure tri_remplacement(**d** t$[1 \ldots n]$: **tab_t**, **r** t$'[1 \ldots n]$: **tab_t**)
    max : **t** ;
    i, j : **entier** ;
Début
    max := maxi(t$[1 \ldots n]$) ;
    $i := 1$;
    Tant que $(i < n)$ **faire**
        j := i_min(t$[1 \ldots n]$) ;
        t$'[i]$:= t$[j]$;
        t$[j]$:= max ;
        $i := i + 1$;
    Fin faire
    t$'[n]$:= max ;
Fin

Tri par remplacement

Complexité

- Opérations élémentaires effectuées dans le pire cas :
  - En dehors de toute boucle, 3 affectations et appel de maxi :
    - 2 affectations avant l'itération et 1 retour après l'itération
    - $2n$ affectations et n additions dans l'itération sur i

    soit $3n+6$ opérations
  - Dans l'itération sur i, 4 affectations, 1 addition et appel de i_min :
    - 2 affectations avant l'itération et 1 retour après l'itération
    - $2n$ affectations et n additions dans l'itération sur i

    soit $(n-1)(3n+8)$ opérations

- Nombre total d'opérations élémentaires :

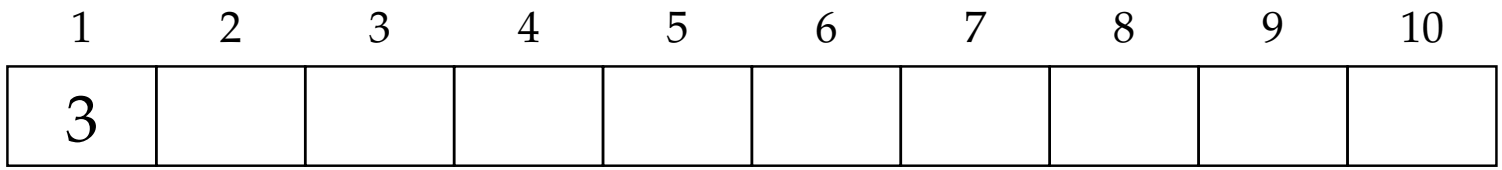
$$3n+6+(n-1)(3n+8) = 3n+6+3n^2+5n-8 = 3n^2+8n-2,$$

majoré asymptotiquement par $c \cdot n^2$, pour $c > 3$

$$\longrightarrow O(n^2)$$

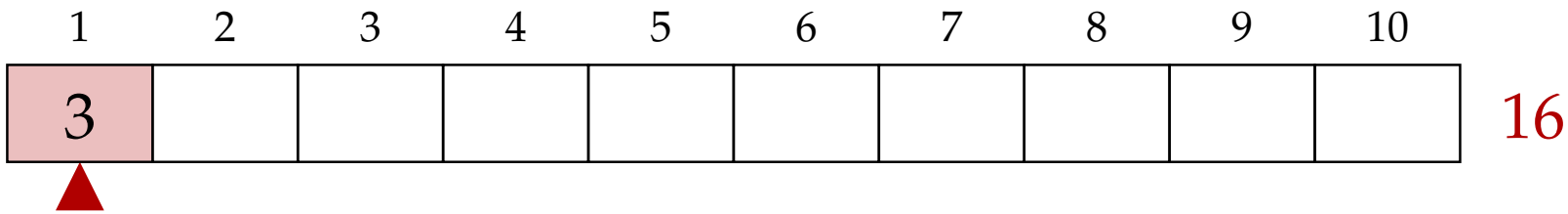
Tri par insertion

Idée générale



Tri par insertion

Idée générale



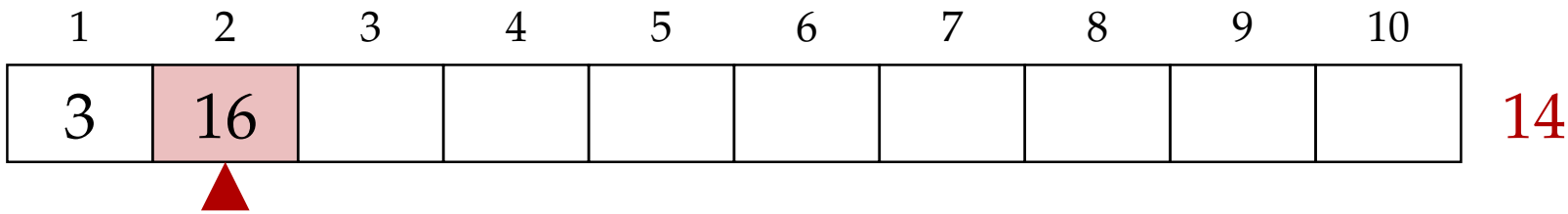
Tri par insertion

Idée générale

1	2	3	4	5	6	7	8	9	10
3	16								

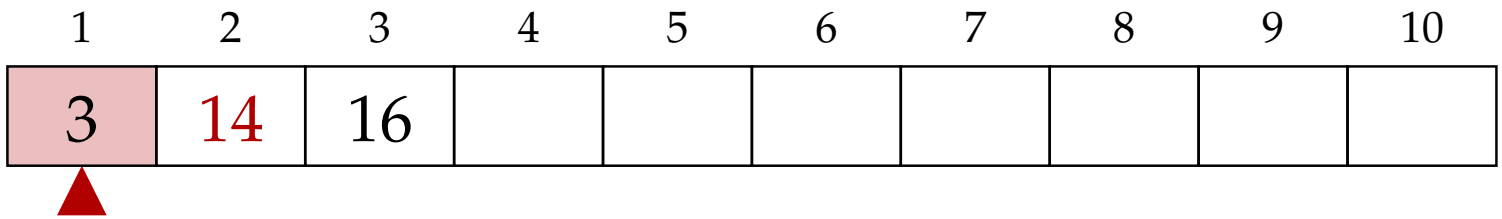
Tri par insertion

Idée générale



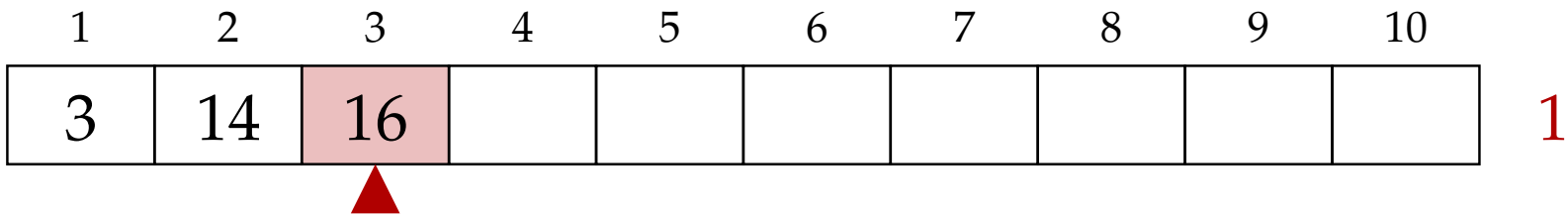
Tri par insertion

Idée générale



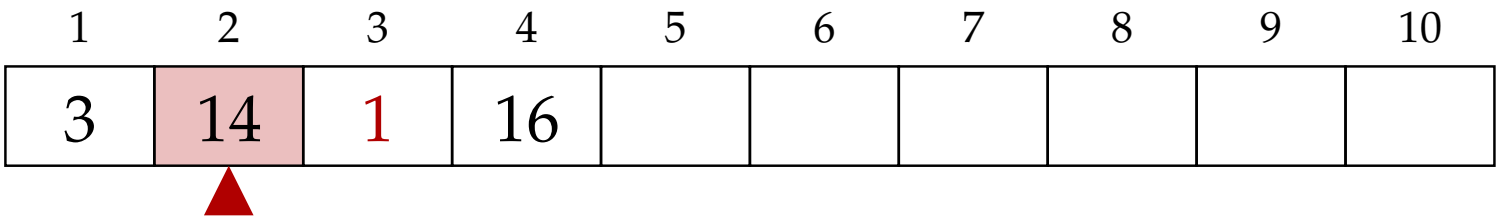
Tri par insertion

Idée générale



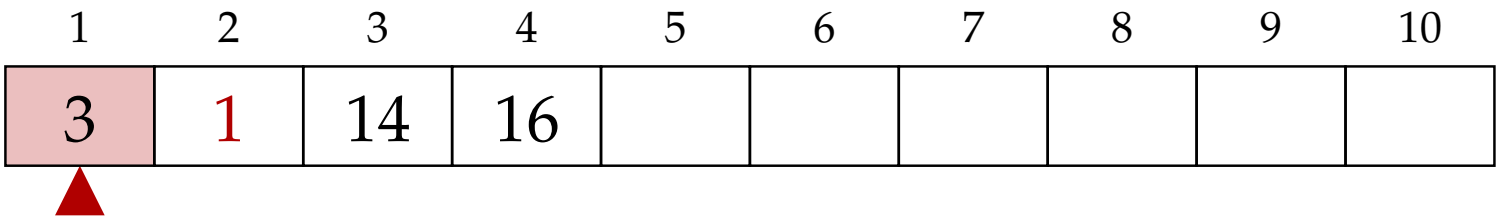
Tri par insertion

Idée générale



Tri par insertion

Idée générale



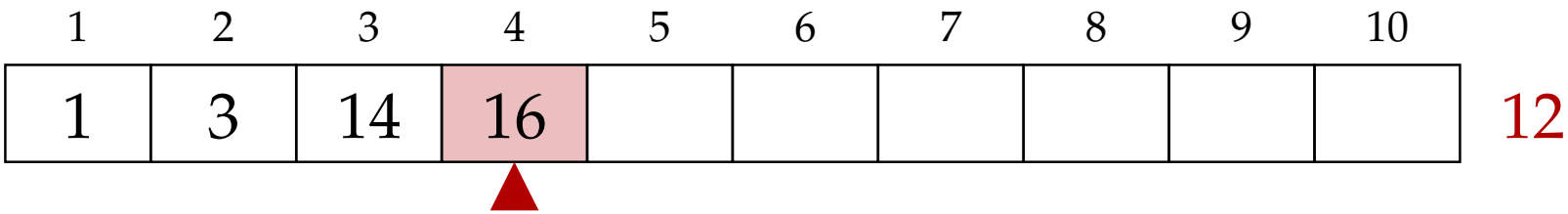
Tri par insertion

Idée générale

1	2	3	4	5	6	7	8	9	10
1	3	14	16						

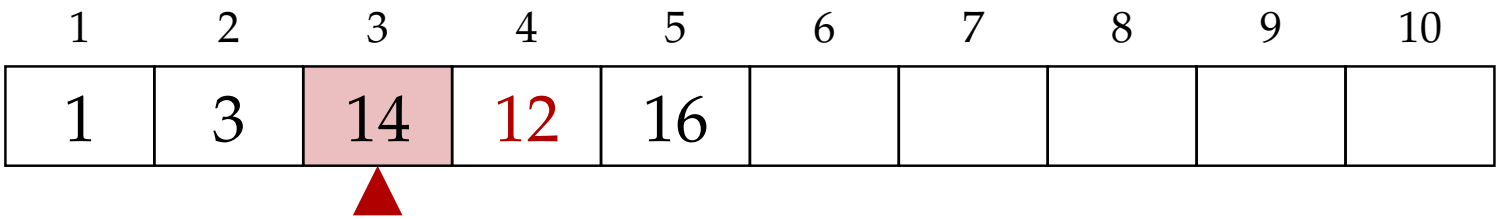
Tri par insertion

Idée générale



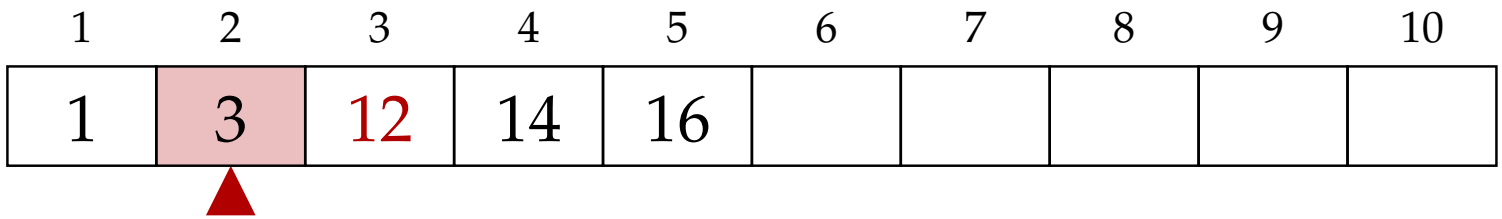
Tri par insertion

Idée générale



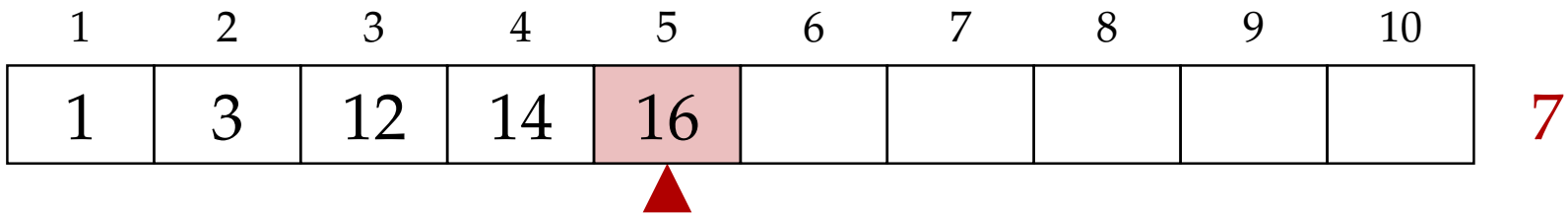
Tri par insertion

Idée générale



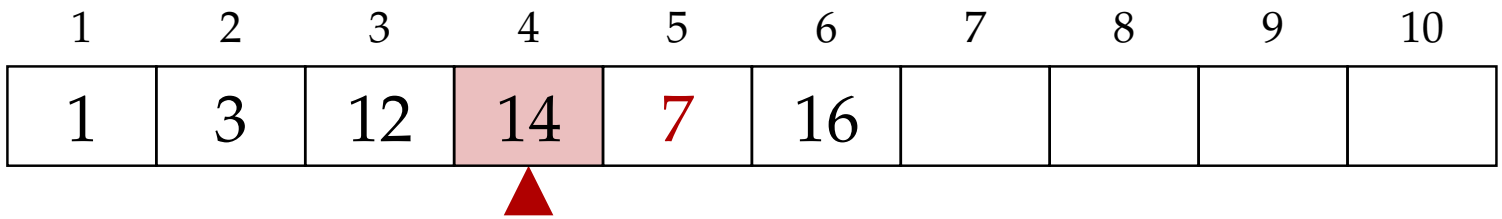
Tri par insertion

Idée générale



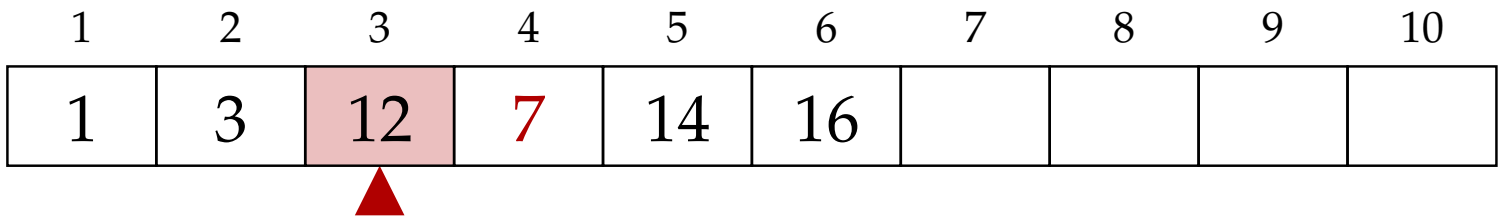
Tri par insertion

Idée générale



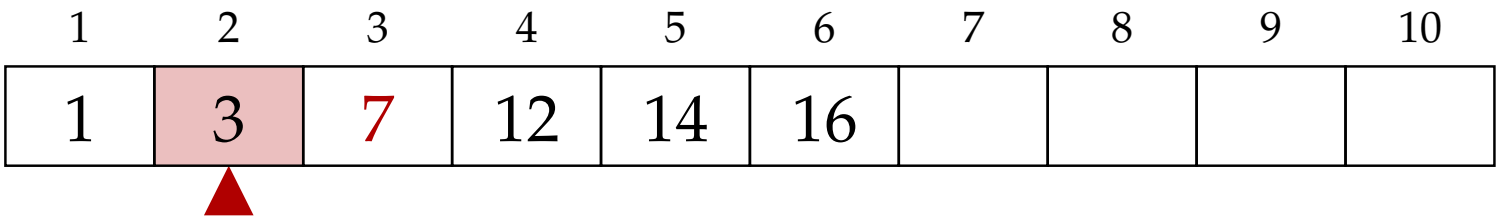
Tri par insertion

Idée générale



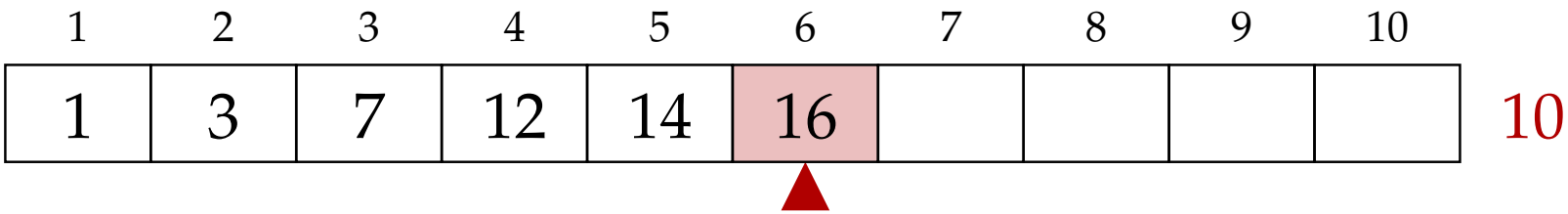
Tri par insertion

Idée générale



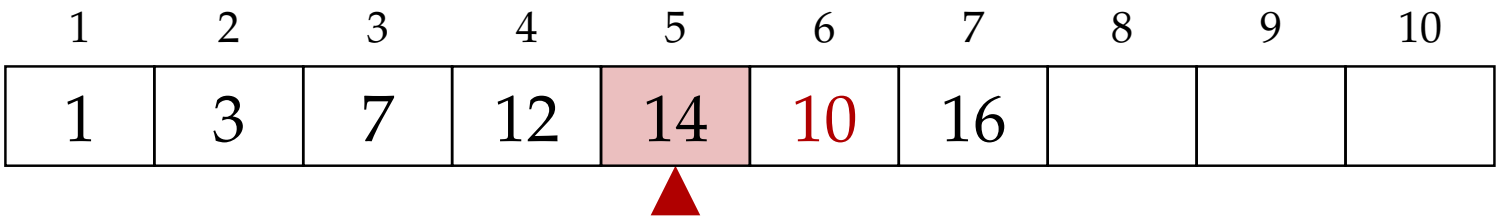
Tri par insertion

Idée générale



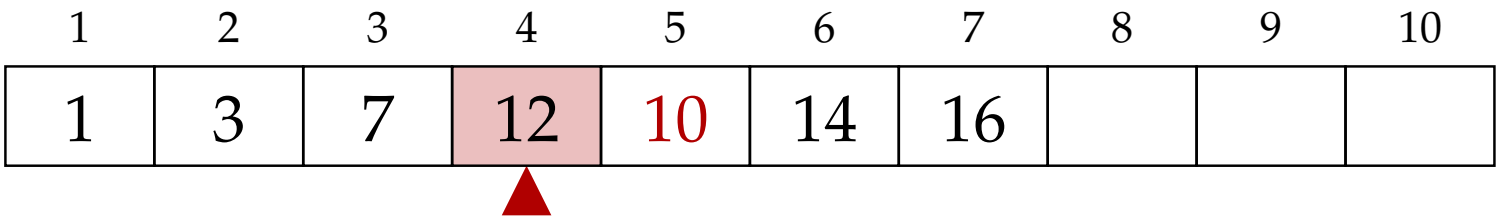
Tri par insertion

Idée générale



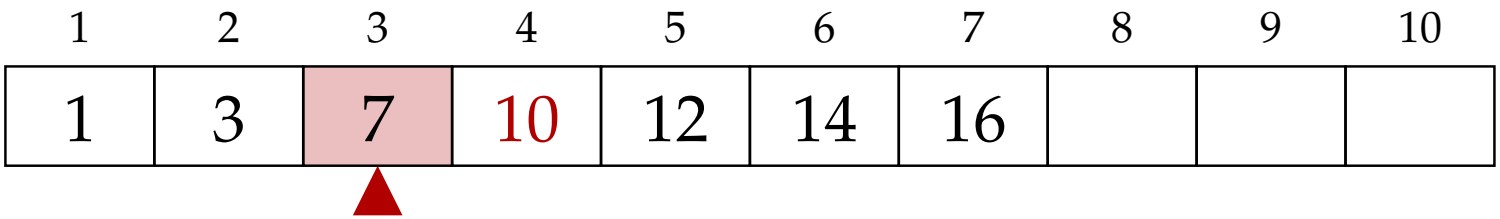
Tri par insertion

Idée générale



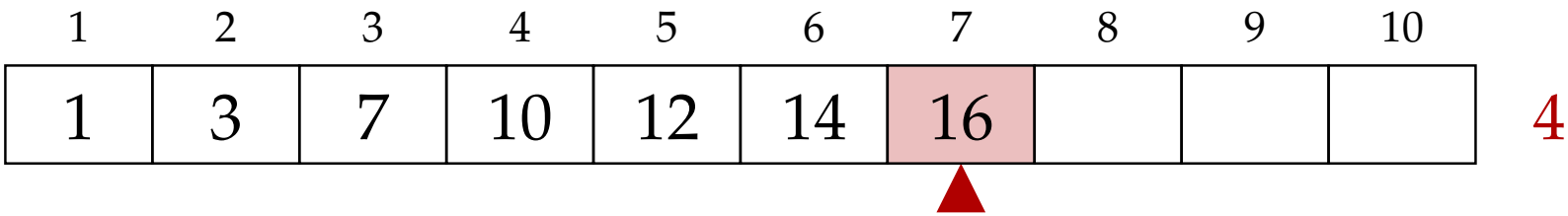
Tri par insertion

Idée générale



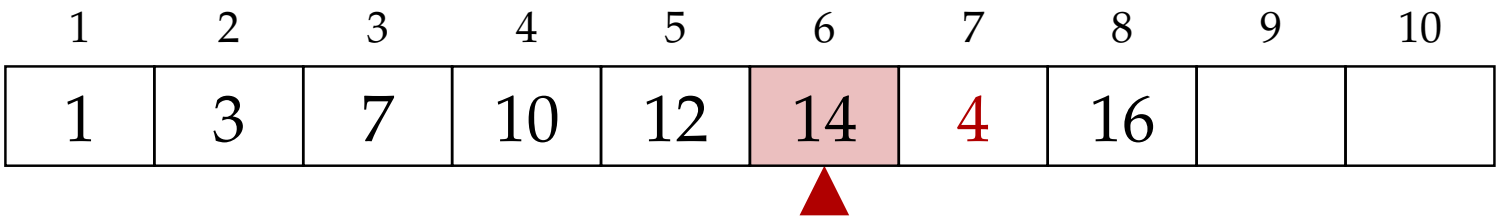
Tri par insertion

Idée générale



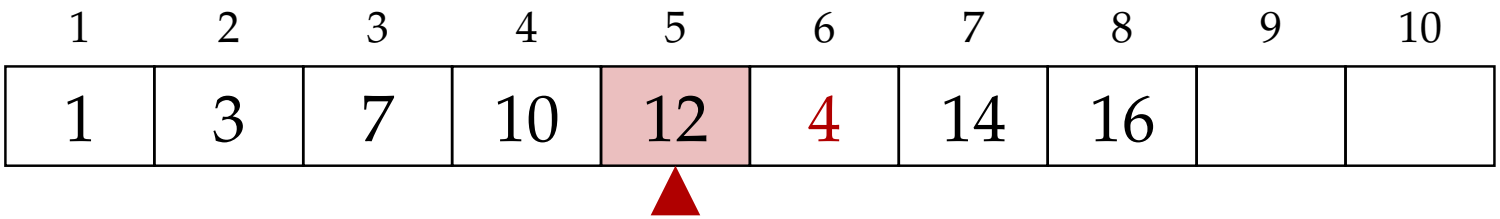
Tri par insertion

Idée générale



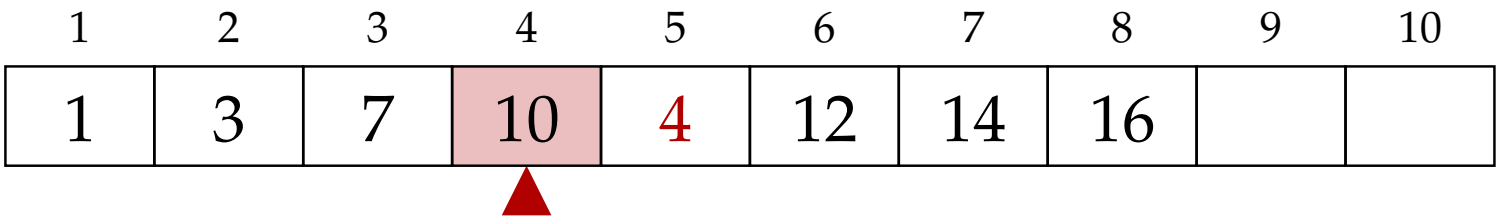
Tri par insertion

Idée générale



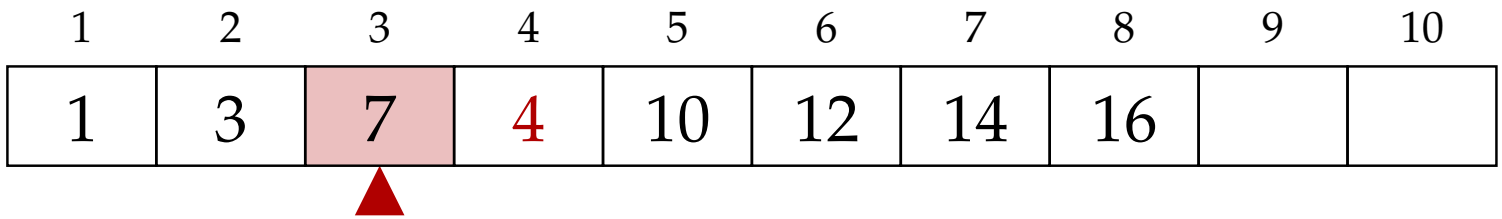
Tri par insertion

Idée générale



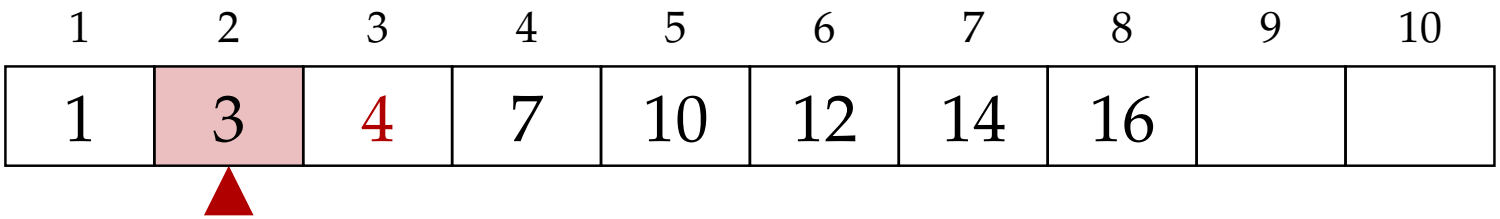
Tri par insertion

Idée générale



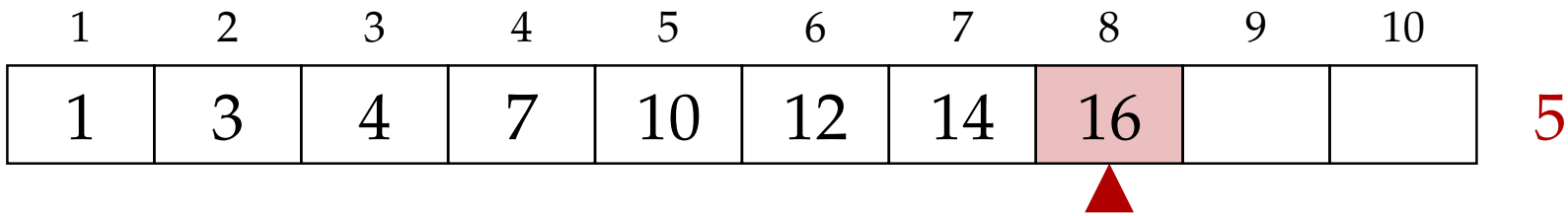
Tri par insertion

Idée générale



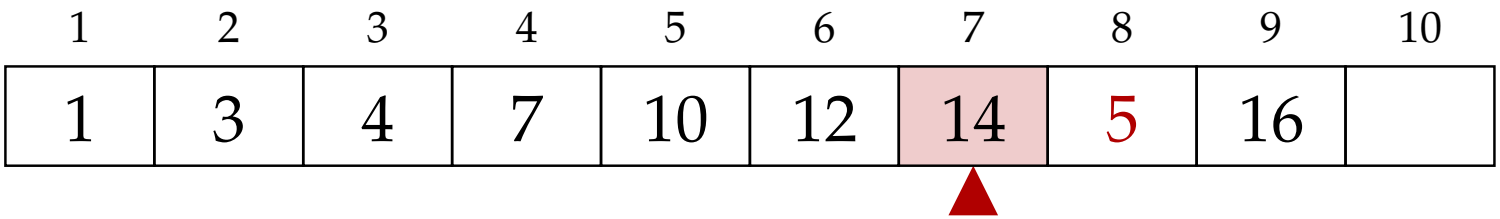
Tri par insertion

Idée générale



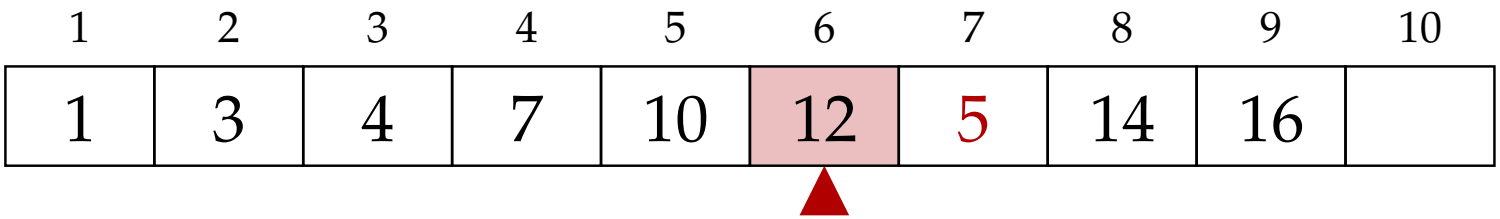
Tri par insertion

Idée générale



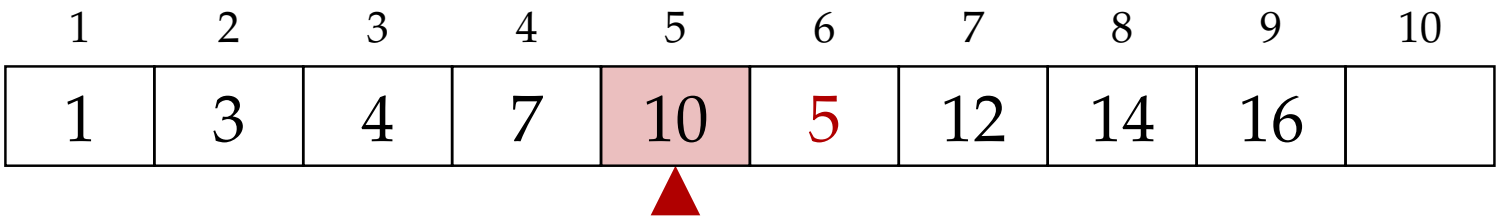
Tri par insertion

Idée générale



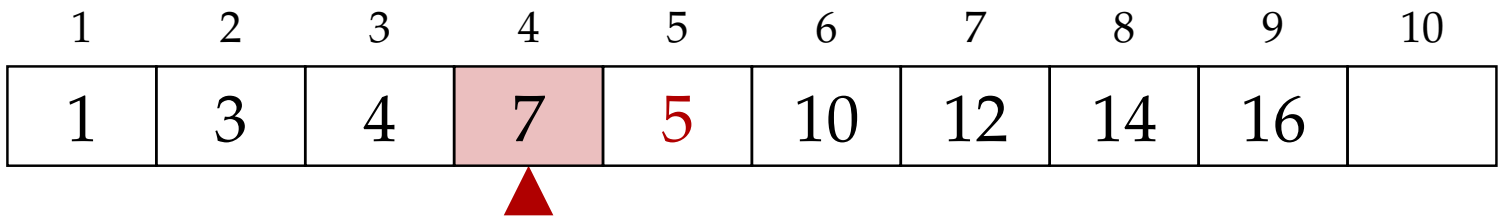
Tri par insertion

Idée générale



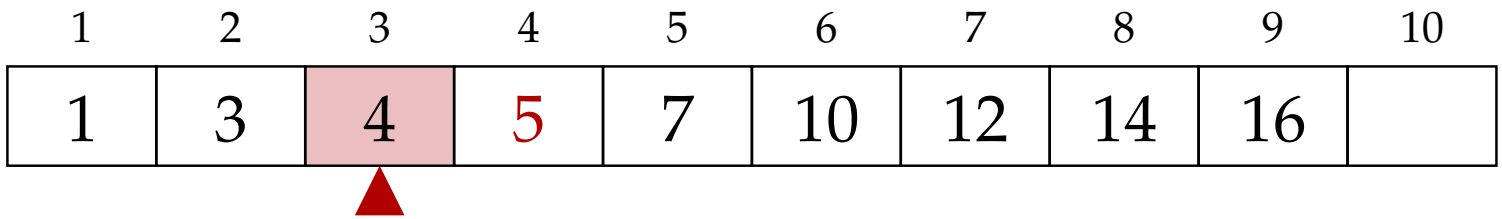
Tri par insertion

Idée générale



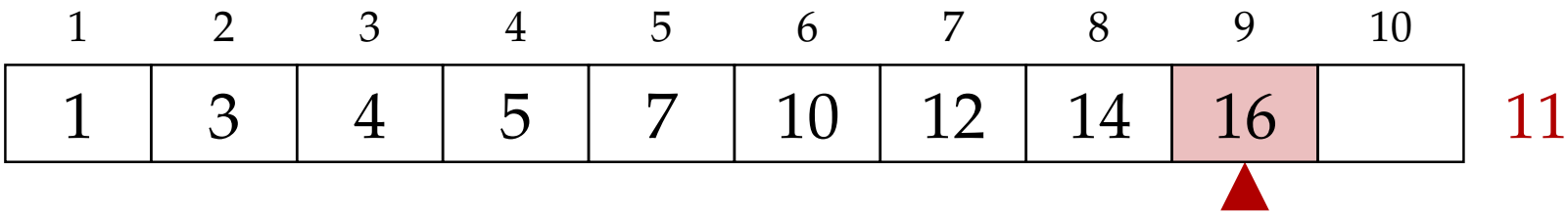
Tri par insertion

Idée générale



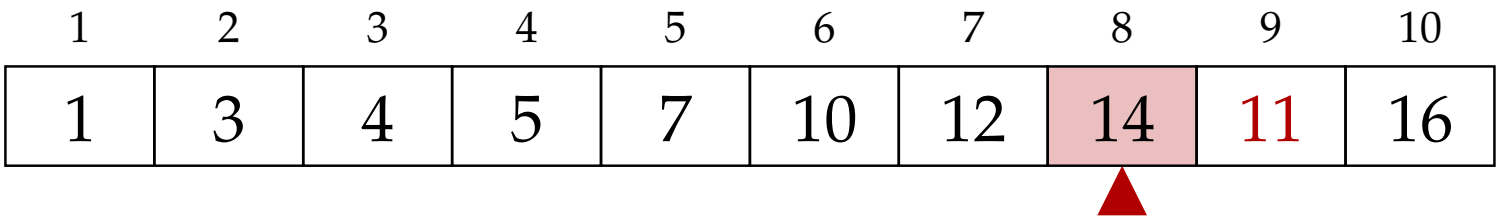
Tri par insertion

Idée générale



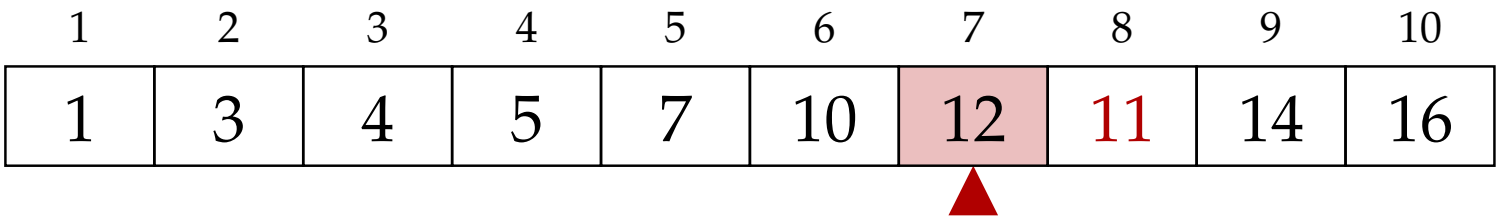
Tri par insertion

Idée générale



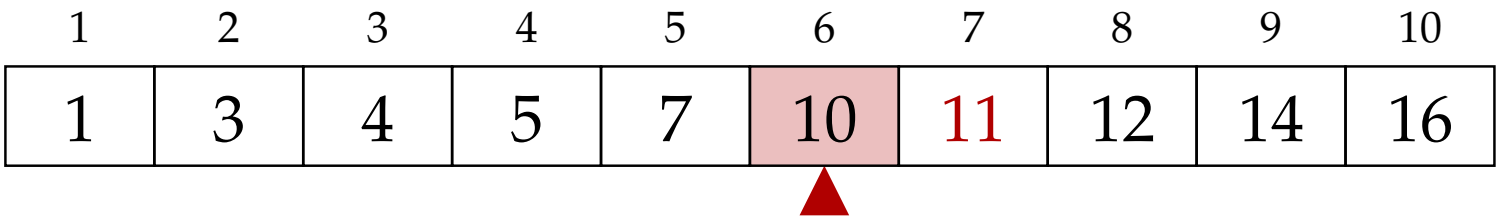
Tri par insertion

Idée générale



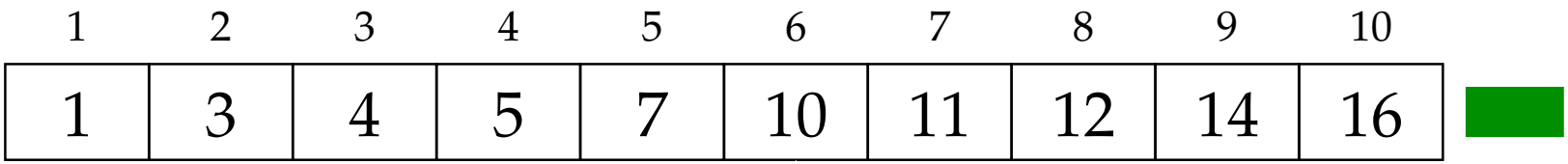
Tri par insertion

Idée générale



Tri par insertion

Idée générale



Tri par insertion

Algorithme

Procédure tri_insertion(tab$[1 \ldots n]$: tab_t, n : **entier**)
    i, j, x : **entier** ;
Début
    $i := 2$;
    Tant que $i \leqslant n$ **faire**
        $x :=$ tab$[i]$;
        $j := i$;
        Tant que $(j > 1)$ et $(\text{tab}[j-1] > x)$ **faire**
            tab$[j] :=$ tab$[j-1]$;
            $j := j - 1$;
        Fin faire
        tab$[j] := x$;
        $i := i + 1$;
    Fin faire
Fin

Tri par insertion

Complexité

- Opérations élémentaires effectuées dans le pire cas :
  - En dehors de toute boucle : 1 affectation.
  - Dans l'itération sur i de $n-1$ étapes (hors itération sur j) : 4 affectations et 1 addition.
  - Dans l'itération sur j imbriquée dans celle sur i :
    $\rightarrow$ 2 affectations et 1 addition
    $\rightarrow$ répétées en tout $\frac{(n-1)\cdot n}{2}$

- Nombre total d'opérations élémentaires :

$$3(\frac{n^2-n}{2})+5(n-1)+1 \;=\; \frac{3n^2-3n+10n-10+2}{2} \;=\; \frac{3n^2+7n-8}{2},$$

majoré asymptotiquement par $c\cdot n^2$, pour $c>3$

$$\longrightarrow O(n^2)$$

Tri par insertion

Application

Vidéo

Récursivité

- Un algorithme (ou fonction) qui s'appelle lui-même (ou elle-même) est récursif
- Exemple : calculer $n!$

```
Fonction fiter(n : entier) : entier
    i, f : entier ;
Début
    f := 1 ;
    i := 2 ;
    Tant que (i ⩽ n) faire
        f := f × i ;
        i := i + 1 ;
    Fin faire
    retour f ;
Fin
```

```
Fonction frec(n : entier) : entier
Début
    Si n = 2 alors
        retour 2 ;
    Sinon
        retour n × fact_rec(n − 1) ;
    Fin si
Fin
```

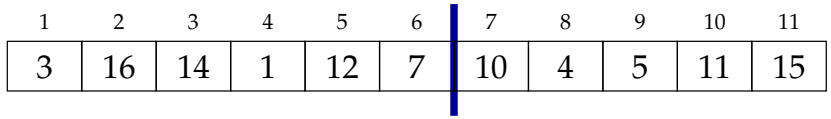
Tri fusion

- Algorithme de tri par comparaison stable
- Technique du "diviser pour régner"
- N'opérant pas en place
- Idée générale :
  1. Si le tableau n'a qu'un élément, il est déjà trié
  2. Sinon, séparer le tableau en 2 parties égales (à 1 près)
  3. trier récursivement les 2 parties avec l'algorithme
  4. Fusionner les deux tableaux triés en un seul tableau trié

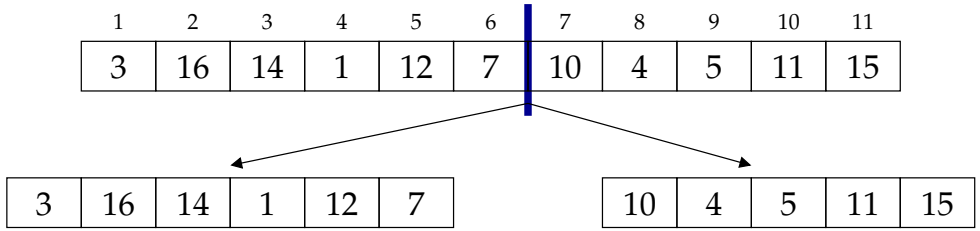
Tri fusion

1	2	3	4	5	6	7	8	9	10	11
3	16	14	1	12	7	10	4	5	11	15

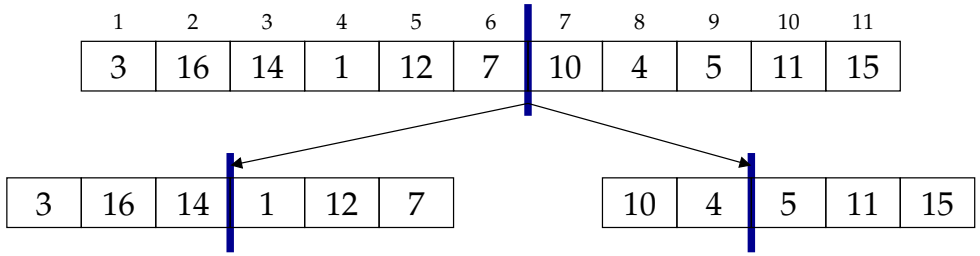
Tri fusion



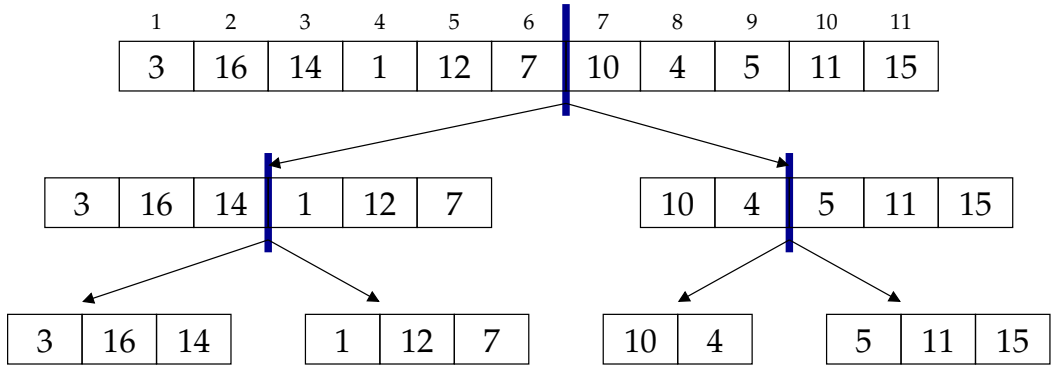
Tri fusion



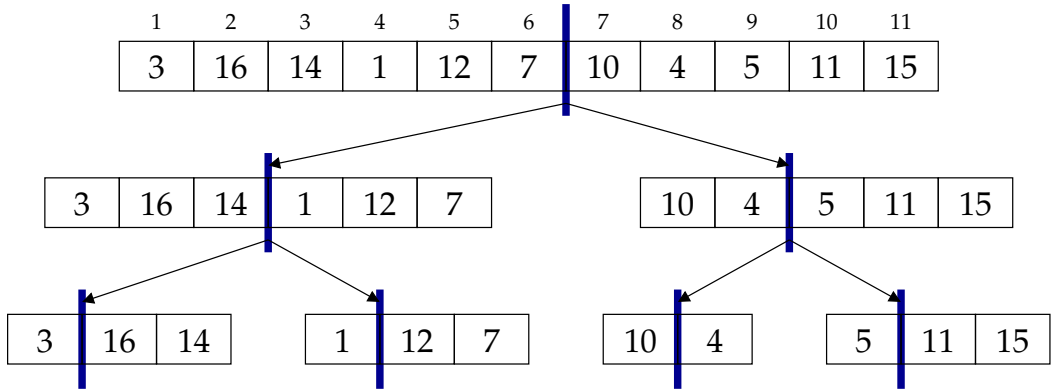
Tri fusion



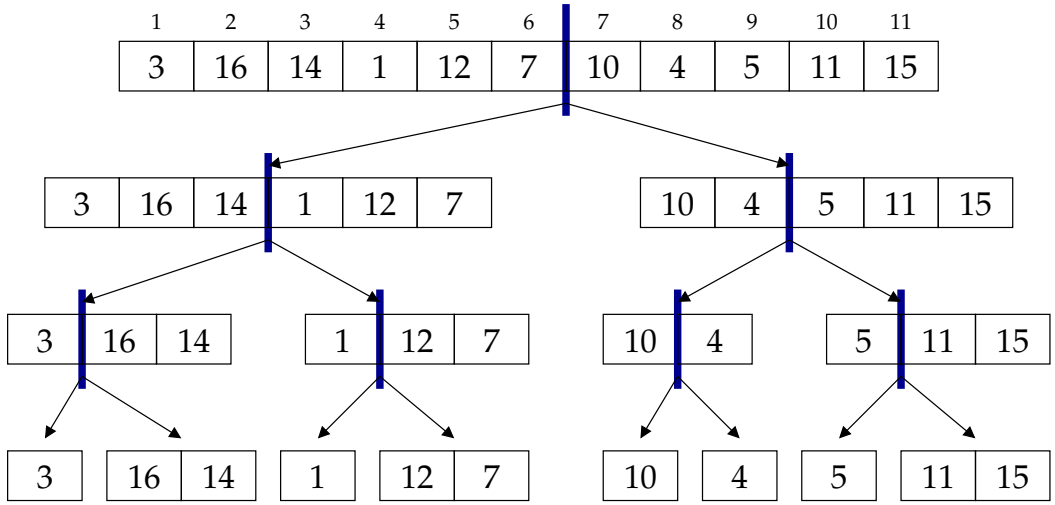
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15

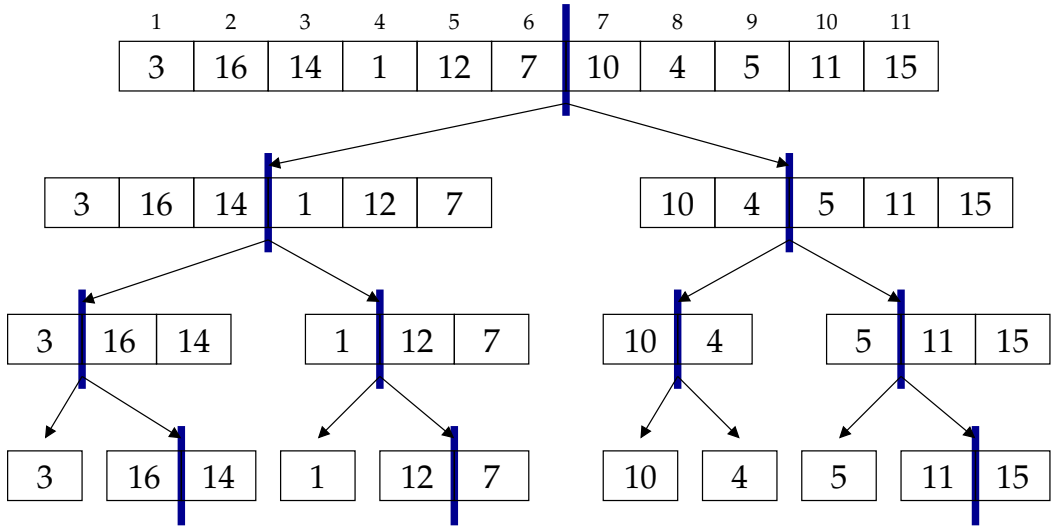
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15

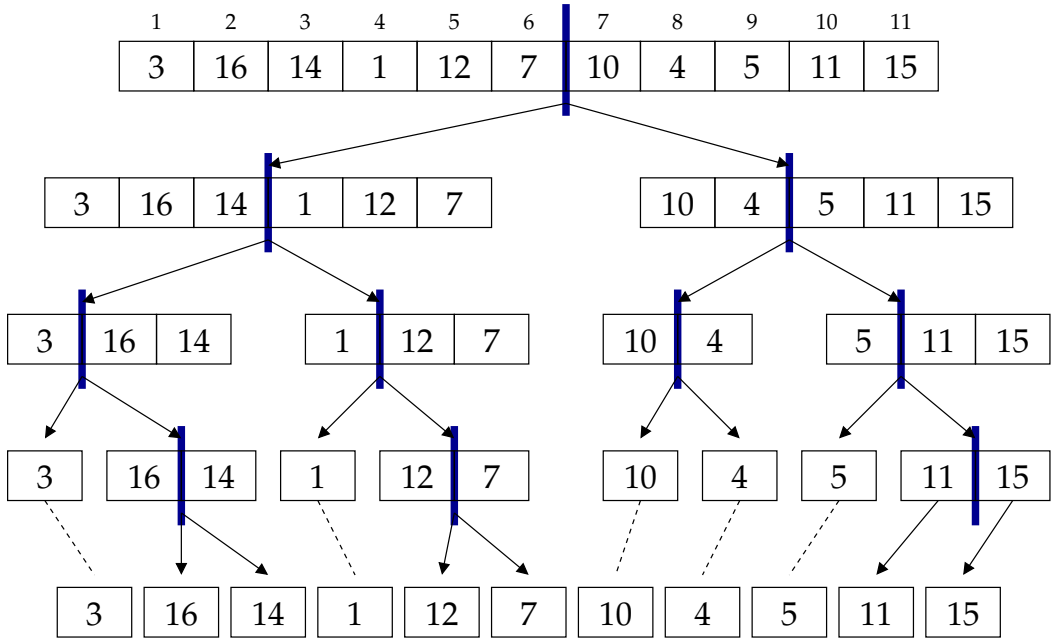
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15

Tri fusion

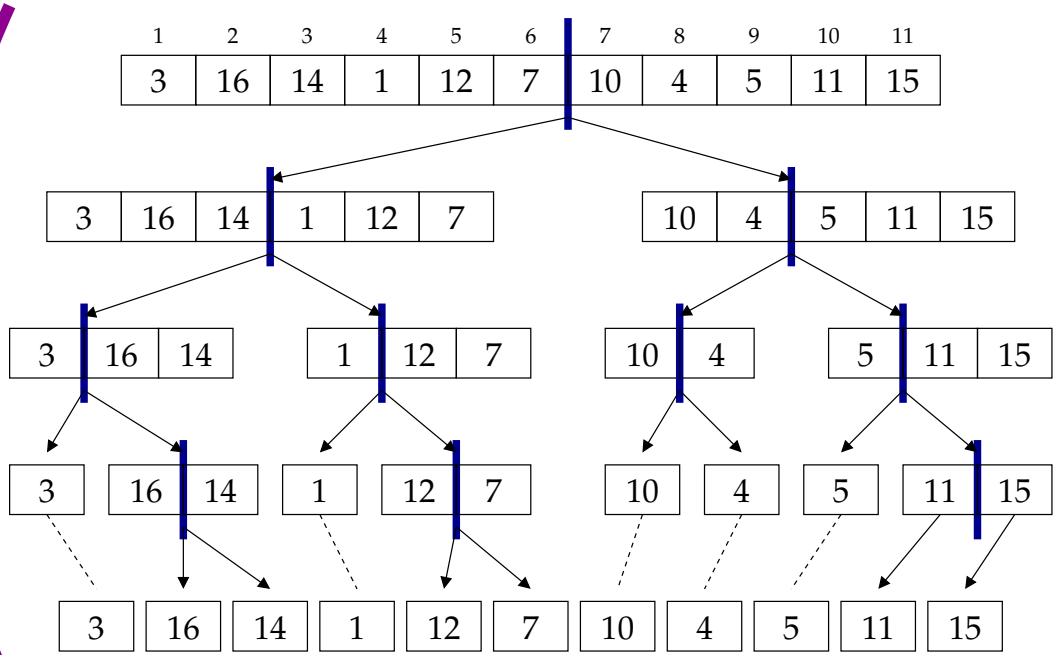

1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15

Tri fusion

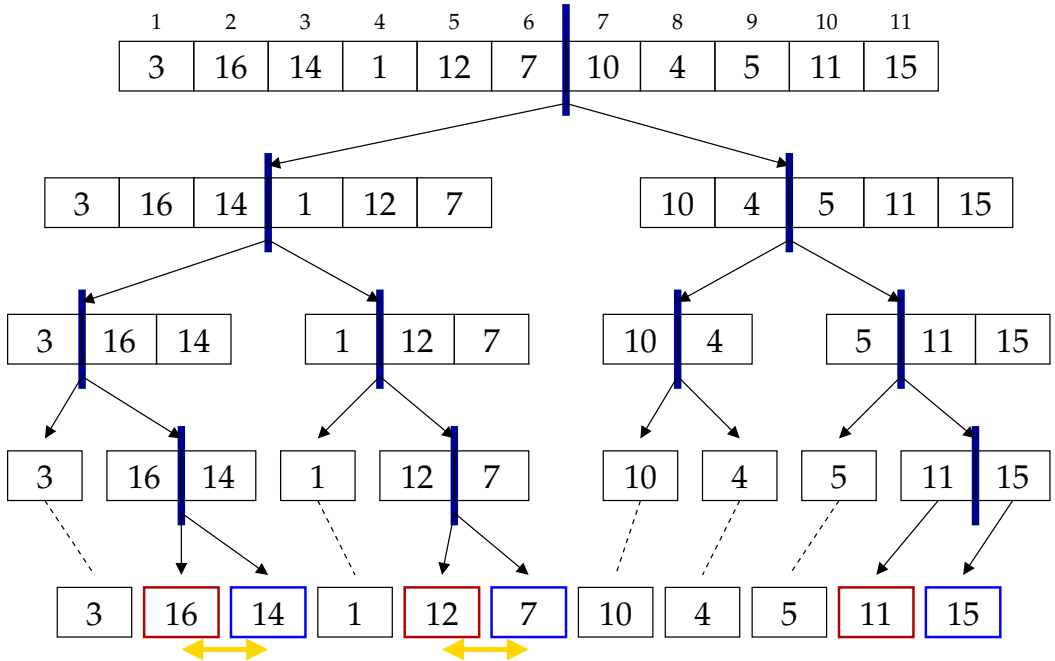

1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3 16 14 1 12 7 10 4 5 11 15

Tri fusion

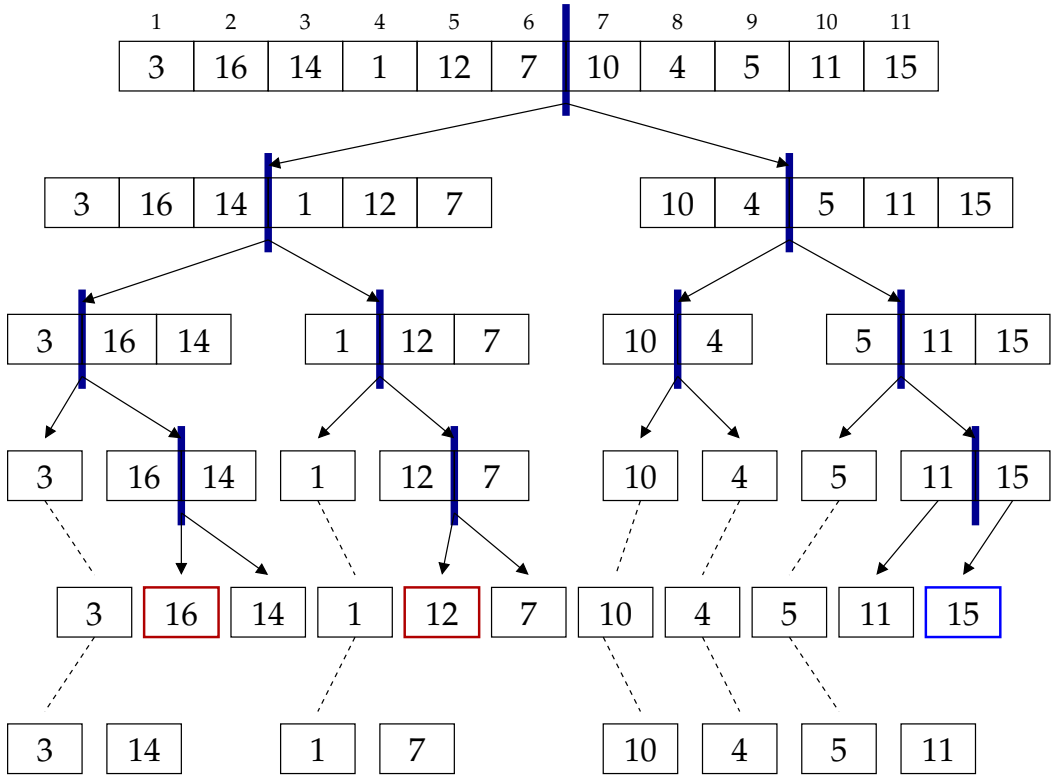
DIVISER



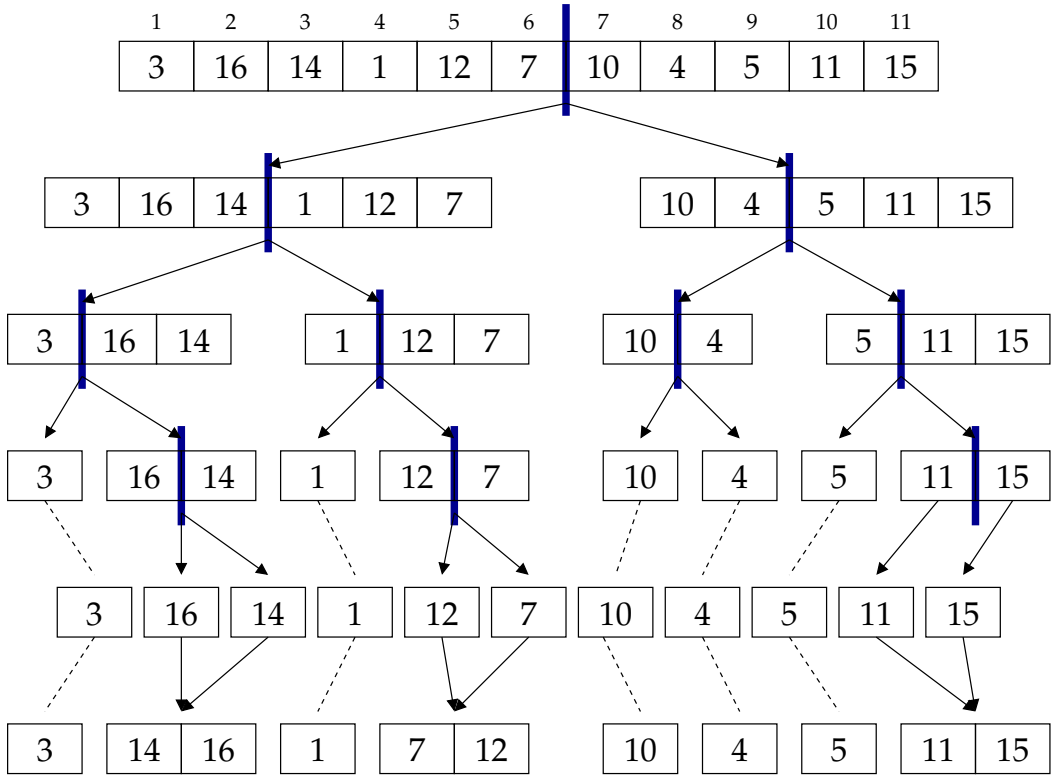
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3 16 14 1 12 7 10 4 5 11 15

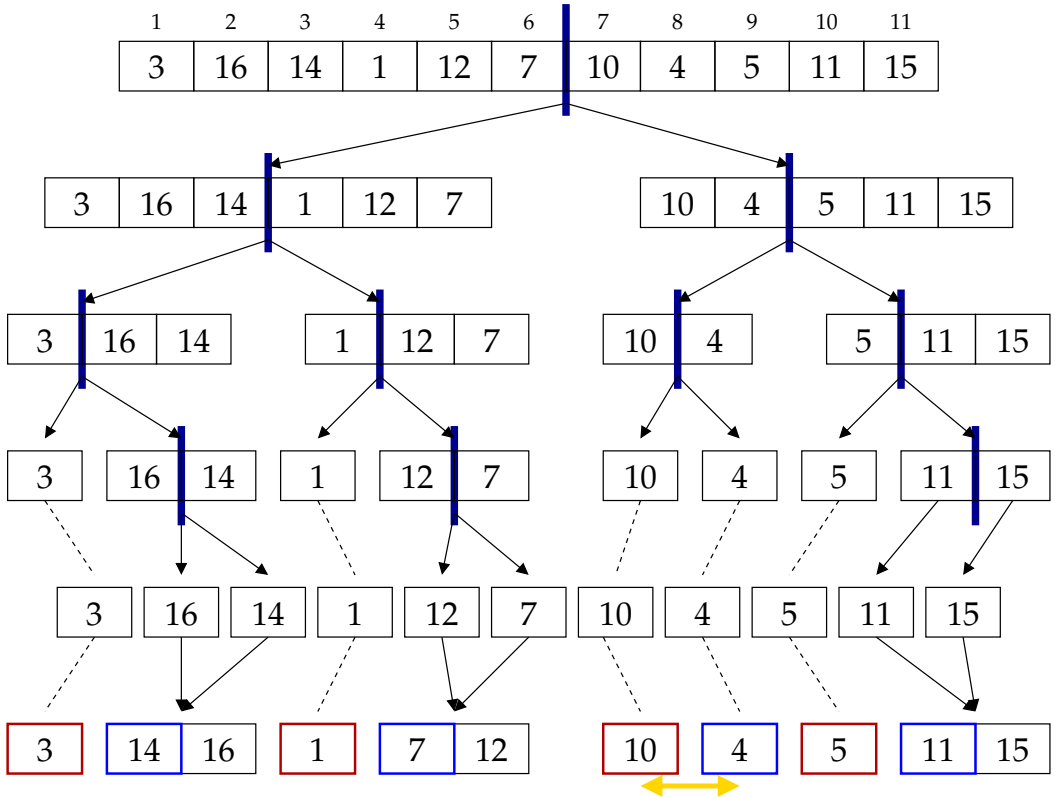
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3 16 14 1 12 7 10 4 5 11 15
3 14
1 7
10 4 5 11

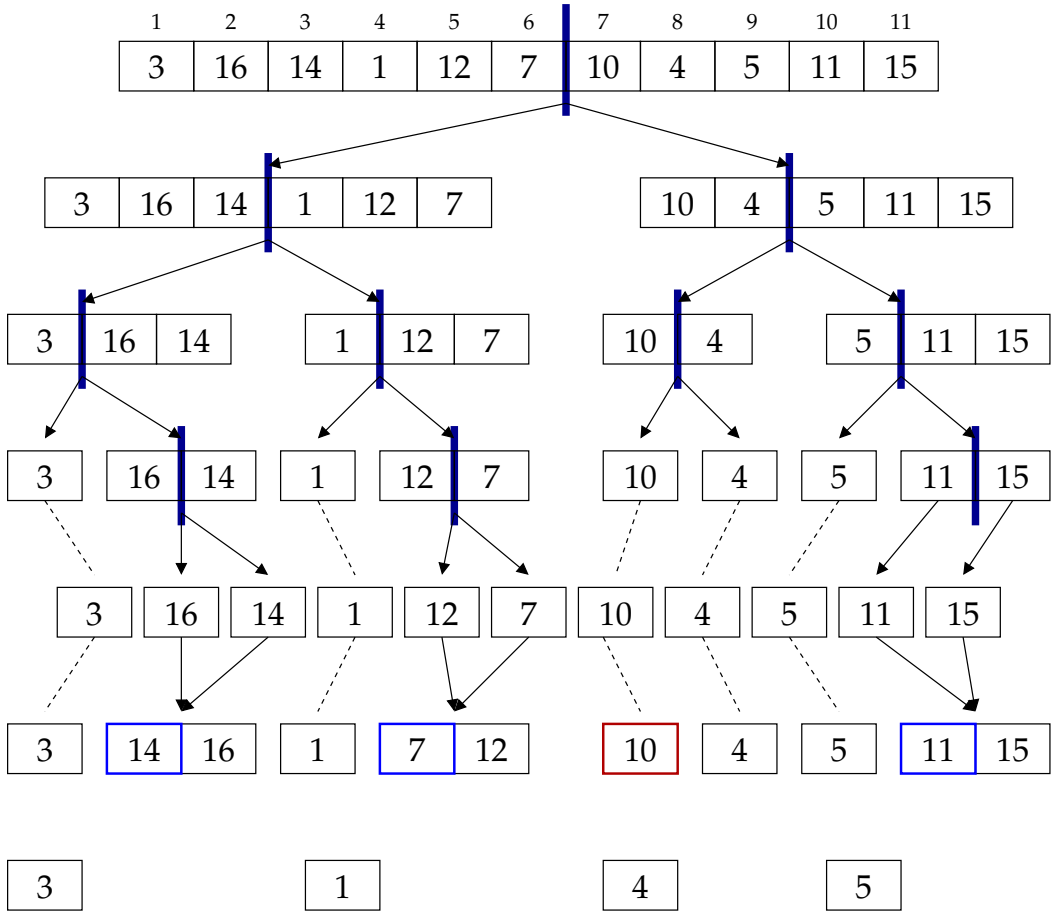
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15

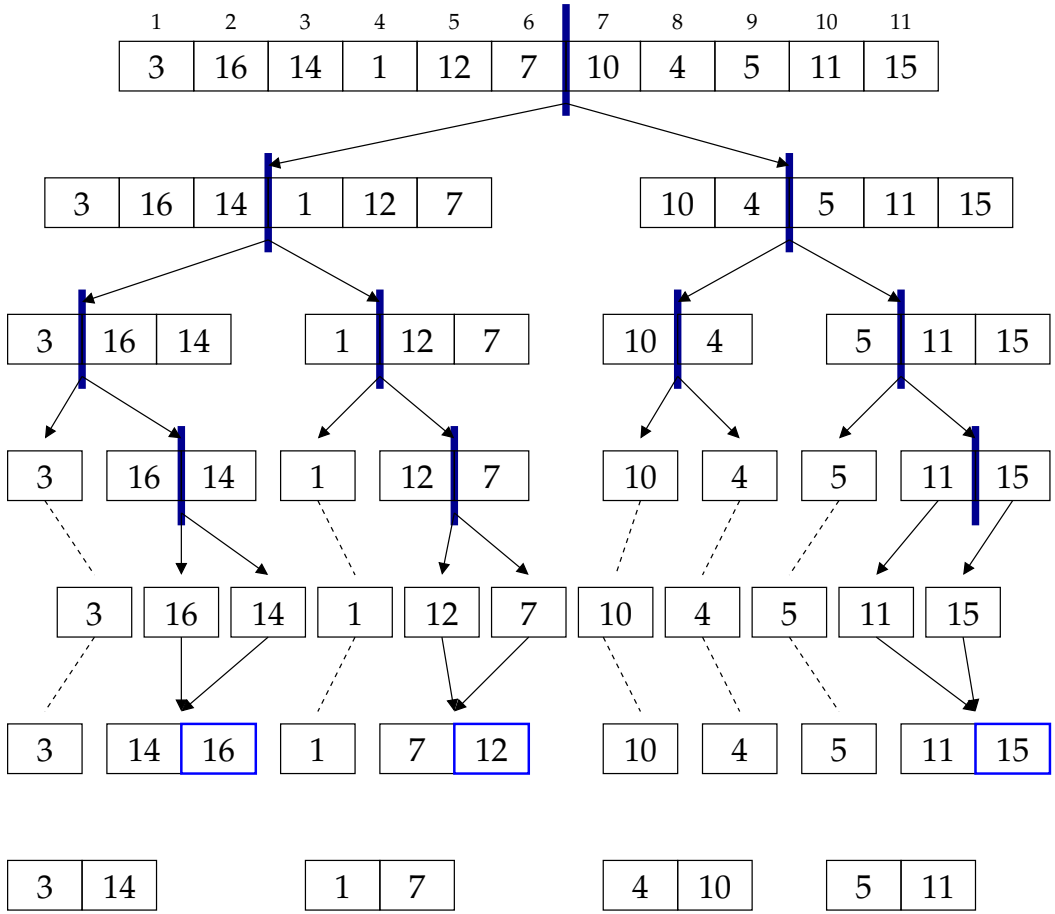
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15

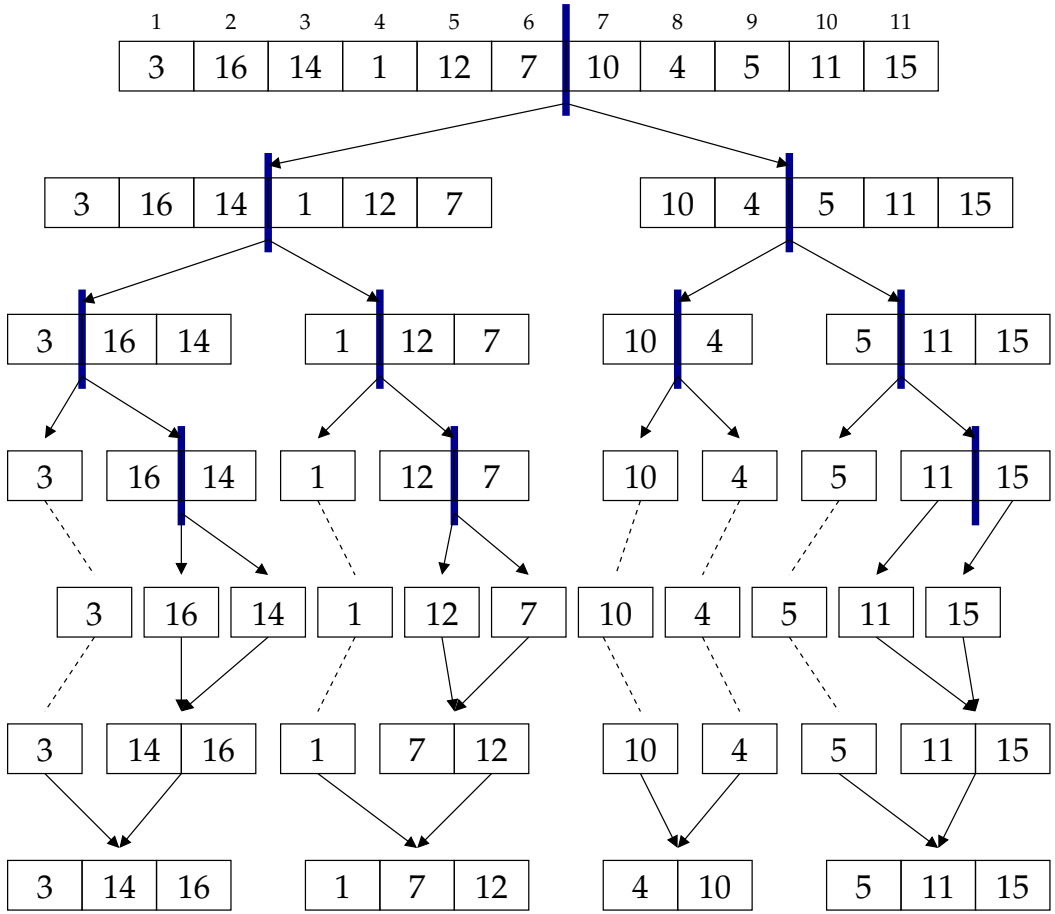
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
1
4
5

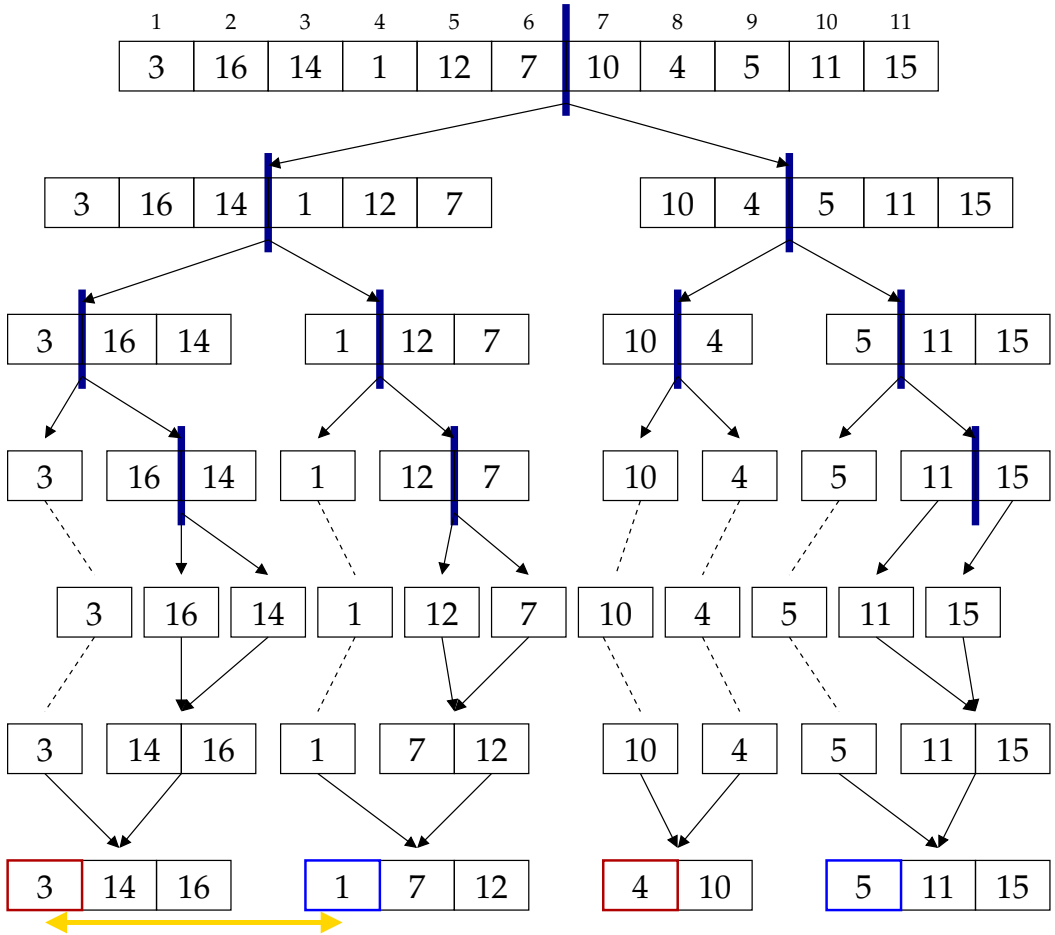
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3 14
1 7
4 10
5 11

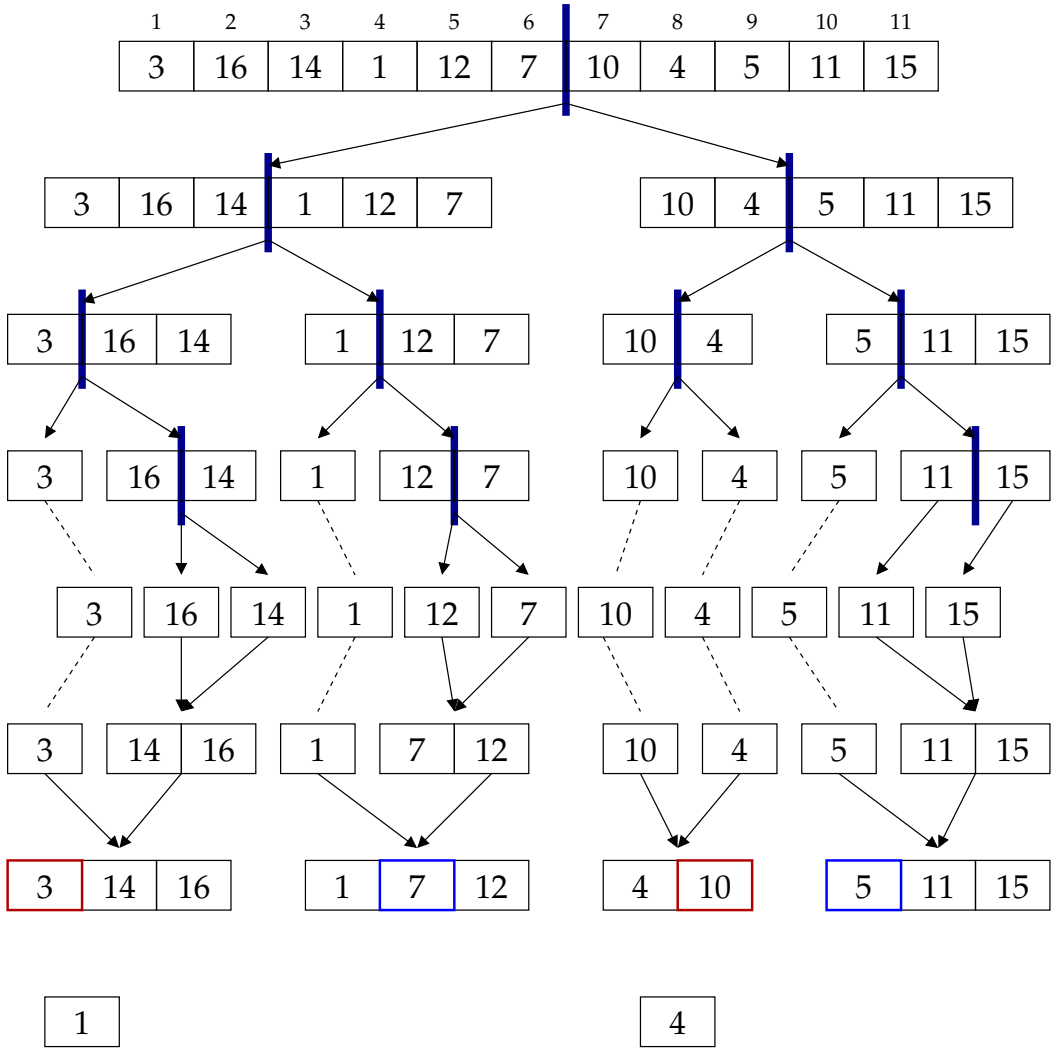
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3 14 16
1 7 12
4 10
5 11 15

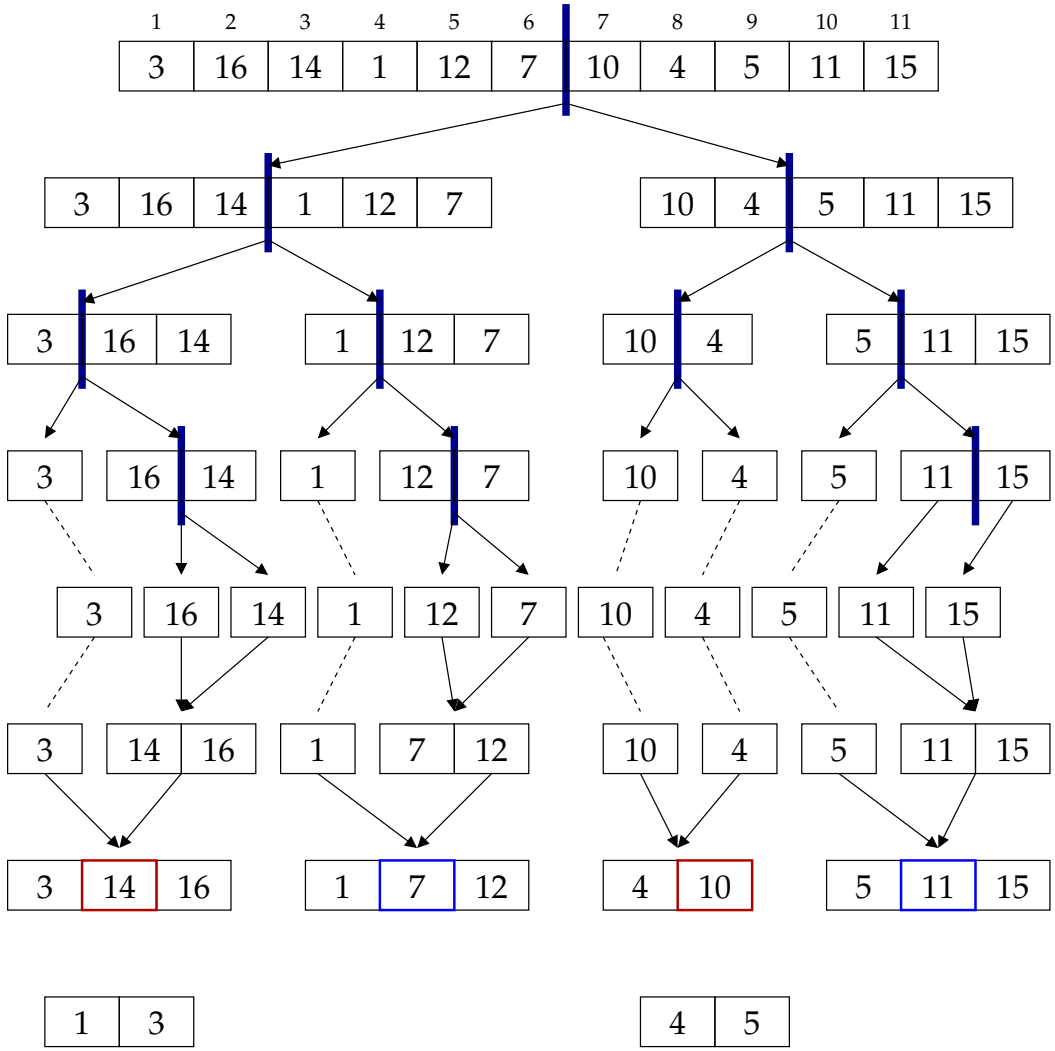
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14
16
1
7
12
4
10
5
11
15

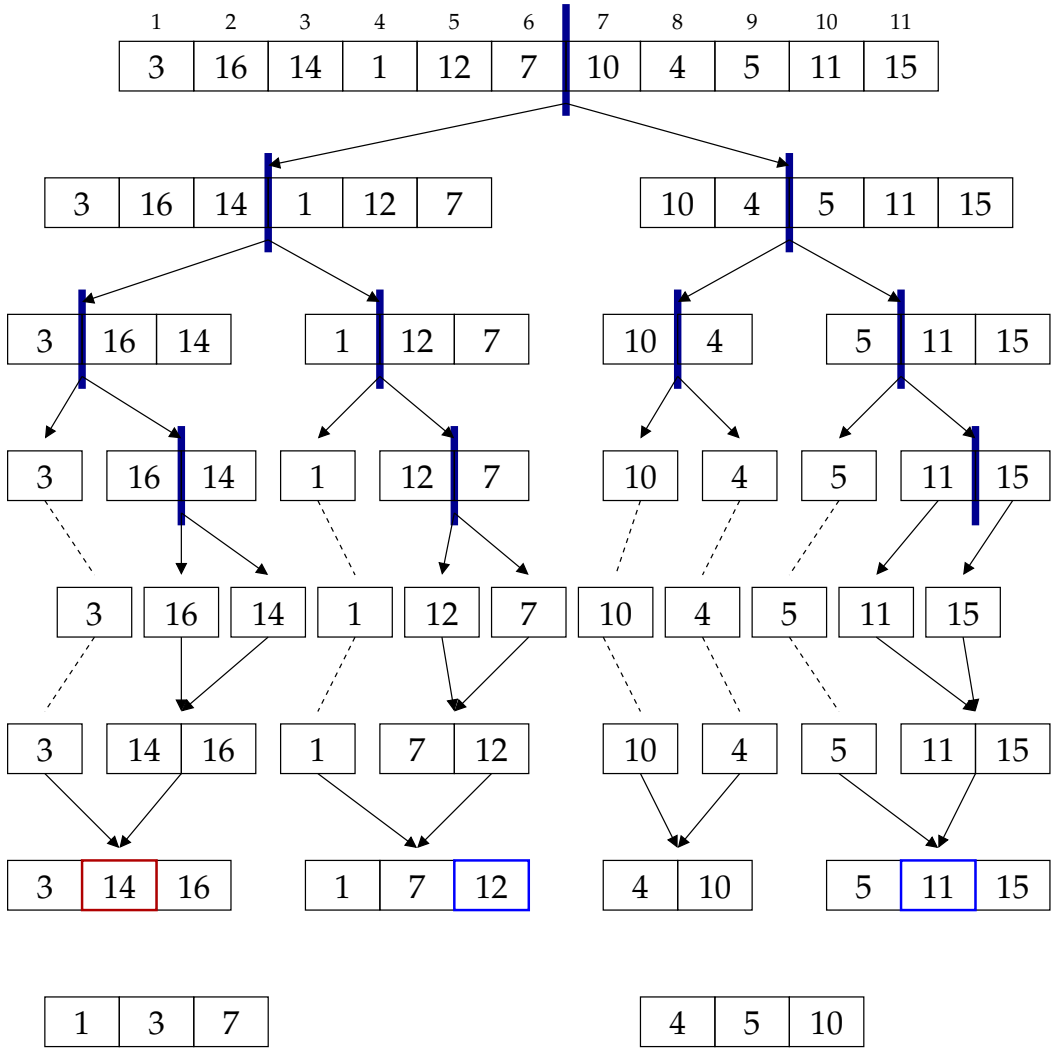
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14
16
1
7
12
4
10
5
11
15
1
4

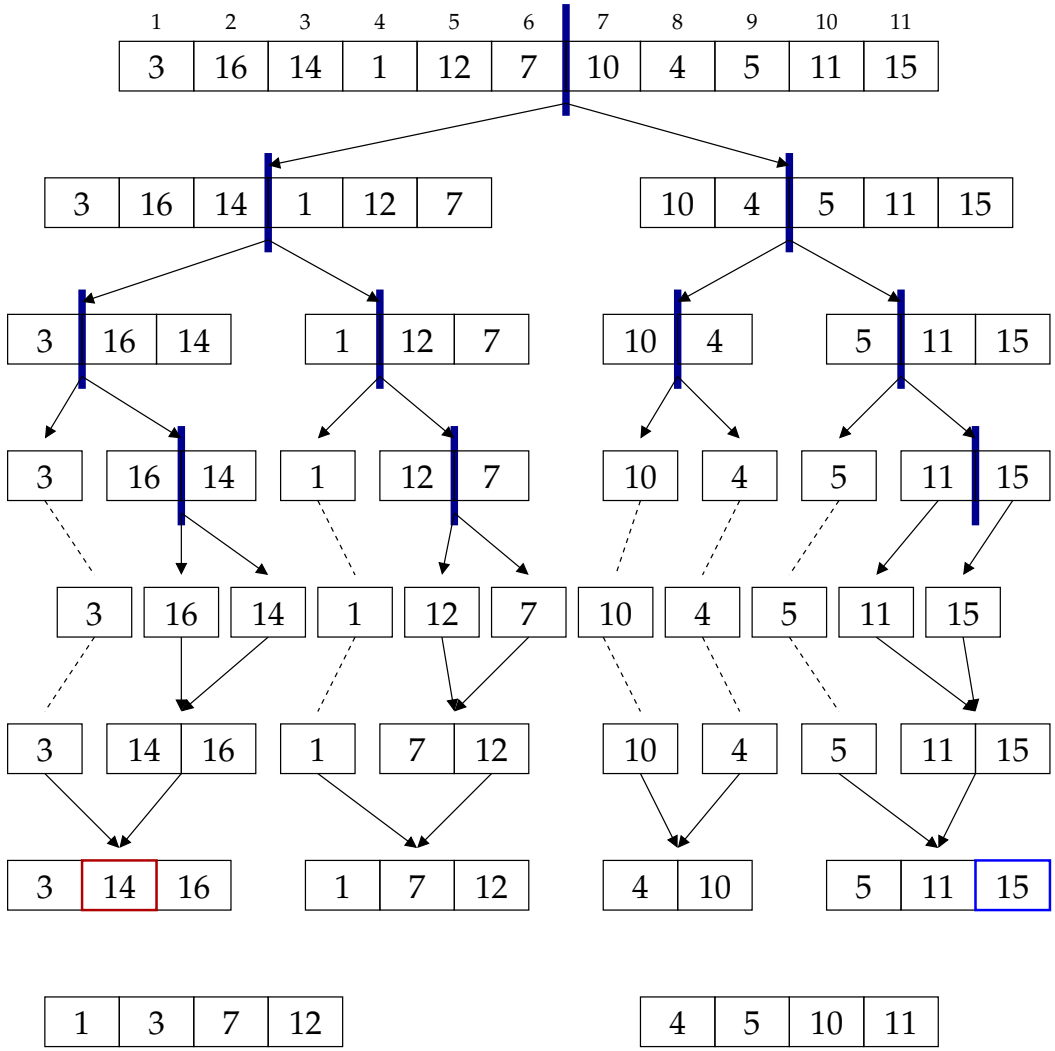
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16
1 7 12
10 4
5 11 15
3 14 16
1 7 12
4 10
5 11 15
1 3
4 5

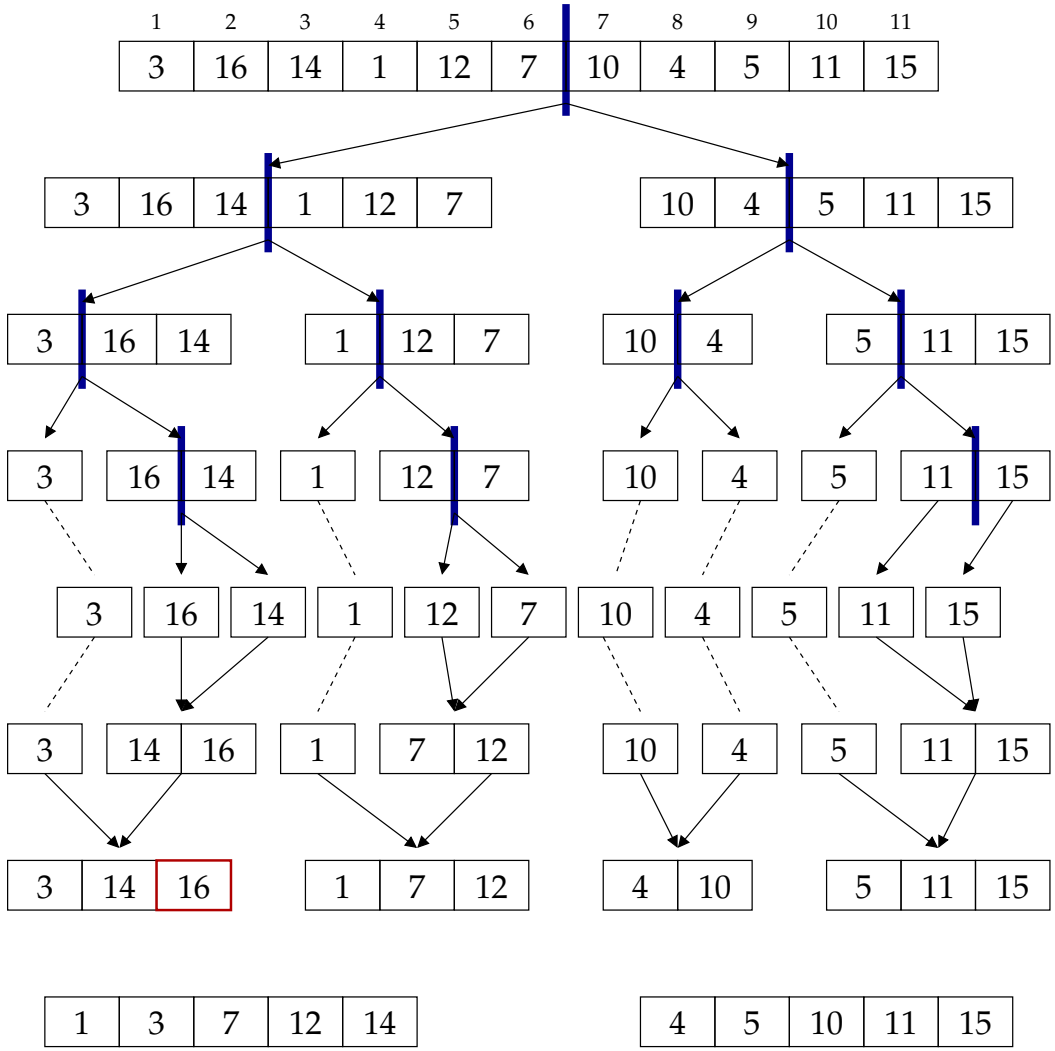
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16
1 7 12
10 4
5 11 15
3 14 16
1 7 12
4 10
5 11 15
1 3 7
4 5 10

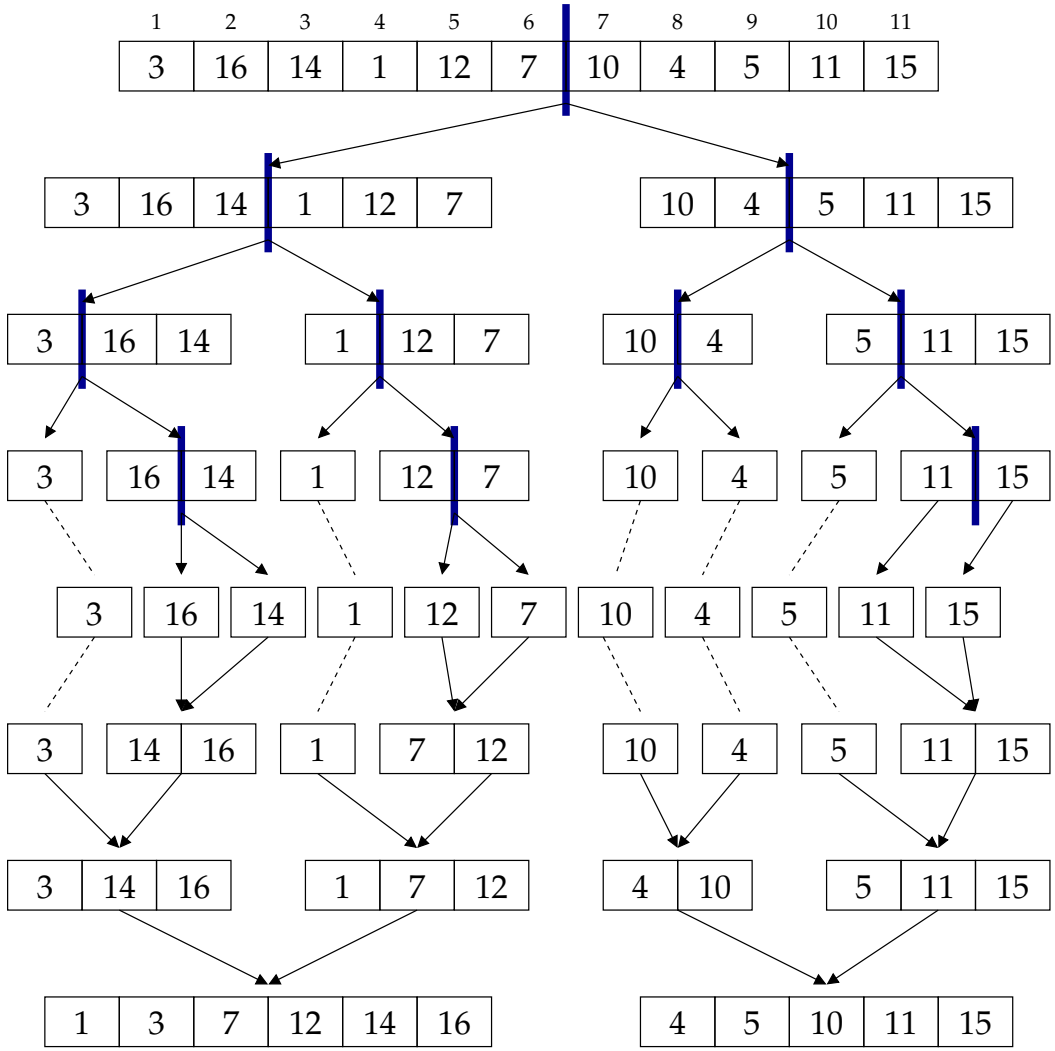
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14
16
1
7
12
4 10
5
11
15
1 3 7 12
4 5 10 11

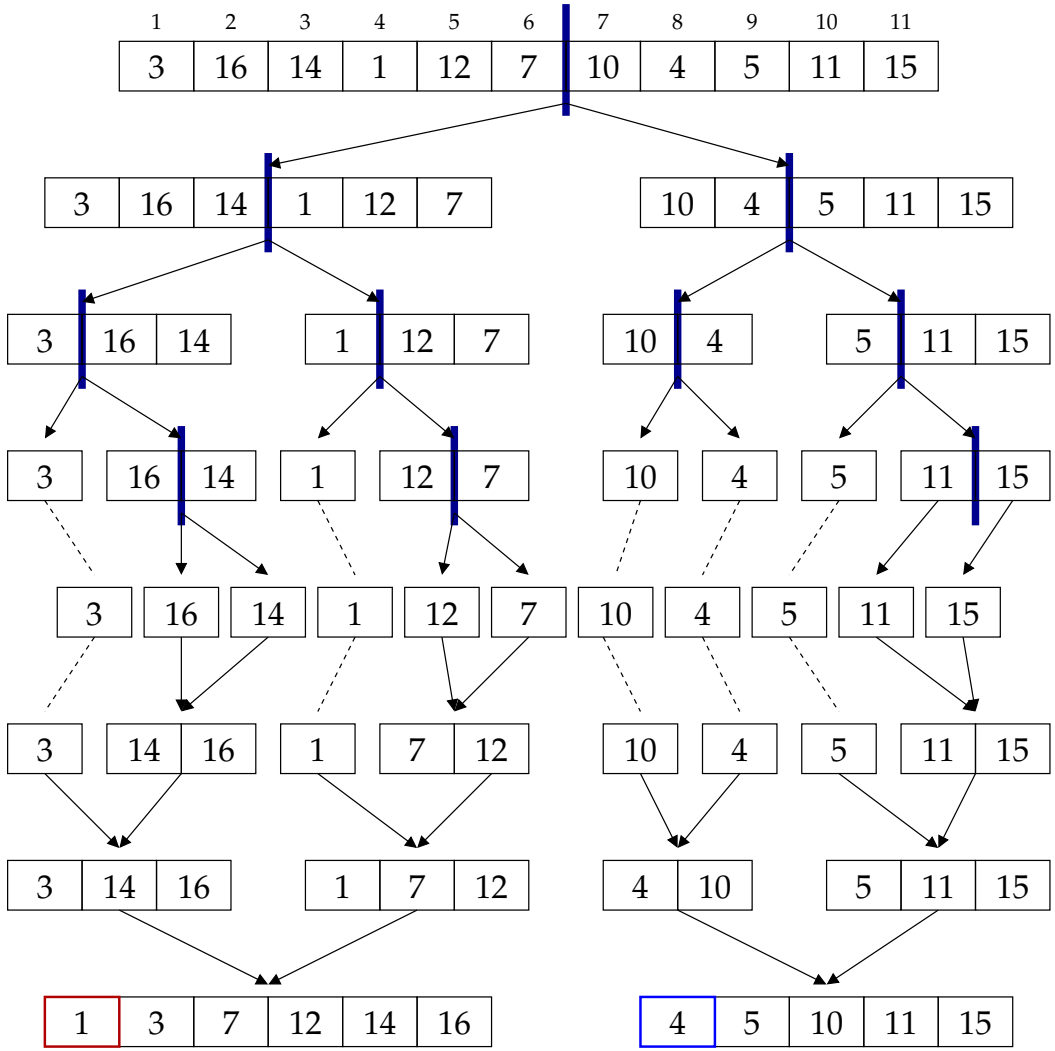
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14
16
1
7
12
4 10
5 11 15
1 3 7 12 14
4 5 10 11 15

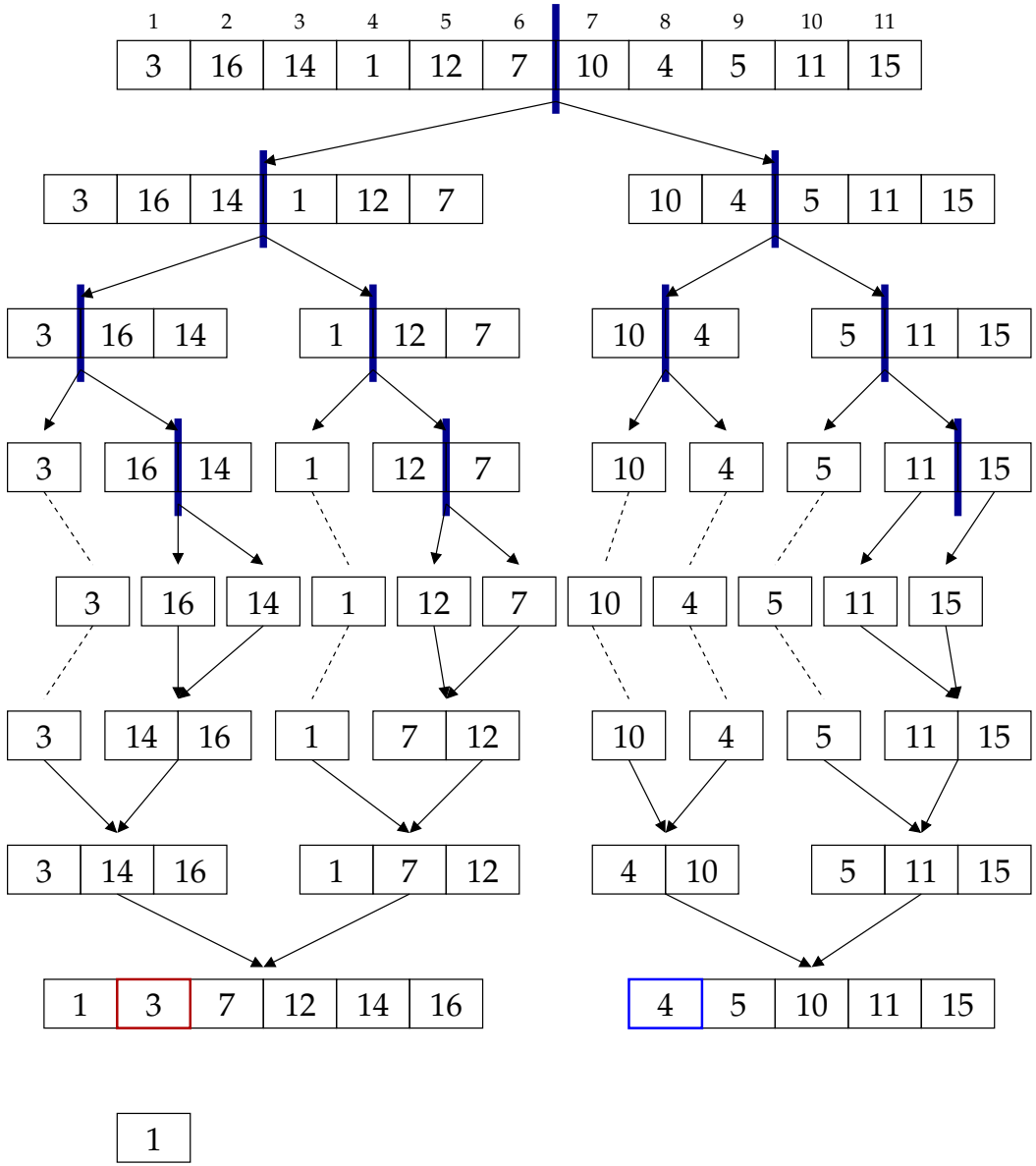
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1
7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15

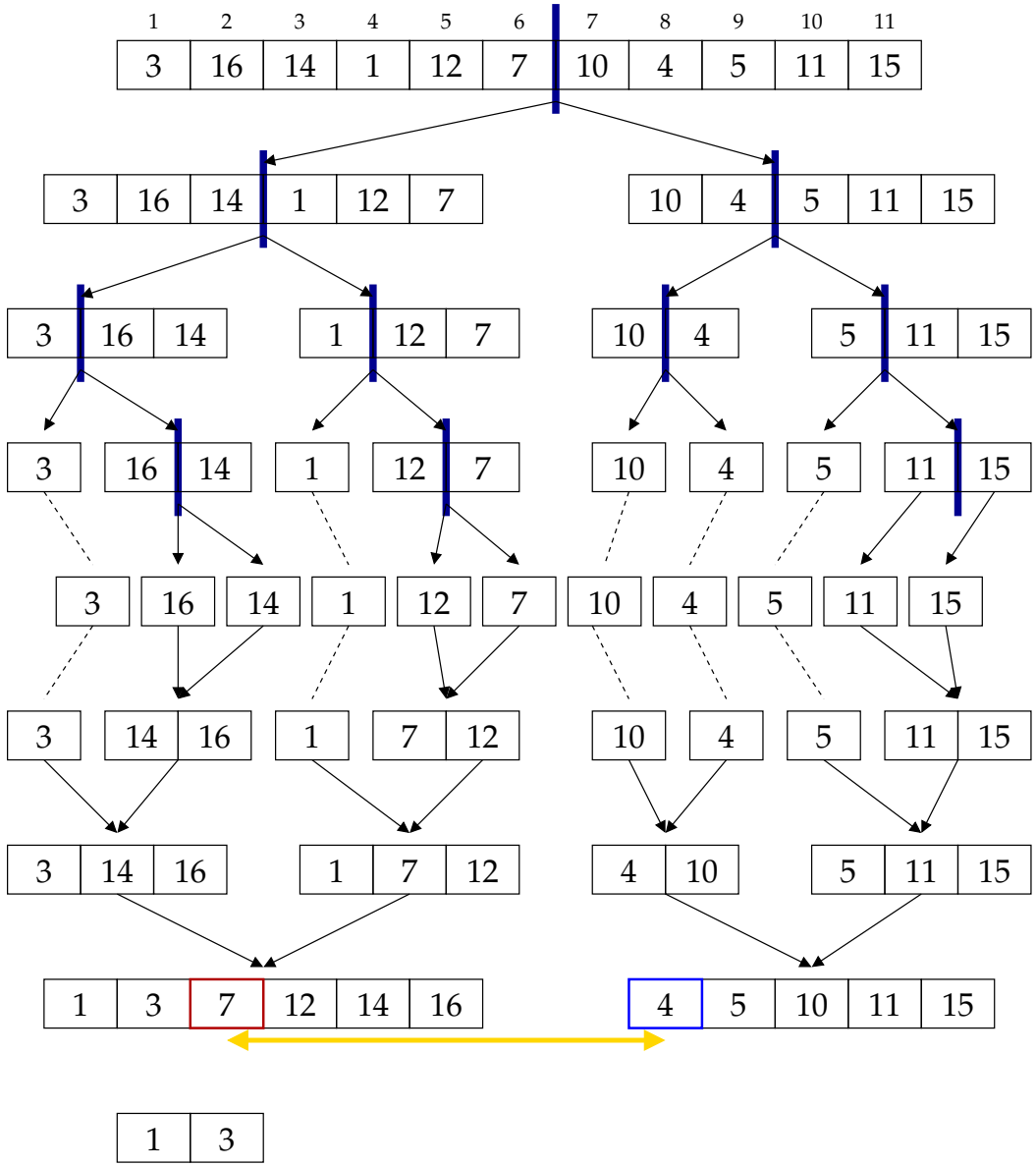
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1
7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15

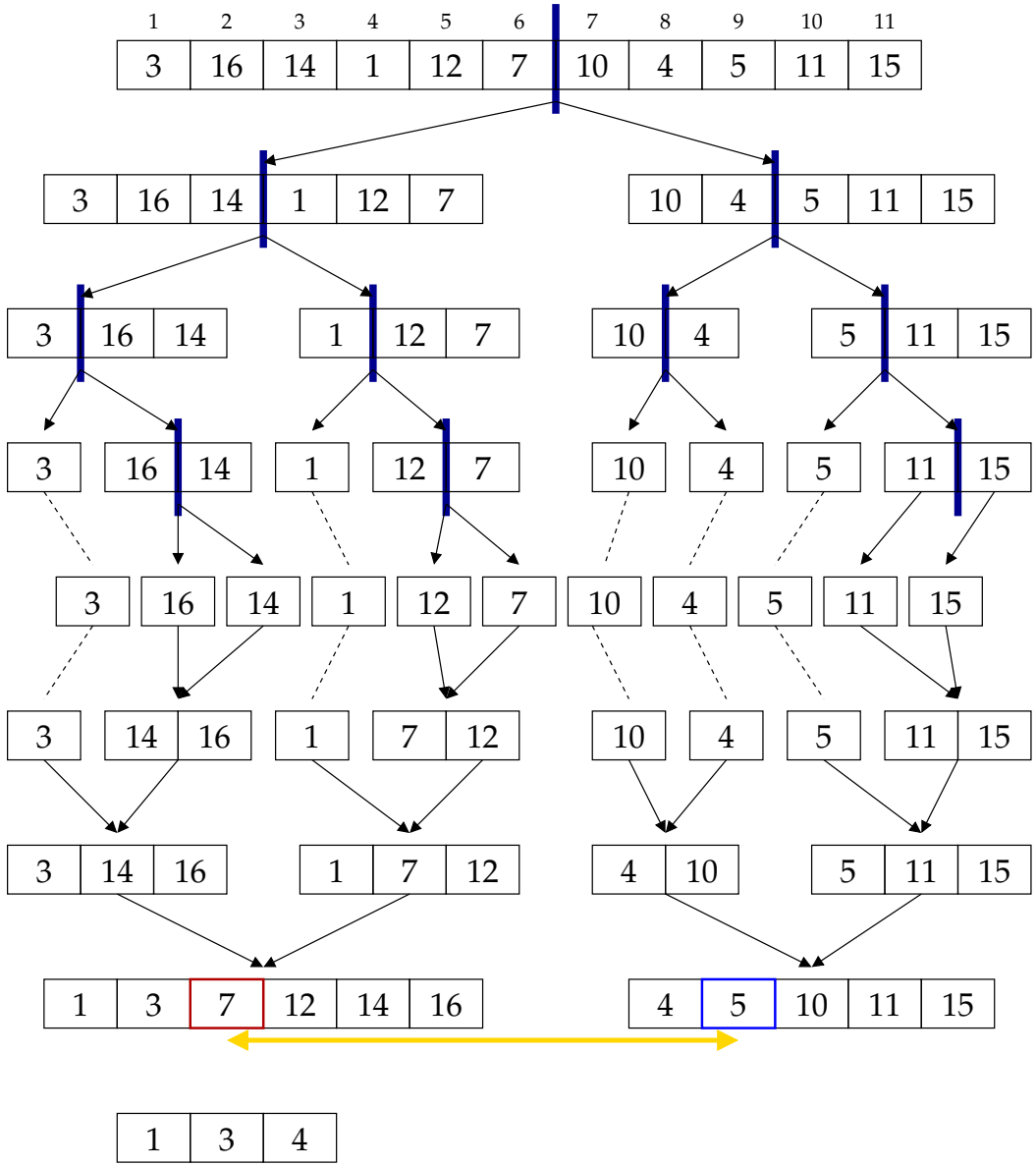
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1
7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1

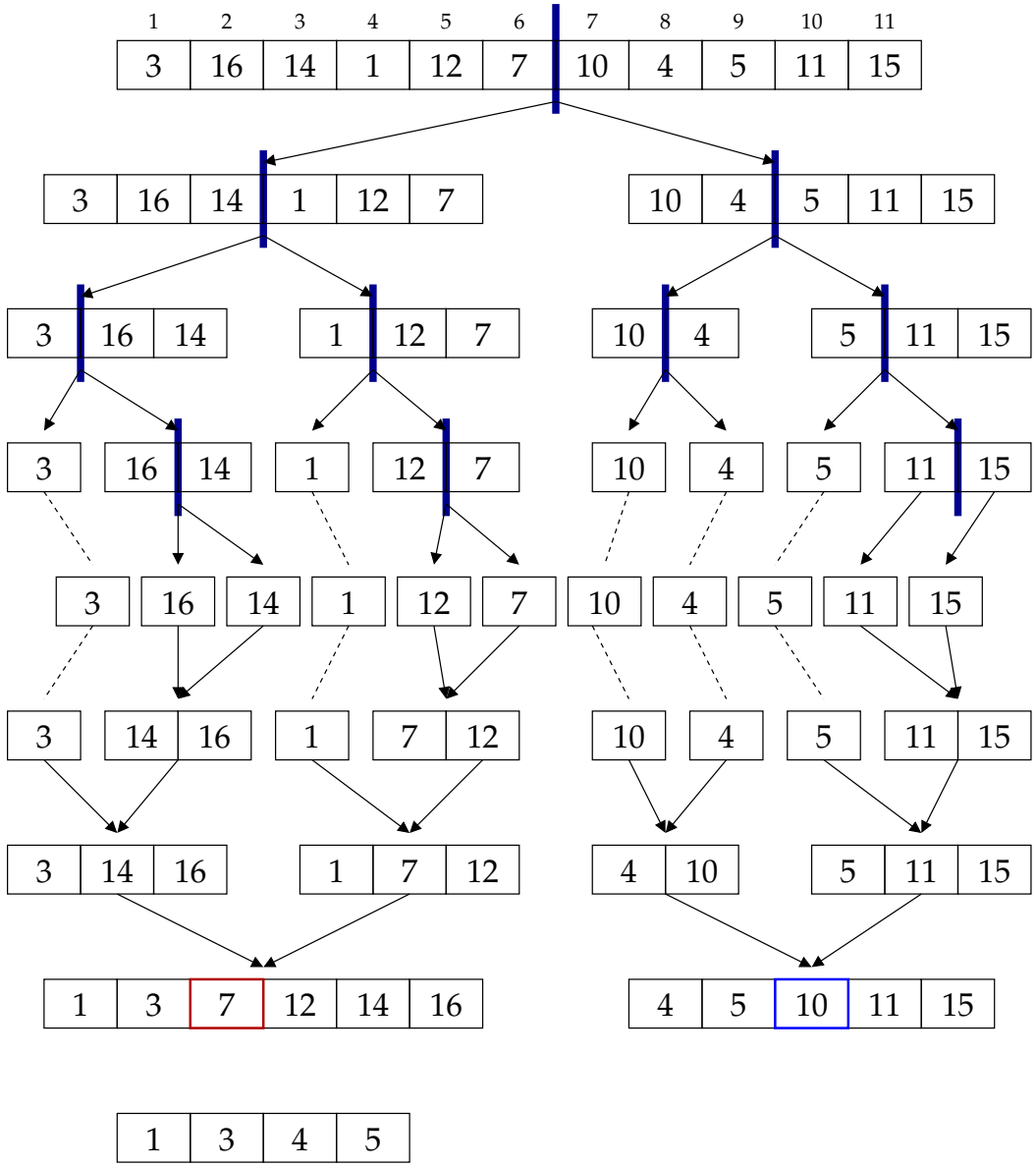
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16
1 7 12
10 4 5 11 15
3 14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3

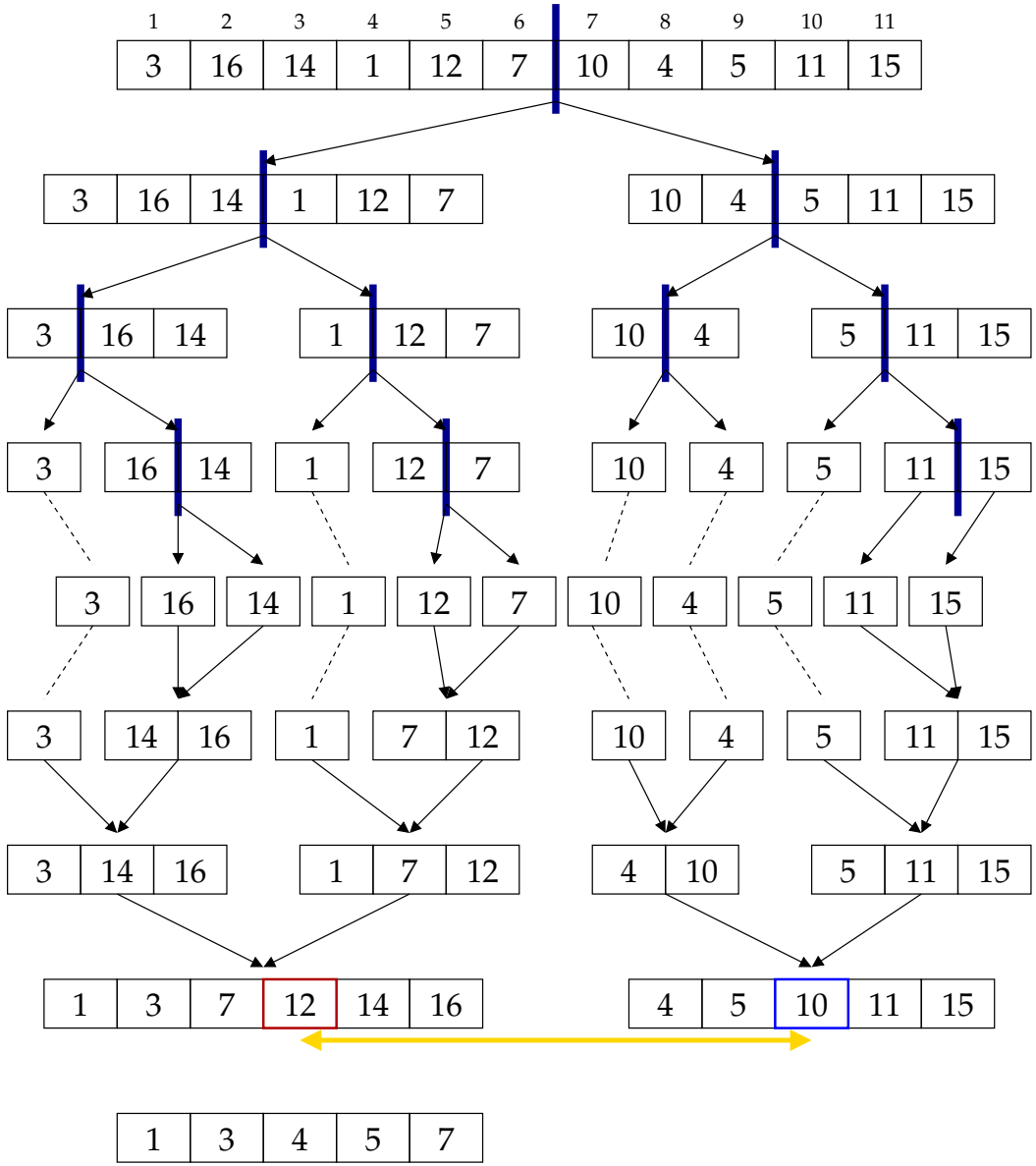
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16 1 7 12 10 4 5 11 15
3 14 16 1 7 12 4 10 5 11 15
1 3 7 12 14 16 4 5 10 11 15
1 3 4

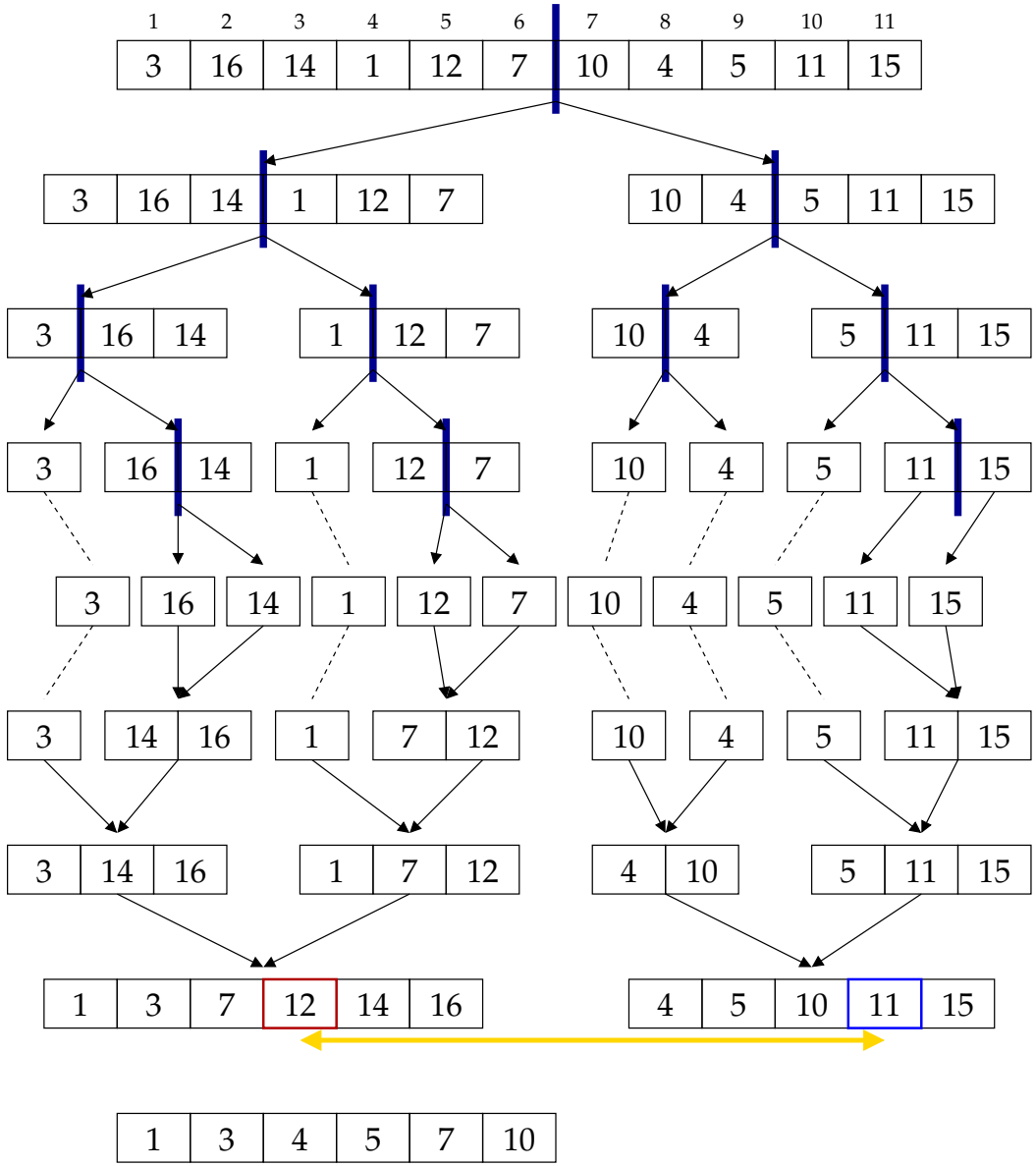
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5

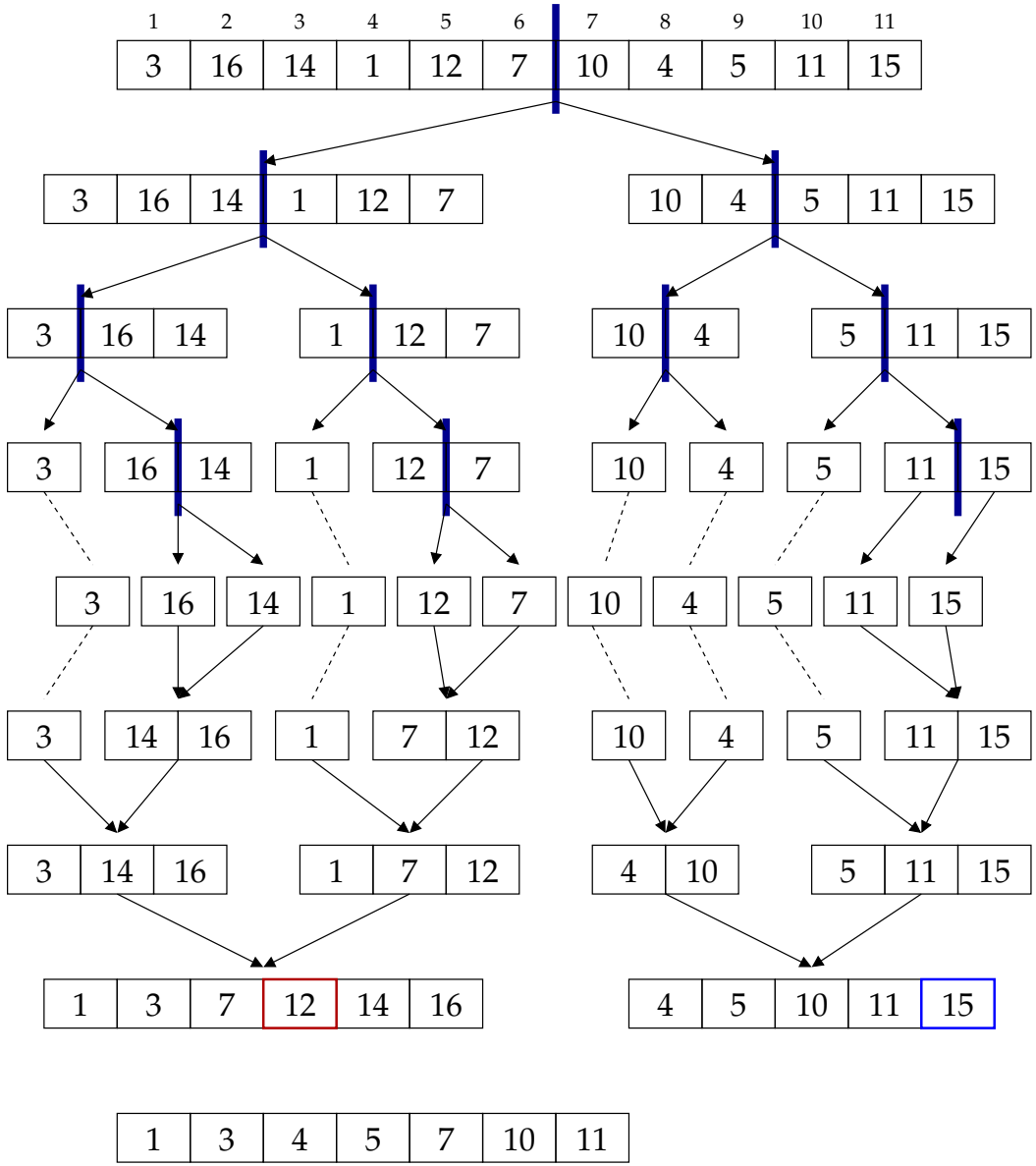
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1
7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7

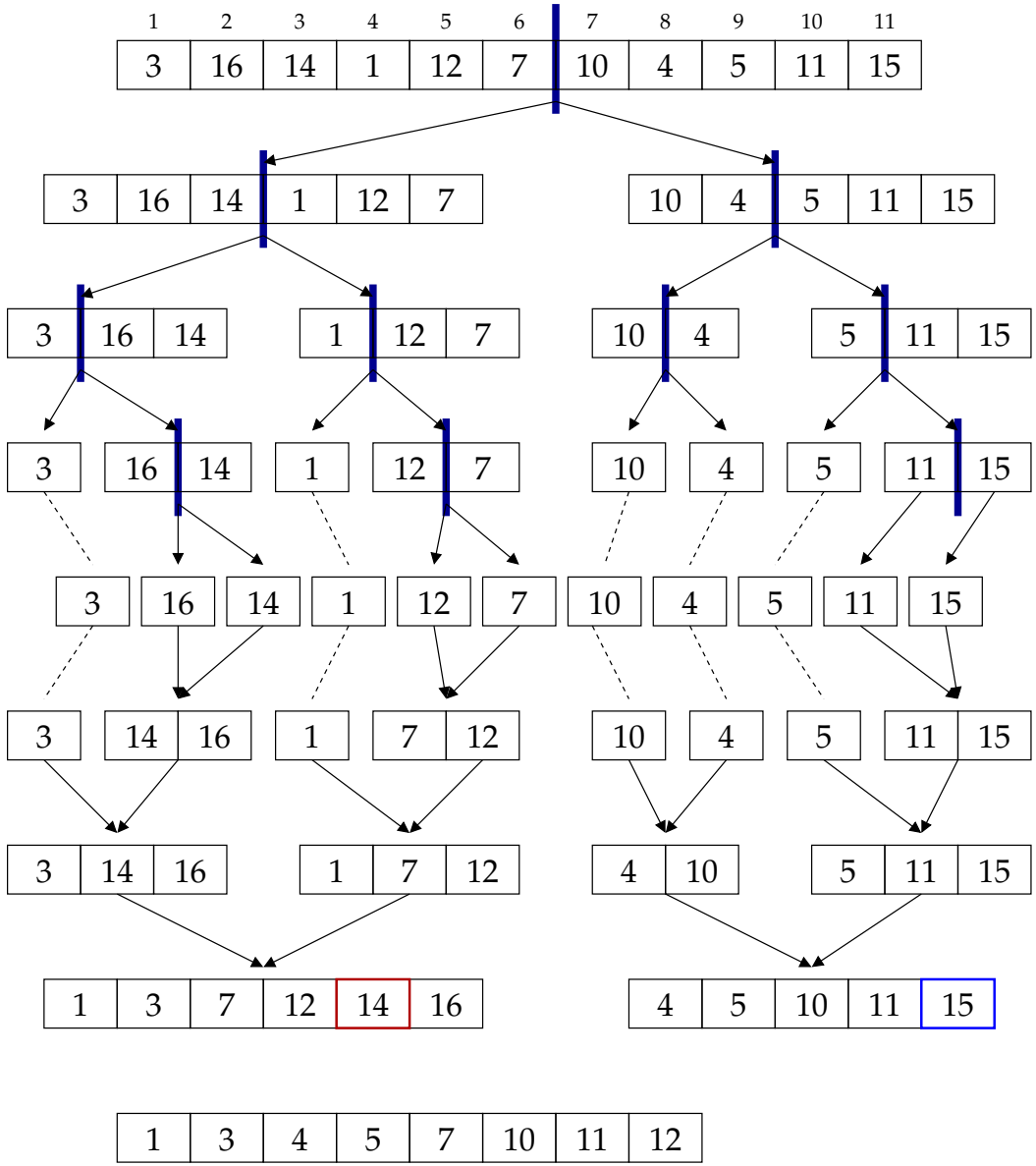
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1
7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10

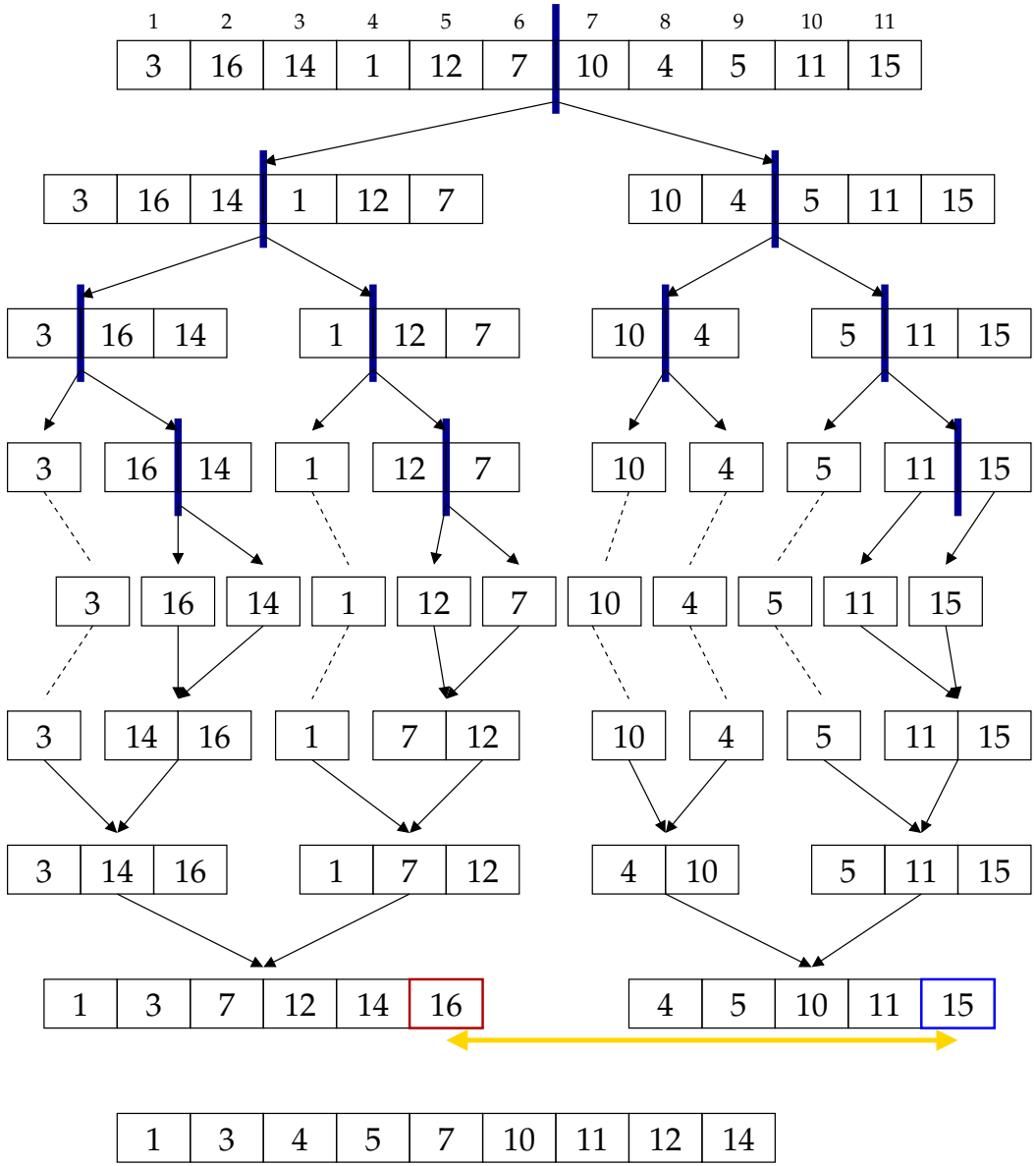
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11

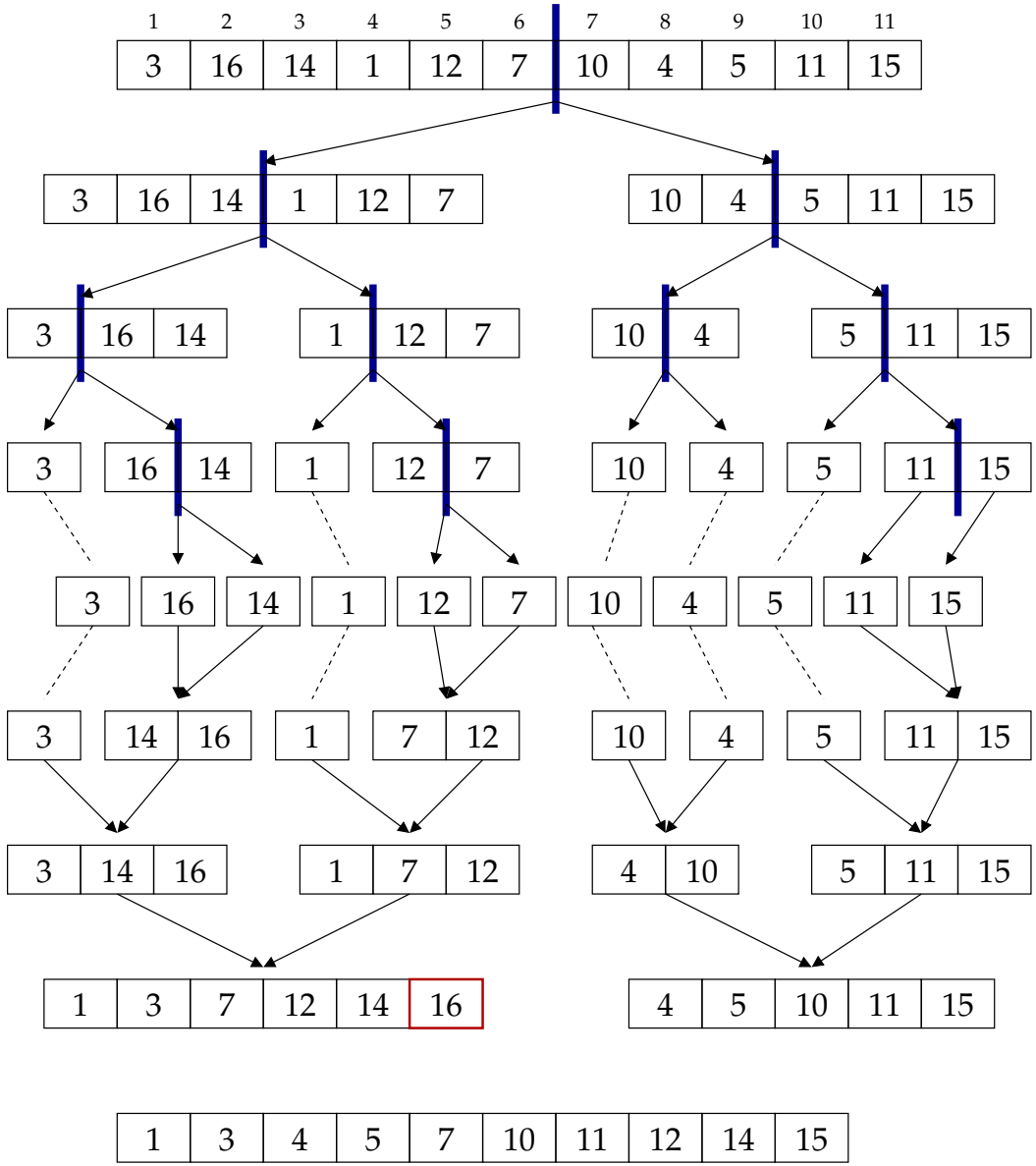
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11 12

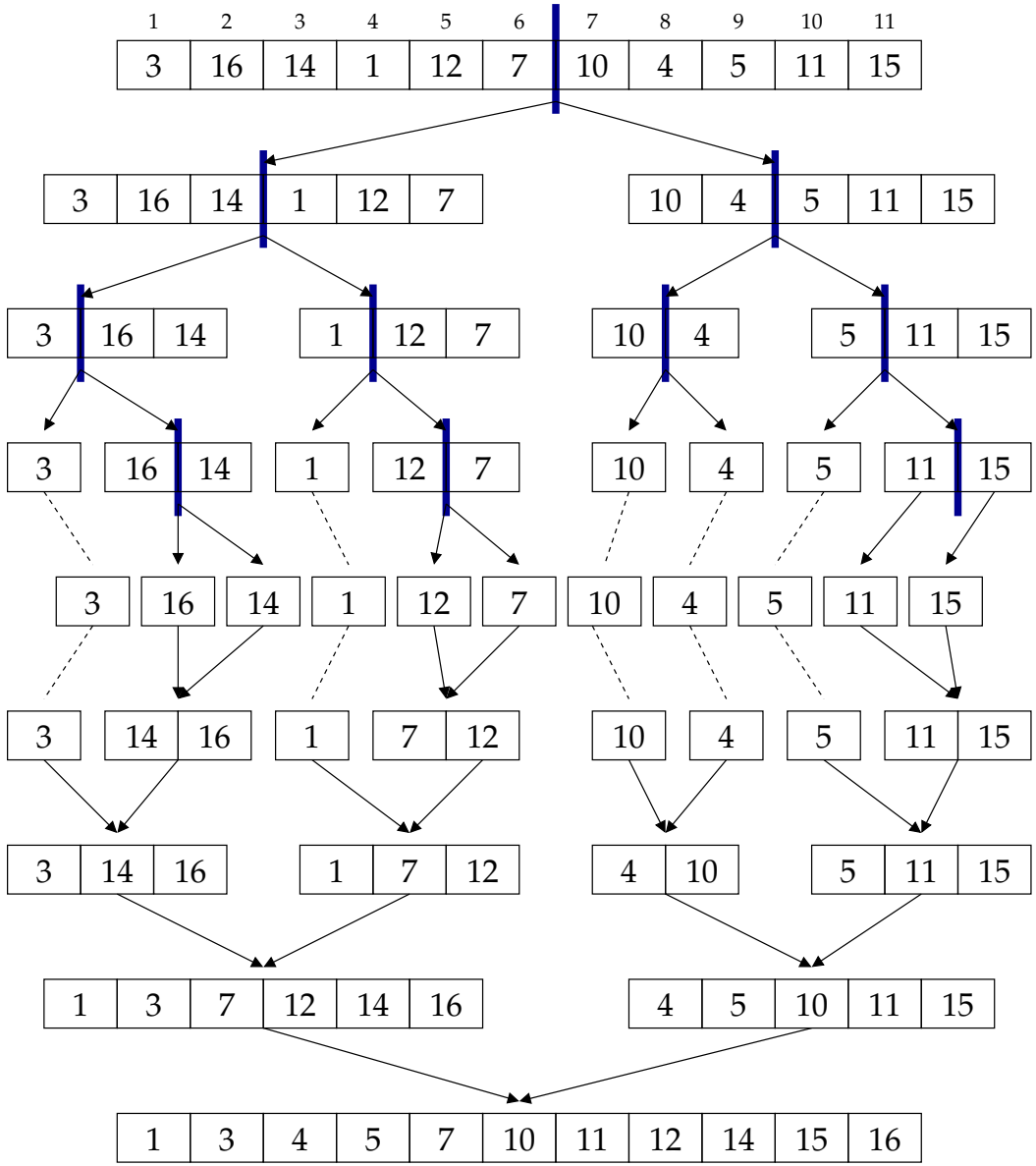
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16
1 7 12
10 4 5 11 15
3 14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11 12 14

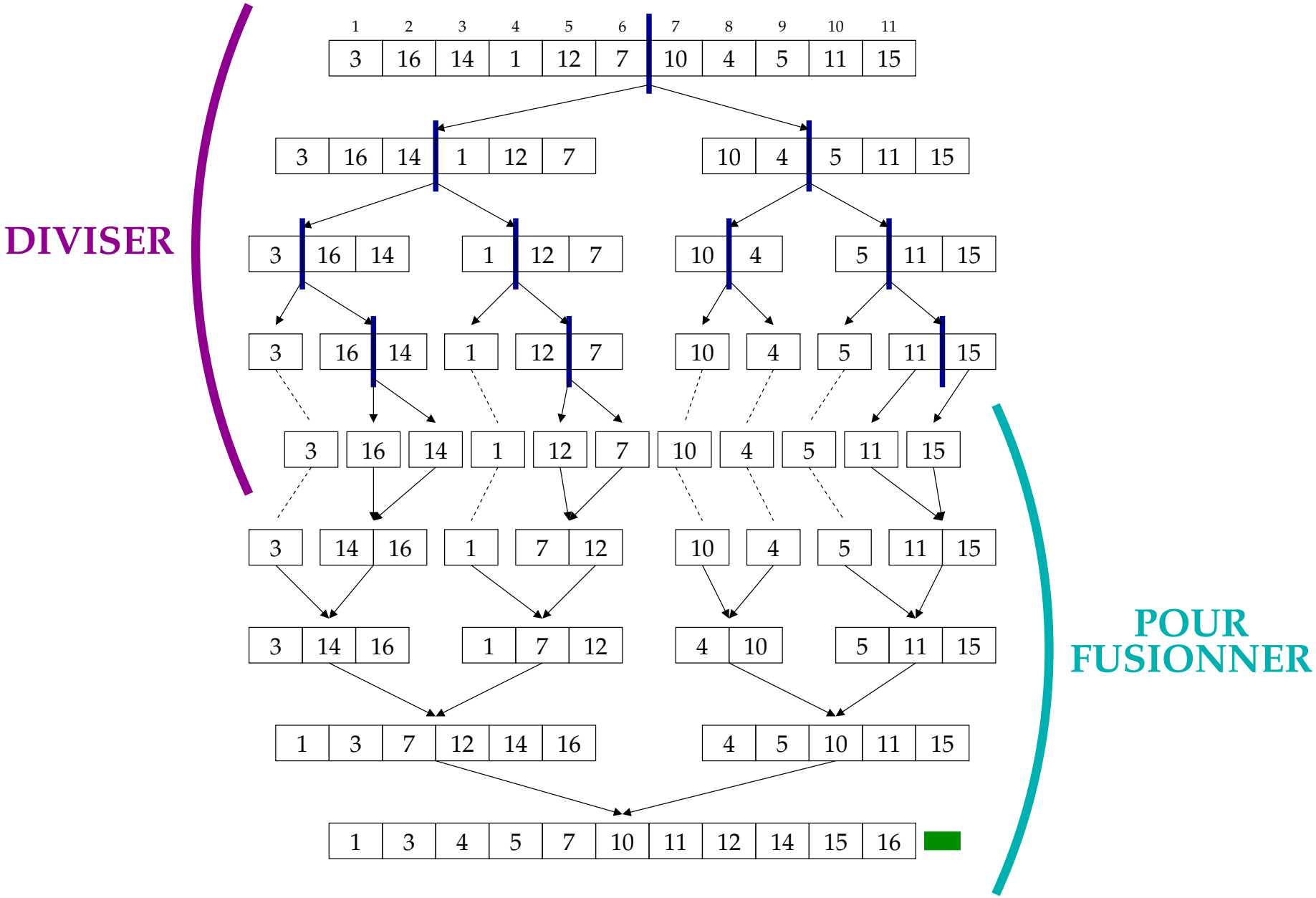
Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11 12 14 15

Tri fusion


1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3
14 16
1
7 12
10
4
5
11 15
3
14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11 12 14 15 16

Tri fusion


DIVISER
POUR FUSIONNER
1 2 3 4 5 6 7 8 9 10 11
3 16 14 1 12 7 10 4 5 11 15
3 16 14 1 12 7
10 4 5 11 15
3 16 14
1 12 7
10 4
5 11 15
3
16 14
1
12 7
10
4
5
11 15
3
16
14
1
12
7
10
4
5
11
15
3 14 16
1 7 12
10 4 5 11 15
3 14 16
1 7 12
4 10
5 11 15
1 3 7 12 14 16
4 5 10 11 15
1 3 4 5 7 10 11 12 14 15 16

Tri fusion

Algorithme

Fonction tri_fusion(tab$[1 \dots n]$: **tab entier**) : **tab entier**
Début
    Si $(n \leqslant 1)$ **alors** **retour** tab ;
    Sinon
        retour fusion(tri_fusion(tab$[1 \dots \lfloor \frac{n+1}{2} \rfloor]$), tri_fusion(tab$[\lceil \frac{n+1}{2} \rceil \dots n]$)) ;
    Fin si
Fin

Fonction fusion($t_1[1 \dots n_1]$, $t_2[1 \dots n_2]$: **tab entier**) : **tab entier**
Début
    Si $(n_1 = 0)$ **alors** **retour** t_2 ;
    Sinon si $(n_2 = 0)$ **alors** **retour** t_1 ;
    Sinon si $(t_1[1] \leqslant t_2[1])$ **alors** **retour** $t_1[1]$ :: fusion($t_1[2 \dots n_1]$, t_2) ;
    Sinon **retour** $t_2[1]$ :: fusion(t_1, $t_2[2 \dots n_2]$) ;
    Fin si
Fin

Tri fusion

Complexité

Qu'en dites-vous ?

Tri fusion

Complexité

Qu'en dites-vous ?

$$O(n \cdot \log_2(n))$$

Tri fusion

Application

Vidéo