Module EA4 – Éléments d'Algorithmique

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique, Math-Info et EIDD
Année universitaire 2013-2014

Rappel des épisodes précédents

Tri par insertion

- $\Theta(n^2)$ comparaisons au pire, $\Theta(n)$ au mieux,
- $\Theta(n^2)$ comparaisons en moyenne,
- trie en place

Rappel des épisodes précédents

Tri par insertion

- $\Theta(n^2)$ comparaisons au pire, $\Theta(n)$ au mieux,
- $\Theta(n^2)$ comparaisons en moyenne,
- trie en place

Tri par fusion

- $\Theta(n \log n)$ comparaisons dans tous les cas,
- la constante cachée dans le Θ est importante,
- ne trie pas en place : complexité en espace $\in \Theta(n)$

Peut-on faire mieux ?

Coût minimal d'un tri par comparaison

Un tri par comparaison est un algorithme de tri qui n'utilise que des comparaisons entre les éléments d'entrée.

Contre-exemple : le tri par dénombrement n'est pas un tri par comparaison

Coût minimal d'un tri par comparaison

Un tri par comparaison est un algorithme de tri qui n'utilise que des comparaisons entre les éléments d'entrée.

Contre-exemple : le tri par dénombrement n'est pas un tri par comparaison

Théorème

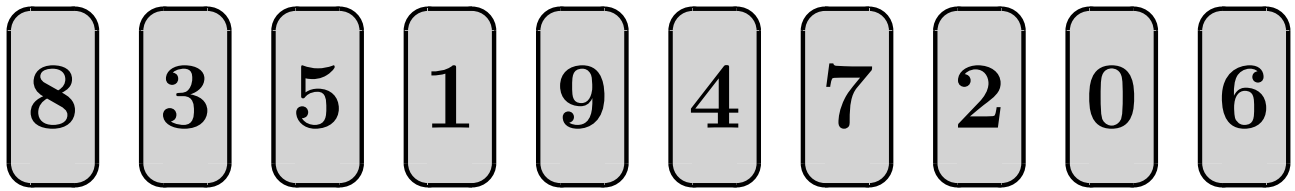
Tout tri par comparaison effectue, en moyenne sur les listes de longueur n, $\Omega(n \log n)$ comparaisons.

Corollaire

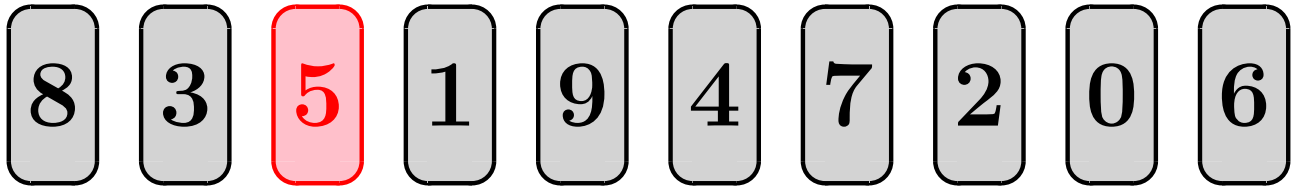
Le tri fusion est un tri par comparaison asymptotiquement optimal.

Tri rapide (*Quicksort*)

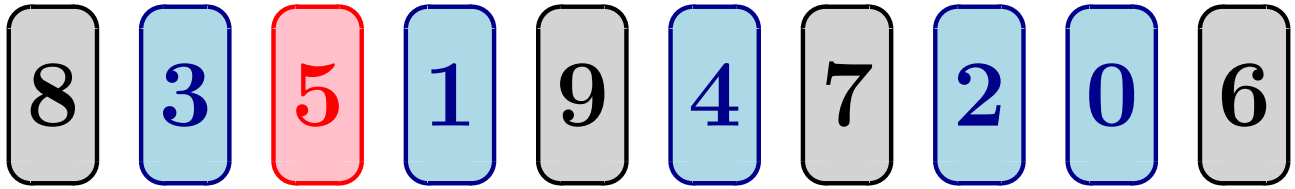
Exemple :



Exemple :

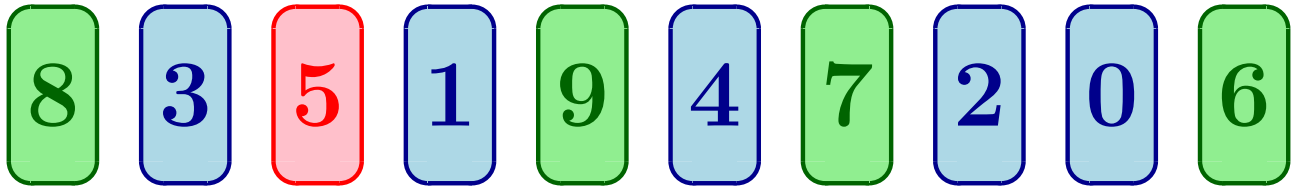


Exemple :



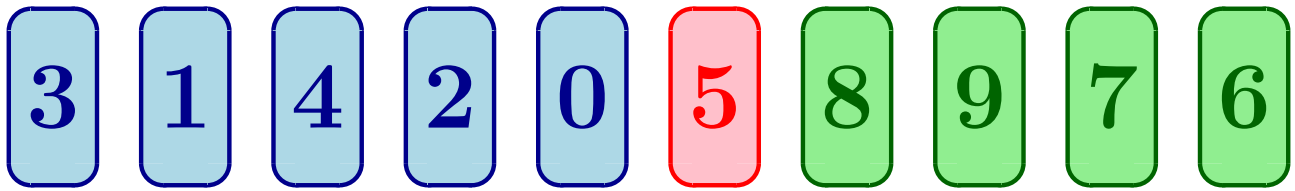
Tri rapide (*Quicksort*)

Exemple :



Tri rapide (*Quicksort*)

Exemple :



Tri rapide (*Quicksort*), version 1

```
def partition(T) :  # les éléments sont supposés distincts
  pivot = T[0]
  gauche = [ elt for elt in T if elt < pivot ]
  droite = [ elt for elt in T if elt > pivot ]
  return pivot, gauche, droite
```

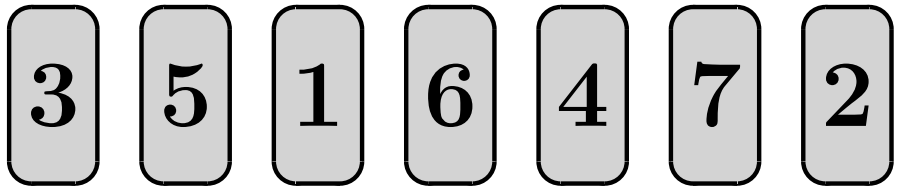
Tri rapide (*Quicksort*), version 1

```
def partition(T) :  # les éléments sont supposés distincts
  pivot = T[0]
  gauche = [ elt for elt in T if elt < pivot ]
  droite = [ elt for elt in T if elt > pivot ]
  return pivot, gauche, droite

def tri_rapide(T) :
  if len(T) < 2 : return T
  pivot, gauche, droite = partition(T)
  return tri_rapide(gauche) + [pivot] + tri_rapide(droite)
```

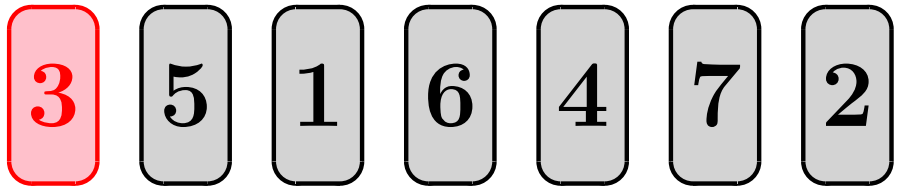
Tri rapide (*Quicksort*), version 1

Exemple :



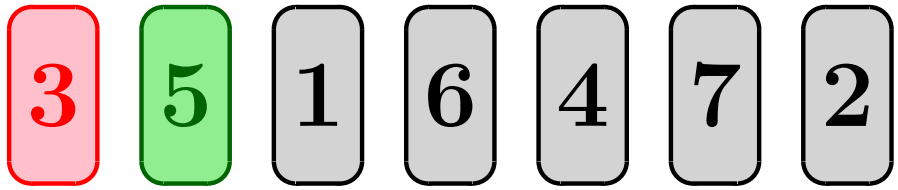
Tri rapide (*Quicksort*), version 1

Exemple :



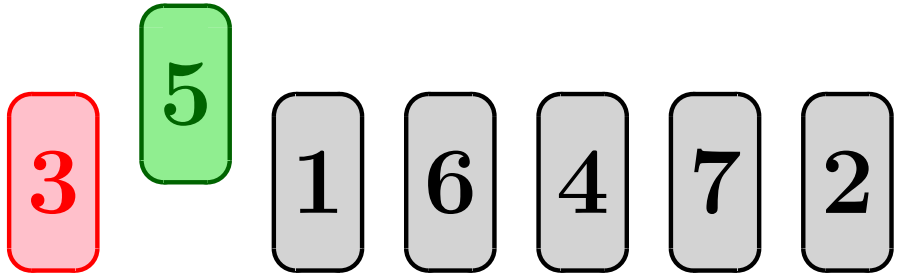
Tri rapide (*Quicksort*), version 1

Exemple :



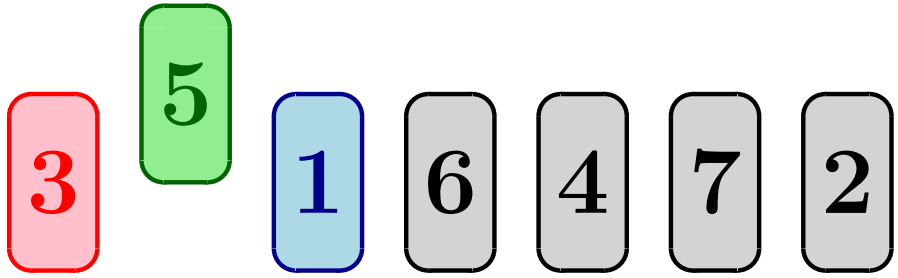
Tri rapide (*Quicksort*), version 1

Exemple :



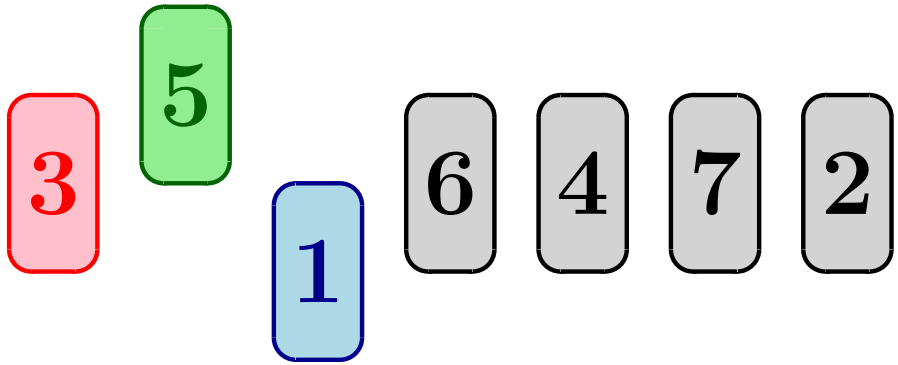
Tri rapide (*Quicksort*), version 1

Exemple :



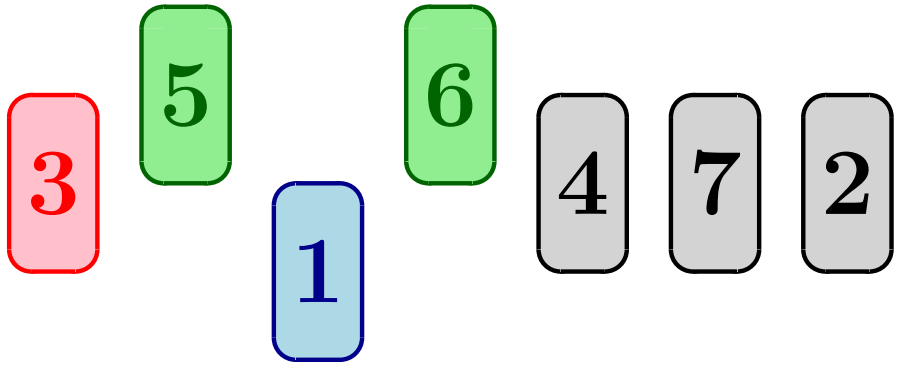
Tri rapide (*Quicksort*), version 1

Exemple :



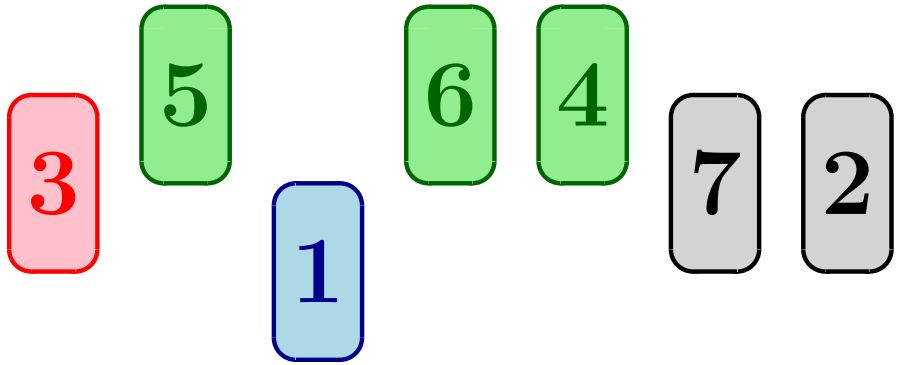
Tri rapide (*Quicksort*), version 1

Exemple :



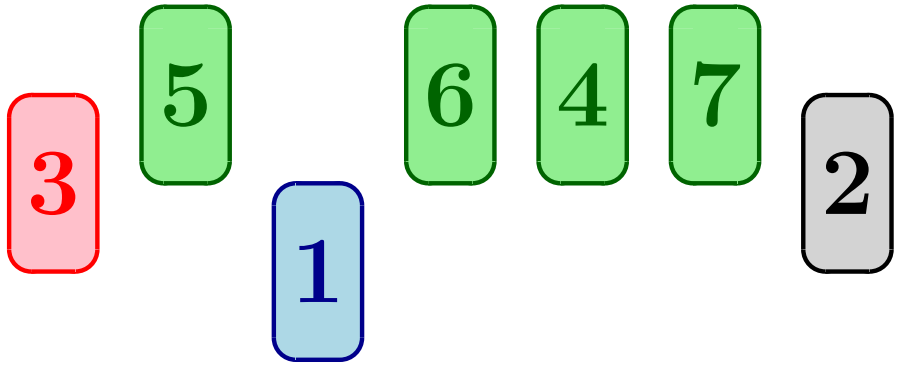
Tri rapide (*Quicksort*), version 1

Exemple :



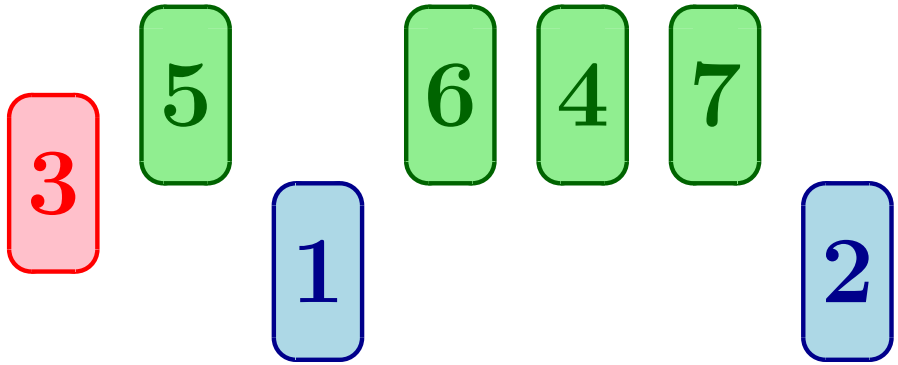
Tri rapide (*Quicksort*), version 1

Exemple :



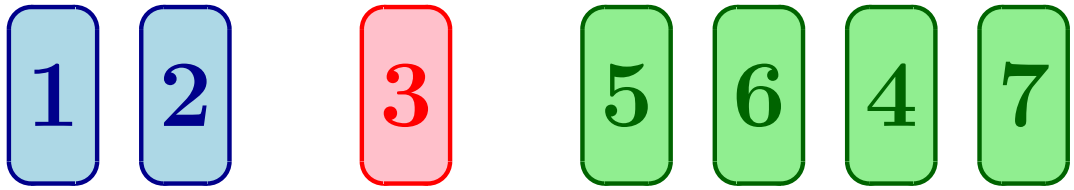
Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

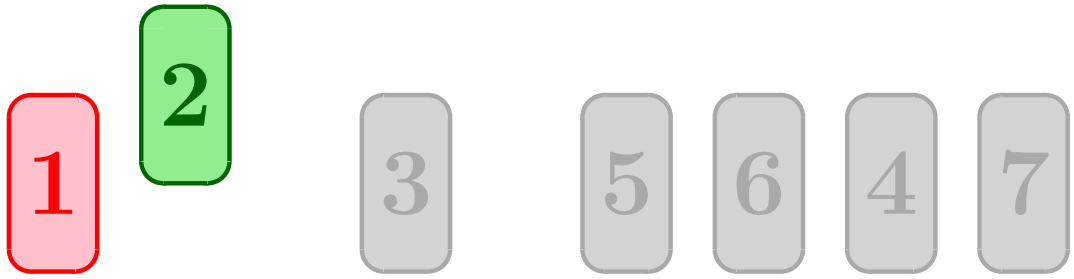
Exemple :

Tri rapide (*Quicksort*), version 1

Exemple :

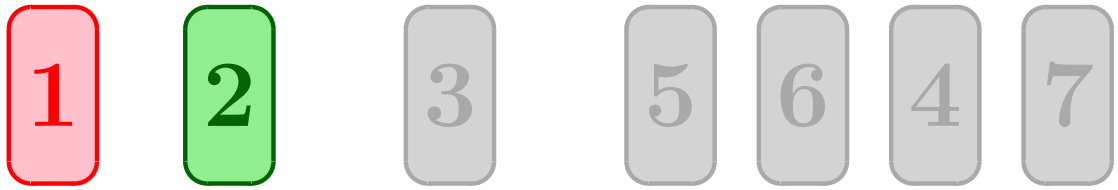
Tri rapide (*Quicksort*), version 1

Exemple :



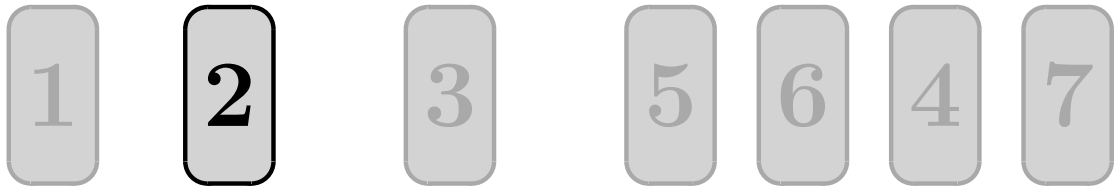
Tri rapide (*Quicksort*), version 1

Exemple :



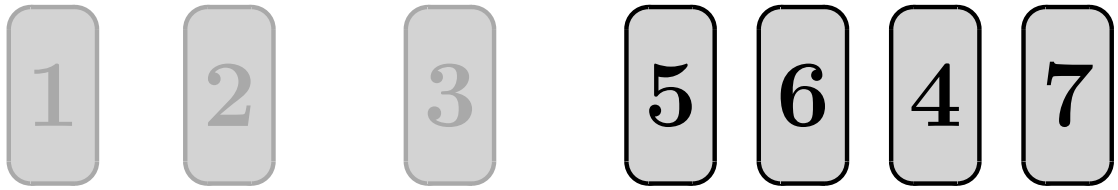
Tri rapide (*Quicksort*), version 1

Exemple :



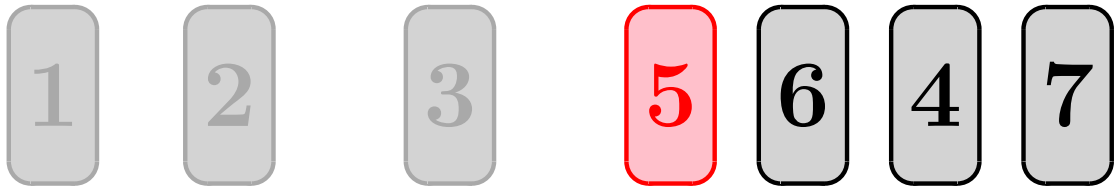
Tri rapide (*Quicksort*), version 1

Exemple :



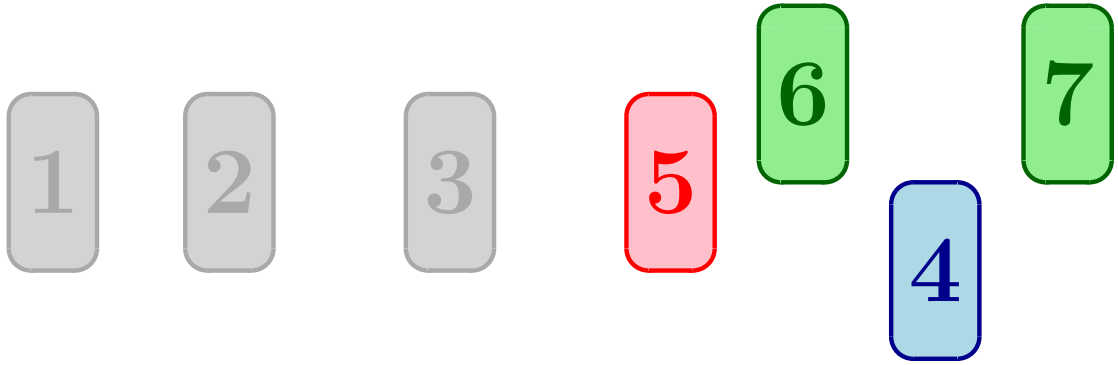
Tri rapide (*Quicksort*), version 1

Exemple :



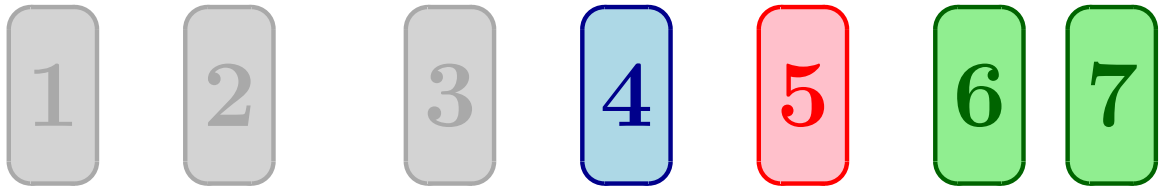
Tri rapide (*Quicksort*), version 1

Exemple :



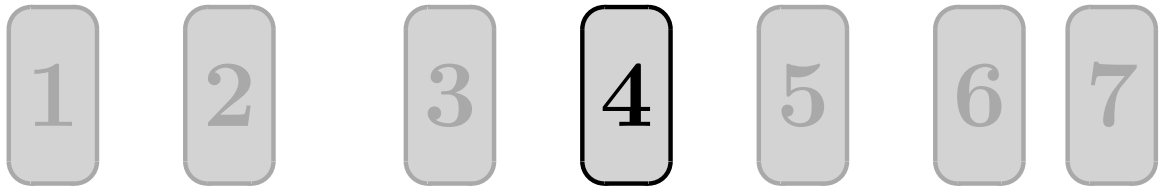
Tri rapide (*Quicksort*), version 1

Exemple :



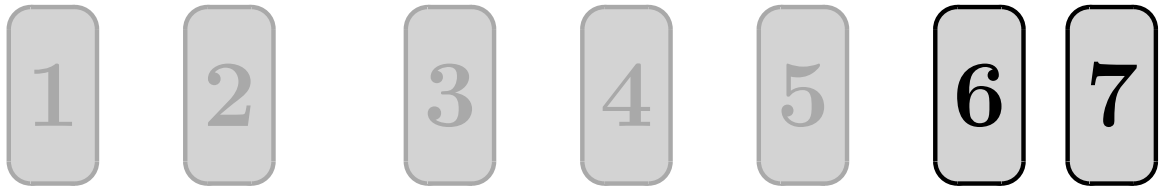
Tri rapide (*Quicksort*), version 1

Exemple :



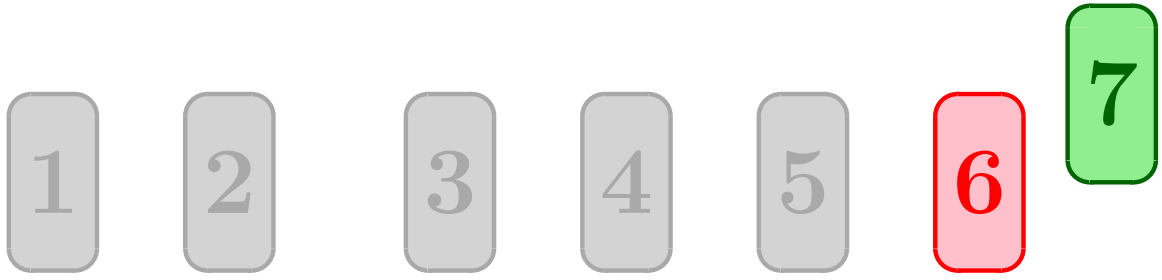
Tri rapide (*Quicksort*), version 1

Exemple :



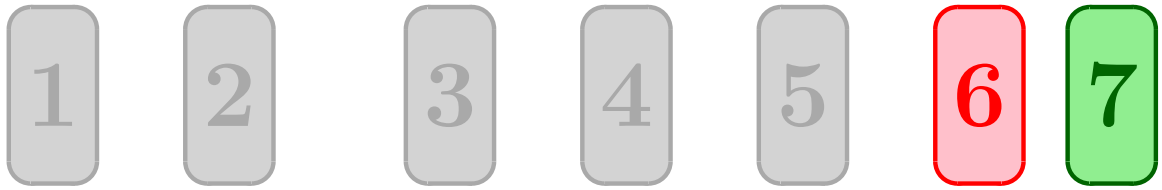
Tri rapide (*Quicksort*), version 1

Exemple :



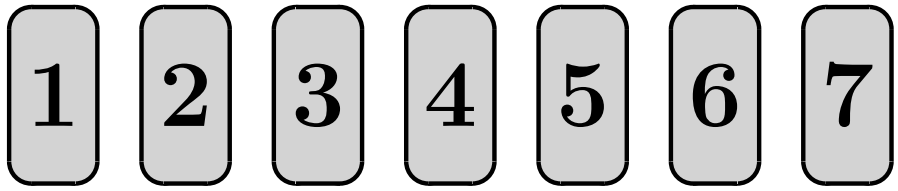
Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

```
def partition(T) :  # les éléments sont supposés distincts
  pivot = T[0]
  gauche = [ elt for elt in T if elt < pivot ]
  droite = [ elt for elt in T if elt > pivot ]
  return pivot, gauche, droite
```

Tri rapide (*Quicksort*), version 1

```
def partition(T) :  # les éléments sont supposés distincts
  pivot = T[0]
  gauche = [ elt for elt in T if elt < pivot ]
  droite = [ elt for elt in T if elt > pivot ]
  return pivot, gauche, droite
```

Complexité de `partition` : $\Theta(n)$ comparaisons

Tri rapide (*Quicksort*), version 1

Complexité de `partition` : $\Theta(n)$ comparaisons

Tri rapide (*Quicksort*), version 1

Complexité de `partition` : $\Theta(n)$ comparaisons

```
def tri_rapide(T) :
  if len(T) < 2 : return T
  pivot, gauche, droite = partition(T)
  return tri_rapide(gauche) + [pivot] + tri_rapide(droite)
```

Tri rapide (*Quicksort*), version 1

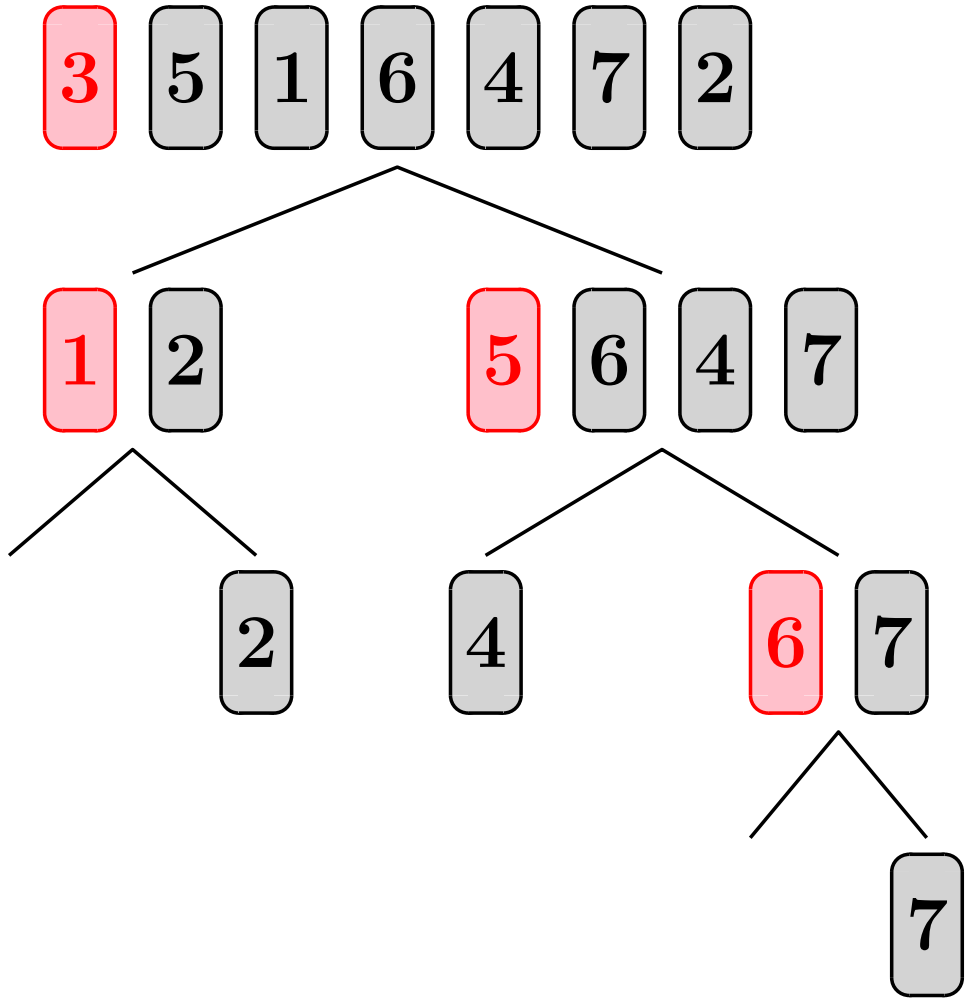
Complexité de `partition` : $\Theta(n)$ comparaisons

```
def tri_rapide(T) :
  if len(T) < 2 : return T
  pivot, gauche, droite = partition(T)
  return tri_rapide(gauche) + [pivot] + tri_rapide(droite)
```

Complexité de `tri_rapide` au pire : $\Theta(n^2)$ comparaisons

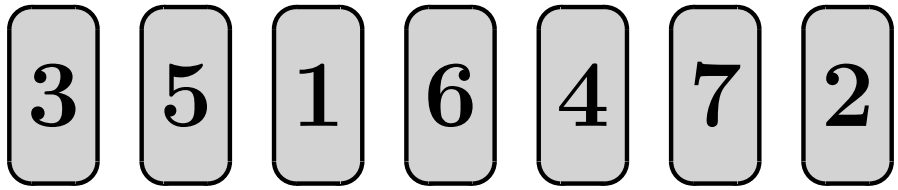
Tri rapide (*Quicksort*), version 1

Exemple :



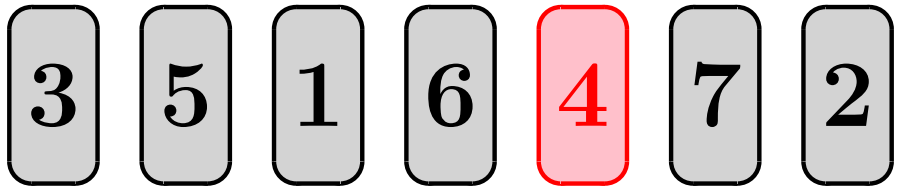
Tri rapide (*Quicksort*), version 1

Exemple :



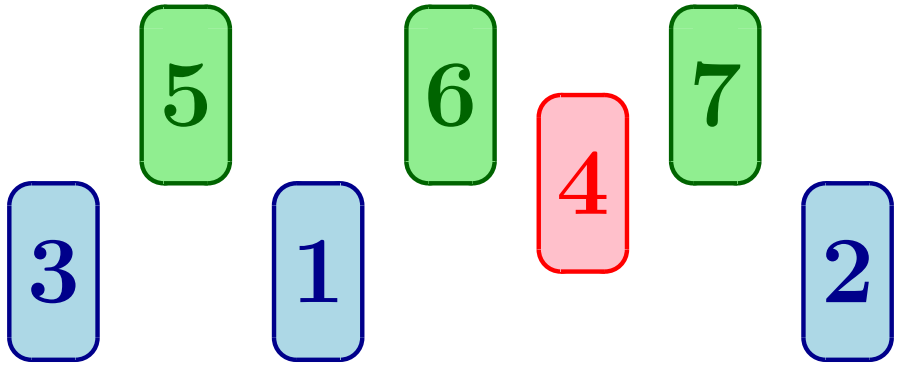
Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Exemple :

Tri rapide (*Quicksort*), version 1

Exemple :

Tri rapide (*Quicksort*), version 1

Exemple :

Tri rapide (*Quicksort*), version 1

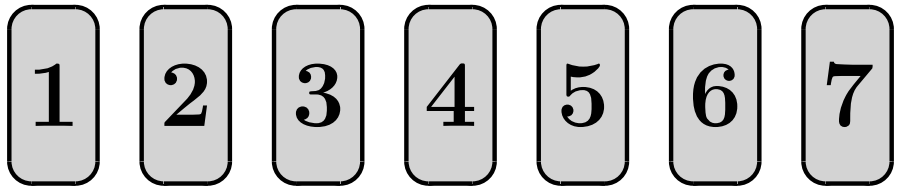
Exemple :

Tri rapide (*Quicksort*), version 1

Exemple :

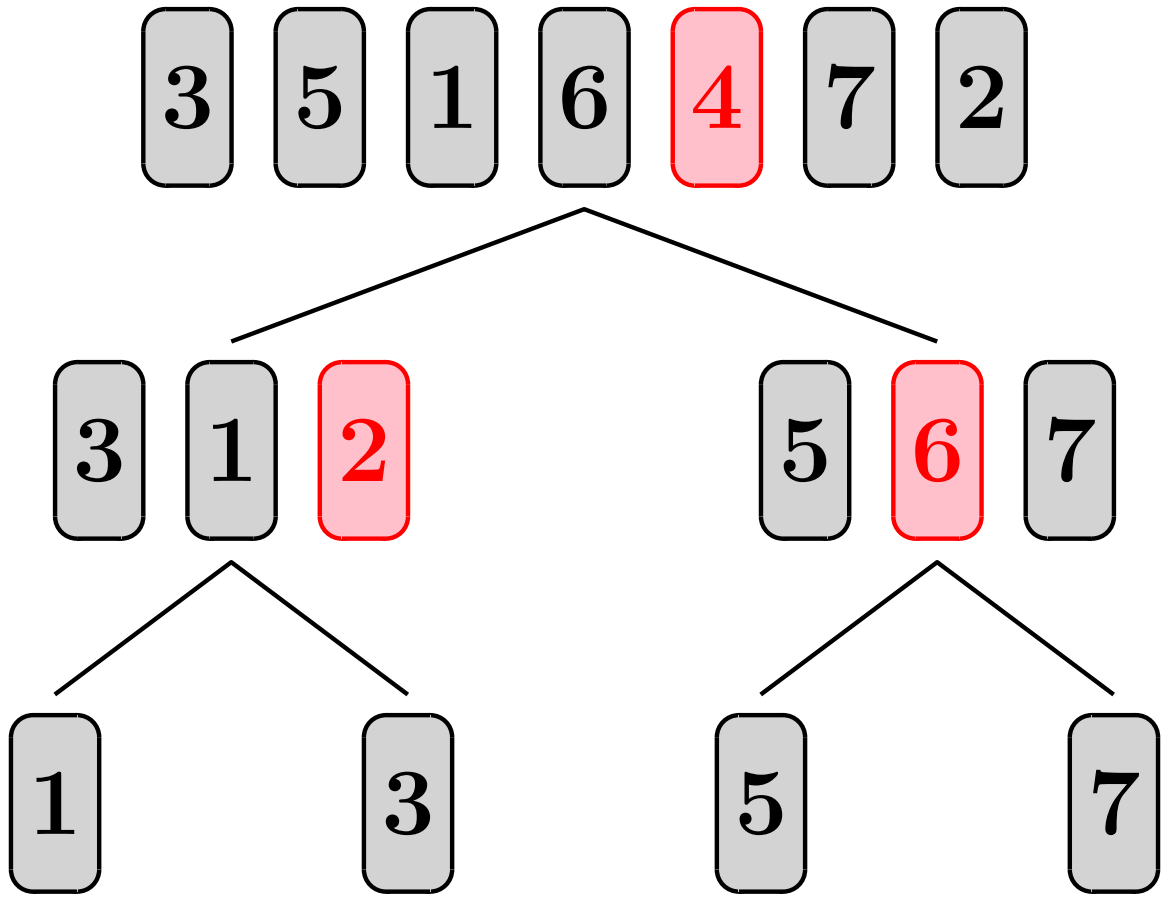
Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Exemple :



Tri rapide (*Quicksort*), version 1

Complexité de `tri_rapide` au pire : $\Theta(n^2)$ comparaisons

Complexité de `tri_rapide` dans le meilleur des cas : $\Theta(n \log n)$ comparaisons

Complexité de `tri_rapide` en moyenne (*admis*) : $\Theta(n \log n)$ comparaisons

Tri rapide (*Quicksort*), version 1

Inconvénients

- `partition` fait deux parcours, là où un seul suffit manifestement
- ne trie pas en place – même les éléments « bien placés » sont déplacés
- les mauvais cas sont des cas « assez probables » : tableaux triés ou presque, à l'endroit ou à l'envers

Tri rapide (*Quicksort*), version 2

```
def tri_rapide(T, debut, fin) : # trie T[debut:fin]
  if fin - debut < 2 : return
  indice_pivot = partition(T, debut, fin)
  tri_rapide(T, debut, indice_pivot)
  tri_rapide(T, indice_pivot + 1, fin)
```

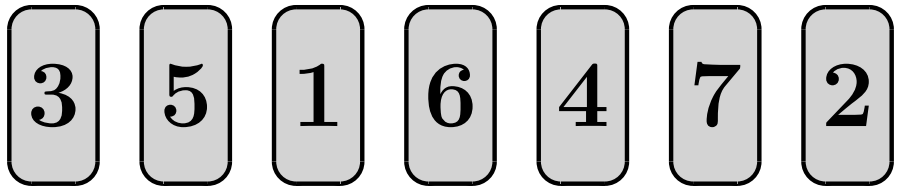
Tri rapide (*Quicksort*), version 2

```
def tri_rapide(T, debut, fin) : # trie T[debut:fin]
  if fin - debut < 2 : return
  indice_pivot = partition(T, debut, fin)
  tri_rapide(T, debut, indice_pivot)
  tri_rapide(T, indice_pivot + 1, fin)

def partition(T, debut, fin) :
  pivot, gauche, droite = T[debut], debut + 1, fin - 1
  while gauche < droite :
    while T[gauche] < pivot : gauche += 1
    while T[droite] > pivot : droite -= 1
    if gauche < droite :
      T[gauche], T[droite] = T[droite], T[gauche]
      gauche, droite = gauche + 1, droite - 1
  T[debut], T[droite] = T[droite], pivot
  return droite
```

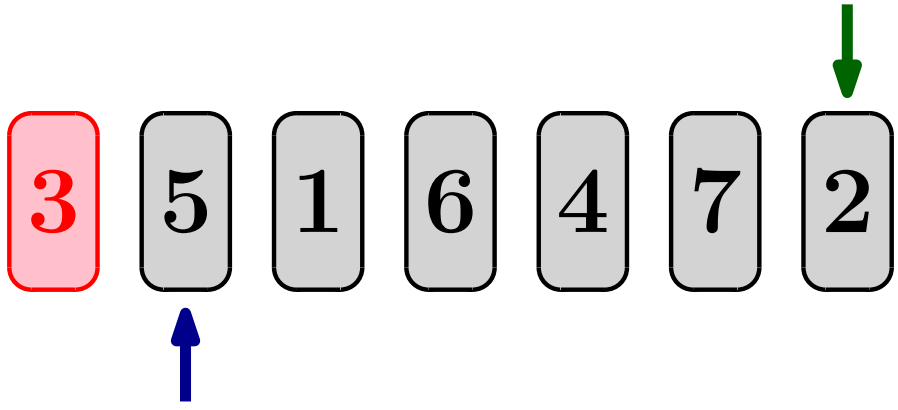
Tri rapide (*Quicksort*), version 2

Exemple :



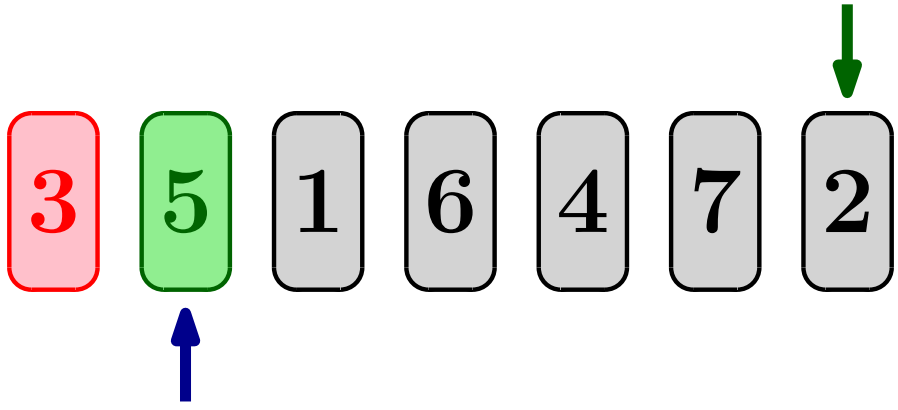
Tri rapide (*Quicksort*), version 2

Exemple :



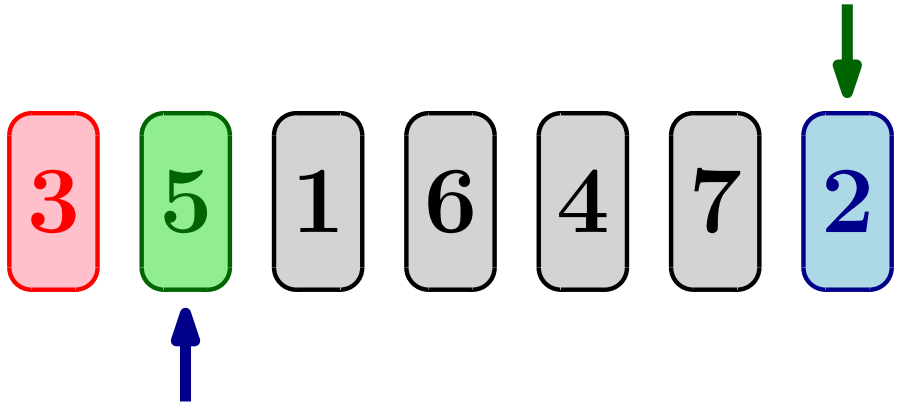
Tri rapide (*Quicksort*), version 2

Exemple :



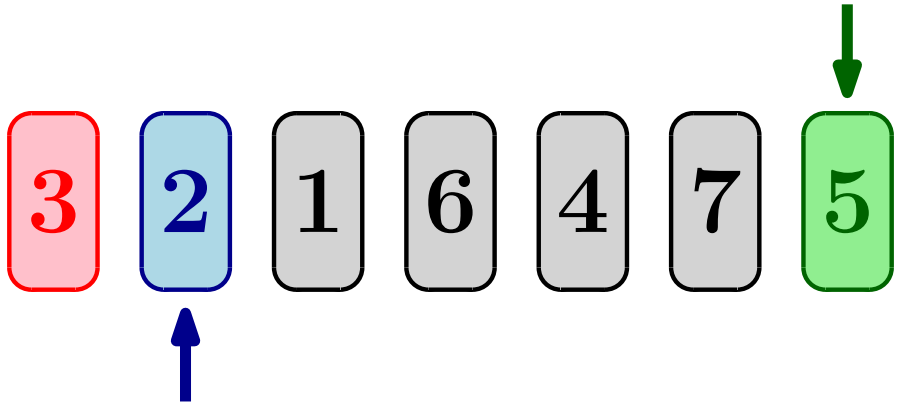
Tri rapide (*Quicksort*), version 2

Exemple :



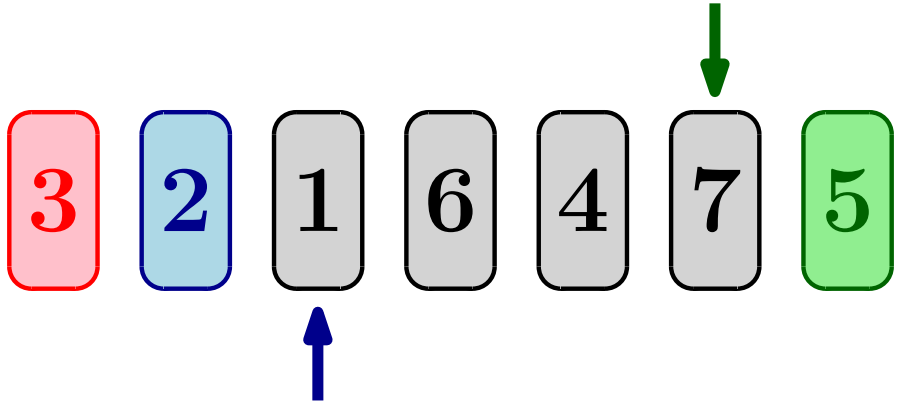
Tri rapide (*Quicksort*), version 2

Exemple :



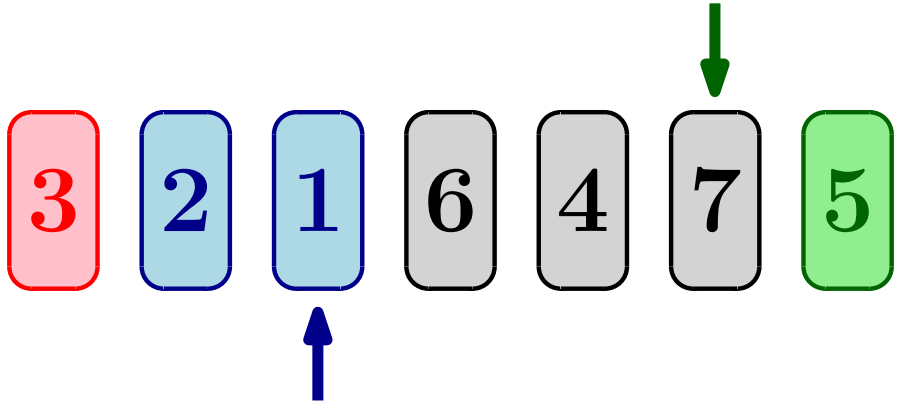
Tri rapide (*Quicksort*), version 2

Exemple :



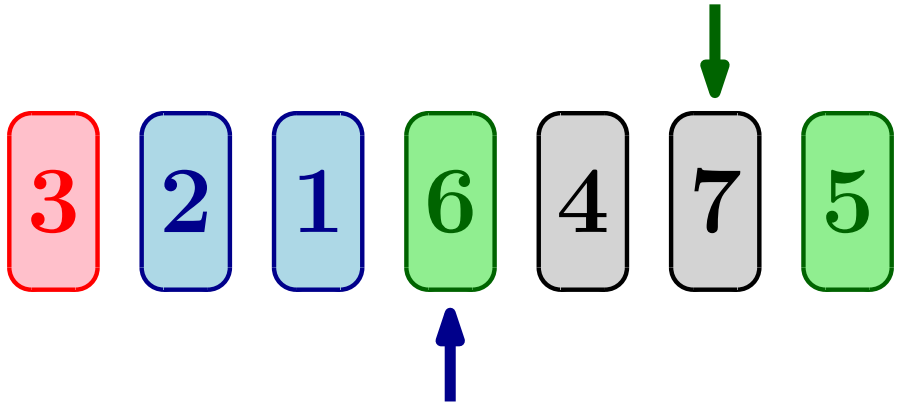
Tri rapide (*Quicksort*), version 2

Exemple :



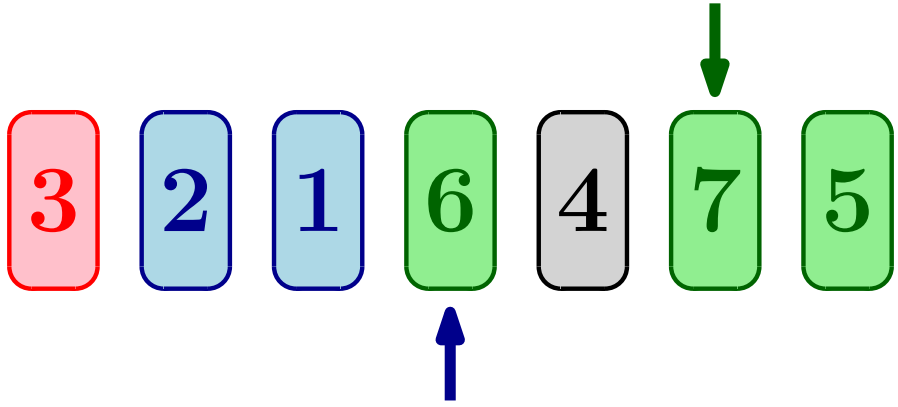
Tri rapide (*Quicksort*), version 2

Exemple :



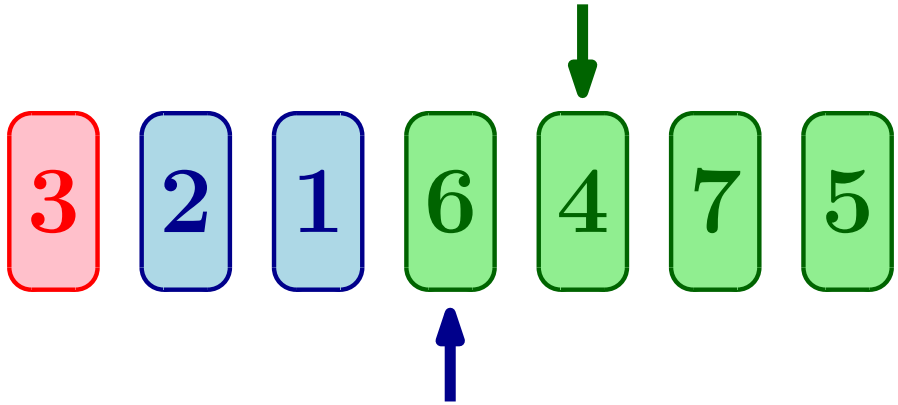
Tri rapide (*Quicksort*), version 2

Exemple :



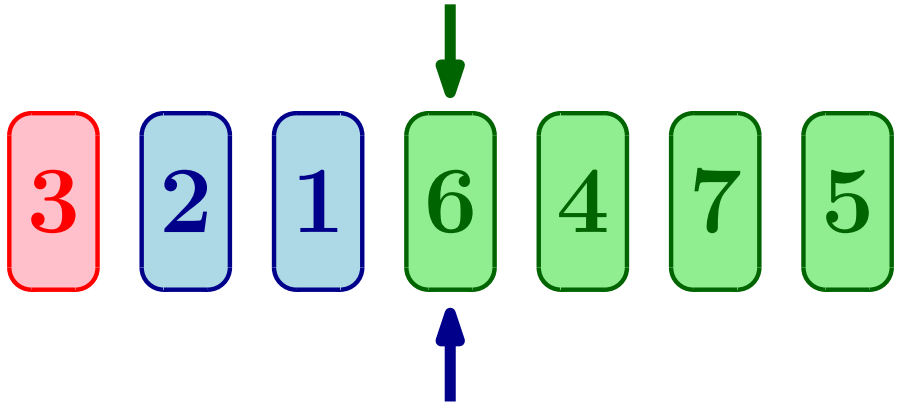
Tri rapide (*Quicksort*), version 2

Exemple :



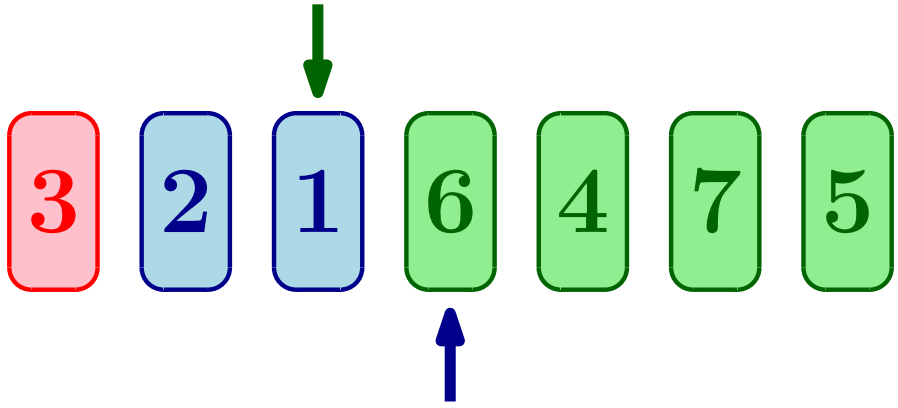
Tri rapide (*Quicksort*), version 2

Exemple :



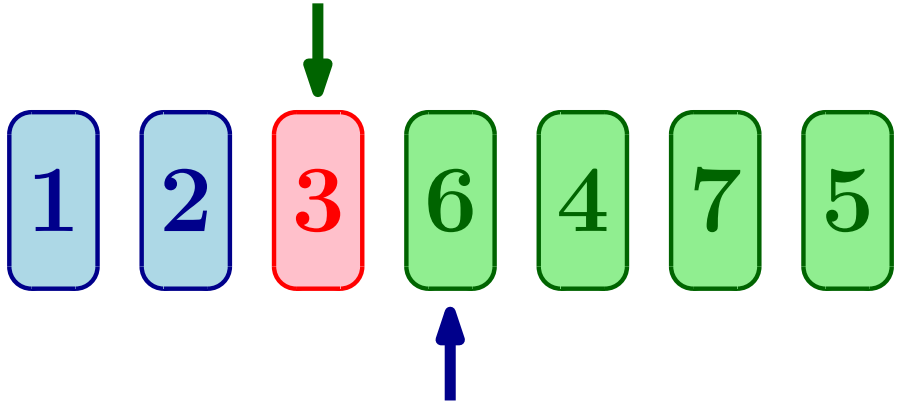
Tri rapide (*Quicksort*), version 2

Exemple :



Tri rapide (*Quicksort*), version 2

Exemple :



Tri rapide (*Quicksort*), version 2

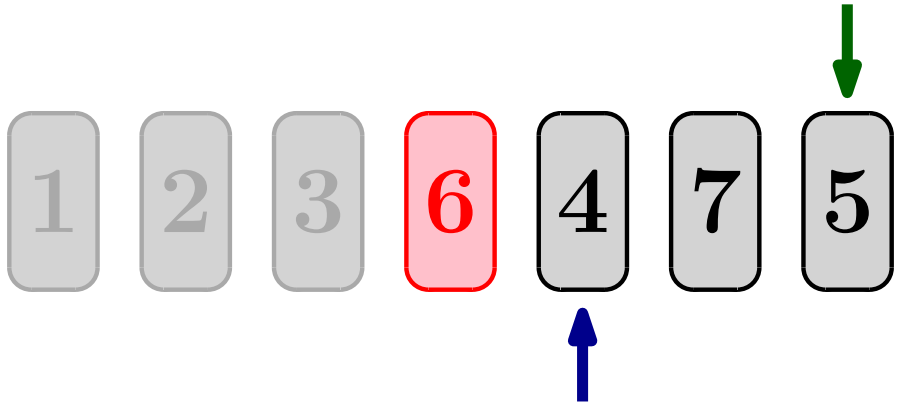
Exemple :

Tri rapide (*Quicksort*), version 2

Exemple :

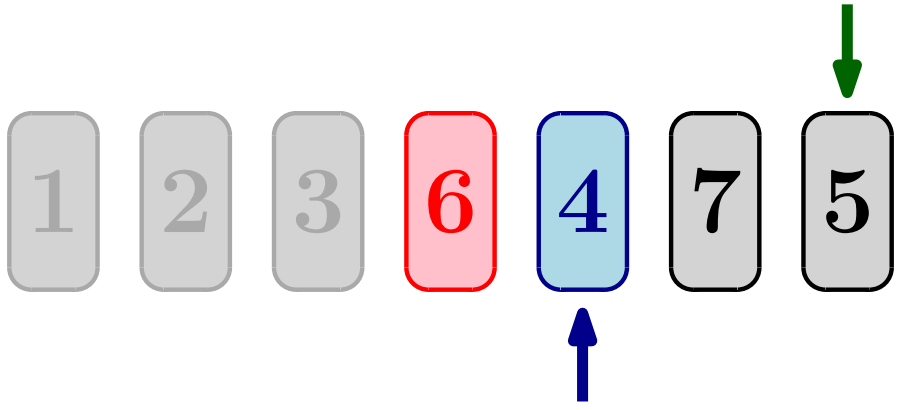
Tri rapide (*Quicksort*), version 2

Exemple :



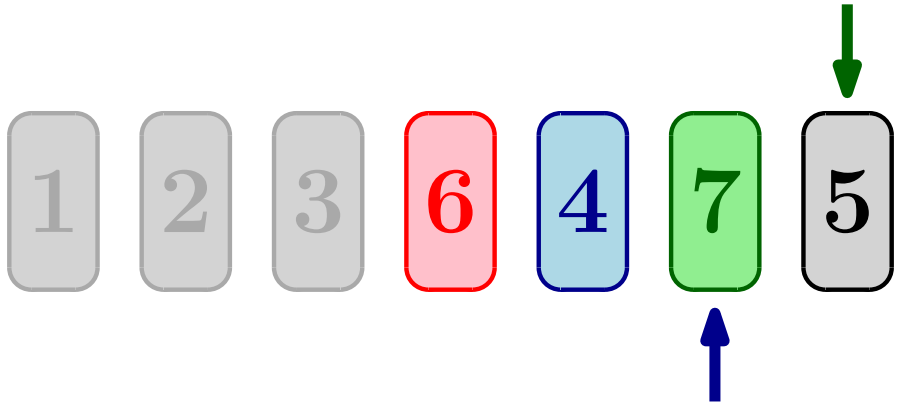
Tri rapide (*Quicksort*), version 2

Exemple :



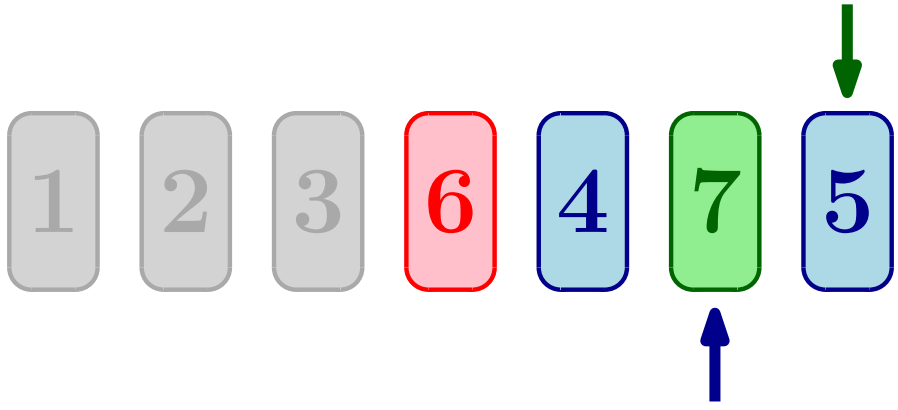
Tri rapide (*Quicksort*), version 2

Exemple :



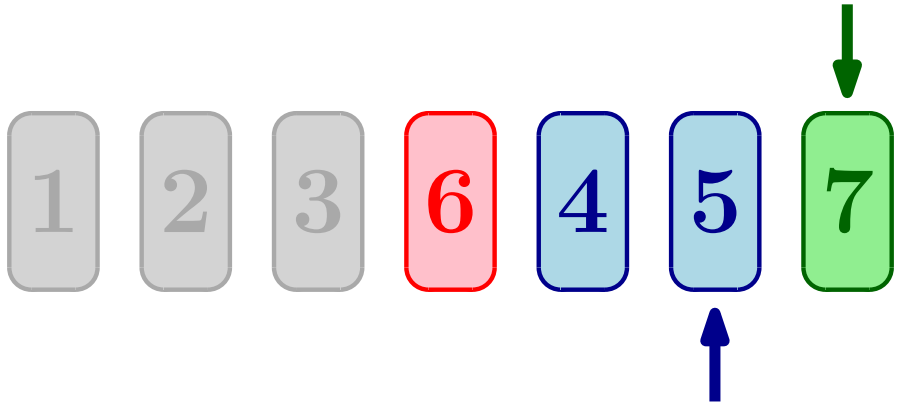
Tri rapide (*Quicksort*), version 2

Exemple :



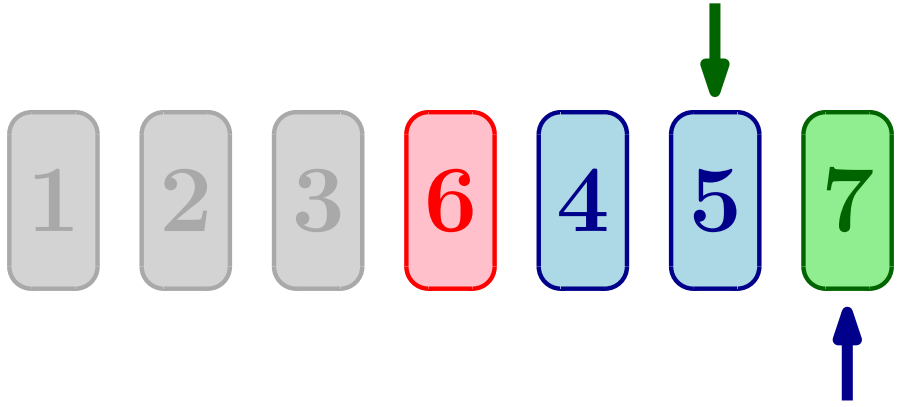
Tri rapide (*Quicksort*), version 2

Exemple :



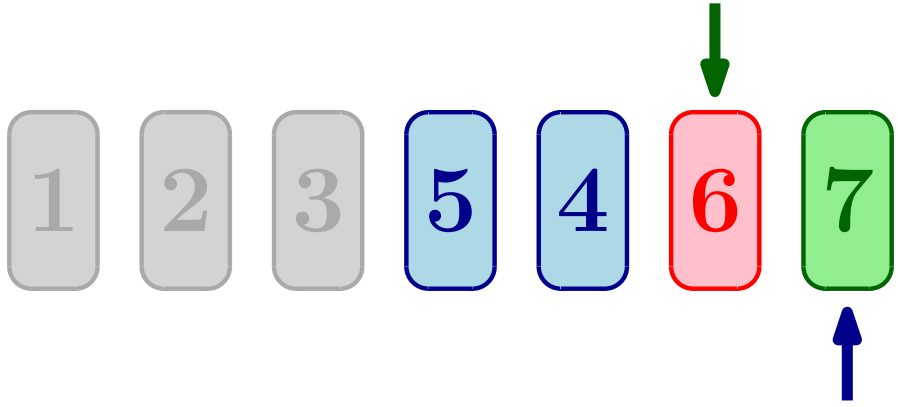
Tri rapide (*Quicksort*), version 2

Exemple :



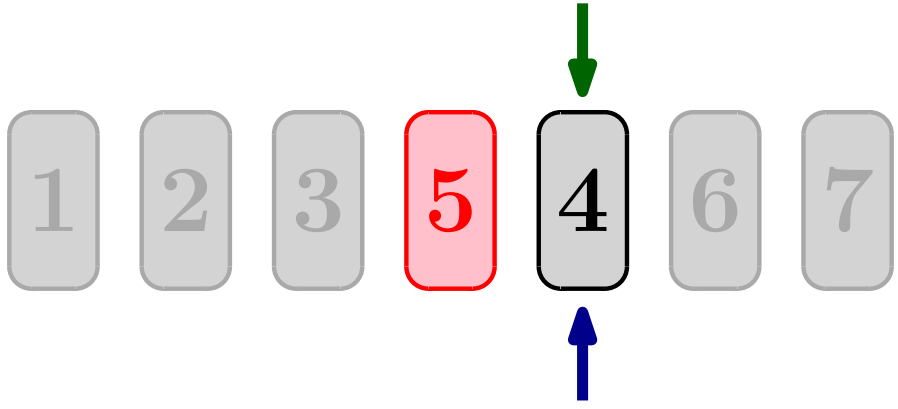
Tri rapide (*Quicksort*), version 2

Exemple :



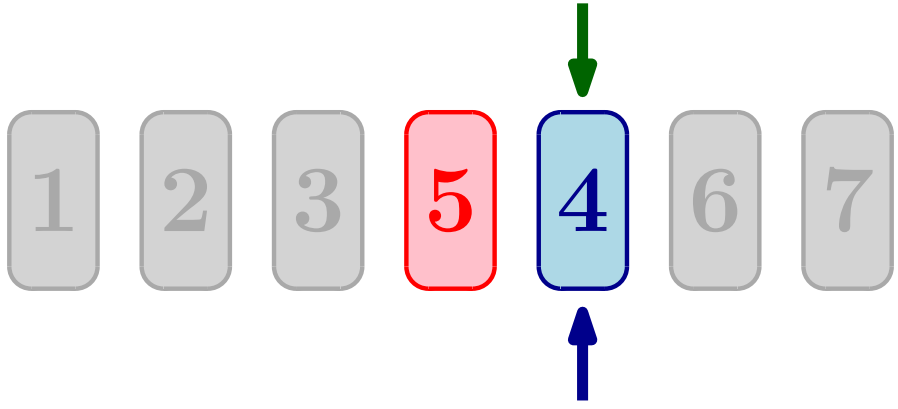
Tri rapide (*Quicksort*), version 2

Exemple :



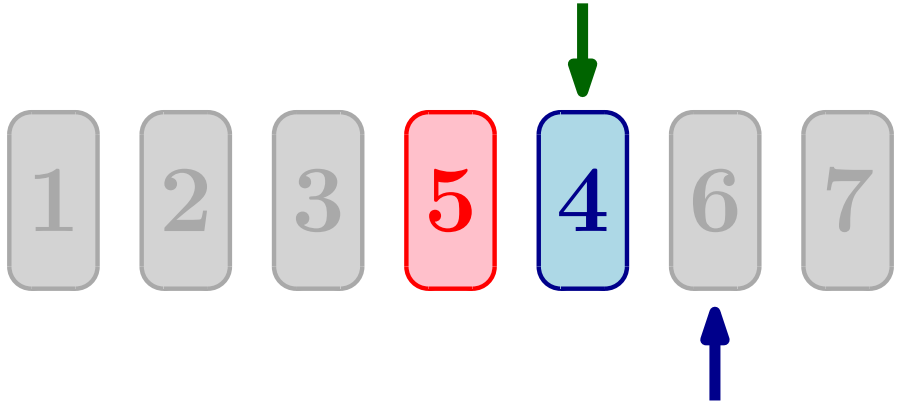
Tri rapide (*Quicksort*), version 2

Exemple :



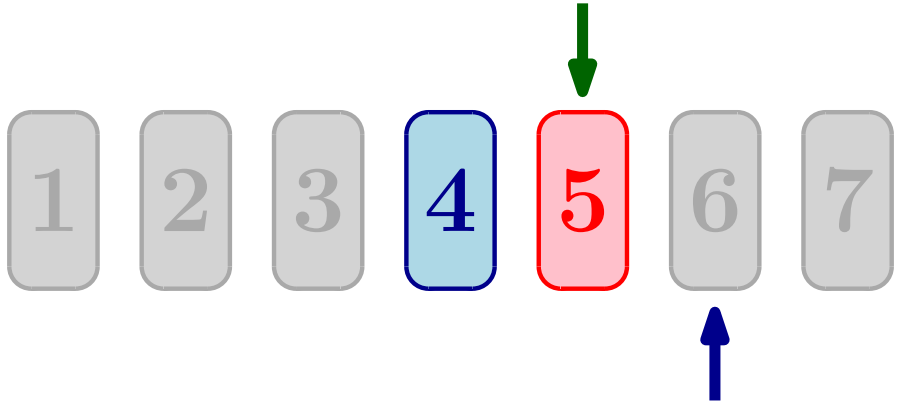
Tri rapide (*Quicksort*), version 2

Exemple :



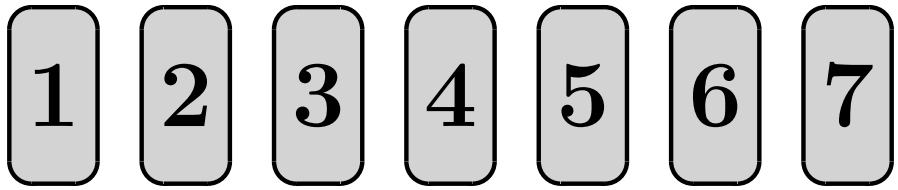
Tri rapide (*Quicksort*), version 2

Exemple :



Tri rapide (*Quicksort*), version 2

Exemple :



Tri rapide, version *randomisée*

Il reste le cas problématique des tableaux (presque) triés

```
def partition(T, debut, fin) :
  alea = random.randint(debut, fin - 1)
  T[debut], T[alea] = T[alea], T[debut]
  pivot, gauche, droite = T[debut], debut + 1, fin - 1
  while gauche < droite :
    while T[gauche] < pivot : gauche += 1
    while T[droite] > pivot : droite -= 1
    if gauche < droite :
      T[gauche], T[droite] = T[droite], T[gauche]
      gauche, droite = gauche + 1, droite - 1
  T[debut], T[droite] = T[droite], pivot
  return droite
```