ISEN
ALL IS DIGITAL!
yncréa

Algorithmes de Tri
CIR 2

Partie 1

Leandro MONTERO
leandro.montero@isen-ouest.yncrea.fr

ISEN
ALL IS DIGITAL!
yncréa

Algorithmes de Tri

- Planning du cours
  - Cours magistraux
  - TDs (à faire en papier SANS l'ordi)
  - TPs (à faire avec l'ordi)

- 4 hs par semaine (15 heures en total)
  - Jours TBA

- Évaluation
  - Mini-rapport noté, surprise et random (date TBA)

    OU
  - Examen final (en papier - date TBA)

ISEN
ALL IS DIGITAL!
yncréa

Algorithmes de Tri

- L'objectif sera de trier un tableau de nombres en ordre croissant.
- Sujet très étudié (et utilisé) !
- On verra plusieurs algorithmes de tri par comparaisons...
- ...et d'autres algorithmes qui n'utilisent pas des comparaisons !

ISEN
ALL IS DIGITAL!
yncréa

Critères d'analyse

- Complexité temporelle:
  - Meilleur des cas
  - Pire des cas
- Complexité spatiale
- Stabilité
- Comportement à l’itération k

ISEN
ALL IS DIGITAL!
yncréa

Les données d'entrée-sortie

- Les données d'entrée de notre problème seront toujours :
  - Un tableau d'entiers (avec doublons)
  - La taille du tableau (toujours dénotée par « *n* »)
  - Les algos (par comparaisons) marcheront toujours avec d'autres types d'entrée, ex : Float, Char, Strings, etc.
- On n'aura pas vraiment de sortie parce que les algos modifient le tableau d'entrée directement !

ISEN ALL IS DIGITAL! | yncréa

Tableau à remplir !

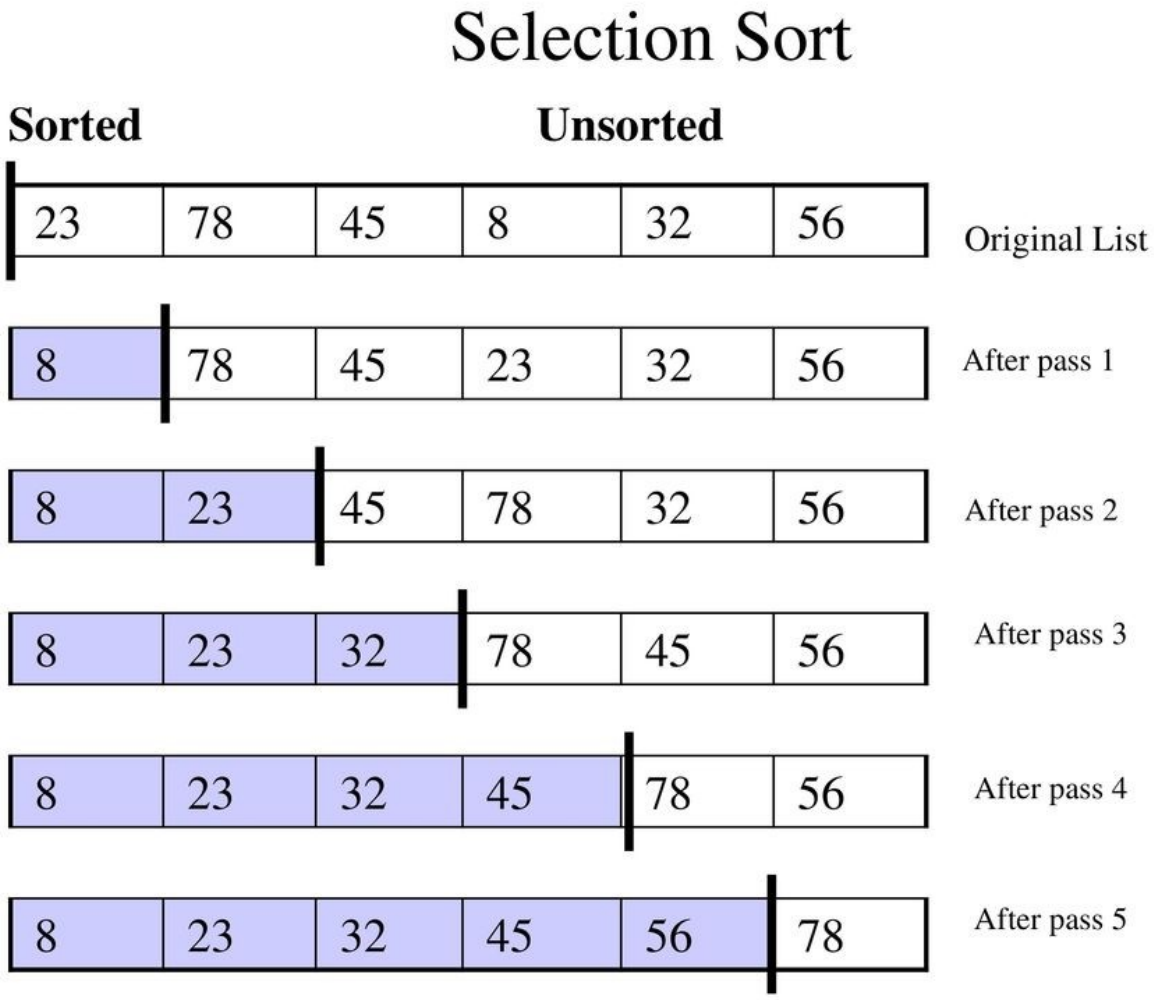
Nom	Complexité Temporelle		Mémoire	Stable	Itération *k* ?
	Meilleur	Pire			
Tri par Sélection					
Tri par Insertion					
Tri à Bulles					
Tri Rapide (Quicksort)					
Tri Fusion (Mergesort)					
Tri Par Tas (Heapsort)					

ISEN ALL IS DIGITAL! | yncréa

Tri par Sélection

- C’est le tri le plus simple à comprendre!
- Idée :
  - Chercher le minimum du tableau et le swapper avec le 1er élément
  - Chercher le minimum suivant et le swapper avec le 2ème élément
  - Etc etc jusqu’à ce qu’on arrive au dernier élément

Tri par Sélection

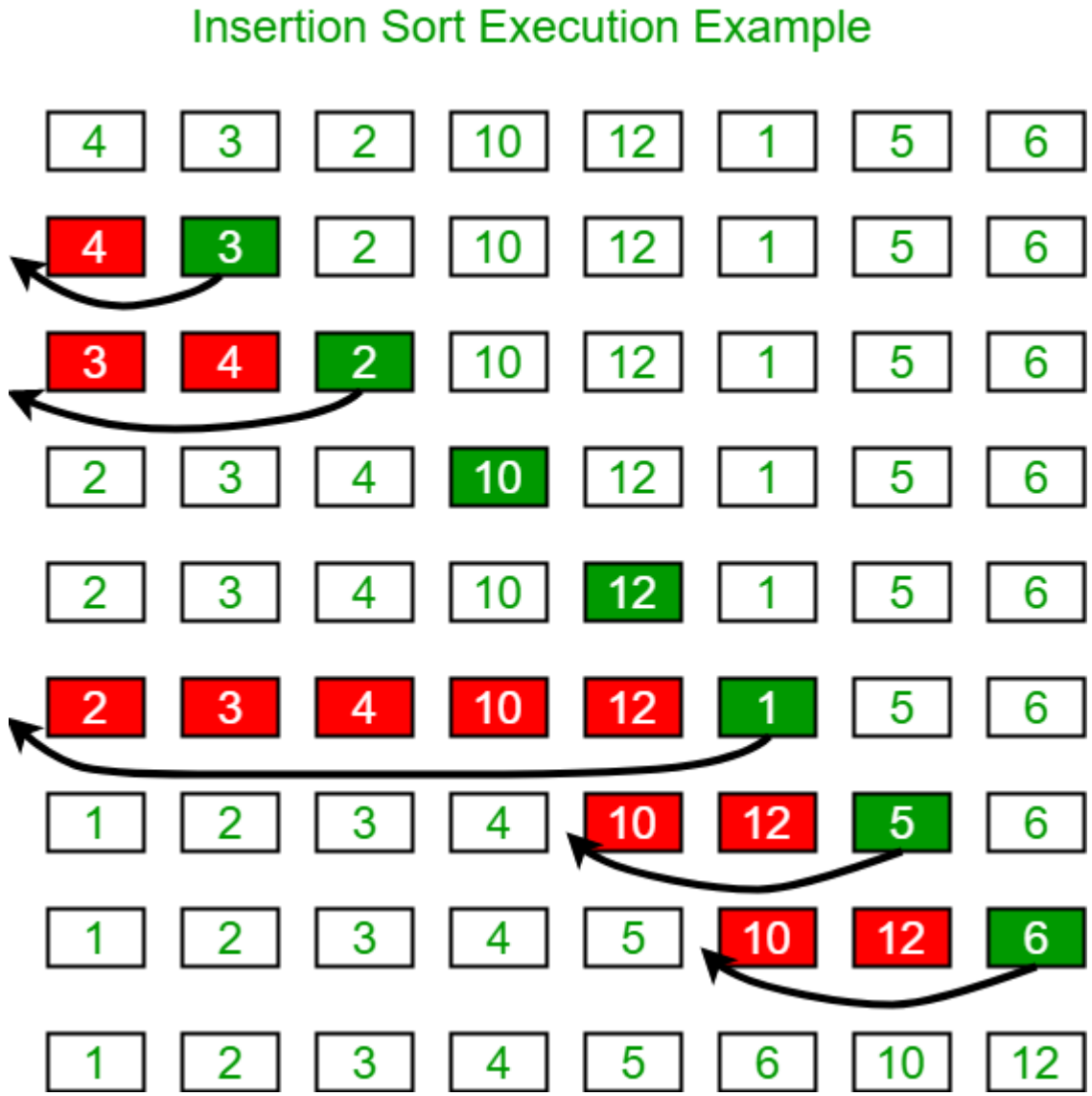


ISEN
ALL IS DIGITAL!
yncréa

Tri par Insertion

- C’est le tri qu’on utilise quand on joue aux cartes
- Très rapide pour les petites entrées donc souvent utilisé dans plusieurs langages de programmation
- Idée :
  - Regarder le tableau de gauche a droite jusqu’à ce qu’on trouve un élément pas trié
  - Placer cet élément dans sa bonne position en revenant en arrière
  - Répéter jusqu’à ce qu’on arrive au dernier élément

Tri par Insertion



Tri à Bulles

- C’est le tri qu’on n’utilise JAMAIS :D
- Intéressant du point de vue pédagogique
- Idée :
  - Avancer le long du tableau de gauche à droite (jusqu’à la fin) en comparant deux éléments consécutifs
  - À chaque fois qu’ils ne sont pas dans le bon ordre, les swapper
  - Recommencer dès le début du tableau jusqu’à ce qu’il soit trié

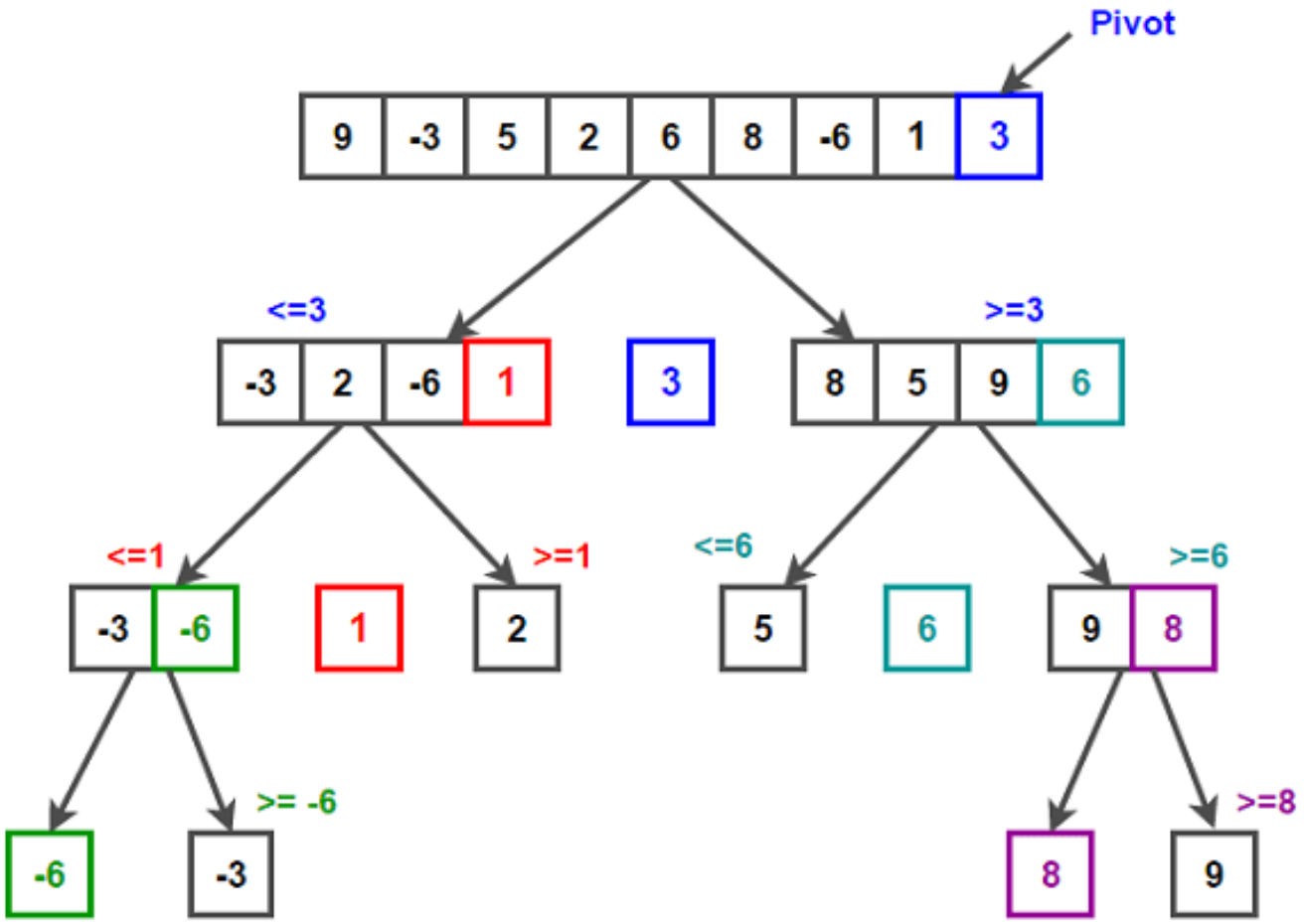
Tri à Bulles

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 6	0	1	2	**3**					
		1	**2**						

Tri Rapide (Quicksort)

- Un des algos le plus utilisé dans les langages de programmation
- Utilise la technique appelée « Divide & Conquer »
- Idée :
  - « Choisir » un élément du tableau (appelé pivot)
  - Mettre tous les éléments inférieurs au pivot à gauche et les autres à droite
  - Répéter la même procédure avec chaque sous-partie du tableau

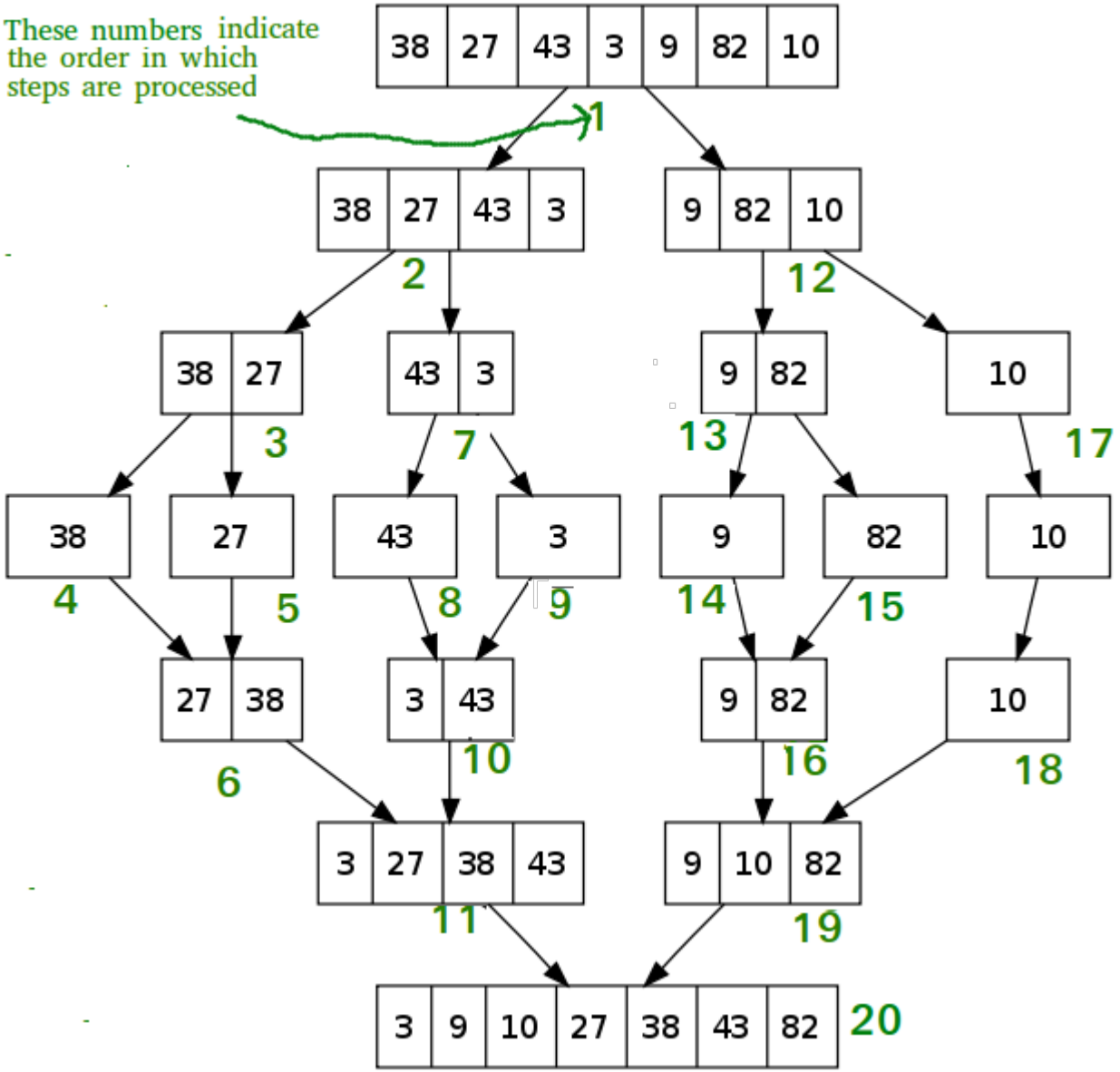
Tri Rapide (Quicksort)



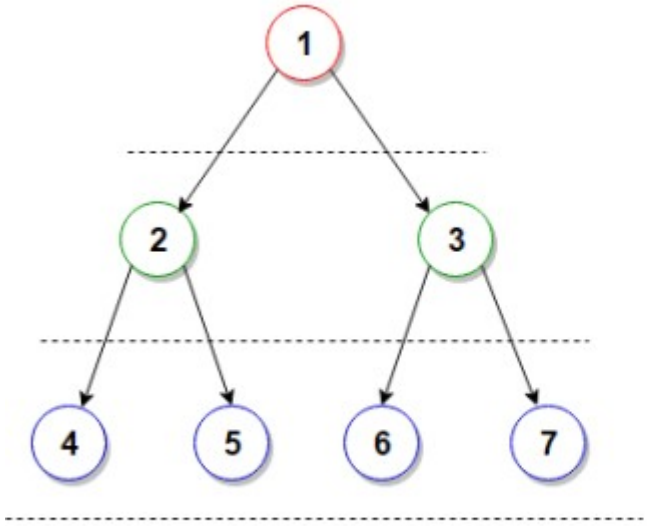
Tri Fusion (Mergesort)

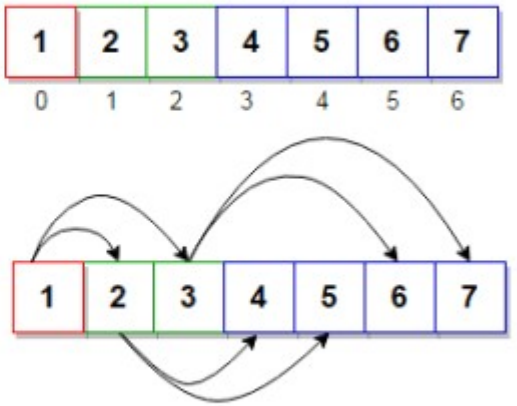
- (Aussi) Un des algos le plus utilisé dans les langages de programmation (avec Quicksort)
- (Aussi) Utilise la technique appelée « Divide & Conquer »
- Idée :
  - Si le tableau a un seul élément, il est déjà trié
  - Sinon, diviser le tableau en deux moitiés de ~même taille et répéter le processus
  - Fusionner les deux tableaux triés dans un seul tableau trié

Tri Fusion (Mergesort)



Tri Par Tas (Heapsort)

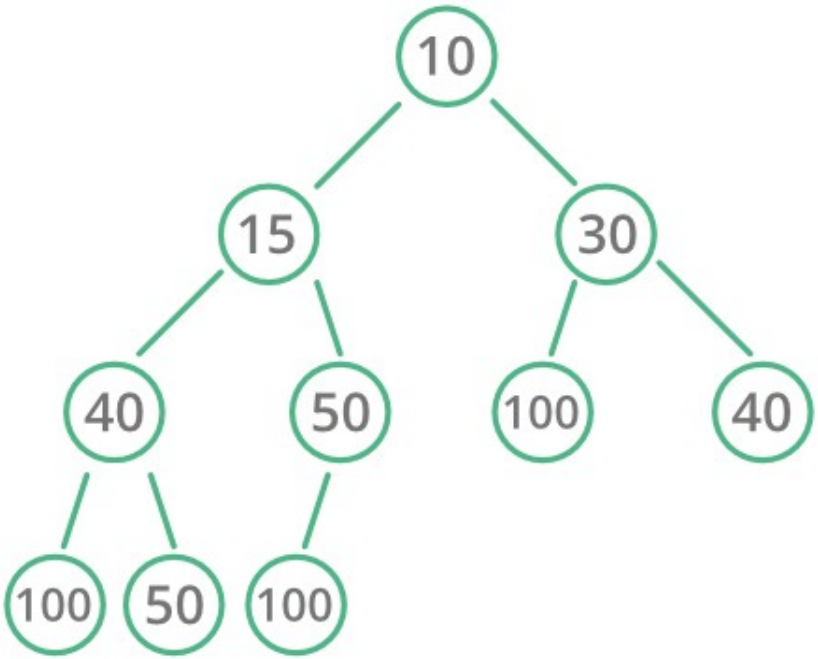




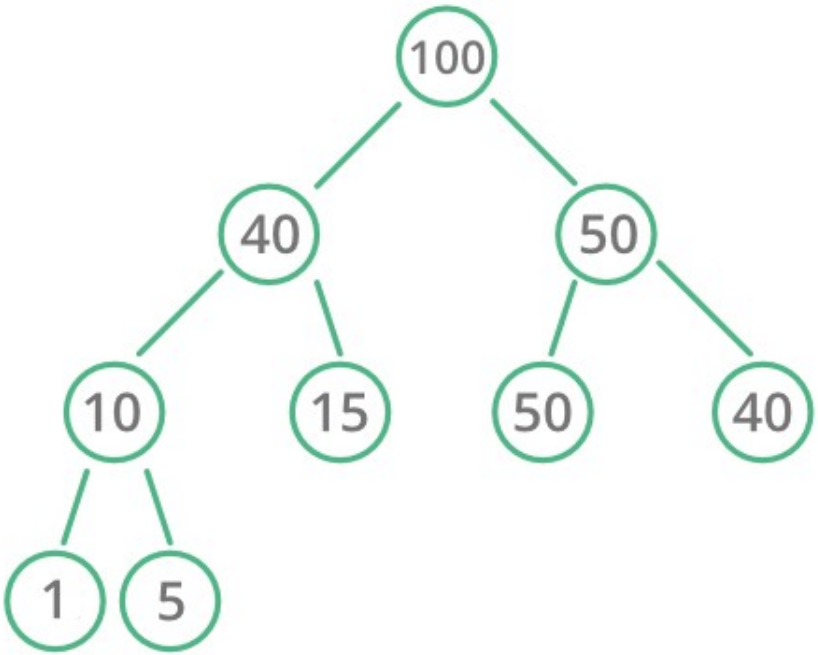
On peut facilement représenter un arbre binaire avec un tableau tel que un nœud i dans l'arbre a deux fils :

- fils à gauche : indice 2*i +1 dans le tableau
- fils à droite : indice 2*i +2 dans le tableau

Tri Par Tas (Heapsort)



Min Heap



Max Heap

Un tas T (ou Heap) est une structure de données telle que :

- T est un arbre binaire complet à gauche
- Chaque nœud vérifie que sa valeur est ≥ à celle de ses fils, on l'appelle Max heap (sinon avec ≤ est Min Heap)

Ne pas confondre la structure de données « tas » (ou « heap ») avec le « tas » (ou « heap ») de mémoire dynamique!!

Tri Par Tas (Heapsort)

- Moins utilisé que le Quicksort et Mergesort, par contre, utilisé dans des systèmes en temps réel (Linux kernel,...)

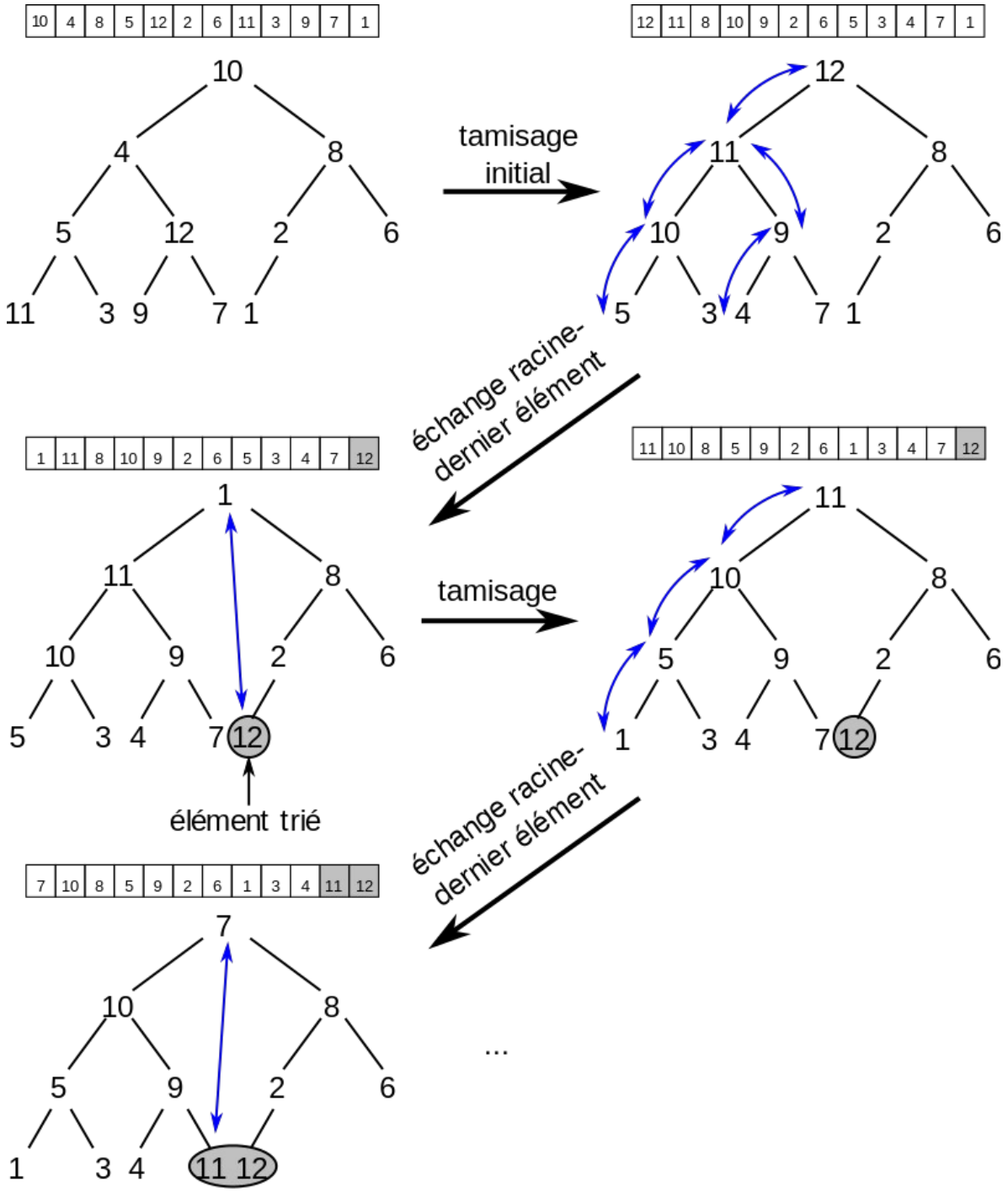
- Idée :

  - Construire un tas à partir du tableau d'entrée

  - Swapper la racine du tas (le premier élément, c'est aussi le plus grand) avec le dernier fils du dernier niveau (le dernier élément du tableau)

  - Refaire le tas (sans prendre en compte ce dernier élément)

  - Répéter les derniers deux étapes


10 4 8 5 12 2 6 11 3 9 7 1
tamisage initial
12 11 8 10 9 2 6 5 3 4 7 1
échange racine-dernier élément
1 11 8 10 9 2 6 5 3 4 7 12
tamisage
11 10 8 5 9 2 6 1 3 4 7 12
élément trié
échange racine-dernier élément
7 10 8 5 9 2 6 1 3 4 11 12
...

ISEN ALL IS DIGITAL! | yncréa

Comparaison finale

Nom	Complexité Temporelle		Mémoire	Stable
	Meilleur	Pire		
Tri par Sélection	$O(n^2)$	$O(n^2)$	O(1)	Non
Tri par Insertion	O(n)	$O(n^2)$	O(1)	Oui
Tri à Bulles	$O(n^2)$ v1.0 O(n) v2.0	$O(n^2)$	O(1)	Oui
Tri Rapide (Quicksort)	O(n log(n))	$O(n^2)$ Cas moyen O(n log(n))	O(n) Cas moyen O(log(n))	Non
Tri Fusion (Mergesort)	O(n log(n))	O(n log(n))	O(n)	Oui
Tri Par Tas (Heapsort)	O(n log(n))	O(n log(n))	O(1)	Non

Quelques mentions spéciales !

Slowsort

- N'est pas utilisé, il a été crée comme une blague !
- Utilise la technique « Multiply & Surrender »
- Pseudocode:

```
slowsort(tableau A ; indices i,j){
   if (i >= j){
      return
   }
   m =(i+j)/2
   slowsort(A,i,m)
   slowsort(A,m+1,j)
   if (A[j] < A[m]){
      swap A[j] and A[m]
   }
   slowsort(A,i,j-1)
}
```

- Complexité :

Quelques mentions spéciales !

Slowsort

- N'est pas utilisé, il a été crée comme une blague !
- Utilise la technique « Multiply & Surrender »
- Pseudocode:

```
slowsort(tableau A ; indices i,j){
   if (i >= j){
      return
   }
   m =(i+j)/2
   slowsort(A,i,m)
   slowsort(A,m+1,j)
   if (A[j] < A[m]){
      swap A[j] and A[m]
   }
   slowsort(A,i,j-1)
}
```

- Complexité : Ω(n^(log(n)/2+ε)) avec ε > 0

Quelques mentions spéciales !

Bogosort (a.k.a. Tri Stupide)

- Bien évidemment, ça sert à rien !
- Pseudocode:

```
bogosort(tableau A){
   while (A n'est pas trié){
      mélanger aléatoirement les éléments de A
   }
}
```

- Complexité
  - Meilleur des cas :
  - Cas moyen :
  - Pire des cas :

Quelques mentions spéciales !

Bogosort (a.k.a. Tri Stupide)

- Bien évidemment, ça sert à rien !
- Pseudocode:

```
bogosort(tableau A){
   while (A n'est pas trié){
      mélanger aléatoirement les éléments de A
   }
}
```

- Complexité
  - Meilleur des cas : O(n)
  - Cas moyen : O(n*n!)
  - Pire des cas : O(+∞)