

# Tri par Fusion ( Merge sort )

## *Algorithme “Diviser pour régner”*

### PARTICIPANTS :

Lezoul imene  
Yasmine Fares  
SEKKAL Wassila  
ESCHROUGUI Yousra  
Hadj Mohammed douae  
Galleze nada



**Professeur : Mme AROUSSI**

# 1 - Environnement Utilisé :

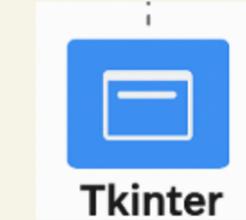
---

## **ENVIRONNEMENT MATÉRIEL :**

- -Processeur : Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz
- -RAM: 16.00 GB

## **ENVIRONNEMENT LOGICIEL :**

- Os utilisé : Windows 11 PRO
- Langage : Python 3.x
- Bibliothèque : Tkinter, Interface graphique (fenêtre, canvas, boutons...)  
tkinter.messagebox (gestion des messages d'erreur et d'information)  
math.ceil (opérations mathématiques)
- Outils : VS Code / PyCharm / IDLE (éditeur Python par défaut)
- Paradigmes de programmation utilisés  
Programmation orientée objet : classe MergeSortTreeSteps



Cette classe gère :

- l'interface graphique,
- la génération des étapes du tri,
- l'affichage de l'arbre,
- la gestion des interactions utilisateur

# 1 - Environnement Utilisé :

---

## ***Programmation événementielle :***

- *Actions déclenchées par l'utilisateur :*

- *Charger*
- *Valider et générer étapes*
- *Étape suivante*
- *Précédent*
- *Réinitialiser*

## ***ASSISTANCE DU CHATBOT IA***

*ChatGPT (OpenAI) a contribué à environ 50% du projet :*

- *corrections du code,*
- *amélioration de la visualisation graphique,*
- *aide à structurer les fonctions,*
- *optimisation de l'affichage des étapes de merge sort.*

## 2 - Introduction : Qu'est-ce que le tri fusion ?

---

Le Tri de Fusion est un algorithme de tri performant fondé sur le principe du

→ Diviser pour régner.

Il consiste à découper récursivement une liste en sous-listes, les trie individuellement, puis les fusionner pour obtenir un ensemble ordonné .

# 3 - Structures de Données et Opérations :

- **le principe de tri de fusion :**

## A. Étape 1 : Division

- On divise le tableau en deux
- Puis chaque moitié est de nouveau divisée
- Jusqu'à obtenir des sous-tableaux d'un seul élément

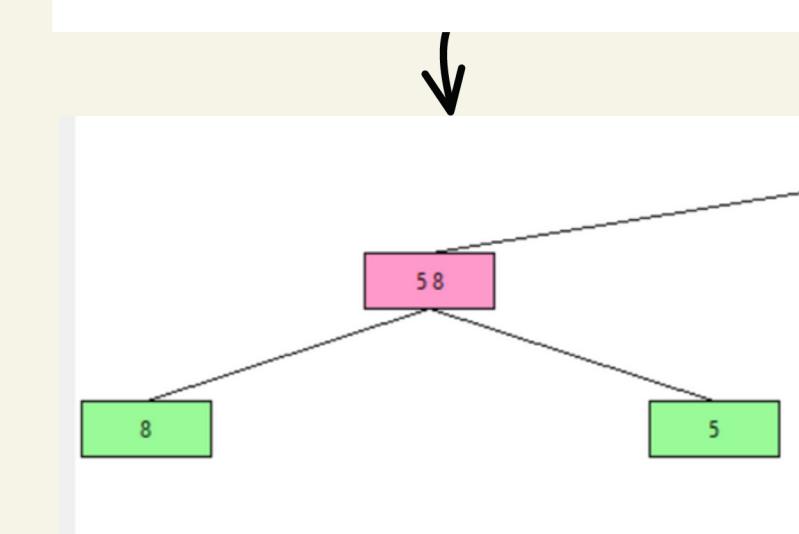
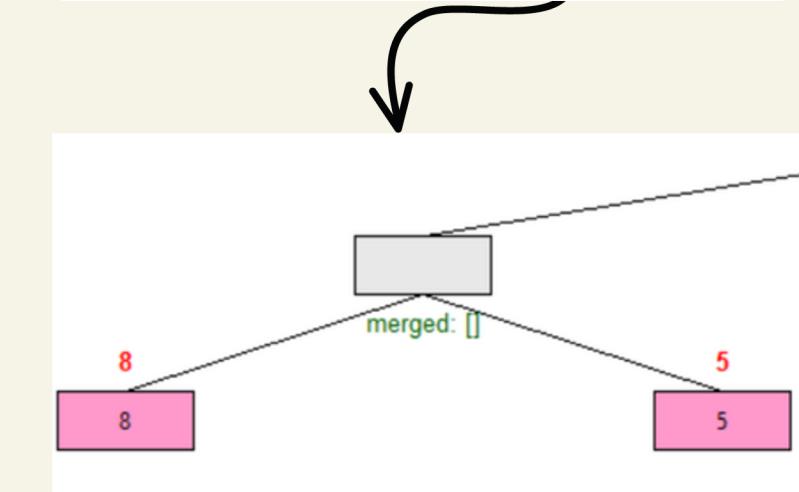
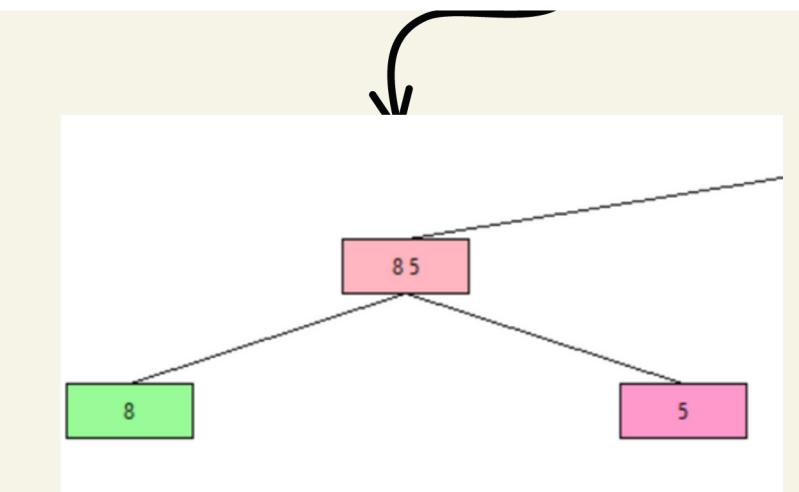
## B. Étape 2 : Fusion(Merge)

- On compare les éléments un par un
- On prend toujours l'élément le plus petit
- On recompose un tableau trié

## B. Étape 2 : Combiner

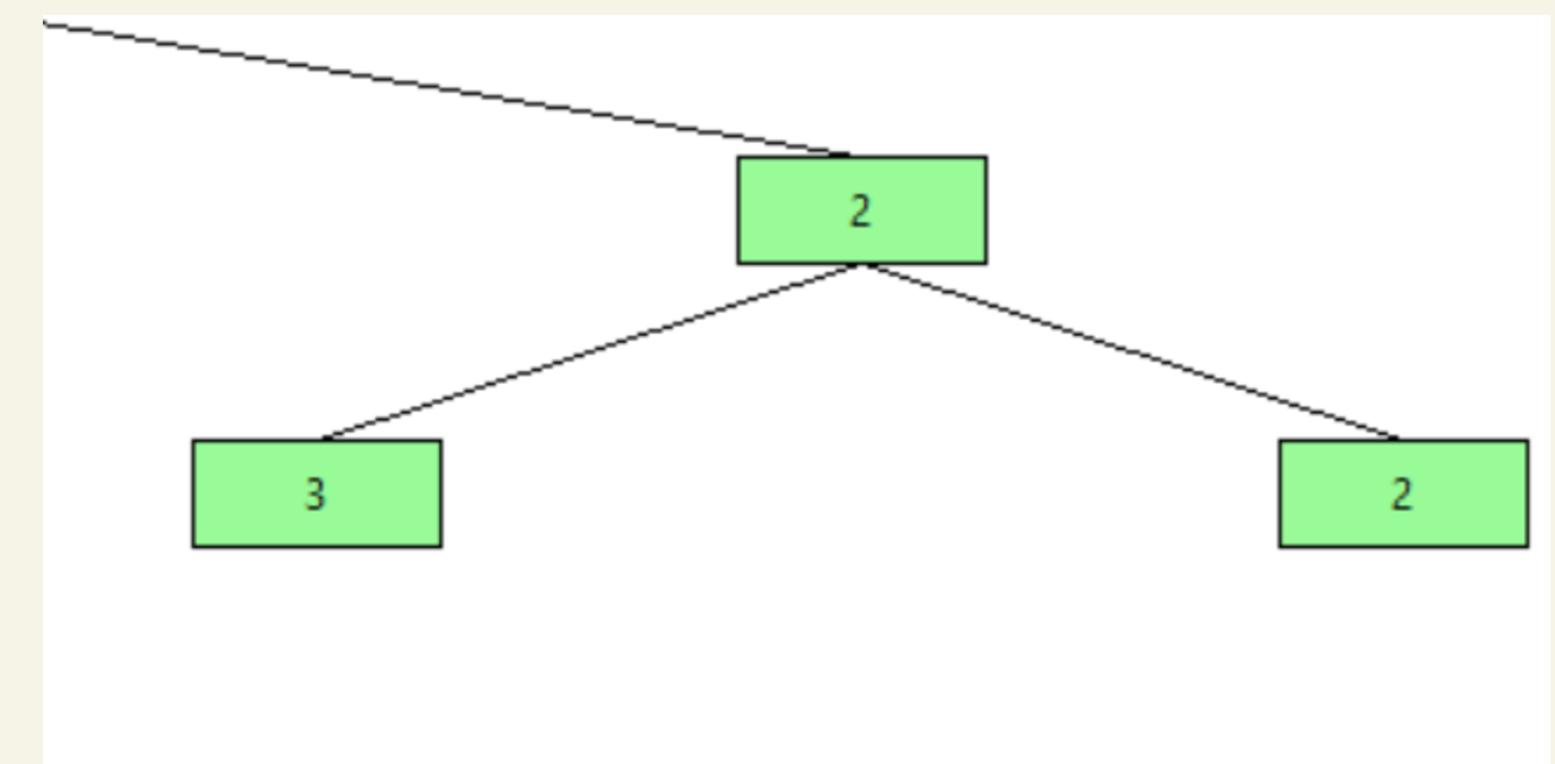
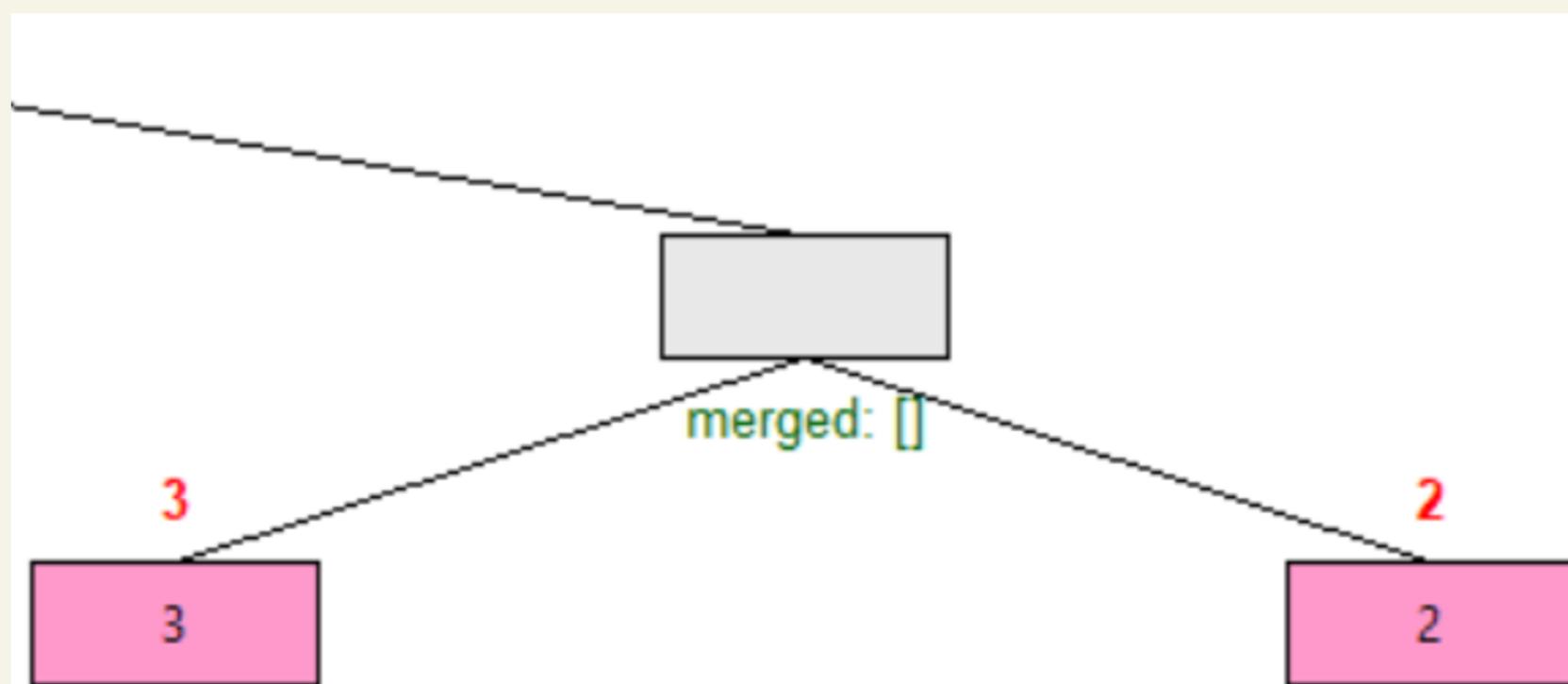
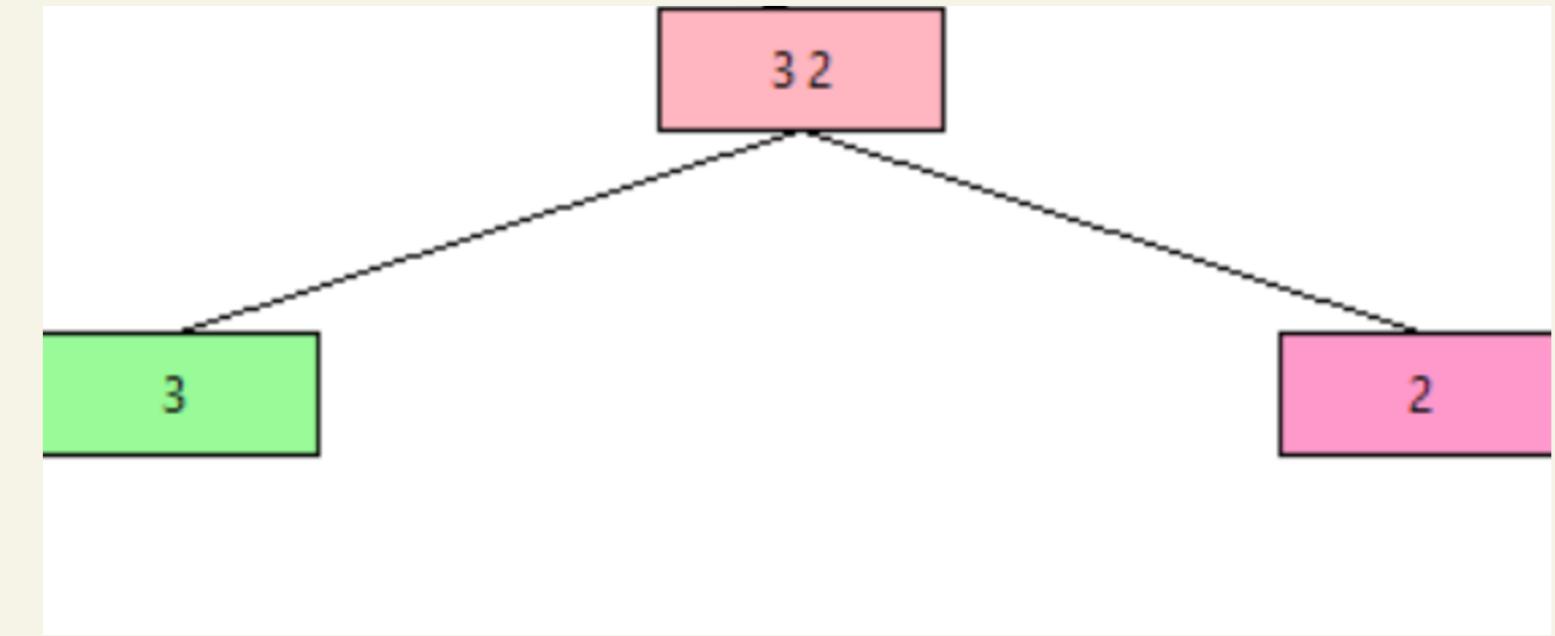
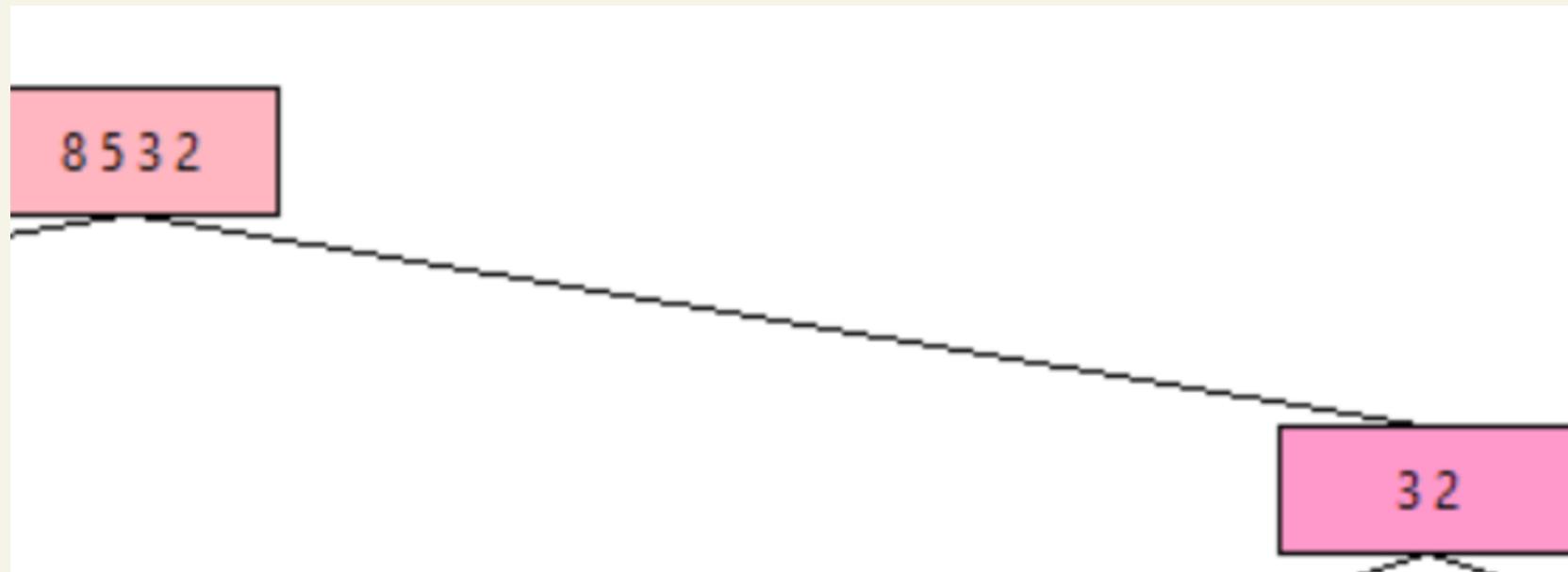
- fusionner les deux moitiés triées pour obtenir le tableau final trié.

## Exemple :



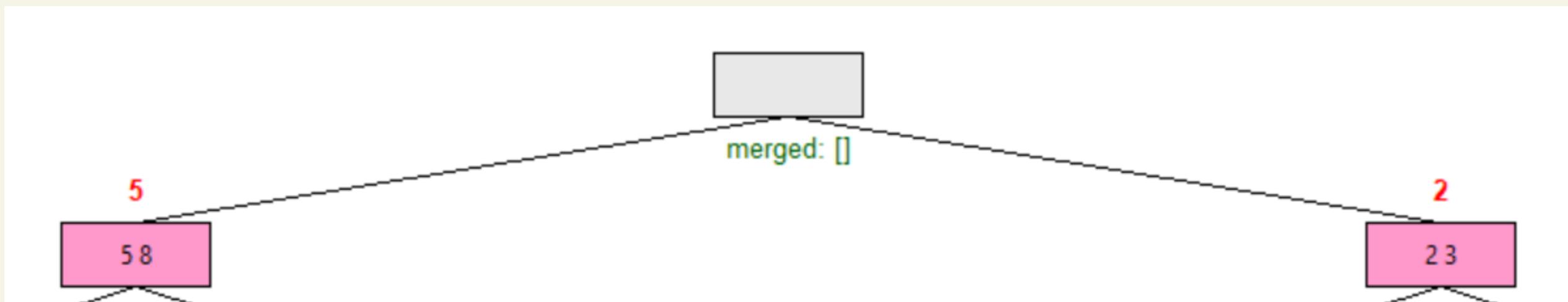
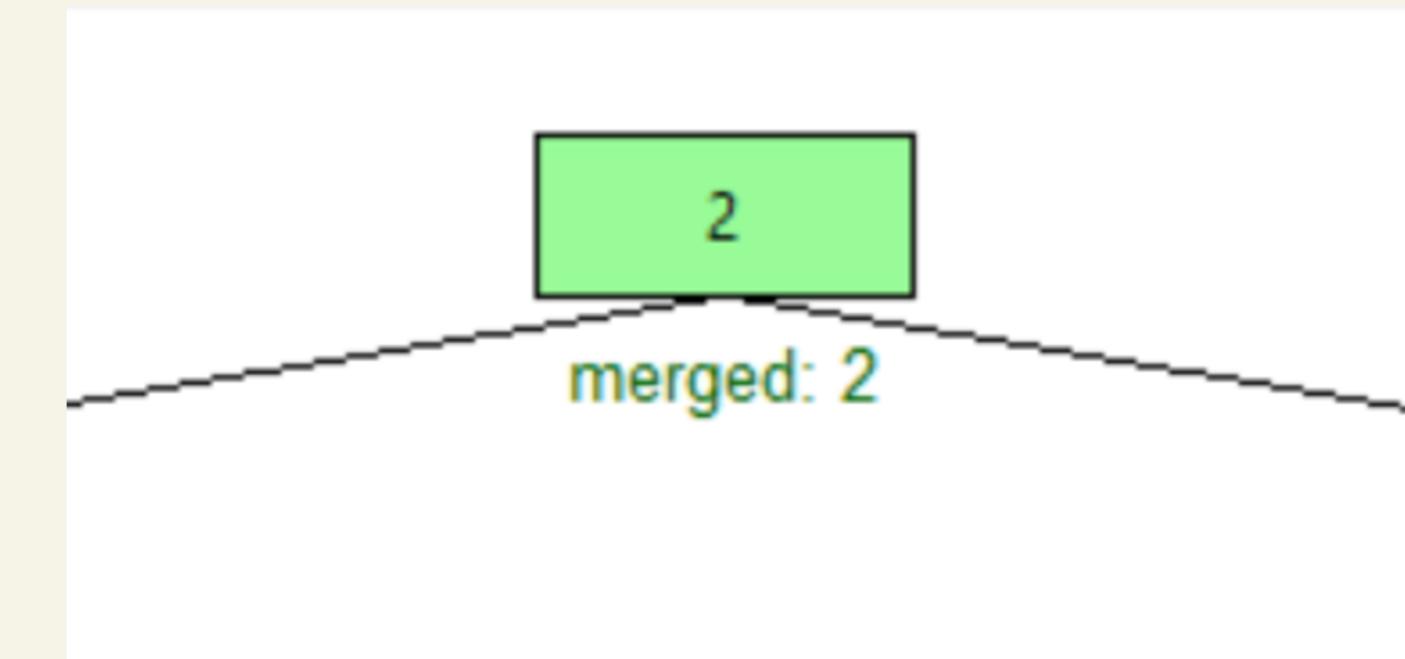
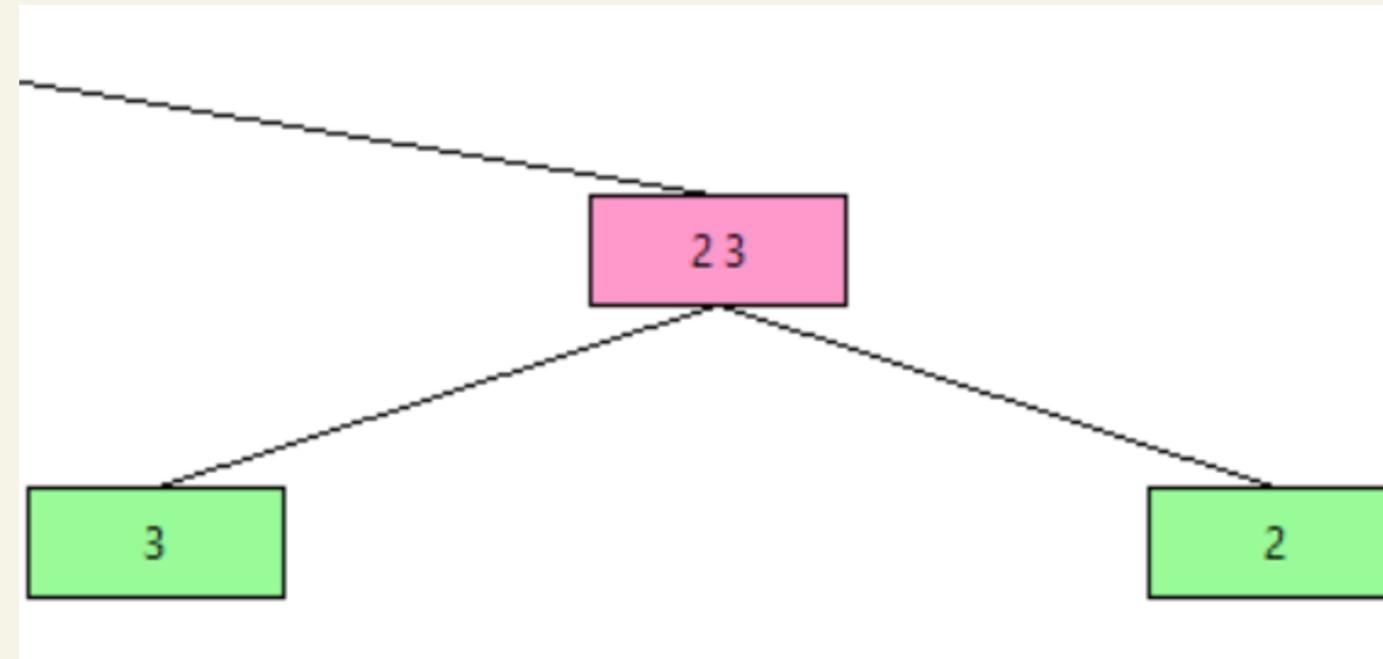
# 3 - Structures de Données et Opérations :

---



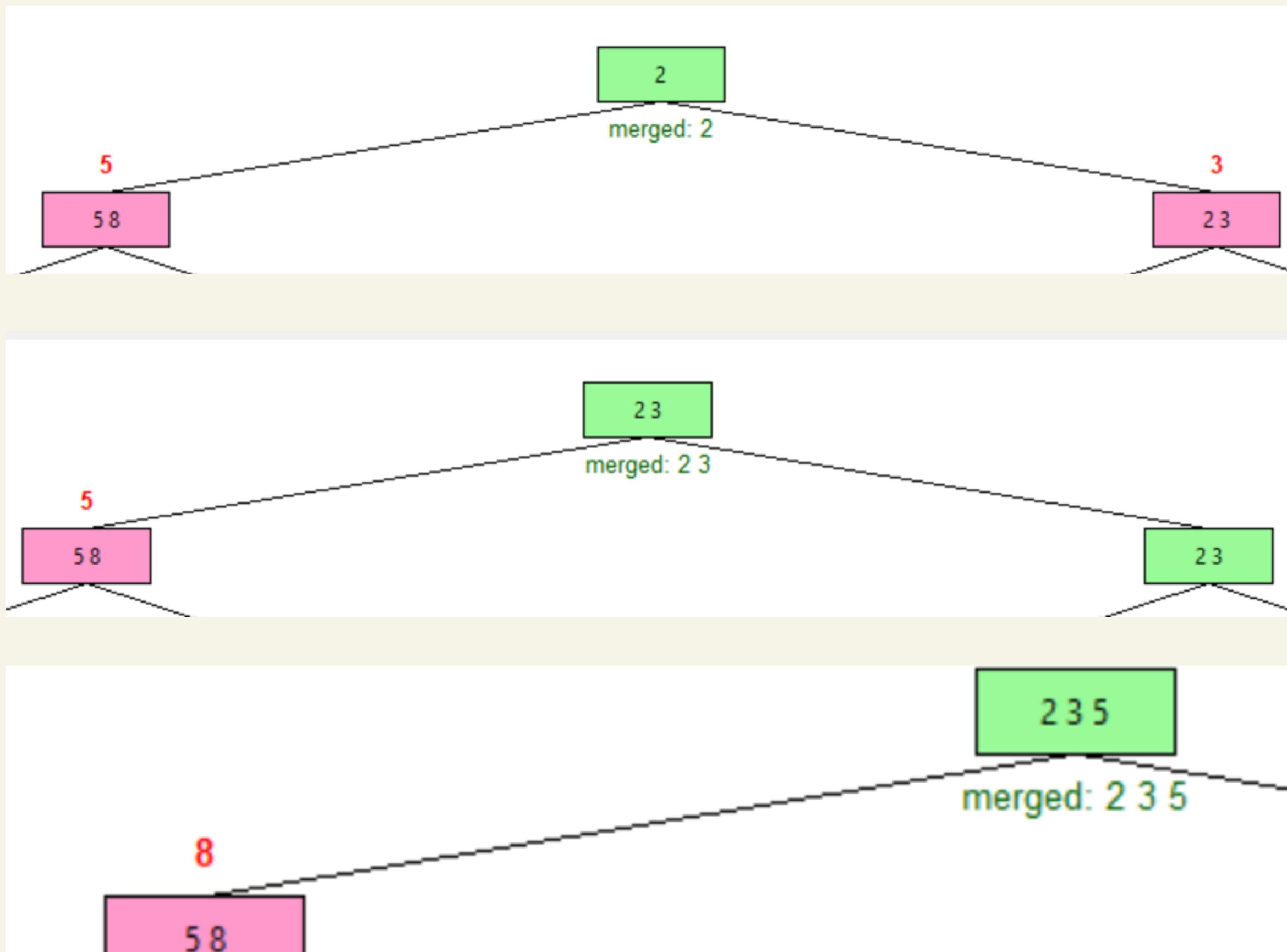
# 3 - Structures de Données et Opérations :

---



### 3- STRUCTURES DE DONNÉES ET OPÉRATIONS :

---

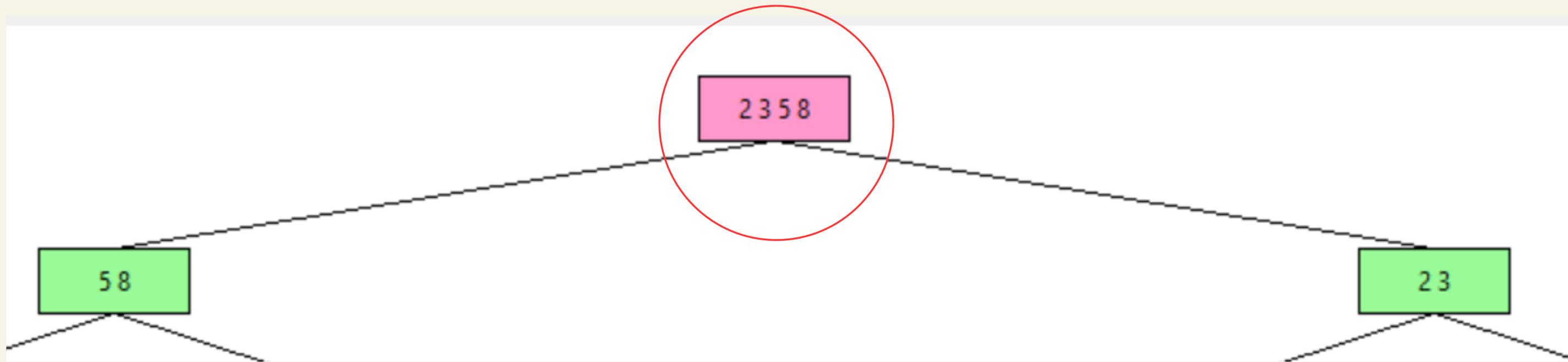


## 3 - Structures de Données et Opérations :

---

### C. Étape 3 : Résultat final

- Le tableau entièrement trié



## 3 - Structures de Données et Opérations :

---

- **Complexité du Tri par Fusion :**

### Diviser le tableau

- À chaque étape, on divise le tableau en deux moitiés
- Nombre de divisions :  $\log_2(n)$  niveaux(car on divise par 2 à chaque étape), donc profondeur de récursion =  $\log n$

### Fusionner les sous-tableaux

- Fusionner deux sous-tableaux de taille totale  $n$  nécessite de parcourir tous les éléments →  $O(n)$
- Chaque niveau de récursion effectue exactement  $n$  comparaisons/fusions

### Complexité en temps totale

- Chaque niveau :  $O(n)$
- Nombre de niveaux :  $\log n$
- Donc Meilleur, moyen et pire cas :  $O(n \log n)$

## 3 - Structures de Données et Opérations :

---

- Les algorithmes des opérations principales de tri de fusion :

1) DIVISER (*split*)

→ C'est dans la fonction *generate\_steps* que tu divises le tableau en deux

```
if left > right:  
    return []  
if left == right:  
  
    self.steps.append(('leaf', (left,right), [arr[left]]))  
    return [arr[left]]  
  
self.steps.append(('split', (left,right), arr[left:right+1].copy()))  
mid = (left+right)//2  
L = self.generate_steps(arr, left, mid)  
R = self.generate_steps(arr, mid+1, right)
```

## 3 - Structures de Données et Opérations :

---

- **Les algorithmes des opérations principales de tri de fusion :**

2) *RÉGNER* (*appel récursif*)

→ *Conquête* (*Tri récursif de chaque côté*)

*Chaque sous-tableau est trié indépendamment grâce à l'appel récursif.*

```
L = self.generate_steps(arr, left, mid)
R = self.generate_steps(arr, mid+1, right)
```

# 3 - Structures de Données et Opérations :

---

- **Les algorithmes des opérations principales de tri de fusion :**

3) COMBINER (*fusionner les deux moitiés triées*)

→ *Grâce à la fonction merge(left, right).*

```
i = j = 0
merged = []
while i < len(L) and j < len(R):

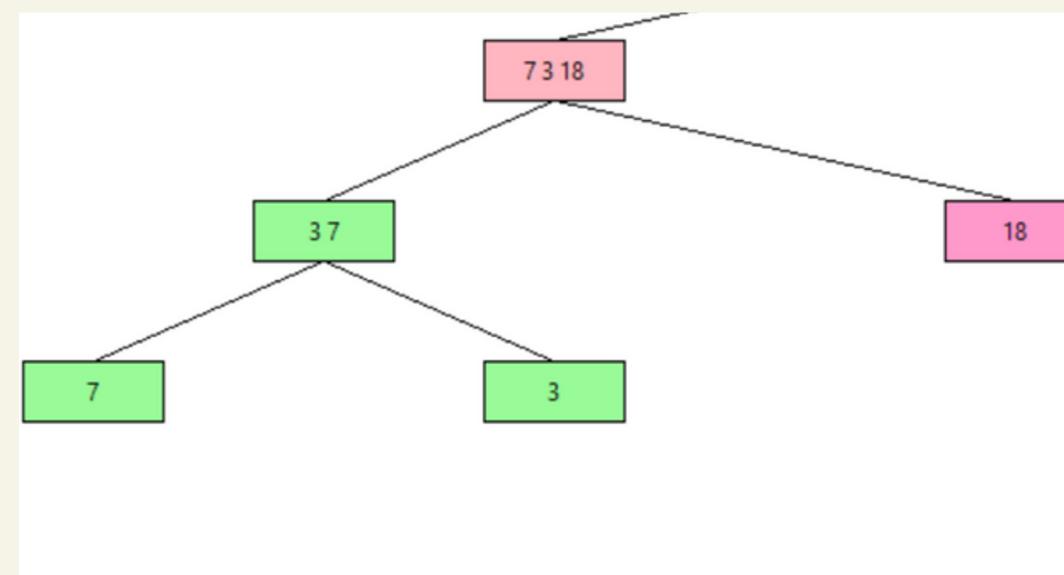
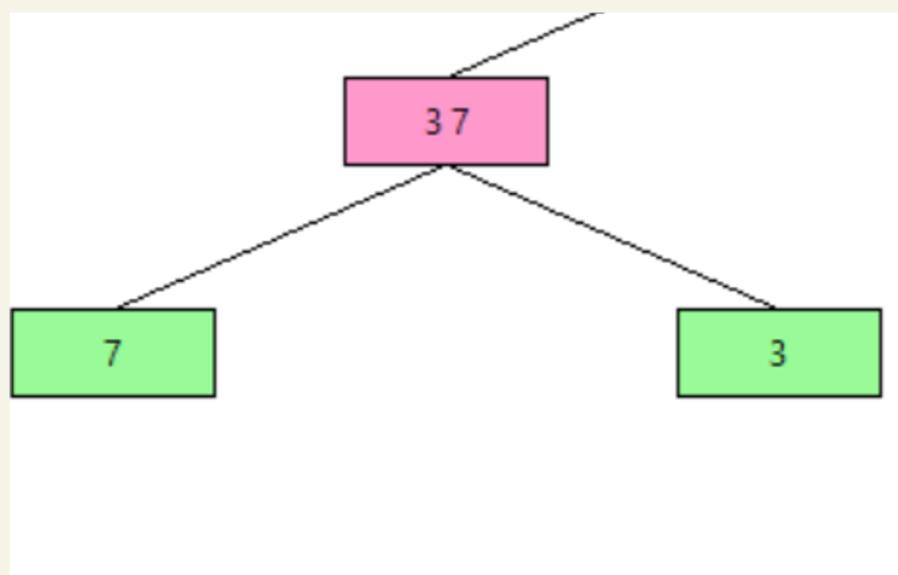
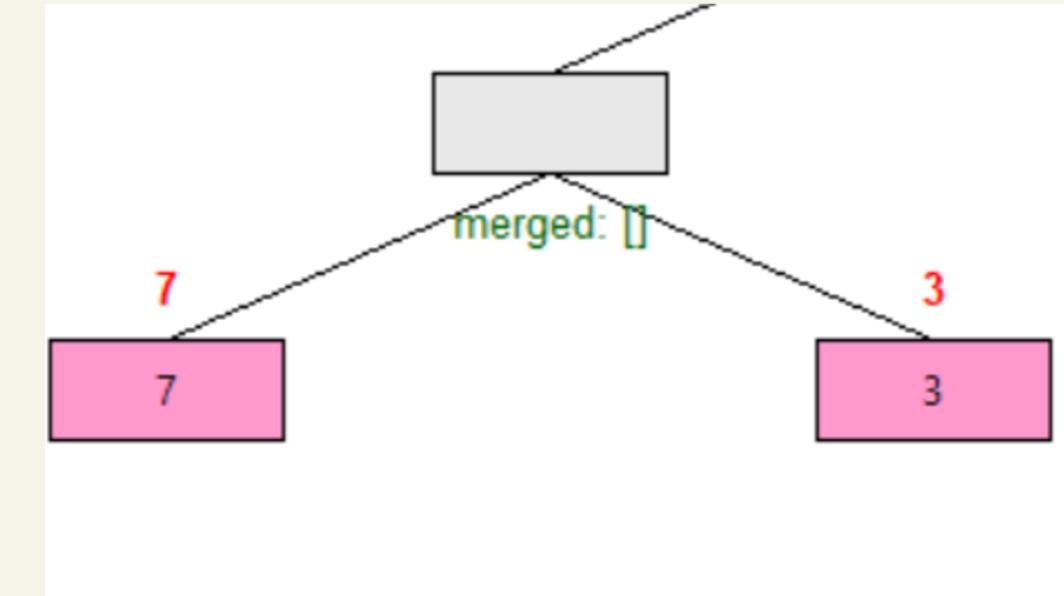
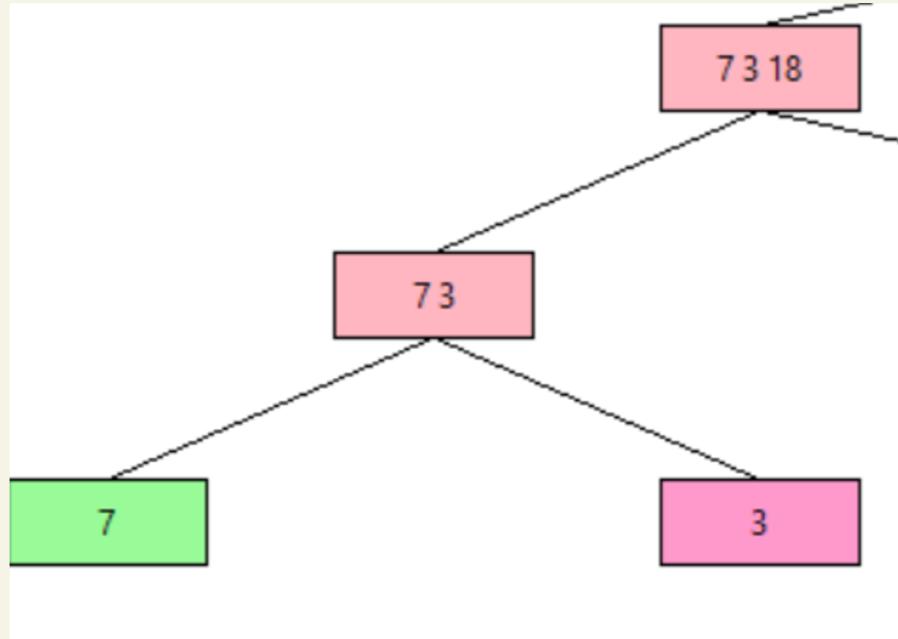
    self.steps.append(('compare', (left,right), L[i], R[j], merged.copy()))
    if L[i] <= R[j]:
        merged.append(L[i]); i += 1
    else:
        merged.append(R[j]); j += 1

    while i < len(L):                                (parameter) left: Any
        self.steps.append(('compare', (left,right), L[i], None, merged.copy()))
        merged.append(L[i]); i += 1
    while j < len(R):
        self.steps.append(('compare', (left,right), None, R[j], merged.copy()))
        merged.append(R[j]); j += 1

self.steps.append(('merge_complete', (left,right), merged.copy()))
return merged
```

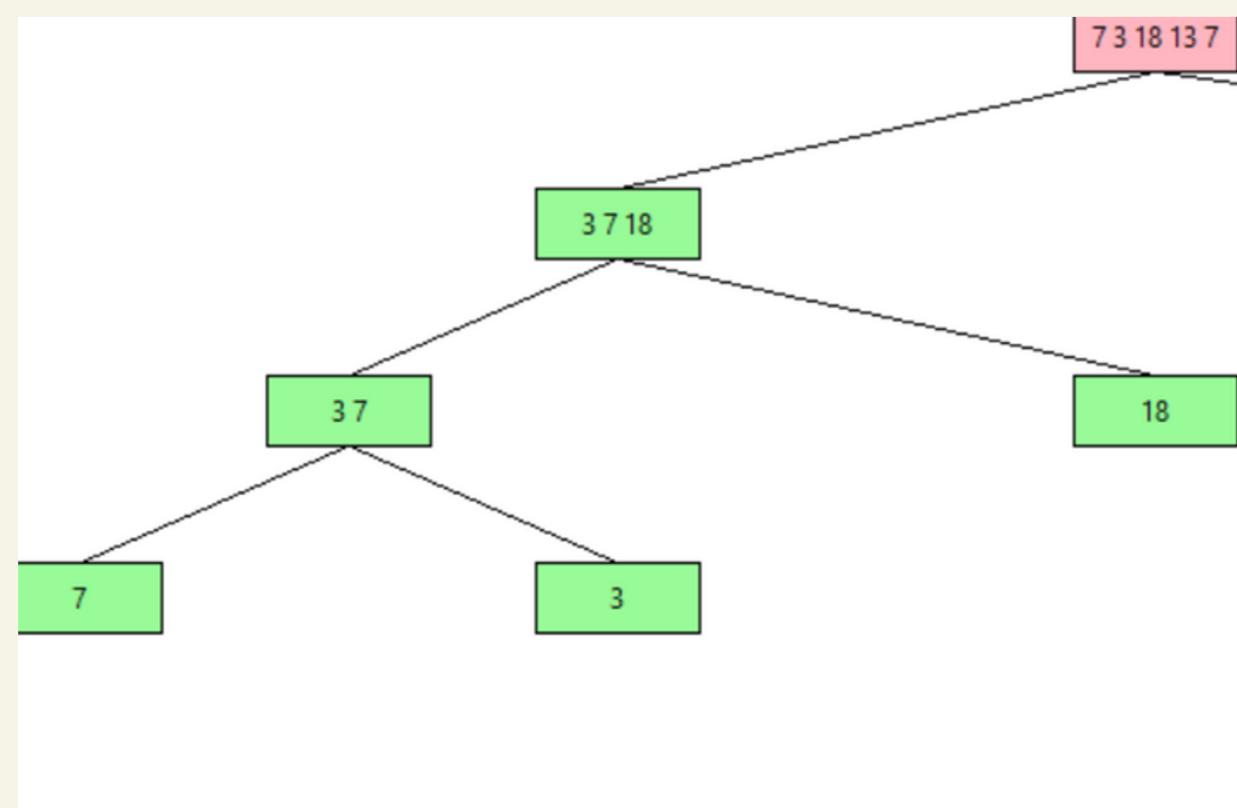
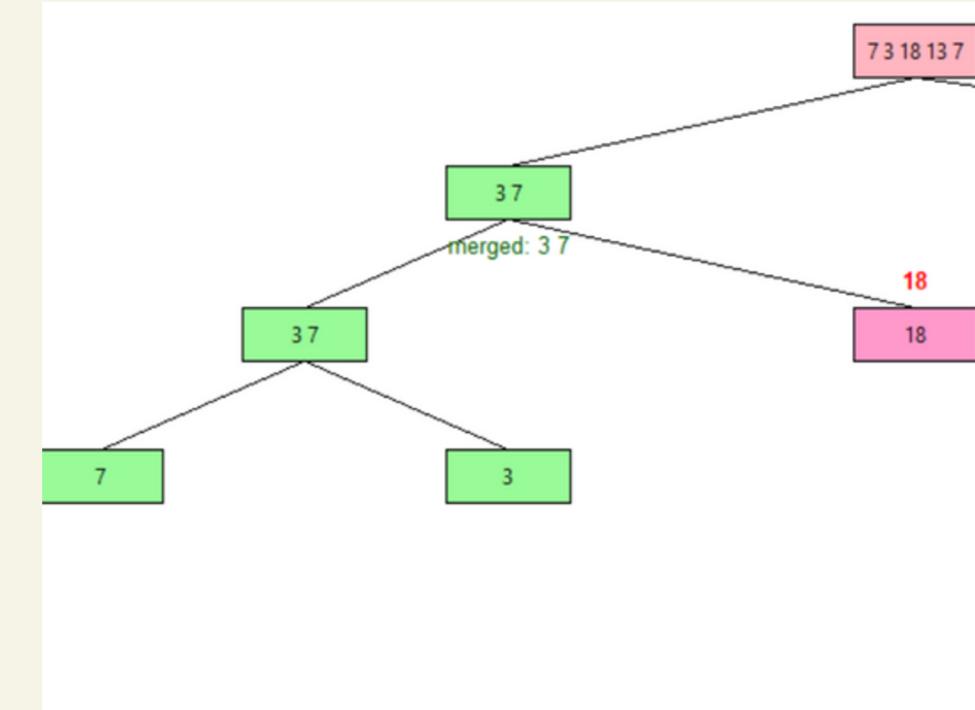
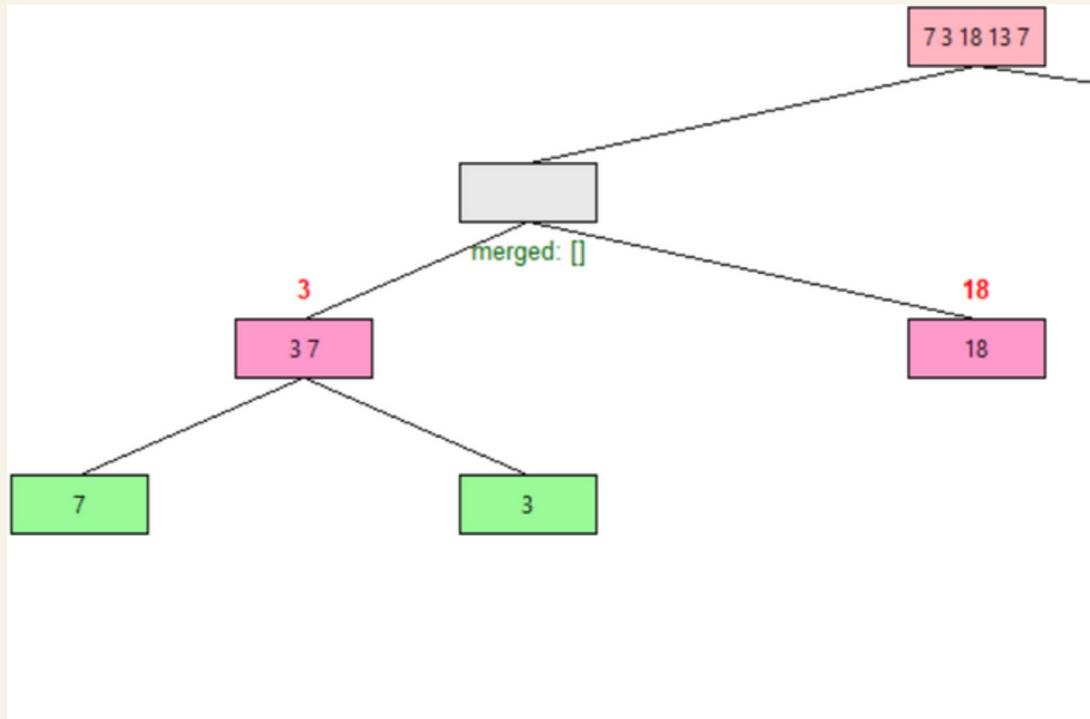
## 4 - Résultat :

### L'Exécution de l'exemple du PDF :



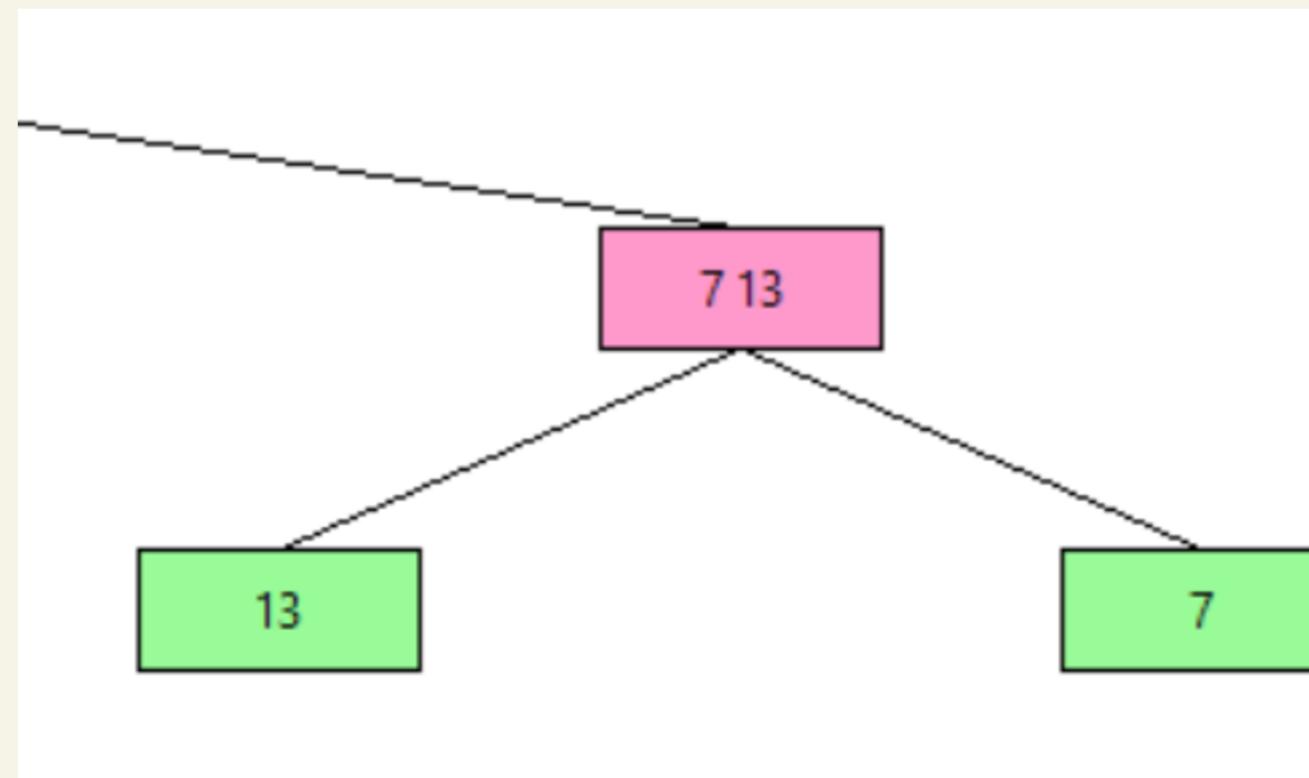
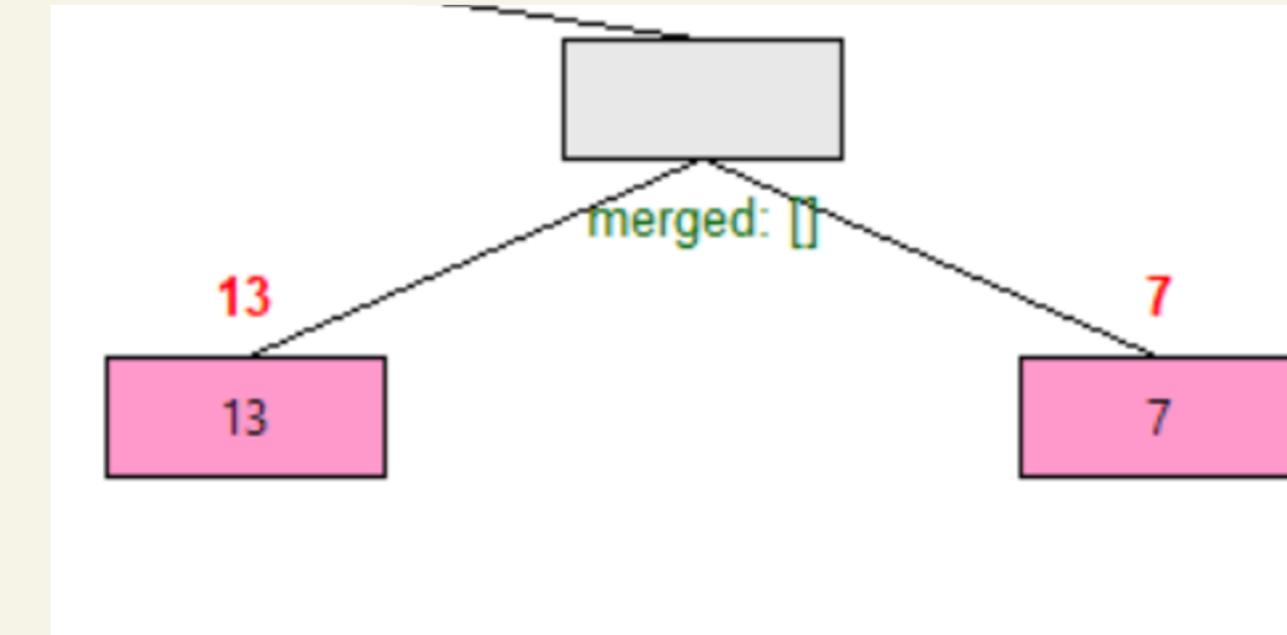
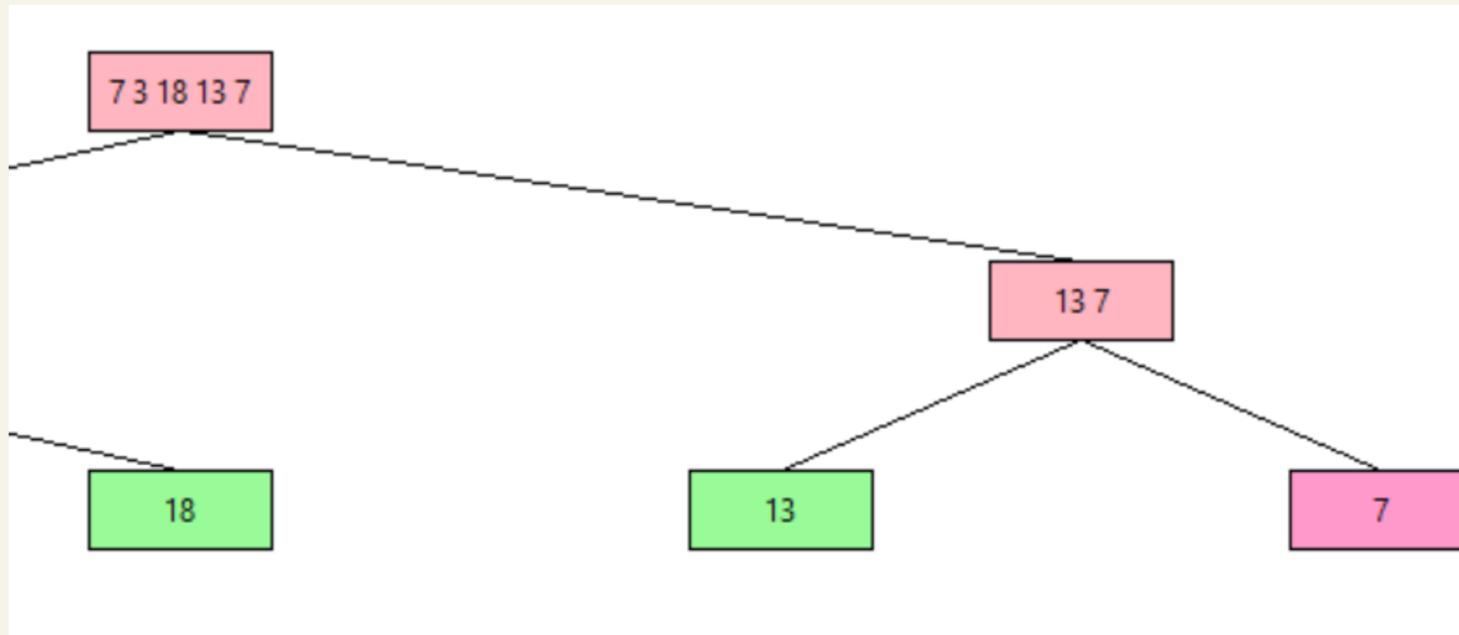
## 4 - Résultat :

### L'Exécution de l'exemple du PDF :



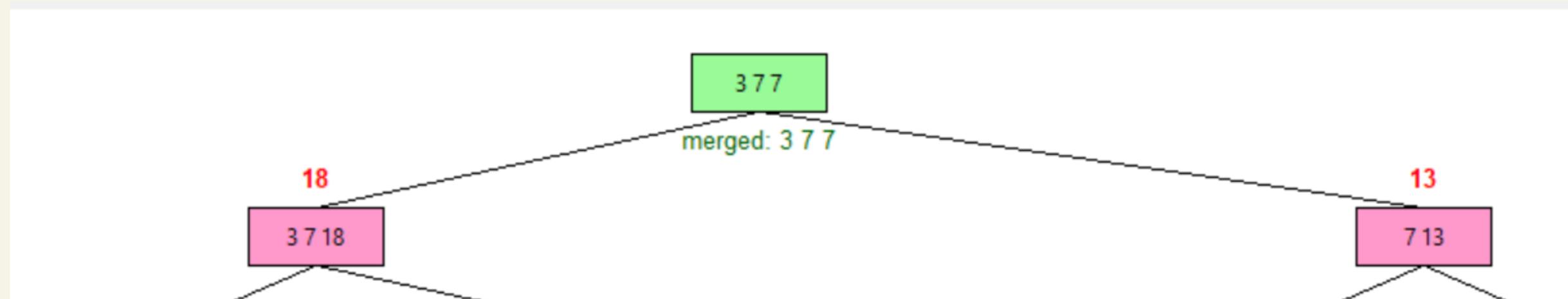
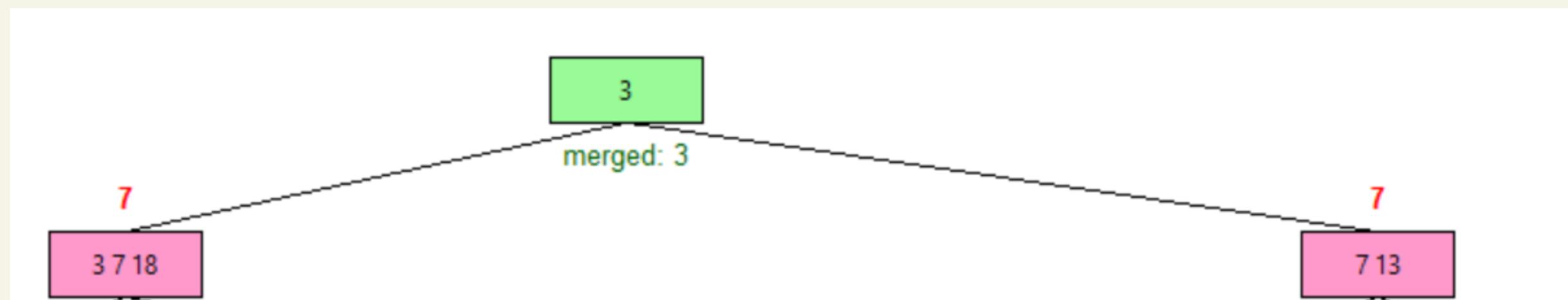
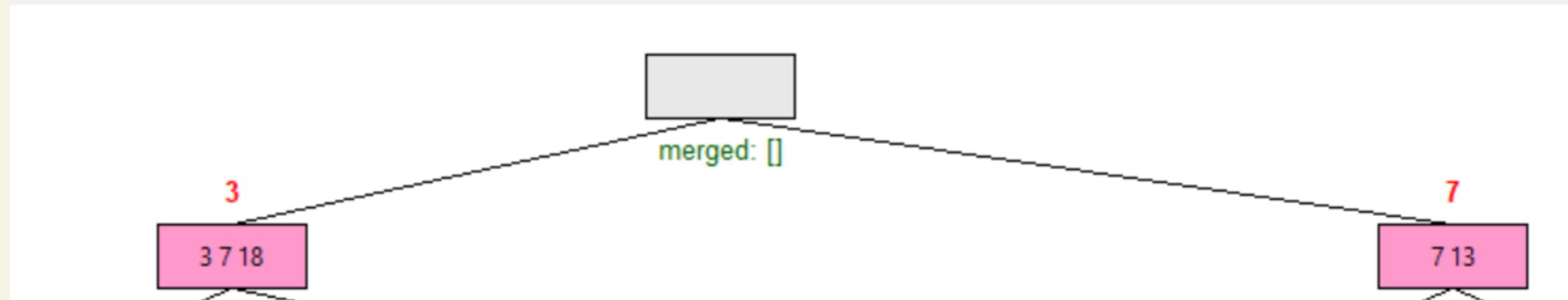
## 4 - Résultat :

### L'Exécution de l'exemple du PDF :



## 4 - Résultat :

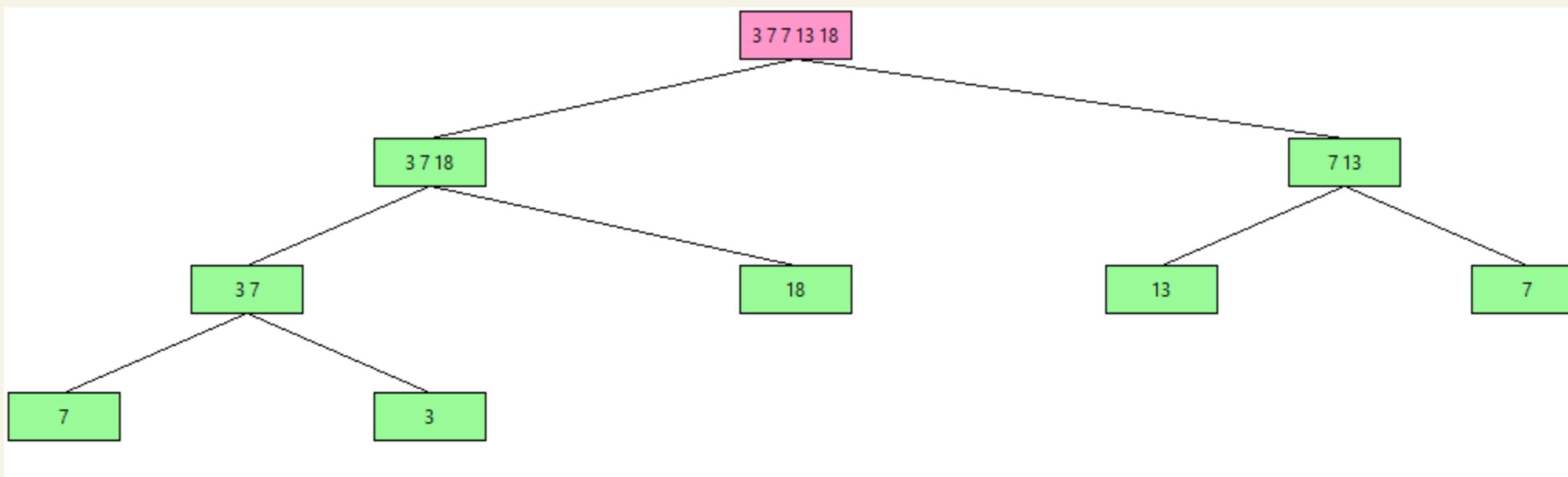
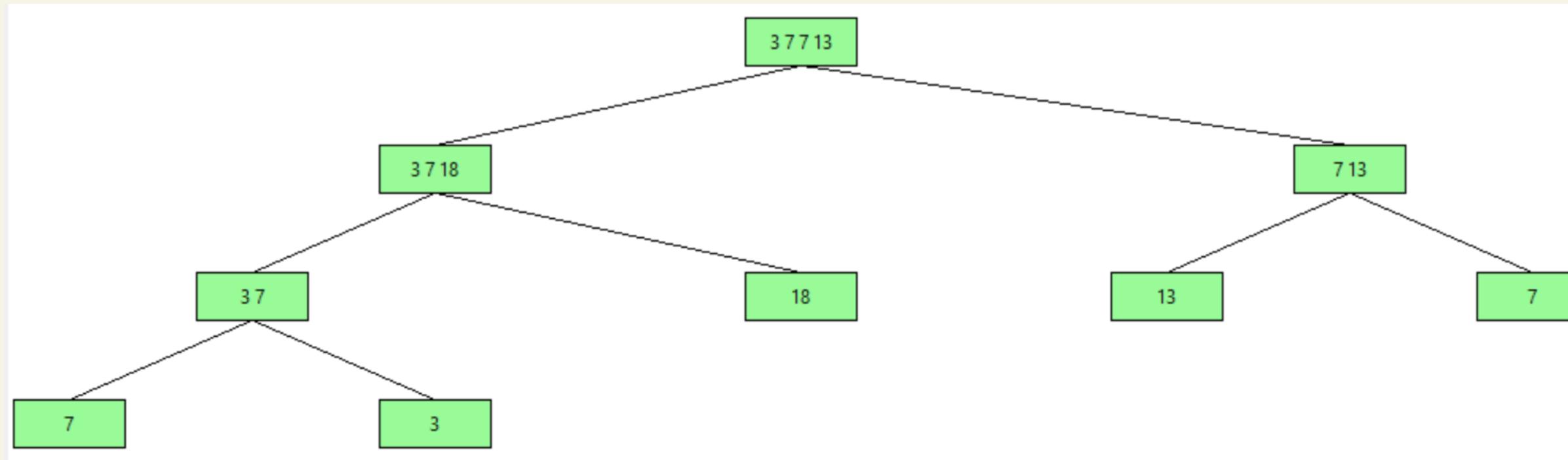
### L'Exécution de l'exemple du PDF :



## 4 - Résultat :

---

### L'Exécution de l'exemple du PDF :



# 5 - Les Avantages et Inconvénients du Tri Fusion

| Avantages  | Inconvénients   |
|--|---|
| <ul style="list-style-type: none"><li>• Complexité garantie : <math>O(n \log n)</math> dans tous les cas</li></ul> | <ul style="list-style-type: none"><li>• Nécessite un espace mémoire supplémentaire (tableau temporaire)</li></ul>     |
| <ul style="list-style-type: none"><li>• Stable : conserve l'ordre des éléments égaux</li></ul>                     | <ul style="list-style-type: none"><li>• Moins rapide que Quicksort en pratique pour les tableaux en mémoire</li></ul> |
| <ul style="list-style-type: none"><li>• Efficace pour les grandes listes ou tableaux liés</li></ul>                | <ul style="list-style-type: none"><li>• Mise en œuvre plus complexe que les tris simples</li></ul>                    |
| <ul style="list-style-type: none"><li>• Facile à paralléliser</li></ul>  | <ul style="list-style-type: none"><li>• Nombreux accès mémoire dus aux copies de sous-tableaux</li></ul>              |
| <ul style="list-style-type: none"><li>• Comportement prévisible et régulier</li></ul>                              | <ul style="list-style-type: none"><li>• Peut être moins adapté aux très petits tableaux</li></ul>                     |

*Merci Pour Votre Attention*