**I. Analyse de la Complexité Temporelle (Time Complexity Analysis)**

La complexité temporelle d'un algorithme mesure comment le temps d'exécution de cet algorithme évolue en fonction de la taille de son entrée (dans notre cas, la taille de la liste triée). On exprime cette complexité à l'aide de la notation "Grand O" (O).

1. **Recherche Binaire : Complexité O(log n)**
   * L'algorithme de recherche binaire divise l'espace de recherche par deux à chaque étape de la boucle while.
   * Par exemple :
     + Si la liste a 16 éléments, après une comparaison il n'y en a plus que 8 à chercher
     + Encore une comparaison => 4
     + Encore une comparaison => 2
     + Encore une comparaison => 1
   * Le nombre d'étapes nécessaires pour trouver la cible (ou confirmer son absence) est donc proportionnel au logarithme (en base 2) du nombre d'éléments (n) de la liste.
   * Cela s'exprime par la notation O(log n).
     + *English:* The algorithm divides the search space in half at every step of the while loop. The number of steps is proportional to the logarithm (base 2) of the number of elements, resulting in O(log n) time complexity.
   * *Key words:* *O(log n), logarithmic time, divide and conquer*
2. **Pourquoi Logarithmique ?**
   * Si la taille de l'entrée est multipliée par 2, le temps d'exécution n'augmente que d'une unité.
   * Autrement dit : si vous doublez la taille de la liste, la recherche prendra une seule étape de plus.
   * C'est ce qui rend cet algorithme très efficace pour les grandes listes.
     + *English:* If the input size doubles, the execution time only increases by one unit. This makes the algorithm very efficient for large lists.
     + *Key words:* *logarithmic scaling, efficient for large lists*
3. **Comparaison avec la recherche linéaire : Complexité O(n)**
   * La recherche linéaire, qui consiste à parcourir tous les éléments de la liste un par un, a une complexité temporelle O(n).
   * Dans ce cas, le temps d'exécution est directement proportionnel à la taille de la liste.
   * Si on double la taille de la liste, la recherche linéaire prend deux fois plus de temps.
     + *English:* Linear search checks each element one by one and has O(n) time complexity. The execution time is directly proportional to the size of the list.
   * *Key words:* *O(n), linear time, sequential search*
4. **Comparaison en pratique :**
   * Pour une liste de 1024 éléments :
     + Recherche binaire : environ 10 comparaisons (log₂ 1024 = 10).
     + Recherche linéaire : en moyenne 512 comparaisons. (n/2)
   * Pour une liste de 1 million d'éléments :
     + Recherche binaire : environ 20 comparaisons (log₂ 1000000 = 20)
     + Recherche linéaire : en moyenne 500 000 comparaisons
   * On voit ici l'intérêt du logarithmique pour les grandes listes.
   * *English:* These examples demonstrate the advantage of binary search for large lists.
   * *Key words:* *practical comparison, large datasets, efficiency*

**II. Documentation du Travail**

Voici comment documenter notre travail, en suivant la structure du rapport :

* **Problem Representation (Représentation du Problème):**
  + Nous avons décrit le problème en détail : recherche d'une cible dans une liste triée.
  + Nous avons fourni une représentation visuelle : la liste et le concept de division à chaque étape.
* **Solution :**
  + Nous avons décrit l'algorithme de recherche binaire.
  + Nous avons fourni l'extrait de code Python (la fonction binary\_search).
* **Results (Résultats) :**
  + Nous avons montré des exemples d'entrée/sortie de l'algorithme avec différentes cibles et listes triées.
* **Conclusion :**
  + **Efficacité :** L'algorithme de recherche binaire a une complexité temporelle de O(log n). Il est donc très efficace pour la recherche dans de grandes listes triées. Nous avons mentionné sa nature "diviser pour régner".
  + **Comparaison :** L'algorithme de recherche binaire est beaucoup plus rapide que la recherche linéaire (O(n)) pour les grandes listes.
  + **Résumé :** Un bon choix d'algorithme dans beaucoup d'applications.

**Mots clés additionnels pour documenter notre rapport (Key words):**

* **Complexity analysis (Analyse de complexité)**
* **Time complexity (Complexité temporelle)**
* **Big O notation (Notation Grand O)**
* **Logarithmic time (Temps logarithmique)**
* **Linear time (Temps linéaire)**
* **Divide and conquer (Diviser pour régner)**

**Documentation de l'Optimisation du Code**

Nous avons également documenté les améliorations apportées au code :

* Ajout d'un exemple concret dans get\_sorted\_list\_from\_user.
* Meilleure gestion des erreurs dans get\_sorted\_list\_from\_user.
* Sauts de ligne pour plus de lisibilité.