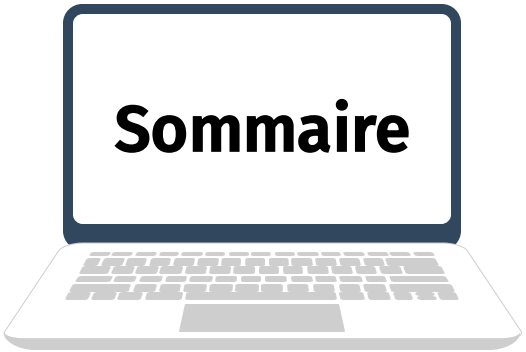
Algorithme de tri et algorithme de recherche

Présenté par:

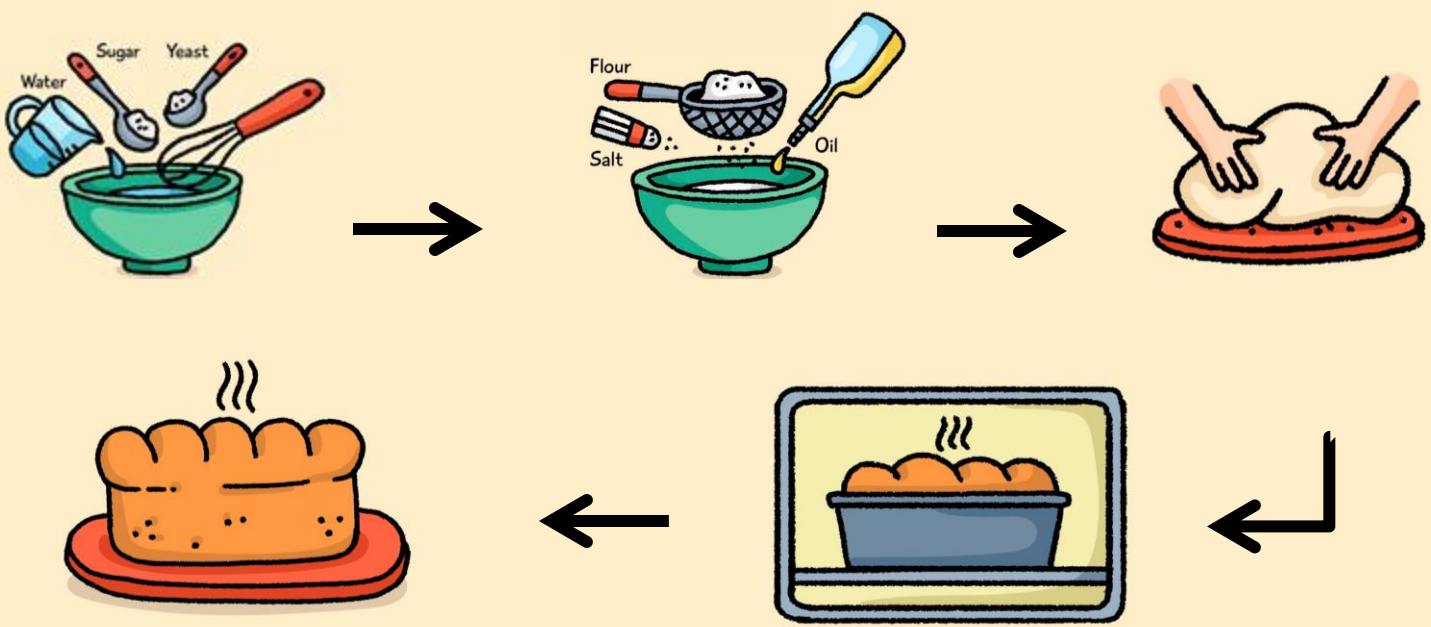
BAAZIZ Wissal

Sommaire

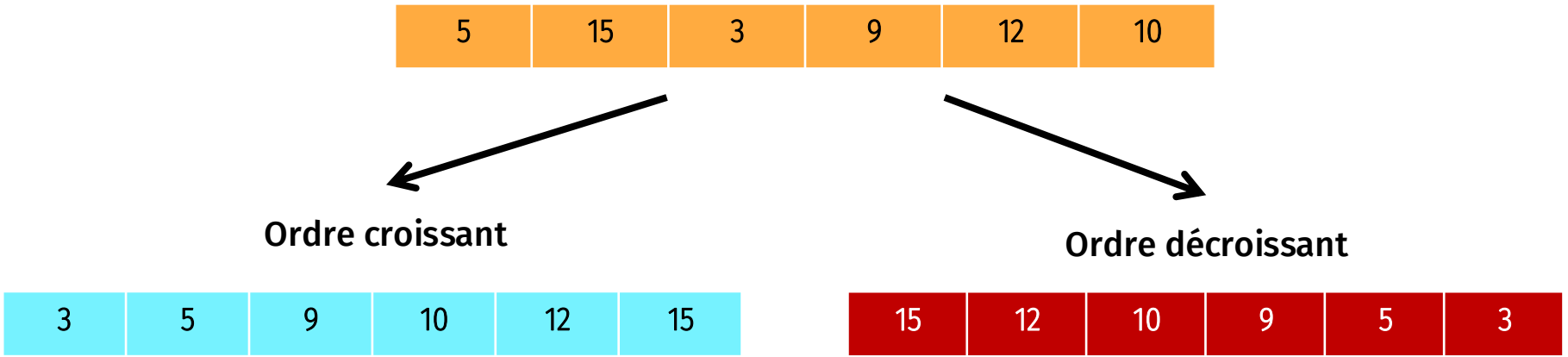


Introduction :

1. C’est quoi un algorithme ?



Définition : Algorithme de tri



Définition : Algorithme de tri

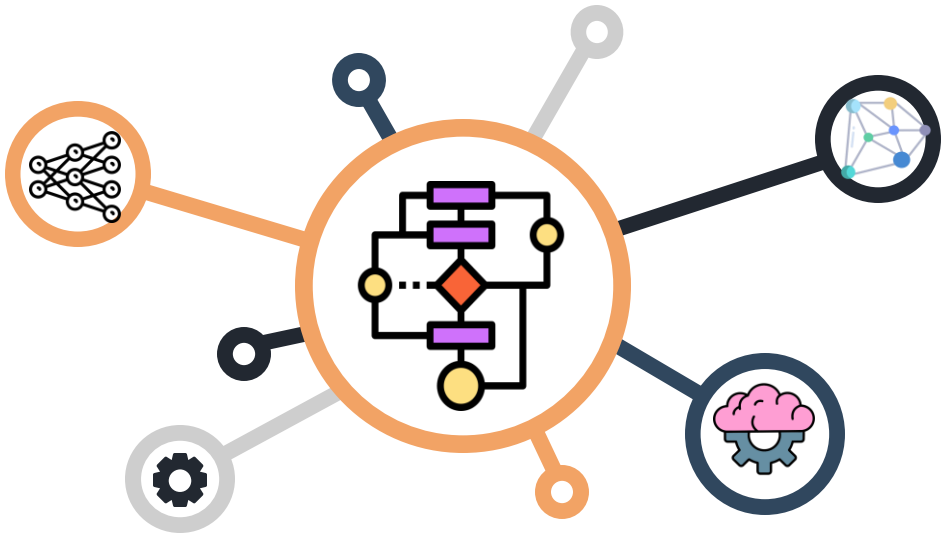
Un algorithme de tri est une séquence d'instructions permettant d'organiser des données dans un ordre spécifique, comme croissant ou décroissant. Ces algorithmes sont largement utilisés en informatique pour faciliter la recherche et la gestion de données. Il existe de nombreuses méthodes de tri, chacune ayant ses propres avantages et inconvénients.

Types d’algorithmes de tri

Tri par tas

Tri par comptage

Tri à bulles



Tri rapide

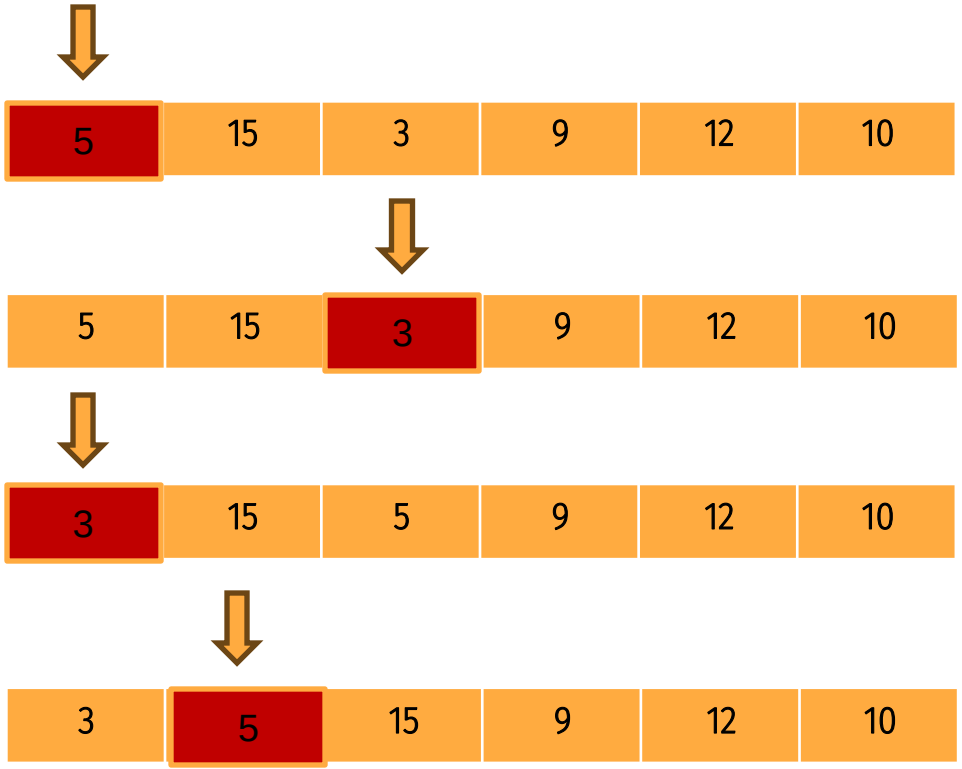
Tri par sélection

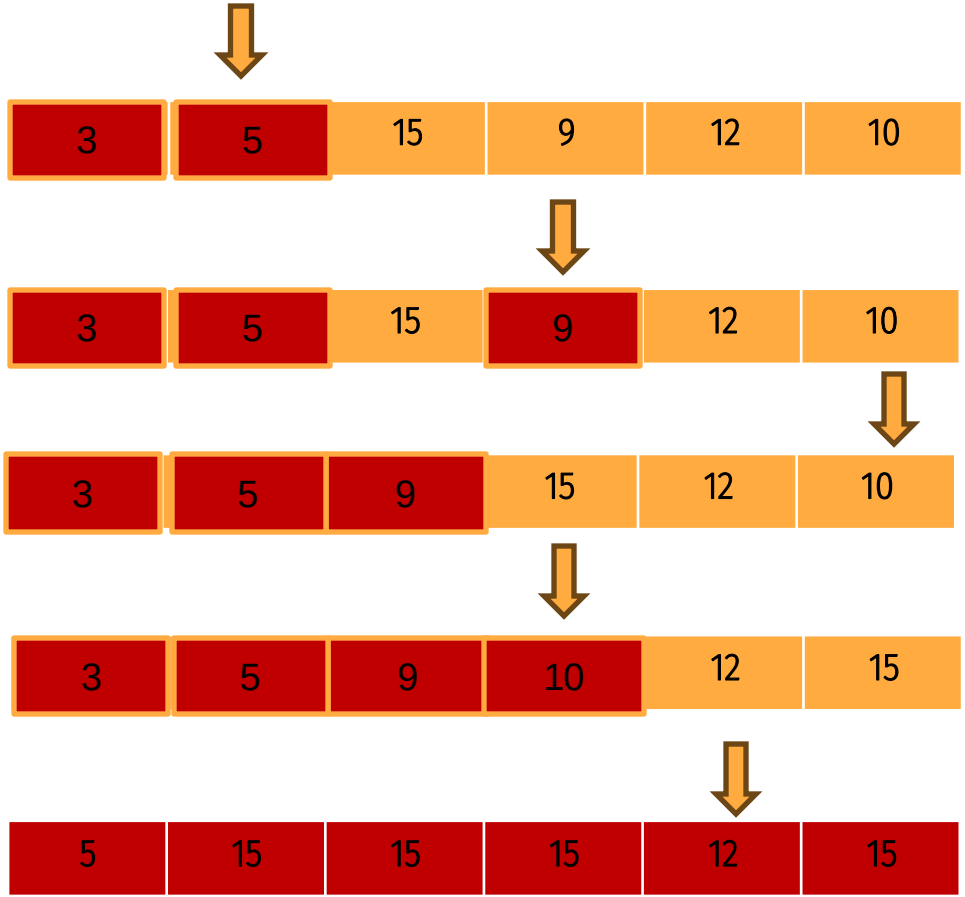
Tri fusion

Tri par peigne

Tri par Insertion

Tri par selection




3
5
15
9
12
10
3
5
15
9
12
10
3
5
9
15
12
10
3
5
9
10
12
15
5
15
15
15
12
15

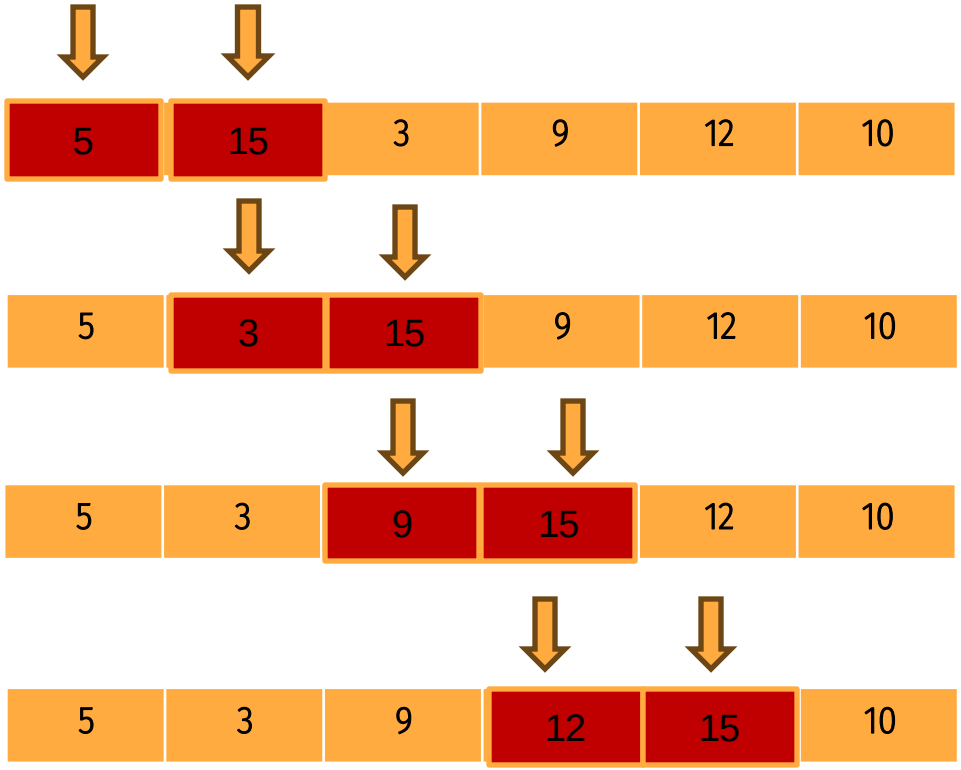
Définition : tri par selection

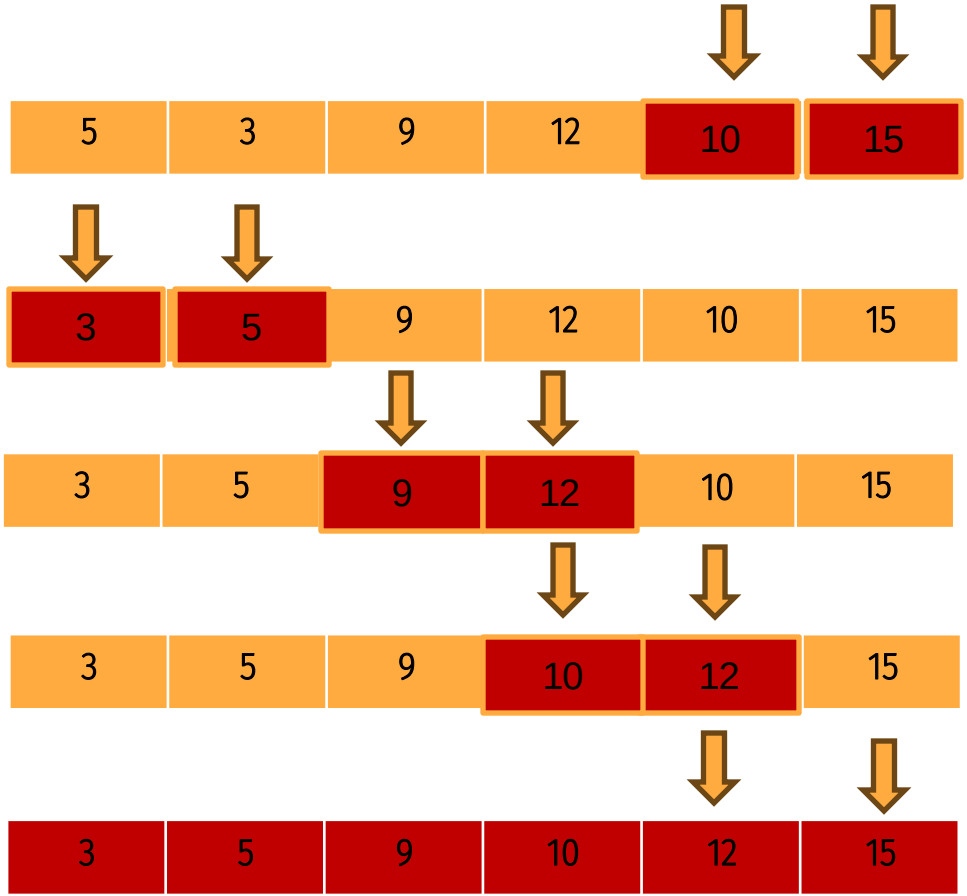
Le tri par sélection (ou tri par extraction) est un algorithme de tri par comparaison. Cet algorithme est simple, mais considéré comme inefficace car il s'exécute en temps quadratique en le nombre d'éléments à trier, et non en temps pseudo linéaire.

Pseudo code:

```
procédure tri_selection(tableau t)
      n ← longueur(t)
      pour i de 0 à n - 2
          min ← i
          pour j de i + 1 à n - 1
              si t[j] < t[min], alors min ← j
          fin pour
          si min ≠ i, alors échanger t[i] et t[min]
      fin pour
  fin procédure
```

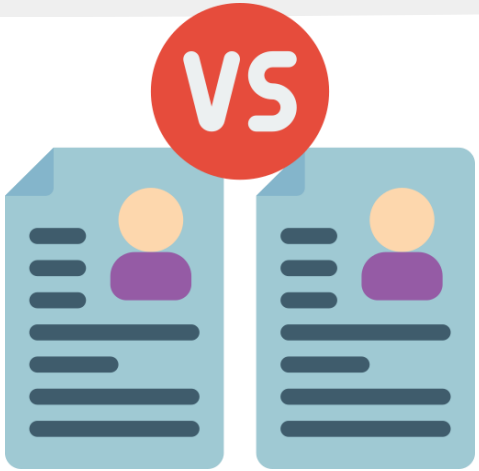
Tri a bulles




5 3 9 12 10 15
3 5 9 12 10 15
3 5 9 12 10 15
3 5 9 10 12 15
3 5 9 10 12 15

Définition : tri a bulles

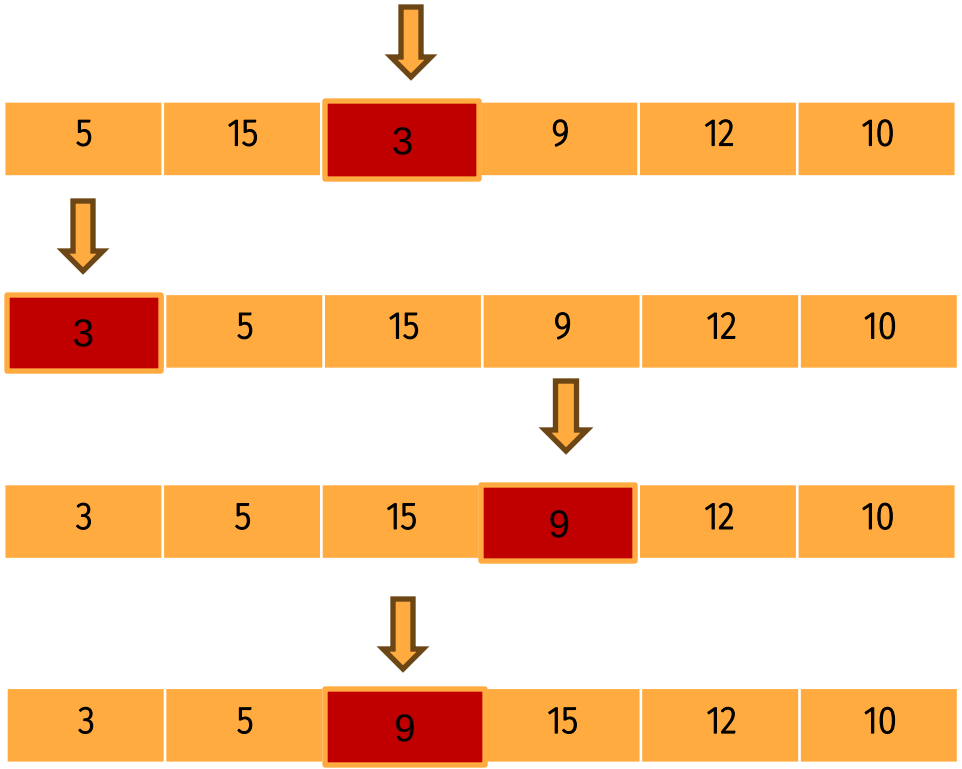
Le **tri à bulles** ou **tri par propagation**[1] est un algorithme de tri. Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés. Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

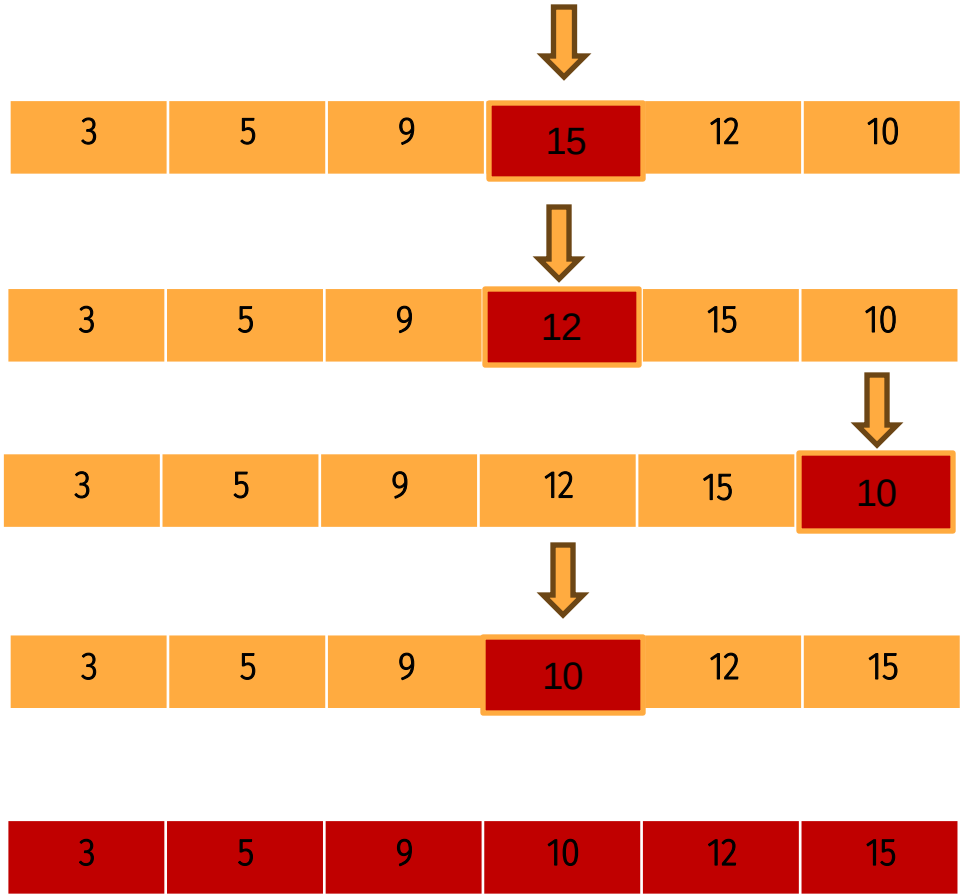


Pseudo code:

```
tri_à_bulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               (T[j+1], T[j]) ← (T[j], T[j+1])
```

Tri par insertion




3 5 9 15 12 10
3 5 9 12 15 10
3 5 9 12 15 10
3 5 9 10 12 15
3 5 9 10 12 15

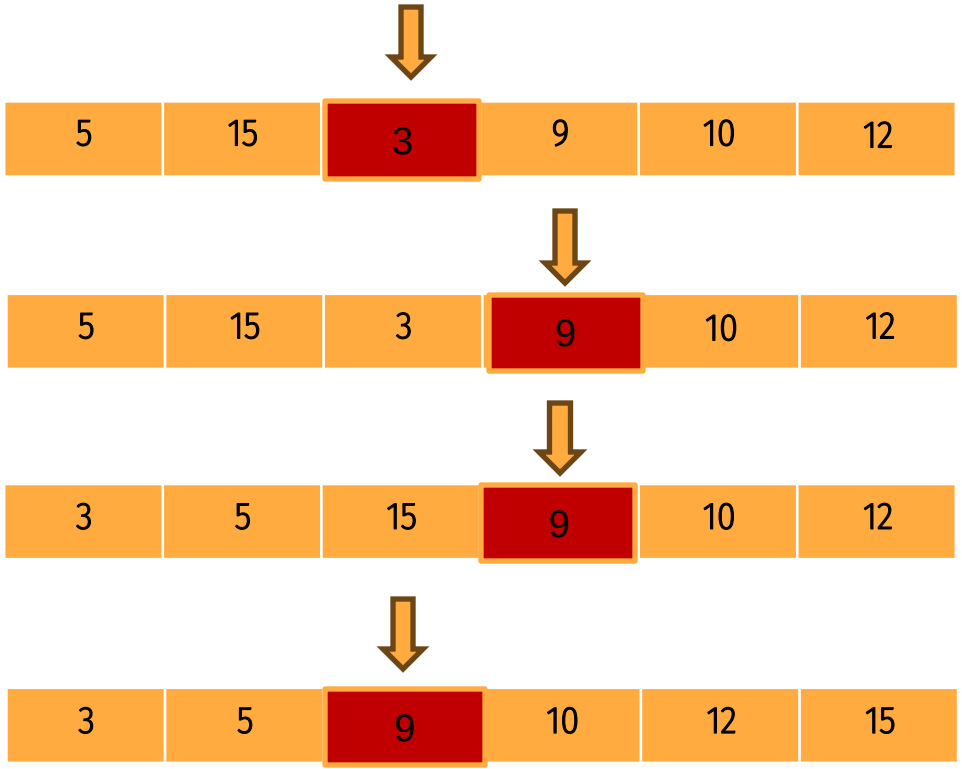
Pseudo code:

```
procédure tri_insertion(tableau T)

    pour i de 1 à taille(T) - 1

        # mémoriser T[i] dans x
        x ← T[i]

        # décaler les éléments T[0]..T[i-1] qui sont plus grands que x, en partant de T[i-1]
        j ← i
        tant que j > 0 et T[j - 1] > x
            T[j] ← T[j - 1]
            j ← j - 1

        # placer x dans le "trou" laissé par le décalage
        T[j] ← x
```

Définition : tri par insertion

En informatique, le **tri par insertion** est un algorithme de tri classique. La plupart des personnes l'utilisent naturellement pour trier des cartes à jouer[1].
En général, le tri par insertion est beaucoup plus lent que d'autres algorithmes comme le tri rapide (ou *quicksort*) et le tri fusion pour traiter de grandes séquences, car sa complexité asymptotique est quadratique.
Le tri par insertion est cependant considéré comme l'algorithme le plus efficace sur des entrées de petite taille. Il est aussi efficace lorsque les données sont déjà presque triées. Pour ces raisons, il est utilisé en pratique en combinaison avec d'autres méthodes comme le tri rapide.

Tri Rapide



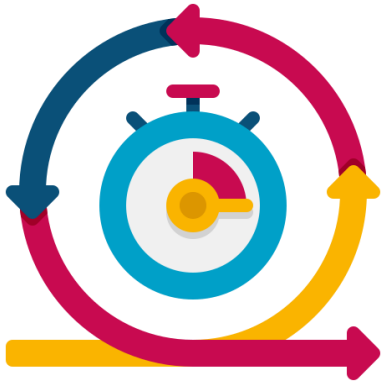
Pseudo code:

```
Fonction tri_rapide(tableau, debut, fin)
    Si debut < fin
        // Partitionne le tableau et retourne l'indice du pivot
        pivot_index <- partitionner(tableau, debut, fin)

        // Trie récursivement les sous-tableaux de gauche et de droite du pivot
        tri_rapide(tableau, debut, pivot_index - 1)
        tri_rapide(tableau, pivot_index + 1, fin)


Fonction partitionner(tableau, debut, fin)
    // Choisissez un élément pivot (par exemple, le dernier élément)
    pivot <- tableau[fin]
```

```
// Initialisez l'indice du pivot à la position de départ
  pivot_index <- debut

  // Parcourez le tableau et déplacez les éléments plus petits que le pivot à gauche
  Pour i de debut à fin - 1
      Si tableau[i] <= pivot
          Échanger tableau[i] et tableau[pivot_index]
          pivot_index <- pivot_index + 1

  // Placez le pivot à sa position finale en échangeant avec l'élément à l'indice pivot_index
  Échanger tableau[fin] et tableau[pivot_index]

  Retourner pivot_index
```

Définition : tri Rapide

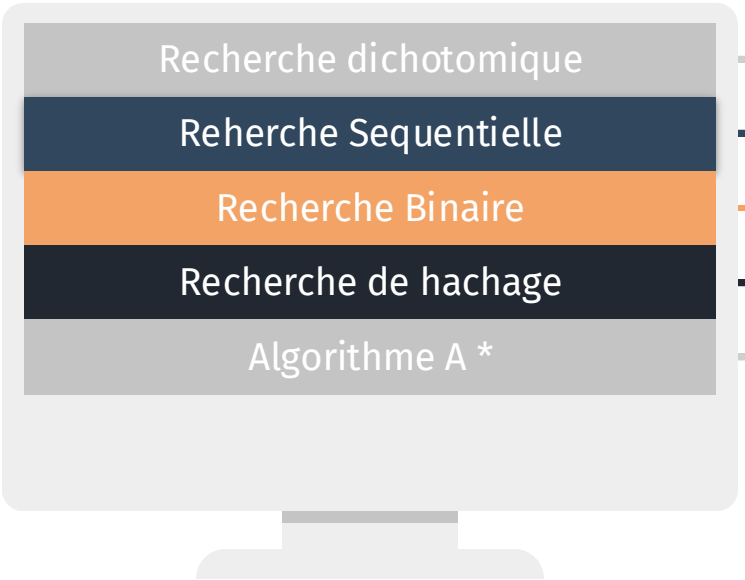
L'algorithme de tri rapide (QuickSort en anglais) est un algorithme de tri efficace basé sur la méthode de "diviser pour régner". Il fonctionne en sélectionnant un élément pivot dans le tableau et en partitionnant le tableau de manière à placer tous les éléments plus petits que le pivot à gauche et tous les éléments plus grands à droite. Ensuite, il récursivement trie les sous-tableaux de gauche et de droite.



Définition : Algorithme de Recherche

En informatique, un algorithme de recherche est un type d'algorithme qui, pour un domaine, un problème de ce domaine et des critères donnés, retourne en résultat un ensemble de solutions répondant au problème. Supposons que l'ensemble de ses entrées soit divisible en sous-ensemble, par rapport à un critère donné, qui peut être, par exemple, une relation d'ordre. De façon générale, un tel algorithme vérifie un certain nombre de ces entrées et retourne en sortie une ou plusieurs des entrées visées. L'ensemble de toutes les solutions potentielles dans le domaine est appelé espace de recherche.

Types d’algorithmes de recherche



La recherche dichotomique est un algorithme de recherche efficace qui divise continuellement l'espace de recherche par deux jusqu'à trouver la solution.

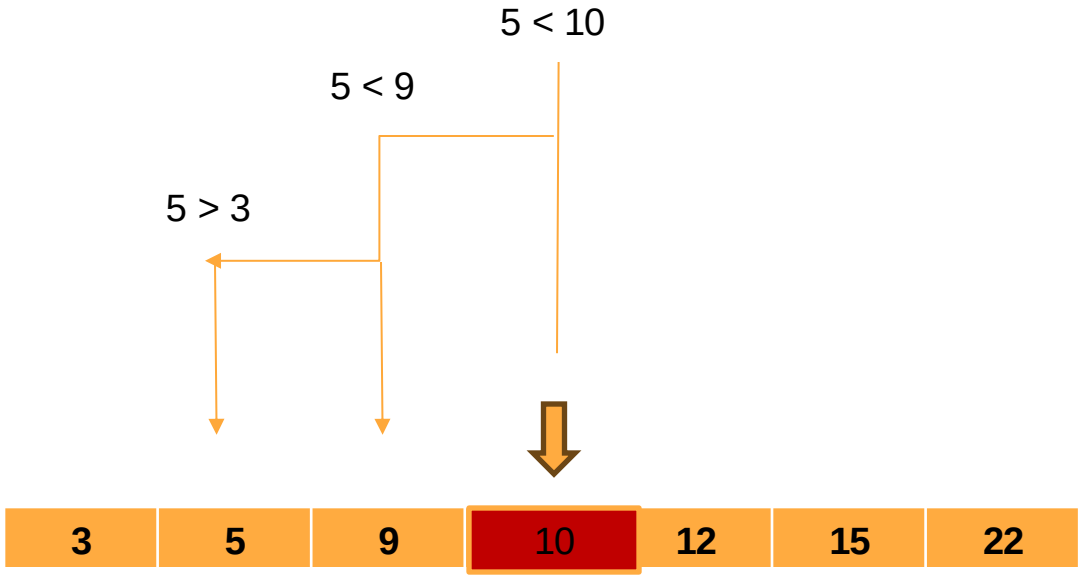
La recherche séquentielle parcourt chaque élément de la liste un par un jusqu'à trouver la cible recherchée.

La recherche binaire est un algorithme de recherche qui divise de manière répétée une liste triée par deux pour trouver efficacement un élément spécifique.

La recherche de hachage est une technique qui associe une clé à une valeur dans une table de hachage pour accéder rapidement à cette valeur, en évitant une recherche séquentielle.

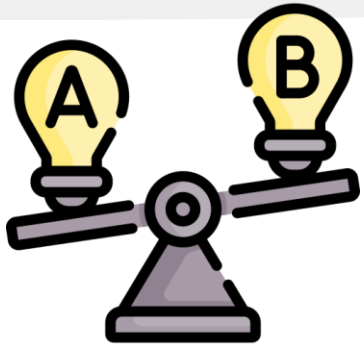
L'algorithme A* est une méthode de recherche qui trouve le chemin optimal dans un graphe en utilisant une combinaison de coût réel et d'estimation heuristique pour guider la recherche.

Recherche dichotomique



Définition : Recherche dichotomique

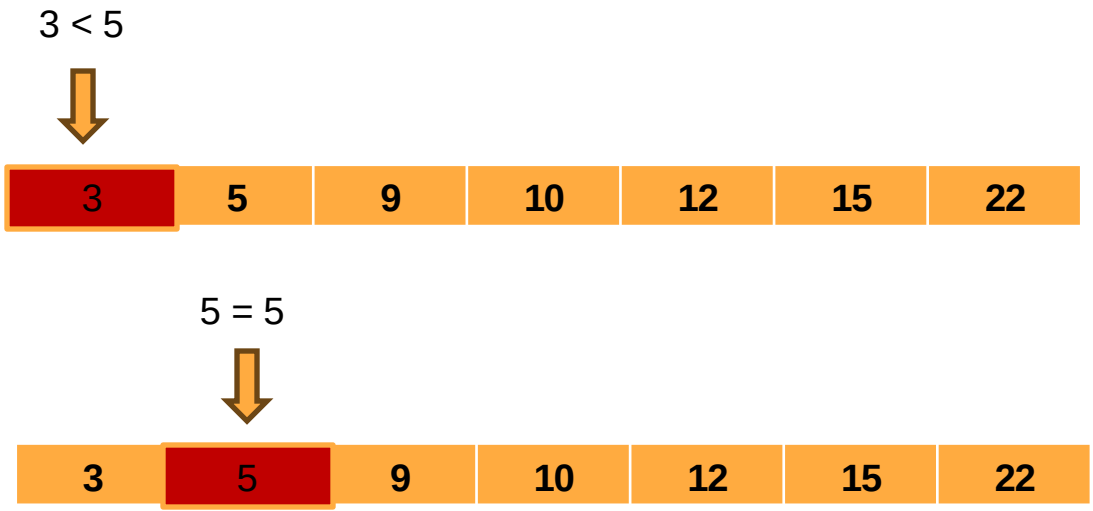
La recherche dichotomique, ou recherche par dichotomie1 est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié. Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.
Le nombre d'itérations de la procédure, c'est-à-dire le nombre de comparaisons, est logarithmique en la taille du tableau. Il y a de nombreuses structures spécialisées (comme les tables de hachage) qui peuvent être recherchées plus rapidement, mais la recherche dichotomique s'applique à plus de problèmes.



Pseudo code:

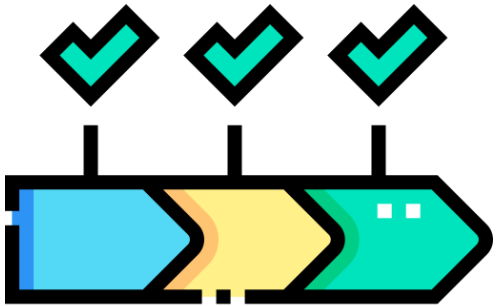
```
 def recherche_dichotomique_recursive2(element, liste_triee):

if len(liste_triee)==1 :
    return 0
m = len(liste_triee)//2
if liste_triee[m] == element :
    return m
elif liste_triee[m] > element :
    return recherche_dichotomique_recursive2(element, liste_triee[:m])
else :
    return m + recherche_dichotomique_recursive2(element, liste_triee[m:])
```

Recherche Sequentielle



Définition : Recherche sequentielle

La recherche séquentielle ou recherche linéaire est un algorithme pour trouver une valeur dans une liste. Elle consiste simplement à considérer les éléments de la liste les uns après les autres, jusqu'à ce que l'élément soit trouvé, ou que toutes les cases aient été lues. Elle est aussi appelée recherche par balayage.



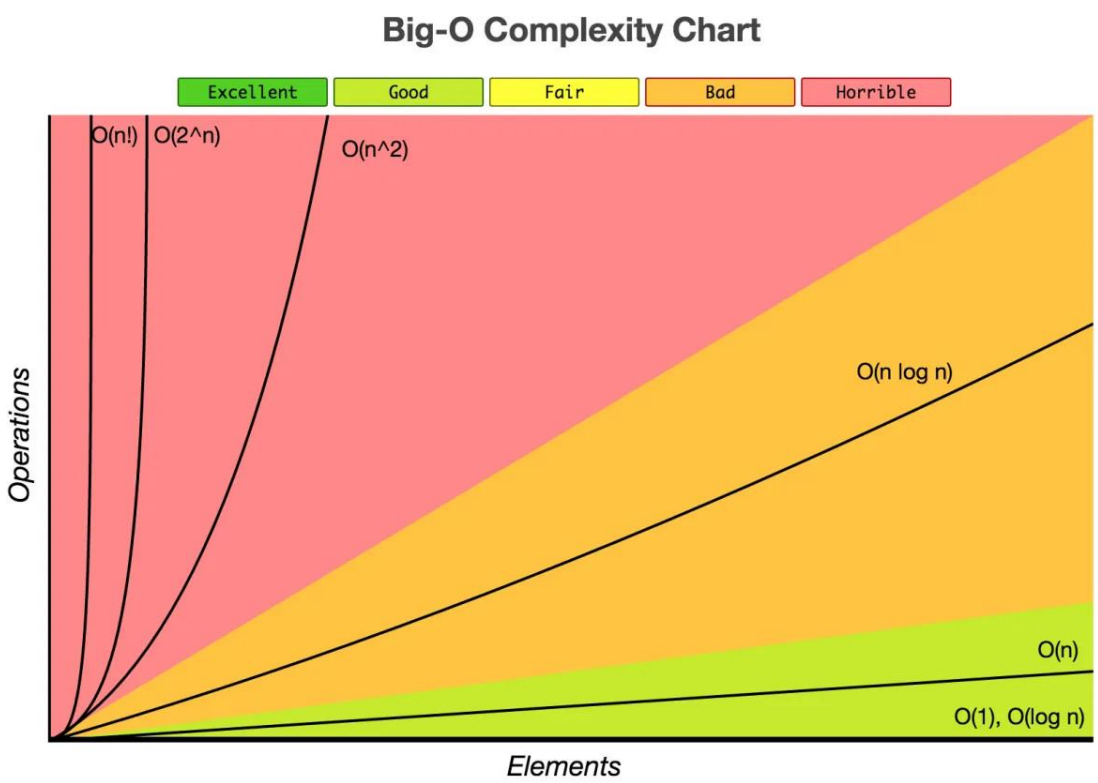
Pseudo code:

```
Fonction recherche_sequentielle(liste, element_recherche)
    Pour chaque élément dans la liste
        Si élément == element_recherche
            Retourner l'indice de l'élément
    Fin Pour
    Retourner -1  // Retourne -1 si l'élément n'est pas trouvé dans la liste
Fin Fonction
```

Big O

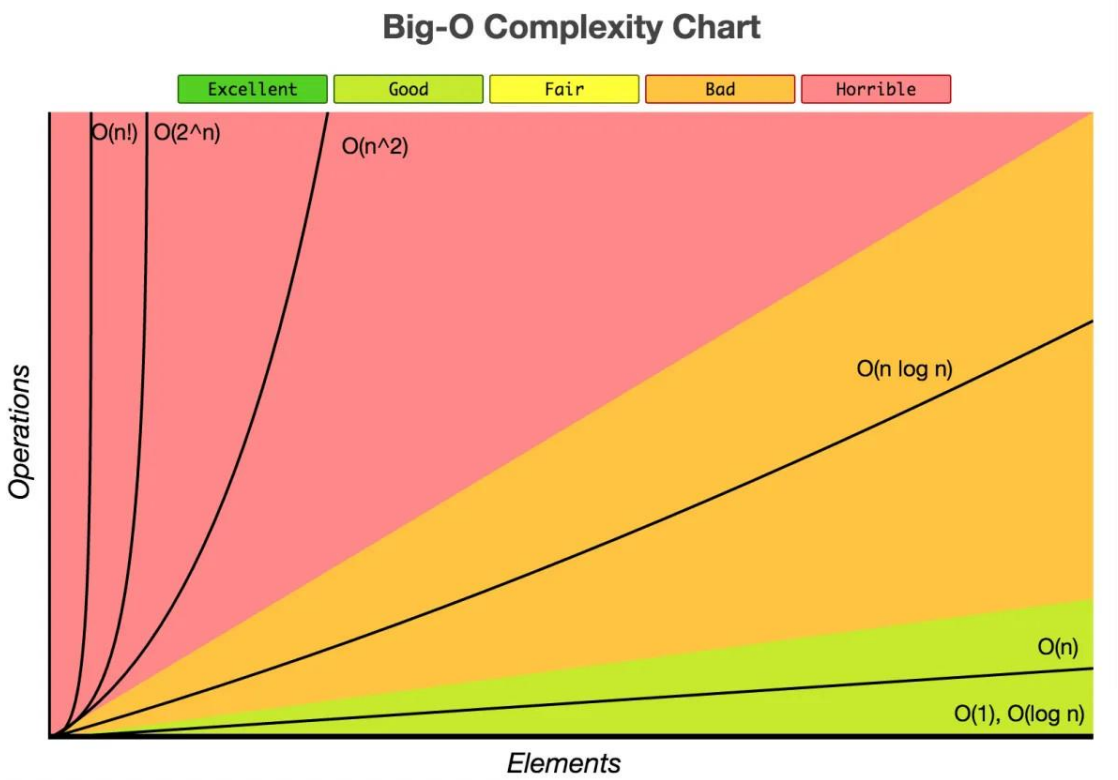
La notation "Big O" est un concept couramment utilisé en informatique pour analyser la complexité des algorithmes en termes de temps d'exécution ou d'utilisation de la mémoire.
La notation "Big O" est généralement utilisée pour décrire la limite supérieure de la croissance d'une fonction en fonction de la taille de l'entrée. Par exemple, si l'on dit qu'un algorithme a une complexité temporelle en O(n), cela signifie que le temps d'exécution de l'algorithme augmente linéairement avec la taille de l'entrée (n). De même, si l'on dit qu'un algorithme a une complexité spatiale en O(1), cela signifie que l'utilisation de la mémoire de l'algorithme reste constante indépendamment de la taille de l'entrée.

Big O



1. Tri par insertion (Insertion Sort) :
   1. Complexité temporelle : O(n^2) dans le pire des cas
   2. Complexité spatiale : O(1)
2. Tri à bulles (Bubble Sort) :
   1. Complexité temporelle : O(n^2) dans le pire des cas
   2. Complexité spatiale : O(1)
3. Tri par sélection (Selection Sort) :
   1. Complexité temporelle : O(n^2) dans le pire des cas
   2. Complexité spatiale : O(1)
4. Tri rapide (Quick Sort) :
   1. Complexité temporelle : O(n^2) dans le pire des cas, O(n log n) en moyenne
   2. Complexité spatiale : O(log n) en moyenne (pour la pile d'appels)

Big O/algorithmes de recherche



Algorithmes de recherche :
1.Recherche séquentielle (Sequential Search) :
1.Complexité temporelle : O(n) dans le pire des cas (lorsque l'élément recherché est à la fin)
2.Complexité spatiale : O(1)
2.Recherche binaire (Binary Search) :
1.Complexité temporelle : O(log n) dans le pire des cas (tableau trié requis)
2.Complexité spatiale : O(1)

Conclusion

Les algorithmes de recherche et de tri sont cruciaux en informatique. Les algorithmes de recherche permettent de localiser rapidement des éléments dans les données, essentiels pour la recherche d'informations en ligne ou la gestion de bases de données. D'autre part, les algorithmes de tri organisent efficacement les données, améliorant les performances et simplifiant la gestion. Ils sont conçus pour être efficaces, minimisant le temps et les ressources nécessaires. De plus, ces algorithmes sont des composants fondamentaux de nombreuses solutions informatiques, résolvant des problèmes complexes. En les optimisant, on peut atteindre des performances exceptionnelles dans des contextes spécifiques. En somme, ils sont au cœur de nombreuses applications informatiques, ayant un impact direct sur la performance et l'efficacité des système