

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Recherche dichotomique

Christophe Viroulaud

Première - NSI

1. Problématique

2. Recherche classique dans un tableau

3. Recherche dans un tableau trié

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Rechercher un élément dans un tableau est une opération courante. Cette tâche a un coût qui dépend de la taille du tableau. Cependant, si le tableau est déjà trié il est possible d'accélérer grandement la recherche.

Comment implémenter une recherche efficace dans un tableau trié ?

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

1. Problématique

2. Recherche classique dans un tableau

2.1 Génération des données

2.2 Recherche dans les données

3. Recherche dans un tableau trié

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

On demande à dix mille personnes de penser à un nombre entre 0 et 1000000 et on stocke les résultats dans un tableau au fur et à mesure des réponses.

Activité 1 : Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1 entiers = [randint(0, 1000000) for _ in range(100000)]
```

Jeu de données

1. Problématique
2. Recherche classique dans un tableau
 - 2.1 Génération des données
 - 2.2 Recherche dans les données
3. Recherche dans un tableau trié

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Pour vérifier la présence d'une valeur dans les données, il faut parcourir le tableau élément par élément.

3	180	1007	56			2178	8
---	-----	------	----	--	--	------	---

FIGURE 1 – Parcours séquentiel

À retenir

Dans le pire des cas la complexité temporelle de la recherche dépend du nombre d'éléments.

$$O(n)$$

Activité 2 :

1. Écrire la fonction `recherche_classique(tab: list, cherche: int) → bool` qui renvoie `True` si l'entier `cherche` est présent dans le tableau.
2. Tester la fonction : vérifier si le nombre 575000 a été choisi par une personne.
3. À l'aide de la méthode `time` de la bibliothèque `time` mesurer la durée d'exécution de la fonction.

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1 def recherche_classique(tab: list, cherche: int) -> bool:
2     """
3     Renvoie True si 'cherche' est dans 'tab'
4     """
5     for element in tab:
6         if element == cherche:
7             return True
8     # à la fin de la boucle on n'a pas trouvé 'cherche'
9     return False
```

```
1 deb = time()
2 print(recherche_classique(entiers, 575000))
3 fin = time()
4 print(fin-deb)
```

```
1 False
2 0.0066792964935302734
```

Exemple de résultat

1. Problématique
2. Recherche classique dans un tableau
3. Recherche dans un tableau trié
 - 3.1 Des données ordonnées
 - 3.2 Recherche dichotomique
 - 3.3 Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Imaginons maintenant la même expérience mais prenons la peine de trier les éléments au fur et à mesure de leur ajout dans le tableau de données.

3	8	56	180			1007	2178
---	---	----	-----	--	--	------	------

FIGURE 2 – Tableau trié

Activité 3 : Pour simplifier nous allons utiliser la méthode `sort` pour trier les données.

1. Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.
2. Trier le tableau.

Problématique

Recherche
classique dans un
tableau

Génération des données
Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées
Recherche dichotomique
Efficacité

```
1 entiers = [randint(0, 1000000) for _ in range(100000)]  
2 entiers.sort()
```

Jeu de données

1. Problématique

2. Recherche classique dans un tableau

3. Recherche dans un tableau trié

3.1 Des données ordonnées

3.2 Recherche dichotomique

3.3 Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

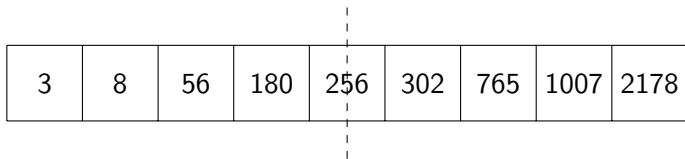
Recherche dichotomique

Efficacité

Les données étant triées, le principe de la dichotomie, pour chercher la présence d'un élément, consiste à :

- ▶ couper le tableau en deux parties égales,
- ▶ ne garder que la partie contenant l'élément,
- ▶ répéter l'opération jusqu'à trouver l'élément ou avoir une partie vide.

Cherchons 765 dans le tableau suivant :



3	8	56	180	256	302	765	1007	2178
---	---	----	-----	-----	-----	-----	------	------

FIGURE 3 – Séparons les données en deux parties

3	8	56	180	256	302	765	1007	2178
---	---	----	-----	-----	-----	-----	------	------

FIGURE 4 – 256 n'est pas le nombre recherché et il est inférieur à 765

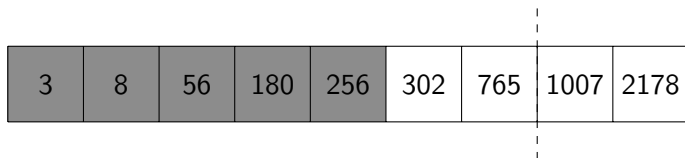


FIGURE 5 – Séparons les données restantes en deux parties

3	8	56	180	256	302	765	1007	2178
---	---	----	-----	-----	-----	-----	------	------

FIGURE 6 – Nous pouvons éliminer la partie supérieure à 765

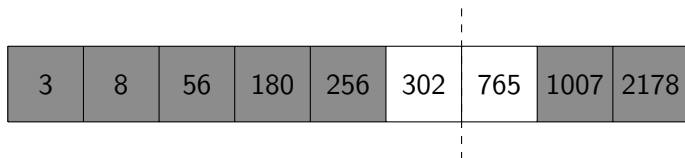


FIGURE 7 – Dernière séparation

3	8	56	180	256	302	765	1007	2178
---	---	----	-----	-----	-----	-----	------	------

FIGURE 8 – 765 a été trouvée en trois itérations

En pratique, on utilise les indices pour trouver le milieu.

0	1	2	3	4	5	6	7	8
3	8	56	180	256	302	765	1007	2178

FIGURE 9 – $\frac{8+0}{2} = 4$ l'indice médian est 4

- 256 n'est pas le nombre recherché,
- il est inférieur au nombre recherché.

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

0	1	2	3	4	5	6	7	8
3	8	56	180	256	302	765	1007	2178

FIGURE 10 – $\frac{8 + 5}{2} = 6$ l'indice médian est 6

- 765 est le nombre recherché,
- la recherche s'arrête.

Activité 4 : Écrire la fonction `recherche_dicho(tab: list, cherche: int) → bool` qui applique le principe de la dichotomie. Pour séparer les données en deux parties (à peu près) égales il faudra calculer l'indice médian de la partie encore valide.

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1 def recherche_dicho(tab: list, cherche: int) -> bool:
2     i_debut = 0
3     i_fin = len(tab)-1
```

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1 while i_fin >= i_debut:  
2     i_milieu = (i_debut+i_fin) // 2
```

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1     if cherche == tab[i_milieu] :  
2         return True
```

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

```
1     elif cherche < tab[i_milieu] :  
2         i_fin = i_milieu-1  
3     else: # cherche > tab[i_milieu]  
4         i_debut = i_milieu+1
```

Problématique

Recherche
classique dans un
tableau

Génération des données
Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées
Recherche dichotomique
Efficacité

```
1 def recherche_dicho(tab: list, cherche: int) -> bool:
2     i_debut = 0
3     i_fin = len(tab)-1
4     while i_fin >= i_debut:
5         i_milieu = (i_debut+i_fin) // 2
6         if cherche == tab[i_milieu] :
7             return True
8         elif cherche < tab[i_milieu] :
9             i_fin = i_milieu-1
10        else: # cherche > tab[i_milieu]
11            i_debut = i_milieu+1
12        # à la fin de la boucle on n'a pas trouvé 'cherche'
13    return False
```


1. Problématique
2. Recherche classique dans un tableau
3. Recherche dans un tableau trié
 - 3.1 Des données ordonnées
 - 3.2 Recherche dichotomique
 - 3.3 Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Problématique

Recherche
classique dans un
tableau

Génération des données
Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées
Recherche dichotomique

Efficacité

À chaque itération la quantité de données (notée **n**) à étudier est divisée par deux. Dans le pire des cas, on divise jusqu'à ce que la taille de la partie restante soit inférieure ou égale à 1.

$$\frac{n}{2^x} = 1$$
$$\Leftrightarrow n = 2^x$$

Activité 5 :

1. Encadrer la valeur de x entre deux entiers, si le tableau contient $n = 10000$ éléments.
2. Mesurer la durée d'exécution de la fonction et la comparer à celle de la recherche classique.

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

$$2^{13} = 8192 < x < 2^{14} = 16384$$

```
1 deb = time()
2 print(recherche_dicho(entiers, 575000))
3 fin = time()
4 print(fin-deb)
```

```
1 False
2 0.0009853839874267578
```

Exemple de résultat

À retenir

La complexité temporelle de la recherche dichotomique
est :

$$\log_2 n = x$$

Code complet

Recherche
dichotomique

Problématique

Recherche
classique dans un
tableau

Génération des données

Recherche dans les données

Recherche dans un
tableau trié

Des données ordonnées

Recherche dichotomique

Efficacité

Le code complet se trouve [ici](#).