

Recherche dichotomique



Objectifs

- But des algorithmes de recherche
- Principes et précondition de la recherche dichotomique
- Algorithme
- Évaluation

Algorithmes de recherche

- Le but est **d'obtenir une sous-collection** de valeurs dans **une collection** de valeurs **par rapport à un critère**
- **Une version plus restrictive** que nous développons ici est de rechercher **si une valeur est présente dans une collection**
- Le retour de l'algorithme est généralement de deux natures :
 - **Un entier** qui représente la position du premier élément trouvé, -1 si non trouvé
 - **Un booléen** qui indique si la valeur est présente ou non

Algorithme intuitif V. 1 – Rappels

```
booleen RechercherValeur(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    pour entier indiceValeurCourante de 0 à p_collection.Capacite - 1 faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
    }
    renvoyer estTrouvee;
}
```

Algorithme intuitif V. 1 – Rappels

```
boolean RechercherValeur(int[] p_collection, int p_valeurAChercher) {
    boolean estTrouvee = faux;
    pour entier indiceValeurCourante de 0 à p_collection.Capacite - 1 faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
    }
    renvoyer estTrouvee;
}
```

Au maximum 22 comparaisons pour un tableau de 11 éléments

Dans le pire des cas on fera 22 comparaisons donc $2 * n$ comparaisons, n étant le nombre d'éléments.

En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * n$, k étant une constante.

On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(n)**.

Algorithme intuitif V. 2 – Rappels

```
booleen RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    entier indiceValeurCourante = 0;
    tant que (non estTrouvee et indiceValeurCourante < p_collection.Capacité) faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
        ++indiceValeurCourante;
    }

    renvoyer estTrouvee;
}
```

Algorithme intuitif V. 2 – Rappels

```
booleen RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    entier indiceValeurCourante = 0;
    tant que (non estTrouvee et indiceValeurCourante < p_collection.Capacité) faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
        ++indiceValeurCourante;
    }

    renvoyer estTrouvee;
}
```

Au maximum 33 comparaisons pour un tableau de 11 éléments

Dans le pire des cas on fera 33 comparaisons donc $3 * n$ comparaisons, n étant le nombre d'éléments.

En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * n$, k étant une constante.

On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(n)**.

Principes de la recherche dichotomique

- La recherche dichotomique, ou recherche par dichotomie (en anglais : binary search), 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.

Principes de la recherche dichotomique

Soit le tableau trié suivant :

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -5

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -5

$3 == -5 \Rightarrow \text{faux}$

$3 < -5 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 10) / 2 \Rightarrow 5$$

 indicePremier

 indiceDernier

 indiceMilieu

Principes de la recherche dichotomique

Recherche de -5

$3 == -5 \Rightarrow \text{faux}$

$3 < -5 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 5 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -5

$-13 == -5 \Rightarrow \text{faux}$

$-13 < -5 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 4) / 2 \Rightarrow 2$$

indicePremier

indiceDernier

indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -5

$-13 == -5 \Rightarrow \text{faux}$

$-13 < -5 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 2 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -5

$-5 == -5 \Rightarrow \text{vrai}$

valeur trouvée !

$4 * 2 + 2 = 10$ comparaisons

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (3 + 4) / 2 \Rightarrow 3$$

indicePremier

indiceDernier

indiceMilieu

2 comparaisons

Principes de la recherche dichotomique

Recherche de -3

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -3

$3 == -3 \Rightarrow \text{faux}$

$3 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 10) / 2 \Rightarrow 5$$

 indicePremier

 indiceDernier

 indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$3 == -3 \Rightarrow \text{faux}$

$3 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 5 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$-13 == -3 \Rightarrow \text{faux}$

$-13 < -3 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 4) / 2 \Rightarrow 2$$

indicePremier

indiceDernier

indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$-13 == -3 \Rightarrow \text{faux}$

$-13 < -3 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 2 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$-5 == -3 \Rightarrow \text{faux}$

$-5 < -3 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Diagram showing the search space: a 2x11 grid where the first row contains indices 0 to 10 and the second row contains values from -99 to 42. A green arrow points down to index 3, a purple arrow points down to index 4, and a yellow double-headed arrow indicates the search range between index 3 and index 4.

$$\text{indiceMilieu} = (3 + 4) / 2 \Rightarrow 3$$

 indicePremier  indiceDernier  indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$-5 == -3 \Rightarrow \text{faux}$

$-5 < -3 \Rightarrow \text{vrai}$

The diagram illustrates a binary search operation on a sorted array of 11 elements. The array is represented by two rows of 11 cells each. The top row contains indices from 0 to 10. The bottom row contains the corresponding values: -99, -18, -13, -5, 2, 3, 7, 12, 24, 39, and 42. A green double-headed arrow is positioned above the cell containing the value -5, indicating it is the target value being searched. A yellow double-headed arrow is positioned below the cell containing the value 2, indicating it is the midpoint element being compared. A yellow single-headed arrow points upwards from the bottom row to the midpoint cell.

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Recherche réduite à un sous-tableau de 1 élément

indicePremier

indiceDernier

indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$2 == -3 \Rightarrow \text{faux}$

$2 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (4 + 4) / 2 \Rightarrow 4$$

indicePremier

indiceDernier

indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

indicePremier <= indiceDernier => **faux**

STOP => Non trouvée !

4 * 4 comparaisons

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



indicePremier

indiceDernier

indiceMilieu

Algorithme de recherche dichotomique

```
boolean RechercherValeurDichotomie(int p_collection, int p_valeurAChercher) {
    boolean estTrouvee = faux;
    entier indicePremier = 0;
    entier indiceDernier = p_collection.Capacité - 1;
    entier indiceMilieu = 0;

    tant que (non estTrouvee et indicePremier <= indiceDernier) {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        si (p_collection[indiceMilieu] == p_valeurAChercher) {
            estTrouvee = vrai;
        } sinon si (p_collection[indiceMilieu] < p_valeurAChercher) {
            indicePremier = indiceMilieu + 1;
        } sinon {
            indiceDernier = indiceMilieu - 1;
        }
    }

    renvoyer estTrouvee;
}
```

Algorithme de recherche dichotomique

- À chaque tour de boucle, on divise la taille de notre problème par 2
- Si n est le nombre d'éléments et k représente la $k^{\text{ième}}$ division, le problème est successivement de taille :
 - n ($n / 2^0$, tour 1)
 - $n/2$ ($n / 2^1$, tour 2)
 - $n/4$ ($n / 2^2$, tour 3)
 - $n/8$ ($n / 2^3$, tour 4)
 - $n/16$ ($n / 2^4$, tour 5)
 - ...
 - $n/2^k$ (tour $k + 1$)

Algorithme de recherche dichotomique

- Dans le pire des cas on va arriver à un tableau d'un élément donc :

$$- 1 \leq \frac{n}{2^k}$$

$$- 2^k \leq n$$

$$- \log(2^k) \leq \log(n) \quad (\log(a^k) = k * \log(a))$$

$$- k * \log(2) \leq \log(n)$$

$$- k \leq \frac{\log(n)}{\log(2)}$$

$$- k \leq \log_2(n)$$

Algorithme de recherche dichotomique

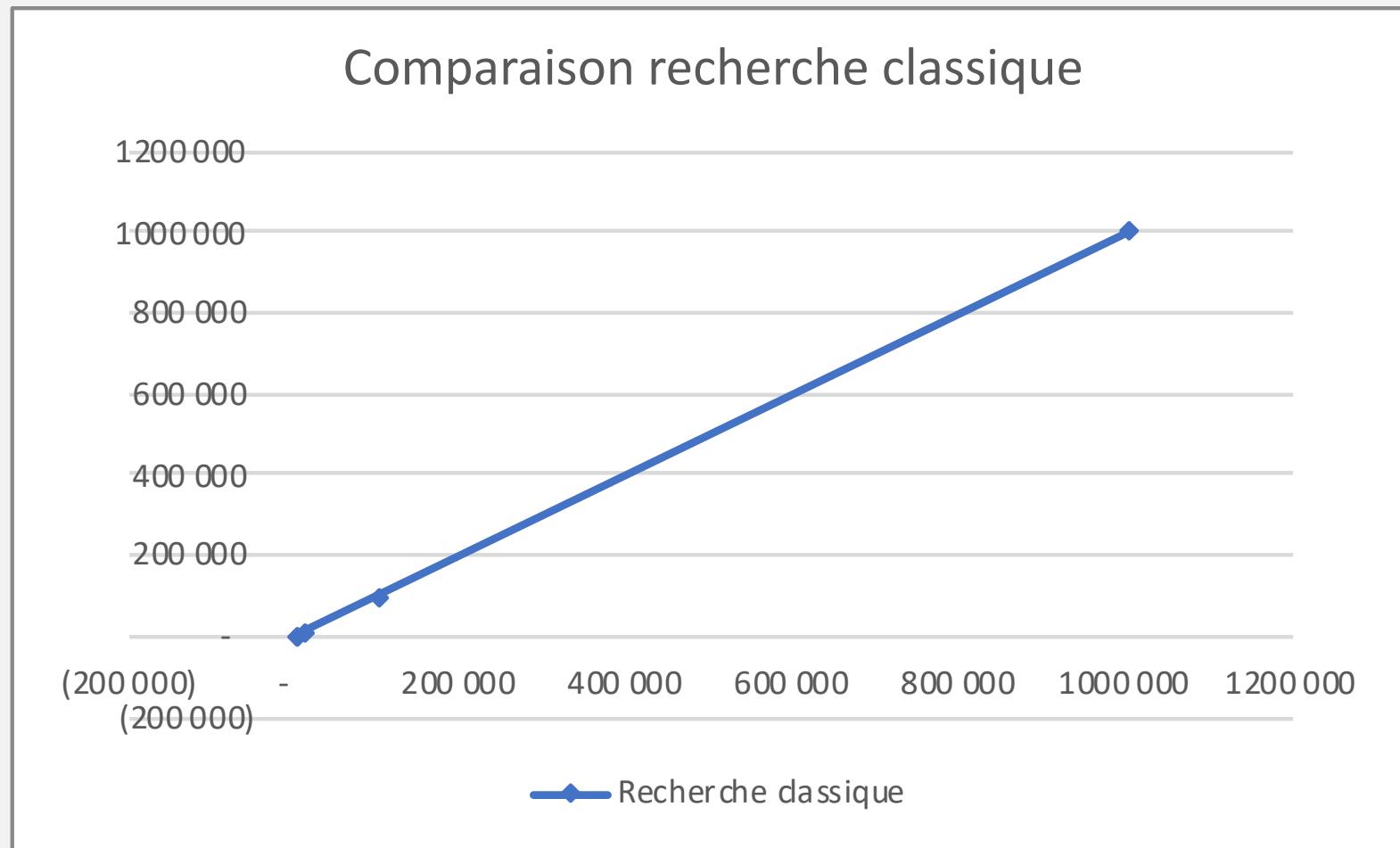
```
booléen RechercherValeurDichomie(int p_collection, int p_valeurAChercher) {
    booléen estTrouvée = faux;
    entier indicePremier = 0;
    entier indiceDernier = p_collection.Capacité - 1;
    entier indiceMilieu = 0;

    tant que (non estTrouvée et indicePremier <= indiceDernier) {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        si (p_collection[indiceMilieu] == p_valeurAChercher) {
            estTrouvée = vrai;
        } sinon si (p_collection[indiceMilieu] < p_valeurAChercher) {
            indicePremier = indiceMilieu + 1;
        } sinon {
            indiceDernier = indiceMilieu - 1;
        }
    }

    renvoyer estTrouvée;
}
```

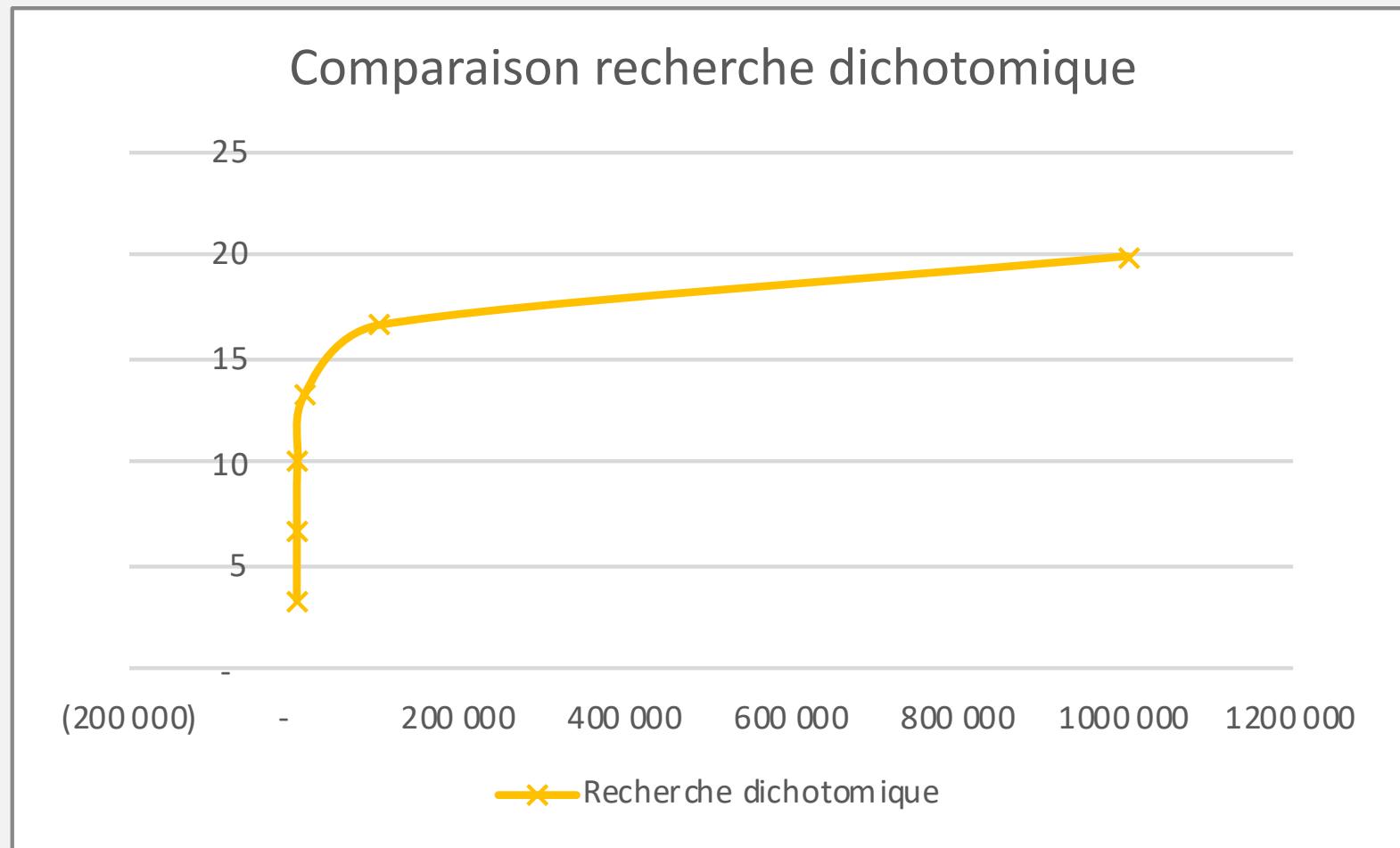
Dans le pire des cas on fera 16 comparaisons donc $4 * \log_2(n)$ comparaisons, n étant le nombre d'éléments. En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * \log_2(n)$, k étant une constante. On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(log₂(n))**.

Intuition sur la complexité des algorithmes



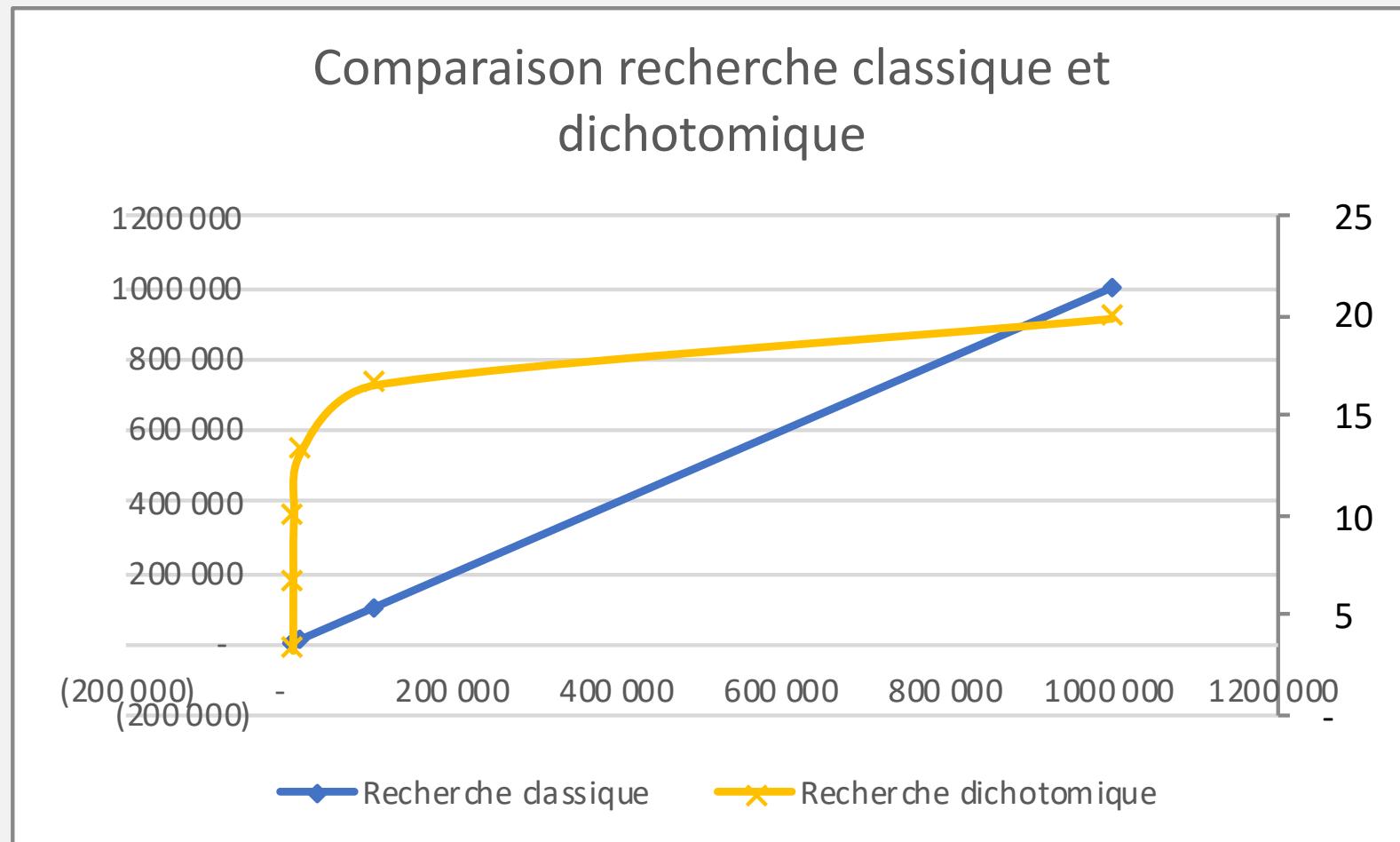
x : quantité de données
y : nombre de comparaisons

Intuition sur la complexité des algorithmes



x : quantité de données
y : nombre de comparaisons

Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons