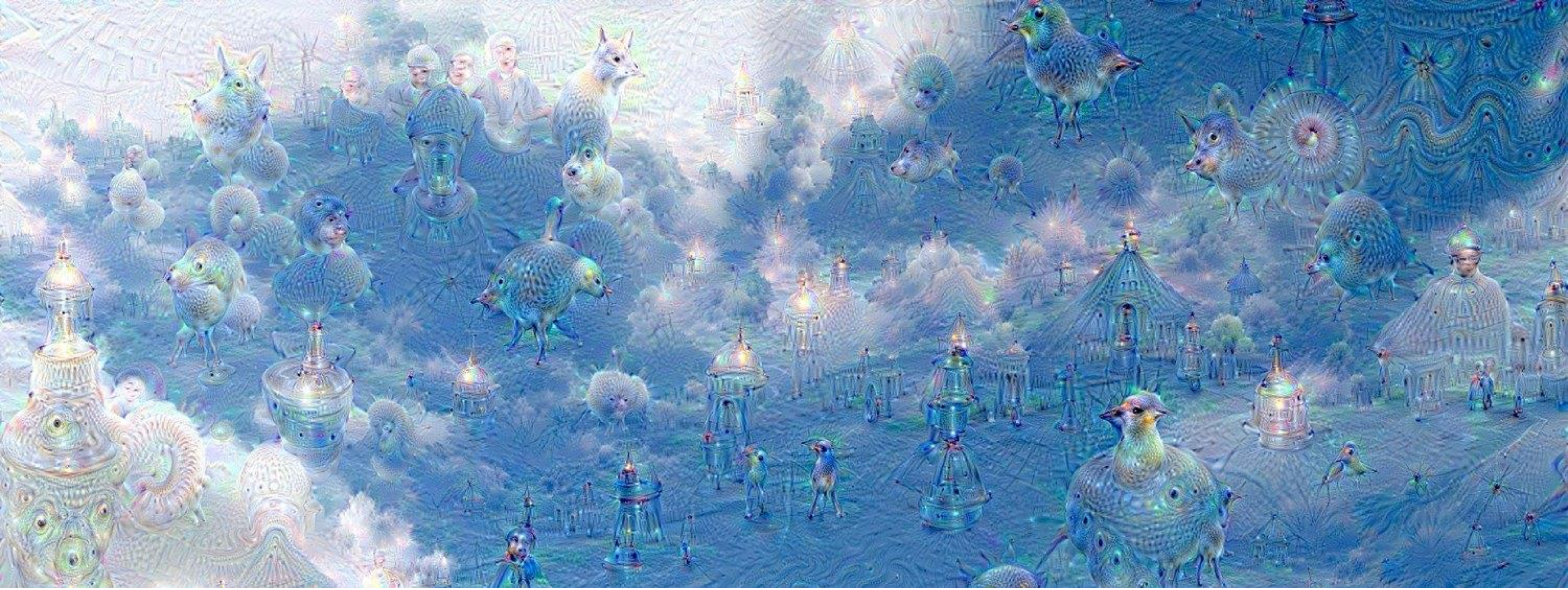


ANALYSE D'ALGORITHMES

Vitesse et efficacité : quelle différence ?

QUELLES SONT LES QUALITÉS D’UN ALGORITHME ?

D’UN BON ALGORITHME ?

PAR RAPPORT À UN AUTRE ?

EST-CE QUE LA VITESSE FAIT TOUT ?

D'AUTRES QUALITÉS EN DEHORS DE LA VITESSE ?

INTEL PENTIUM 4 VS INTEL CORE I7 : QUELLE DIFFÉRENCE ?

EXERCICE : RÉCHAUFFEMENT ALGORITHMIQUE

Obtenez le projet de départ et écrivez les algorithmes suivants :

- Indiquer si, oui ou non, un nombre est pair.
- Trouver tous les nombres pairs de 1 à 100 inclusivement.
- Trouver la quantité de nombres pairs de 1 à 100 000 inclusivement.
- Indiquer si, oui ou non, un tableau d'entiers contient un doublon.
- Calculer la suite de Fibonacci (une valeur à l'index en base 0).
  - 0,1,1,2,3,5,8,13,21 …

Si vous avez le temps, faites une seconde version.

- Les V2 dans le code.

EXERCICE : TROUVER DES NOMBRES PREMIERS

Toujours dans le projet de départ, écrivez un algorithme qui trouve la quantité de nombres premiers de 1 à 100 000.

Maintenant, écrivez un autre algorithme faisant la même chose, mais d’une autre façon. Soyez créatifs.

Deux résultats probables :

- 1 long, mais qui utilise peu de mémoire
- 1 plus rapide, mais qui utilise plus de mémoire.

CRIBLE D'ÉRATOSTHÈNE

1. Créer un tableau de booléens.
   1. Autant de cases que de nombres.
   2. Toutes les cases à « true ».
   3. L'index représente le nombre.
2. Mettre à « false » les cases des multiples de chaque nombre.
3. À la fin, il ne reste que les cases des nombres qui sont à « true ».

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

Source (Voir le GIF)

EXERCICE : VITESSE DE L’ALGORITHME

Prenez les deux versions de l’algorithme de recherche de nombre premier et calculez le temps d’exécution.

Qu’obtenez-vous ?

- Mise en commun dans 3 minutes.

Est-ce que le temps (mesure empirique) est vraiment une bonne manière de mesurer l’efficacité d’un algorithme ?

NOTATION BIG-O

Moyen de mesurer l'efficacité d'un algorithme, indépendamment du processeur sur lequel il s'exécute.

- Intel Pentium 1 vs AMD Threadripper 1950X

Basée sur le nombre d'étapes à faire.

Comment l'algorithme se comporte face à différentes tailles de problèmes ?

- 10 éléments vs 1 000 000 000 d'éléments.
- Mesure de comparaison plutôt qu'une mesure de vitesse.

NOTATION BIG-O – O(N)

O(n)

- Autant d’étapes qu’il y a d’éléments.

ATTENTION!

- La notation Big-O ignore le nombre d’opérations par étape.
- La vitesse d’exécution de ces opérations est dépendante de la vitesse du processeur.

NOTATION BIG-O – O(N)

O(n)

- Autant d'étapes qu'il y a d'éléments.

Algorithmes en O(n)

- Trouver tous les nombres pairs de 1 à 100 inclusivement.
- *Trouver un élément dans une liste.*

Bref, un O(n) contient une boucle qui parcourt tous les éléments du problème.

NOTATION BIG-O – O(N^2)

O(n^2)

- Autant étapes par élément qu'il y a d'éléments.
- N étapes par éléments.

Algorithmes en O(n^2)

- Indiquer si un tableau d'entiers contient un doublon.
- *Indiquer si un nombre est premier.*

Bref, O(n^2) contient une boucle imbriquée.

NOTATION BIG-O – $O(2^N)$

$O(2^n)$

- Traiter chaque élément prend N étapes.

Algorithmes en $O(2^n)$

- Trouver une valeur de la suite de Fibonacci (de manière récursive).

Assez rare (beurk…), et pas à l'examen.

NOTATION BIG-O – O(1)

O(1)

- Une seule étape, peu importe la taille du problème.

Algorithmes en O(1)

- Indiquer si un nombre est pair.
- Trouver le nombre de nombres pairs entre 1 et 100 inclusivement.
- *Obtenir un élément dans un tableau à un index précis.*

Bref, O(1) ne contient pas de boucle.

$O(1)$, $O(N)$, $O(N^2)$ ET $O(2^N)$ SONT DES ALGORITHMES DE FORCE BRUTE.

SOUVENT, IL Y A PLUS EFFICACE QUE LA FORCE BRUTE.

EXERCICE : ALGORITHMES DE RECHERCHE

Toujours dans le projet de départ, écrivez un algorithme qui indique si, oui ou non, un tableau trié contient un élément.

- Faites un algorithme de force brute en O(n).

Maintenant, suivez le guide pour le second algorithme.

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Taille: 10

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Taille: 10

Borne inférieur : 0

Borne supérieur : Taille – 1 = 10 -1 = 9

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Taille: 10

Borne inférieur : 0

Borne supérieur : 9

Milieu : (Borne supérieur – Borne inférieur) / 2 = 4

Pourquoi pas 5 ?

EXERCICE : RECHERCHE DICHOTOMIQUE

Pourquoi 4 est le point milieu et pas 5 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

La ligne rouge représente la séparation correcte (en deux parts égales) des deux tableaux.

La position 5 n'est pas le milieu du tableau, même si c'est effectivement la moitié de la taille du tableau.

EXERCICE : RECHERCHE DICHOTOMIQUE

Pourquoi 4 est le point milieu et pas 5 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Aussi, les divisions d'entiers sur un processeur donnent la valeur « plancher » s'il y a un reste durant la division, d'où la position 4.

Ex : 9 / 2 = 4, 5 / 2 = 2, etc…

EXERCICE : RECHERCHE DICHOTOMIQUE

Et si le tableau est de taille impaire ?

0	1	2	3	4	5	6	7	8
2	7	9	20	25	55	75	100	200

Taille : 9

Borne inférieure : 0

Borne supérieure : Taille – 1 = 8

Milieu : (Borne supérieure – borne inférieure) / 2 = (8 – 0) / 2 = 4

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Taille: 10

Borne inférieur : 0

Borne supérieur : 9

Milieu : 4

Valeur au milieu : 25

20 est plus petit que 25, donc doit regarder avant le milieu (4).

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Nouvelle Borne inférieur : 0

Nouvelle Borne supérieur : Ancien Milieu – 1 = 4 – 1 = 3

Milieu : (Borne supérieure – borne inférieure) / 2 = (3 – 0) / 2 = 1

Valeur au milieu : 7

20 est plus grand que 7, donc doit regarder après le milieu (1).

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7
2	7	9	20	25	55	75	100

ATTENTION!
Formule modifiée!
Pourquoi ?

Nouvelle Borne inférieur : Ancien Milieu + 1 = 1 + 1 = 2

Nouvelle Borne supérieur : 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2

= 2 + (3 – 2) / 2 = 2 + 1 / 2 = 2 + 0 = 2

EXERCICE : RECHERCHE DICHOTOMIQUE

Pourquoi la formule du milieu a changé ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

En fait, c'est la formule qui aurait dû être utilisée dès le départ.

On a 2 bornes : une inférieure et une supérieure. Il faut trouver le milieu entre les deux.

La distance entre deux bornes est égale à la valeur de la borne supérieure moins la valeur de la borne inférieure.

EXERCICE : RECHERCHE DICHOTOMIQUE

Pourquoi la formule du milieu a changé ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

En divisant la distance par 2, on obtient non pas l'index du milieu, mais la distance à parcourir, à partir de la borne inférieure, pour atteindre le milieu.

D'où :

Borne inférieure + (Borne supérieure – Borne inférieure) / 2

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Nouvelle Borne inférieur : Ancien Milieu + 1 = 1 + 1 = 2

Nouvelle Borne supérieur : 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2

= 2 + (3 – 2) / 2 = 2 + 1 / 2 = 2 + 0 = 2

Valeur au milieu : 9

20 est plus grand que 9, donc doit regarder après le milieu (2).

EXERCICE : RECHERCHE DICHOTOMIQUE

Élément à trouver : 20.

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Nouvelle Borne inférieur : Ancien Milieu + 1 = 2 + 1 = 3

Nouvelle Borne supérieur : 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2

= 3 + (3 – 3) / 2 = 3 + 0 / 2 = 3 + 0 = 3

Valeur au milieu : 20

On a trouvé!

EXERCICE : RECHERCHE DICHOTOMIQUE

Que serait-il arrivé si la valeur cherchée était 21 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Borne inférieur : 0

Borne supérieur : 9

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2

= 4

Valeur au milieu : 25

21 est plus petit que 25. Donc, on continue vers le bas.

EXERCICE : RECHERCHE DICHOTOMIQUE

Que serait-il arrivé si la valeur cherchée était 21 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Borne inférieur : 0

Borne supérieur : Ancien Milieu – 1 = 4 – 1 = 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2
= 1

Valeur au milieu : 7

21 est plus grand que 7. Donc, on continue vers le haut.

EXERCICE : RECHERCHE DICHOTOMIQUE

Que serait-il arrivé si la valeur cherchée était 21 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Borne inférieur : Ancien Milieu + 1 = 1 + 1 = 2

Borne supérieur : 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2
= 2

Valeur au milieu : 9

21 est plus grand que 9. Donc, on continue vers le haut.

EXERCICE : RECHERCHE DICHOTOMIQUE

Que serait-il arrivé si la valeur cherchée était 21 ?

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	7	9	20	25	55	75	100	200	500

Borne inférieur : Ancien Milieu + 1 = 2 + 1 = 3

Borne supérieur : 3

Milieu : Borne inférieure + (Borne supérieure – Borne inférieure) / 2
= 3

Valeur au milieu : 20

21 est plus grand que 20. Donc, on continue vers le haut.

EXERCICE : RECHERCHE DICHOTOMIQUE

Que serait-il arrivé si la valeur cherchée était 21 ?

0	1	2	3	4	5	6	7	8	9
2	7	9	20	25	55	75	100	200	500

Borne inférieur : Ancien Milieu + 1 = 3 + 1 = 4

Borne supérieur : 3

OH! La borne inférieure est plus grande que la borne supérieure.

Ce n’est pas normal! On arrête!

DONC C'EST QUOI LA COMPLEXITÉ DE ÇA EN NOTATION BIG-O ?

NOTATION BIG-O – O(LOG N)

O(Log N)

- Plus on avance dans le problème, plus sa taille diminue.
- Souvent, divisée par 2.

Algorithmes en O(Log N)

- Recherche dichotomique.

Bref, O(Log N) contient effectivement une boucle, mais aussi un moyen de réduire le nombre d'éléments à traiter à chaque itération.

ET POUR LE SECOND ALGORITHME DE RECHERCHE DE NOMBRES PREMIERS ?

WOW! C'EST LOIN ÇA!

AVANT ÇA, QUELQUES NOTIONS DE PLUS...

OPÉRATIONS SUR LA NOTATION BIG-O

MULTIPLICATION DANS LA NOTATION BIG-O

Lorsqu’une boucle en contient une autre, on multiplie les notations.

Si une boucle O(n) contient une boucle O(log n), alors on a :

$$O(n * \log n)$$

Si une boucle O(n) contient une autre boucle O(n) :

$$O(n^2)$$

ADDITION DANS LA NOTATION BIG-O

Lorsque des boucles se suivent, on additionne les notations.

Si une boucle O(n) est suivi d'une boucle O(n * log n) et d'une autre O(n), alors on a :

$$O(n) + O(n * \log n) + O(n)$$

Cependant, on ne garde que la boucle la plus longue :

$$O(n * \log n)$$

AJUSTEMENTS DANS LA NOTATION BIG-O

Parfois, une boucle fait 1 itération de moins qu'il y a d'éléments.

On peut alors dire :

$$O(n - 1)$$

Par contre, cette itération de moins est pratiquement invisible, donc on préfère écrire tout simplement :

$$O(n)$$

CONDITIONS DANS LA NOTATION BIG-O

Certains algorithmes ont un comportement différent en fonction de la valeur du problème en entrée.

Par exemple, un algorithme peut s’exécuter en O(1) pour un cas et en O(n) dans tous les autres cas. On sépare alors les notations :

Meilleur cas : O(1), Pire cas : O(n)

DONC…

RAPPEL SUR LE CRIBLE D'ÉRATOSTHÈNE

1. Créer un tableau de booléens.
   1. Autant de cases que de nombres.
   2. Toutes les cases à « true ».
   3. L’index représente le nombre.
2. Mettre à « false » les cases des multiples de chaque nombre.
3. À la fin, il ne reste que les cases des nombres qui sont à « true ».

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

[Source (Voir le GIF)]

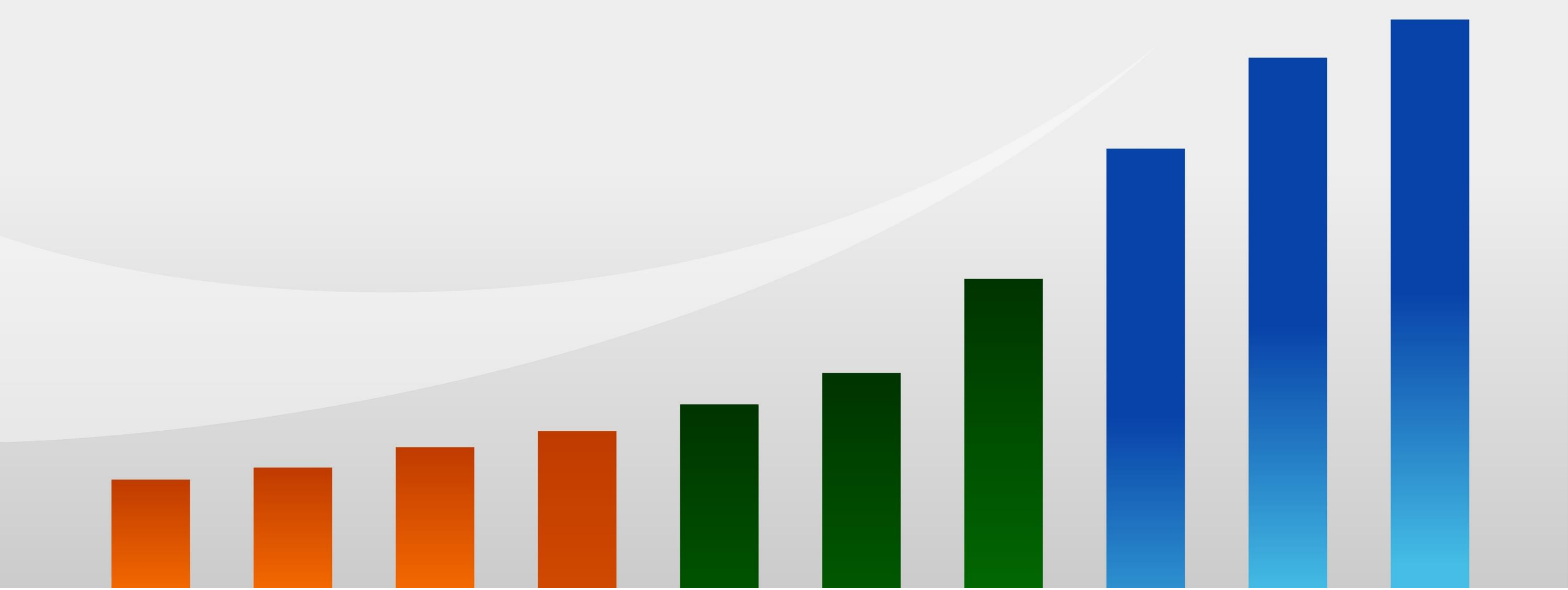
ANALYSE DU CRIBLE D'ÉRATOSTHÈNE

Trois boucles :

- La première est en O(n) → Pour créer le tableau.
- La deuxième est en O(n * log n) → Pour éliminer les multiples.
- La troisième est en O(n) → Pour compter le nombre de booléens à « true. »

On applique les opérations, ce qui donne :

$$O(n * \log n)$$



EFFECTUER DE L'OPTIMISATION

Je préfère vous prévenir…

TYPES D'OPTIMISATIONS

Optimisation technique :

- « i++ » est un peu plus lent que « ++i ».
- Utilisation de la mémoire cache du processeur.

Optimisation algorithmique :

- Réduire la taille du problème au lieu d'utiliser la force brute.
- Changement complet de stratégie.

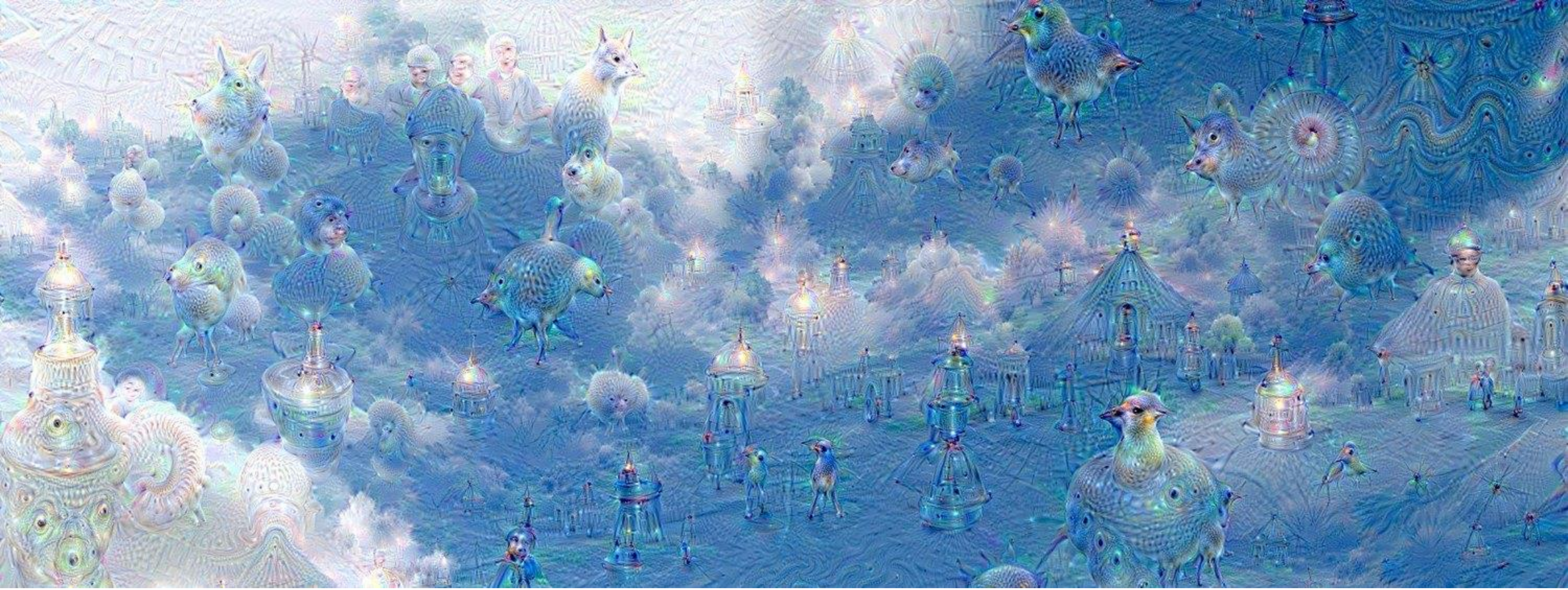
LE BON ALGORITHME POUR LE BON CLIENT

Quels sont les besoins pour l'algorithme ?

- La vitesse est-elle si importante ?
- Est-ce un critère pour le client ?
- Est-ce que l'algorithme est exécuté souvent ?
- Est-ce qu'investir du temps dans un algorithme plus rapide vaut la peine ?
- Est-ce qu'un algorithme de force brute ne serait pas suffisant ?

« Too much optimization, too early, is the root of all evil. »

ADAGE INFORMATIQUE (AUTEUR INCONNU)



FIN