Sylvain, Raphaël
(111 124 564)

Conception et analyse d'algorithmes
IFT-3001

Travail 1

Travail présenté à
Yanick Ouellet

Département d'informatique et de génie logiciel
Univesité Laval
Hiver 2019

Sylvain, Raphaël
111 124 564

Question 1

Description textuelle

L'algorithme se base sur ces observations suivantes :

1. Zéro est l'élément absorbant de la multiplication ;
2. Le résultat est un produit d'éléments de vecteur (une multiplication) ;
3. Lorsque deux zéros sont dans le vecteur, le produit sera toujours zéro ;
4. Nous pouvons émuler la non-production d'un élément, en divisant le produit par celui-ci, pour vu qu'il n' est pas zéro.

L'algorithme débute donc en initialisant une variable pour conserver le produit total (initialisé à 1), ainsi que deux variables "indicatices" (flags). Une pour conserver l'indice d'un élément absorbant et l'autre pour indiquer s'il y a plus d'un élément absorbant.

Puis, l'algorithme parcourt tous les éléments de notre vecteur entrant. S'il trouve pour la première fois un zéro, il affectera l'indice où cet élément se trouve dans le vecteur. S'il en trouve un deuxième il affecte la valeur booléen indiquant plusieurs zéros à vrai.

Cette boucle terminée, il entre dans un des trois embranchements dépendamment de la valeur des variables indicatrices.

Premièrement, si la variable booléenne indiquant que plusieurs zéros ont été trouvés dans le vecteur entrant, alors nous initialisons le vecteur résultat pour qu'il ne contienne que des zéros, puisque nous sommes garantis qu'il y aura une multiplication par zéro pour tous ces éléments.

Sinon, si la variable indiquant l'indice d'un zéro a été définie, alors tous les éléments du vecteur résultat seront à zéro, à l'exception de l'élément à cet indice qui aura le résultat total du produit.

Finalement, si aucune des deux variables indicatrices n'a été définie, alors chaque élément du vecteur résultat sera égal au produit, diviser par l'élément à l'indice correspondant du vecteur entrant.

Analyse de l'algorithme

Puisque l'algorithme est trop complexe pour être analysé à l'aide d'une seule opératoire de base, nous allons séparer l'algorithme en quatre parties et

analyser chacune de ces parties individuellement, puis conclure sur l'entièreté de l'algorithme à partir de ces sous-analyses.

La taille de l'instance est n, soit la cardinalité du vecteur entrant A.

Bloc A

Le bloc A est composé de la première boucle `for` et des déclarations lui précédents.

Nous pouvons choisir comme opération de base le `!productIsAlwaysZero`, car c'est la comparaison qui est effectuée le plus souvent, c'est-à-dire à chaque itération. De plus, la fonction `A.at(i)` à l'intérieur de la boucle s'effectue en temps constant.

Le nombre de fois que cette opération de base peut s'exécuter dépendant du contenu du vecteur entrant ainsi que de sa cardinalité.

Meilleur cas

En meilleur cas, les deux premiers éléments du vecteur sont 0. Si la boucle découvre deux 0, elle se court-circuite. Nous avons donc que :

$$
\begin{aligned}
& C^A_{best}(n) = 3 \\
\Rightarrow \quad & \langle \text{ Donc } \rangle \\
& 1 \leq C^A_{best}(n) = 3 \leq 5 \\
\in \quad & \langle \text{ Définition de } \Theta \text{ avec } c_1 = 1,\ c_2 = 5 \text{ et } n_0 = 0 \rangle \\
& \Theta(1)
\end{aligned}
$$

Pire cas

En pire cas, nous devons parcourir l'ensemble des éléments du vecteur entrant. À chacune des boucles, nous devons effectuer une seule comparaison, puis nous effectuons une dernière comparaison avant de sortir. Nous avons donc :

$$
\begin{aligned}
& C^A_{worst}(n) \\
= \quad & \langle \text{ Définition mathématique } \rangle \\
& \sum_{i=0}^{n-1} 1 + 1
\end{aligned}
$$

$$
\begin{aligned}
&= && \langle \text{ Première règle des sommations } \rangle \\
&(((n-1)-0+1)\times 1)+1 \\
&= && \langle \text{ Simplification } \rangle \\
&n+1 \\
&\Rightarrow && \langle \text{ Donc } \rangle \\
&\tfrac{1}{2}n \leq C^A_{worst}(n) = n+1 \leq 2n \ \forall n \geq 1 \\
&\in && \langle \text{ Définition de } \Theta \text{ avec } c_1 = \tfrac{1}{2},\ c_2 = 2 \text{ et } n_0 = 1 \ \rangle \\
&\Theta(n)
\end{aligned}
$$

Bloc B

Le bloc B est composé uniquement du premier bloc du `if`, soit :

```
B.assign(VECTOR_SIZE, 0);
```

Ce bloc est exécuté uniquement si deux zéros sont trouvés dans le vecteur entrant.

L'opération à utiliser est donc l'appel à la fonction `std::vector::assign`.

La complexité de la fonction `std::vector::assign` est linéaire en tout temps sur la valeur du premier paramètre. Dans notre cas, ce paramètre est équivalent à la cardinalité du vecteur entrant. Donc

$$
\begin{aligned}
&C^B(n) \\
&= && \langle \text{ Définition de la complexité de } C^B \ \rangle \\
&C_{assign}(n)\cdot 1 \\
&= && \langle \text{ Simplification } \rangle \\
&C_{assign}(n) \\
&\in && \langle \text{ Documentation de } \texttt{std::vector::assign} \ \rangle \\
&\Theta(n)
\end{aligned}
$$

Bloc C

Le bloc C est composé uniquement du bloc `else if`, soit :

```
B.assign(VECTOR_SIZE, 0);
B.at(indexOfAbsorbingElement) = totalProductExceptZeroes;
```

Ce bloc est exécuté uniquement si un seul zéro a été trouvé dans le vecteur entrant.

L'opération à utiliser est donc l'appel à la fonction `std::vector::assign`, puisque celle-ci à une complexité linéaire sur la valeur du premier paramètre en tout temps, alors que la fonction `std::vector::at` est constante en tout temps. Donc

$$
\begin{aligned}
& C^C(n) \\
= & \qquad \langle \text{ Définition de la complexité de } C^C \rangle \\
& C_{assign}(n) \cdot 1 \\
= & \qquad \langle \text{ Simplification } \rangle \\
& C_{assign}(n) \\
\in & \qquad \langle \text{ Documentation de } \texttt{std::vector::assign} \rangle \\
& \Theta(n)
\end{aligned}
$$

Bloc D

Le bloc D est composé de la boucle `for` située dans le `else`.

Ce bloc est exécuté que lorsqu'aucun zéro n'a été trouvé dans le vecteur entrant.

L'opération à utiliser est l'appel à la fonction `std::vector::push_back`, car elle a une complexité amortie de $\Theta(1)$ et qu'elle est appelée n fois, soit à une constante près de l'élément appelé le plus souvent. De plus, l'appel à `std::vector::at` à une complexité constante qui est moindre ou égale à la complexité de `std::vector::push_back`.

Donc

$$
\begin{aligned}
& C^D(n) \\
= & \qquad \langle \text{ Définition de la complexité de } C^D \rangle \\
& \sum_{i=0}^{n} C_{push\_back}(n) \\
= & \qquad \langle \text{ Première règle de sommation } \rangle \\
& ((n-1) - 0 + 1) \cdot C_{push\_back}(n) \\
= & \qquad \langle \text{ Simplification } \rangle \\
& n \cdot C_{push\_back}(n)
\end{aligned}
$$

$\simeq$ ⟨ Abus de notation.
Définition de la complexité de `std::vector::push_back` ⟩

$n \cdot 1$

$=$ ⟨ Simplification ⟩

n

$\Rightarrow$ ⟨ Donc ⟩

$n \leq C^D(n) = n \leq n \ \forall n \geq 0$

$\in$ ⟨ Définition de Θ avec $c_1 = c_2 = 1$ et $n_0 = 0$ ⟩

$\Theta(n)$

Résultat final

L'algorithme est composé de 4 blocs. Dont un et un seul bloc entre B, C et D ne peut-être exécuter. Notons $C(n)$ le temps d'exécution de l'algorithme complet. Le bloc A est exécuté dans tous les cas.

En meilleur cas, le bloc B est exécuté après le bloc A. Nous avons donc :

$C_{best}(n)$

$=$ ⟨ Définition de la complexité de C_{best} ⟩

$C^A_{best}(n) + C^B(n)$

$=$ ⟨ Définition de $C^A_{best}(n)$ et $C^B(n)$ ⟩

$3 + C_{assign}(n)$

$\simeq$ ⟨ Abus de notation.
Définition de la complexité de `std::vector::assign` ⟩

$3 + n$

$=$ ⟨ Par la règle du maximum ⟩

n

$\Rightarrow$ ⟨ Donc ⟩

$n \leq C_{best}(n) = n \leq n \ \forall n \geq 0$

$\in$ ⟨ Définition de Θ avec $c_1 = c_2 = 1$ et $n_0 = 0$ ⟩

$\Theta(n)$

Dans le pire cas, le bloc C ou D est exécuté après le bloc A. Nous avons

donc :

$$
\begin{aligned}
& C_{worst_C}(n) \\
= & \quad \langle \text{ Définition de la complexité de } C_{worst_C} \rangle \\
& C^A_{worst}(n) + C^C(n) \\
= & \quad \langle \text{ Définition de } C^A_{best}(n) \text{ et } C^C(n) \rangle \\
& n + 1 + C_{assign}(n) \\
\simeq & \quad \langle \text{ Abus de notation.} \\
& \quad \text{Définition de la complexité de } \texttt{std::vector::assign} \rangle \\
& n + 1 + n \\
= & \quad \langle \text{ Simplification} \rangle \\
& 2n + 1 \\
= & \quad \langle \text{ Par la règle du maximum} \rangle \\
& 2n \\
\Rightarrow & \quad \langle \text{ Donc} \rangle \\
& n \leq C_{worst_C}(n) = 2n \leq 2n \ \forall n \geq 0 \\
\in & \quad \langle \text{ Définition de } \Theta \text{ avec } c_1 = 1,\ c_2 = 2 \text{ et } n_0 = 0 \rangle \\
& \Theta(n)
\end{aligned}
$$

et aussi que :

$$
\begin{aligned}
& C_{worst_D}(n) \\
= & \quad \langle \text{ Définition de la complexité de } C_{worst_D} \rangle \\
& C^A_{worst}(n) + C^D(n) \\
= & \quad \langle \text{ Définition de } C^A_{best}(n) \text{ et } C^D(n) \rangle \\
& n + 1 + n \\
= & \quad \langle \text{ Simplification} \rangle \\
& 2n + 1 \\
= & \quad \langle \text{ Par la règle du maximum} \rangle \\
& 2n \\
\Rightarrow & \quad \langle \text{ Donc} \rangle \\
& n \leq C_{worst_D}(n) = 2n \leq 2n \ \forall n \geq 0
\end{aligned}
$$

$$\in \qquad \langle \text{ Définition de } \Theta \text{ avec } c_1 = 1,\ c_2 = 2 \text{ et } n_0 = 0 \ \rangle$$
$$\Theta(n)$$

Donc, nous avons que cet algorithme est linéaire sur la cardinalité du vecteur entrant pour tous les cas.

Sylvain, Raphaël
111 124 564

Question 2

$$C(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ 2 \cdot C(\lfloor \frac{n}{3} \rfloor) + n & \text{si } n > 1 \end{cases}$$

Résolution récurrence

Nous allons commencer par essayer de résoudre cette récurrence que pour les occurrences de $n = 3^k \ \forall k \in \mathbb{N}$. Donc $k = log_3 n$

Nous aurons alors
$C\left(3^k\right) = 2 \cdot C\left(\left\lfloor \frac{3^k}{3} \right\rfloor\right) + 3^k = 2 \cdot C\left(3^{k-1}\right) + 3^k$
avec comme valeur de base
$C(3^0) = 0$
Résolvons :

$$\begin{aligned}
& C\left(n\right) \\
= & \qquad \langle \text{ Définition de } 3^k \ \rangle \\
& C\left(3^k\right) \\
= & \qquad \langle \ 1^{ere} \text{ induction } \rangle \\
& 2 \cdot C\left(3^{k-1}\right) + 3^k \\
= & \qquad \langle \ 2^e \text{ induction } \rangle \\
& 2 \cdot \left[2 \cdot C\left(3^{k-2}\right) + 3^{k-1}\right] + 3^k \\
= & \qquad \langle \text{ Simplification } \rangle \\
& 2^2 \cdot C\left(3^{k-2}\right) + 2 \cdot 3^{k-1} + 3^k \\
= & \qquad \langle \ 3^e \text{ induction } \rangle \\
& 2^2 \cdot \left[2 \cdot C\left(3^{k-3}\right) + 3^{k-2}\right] + 2 \cdot 3^{k-1} + 3^k \\
= & \qquad \langle \text{ Simplification } \rangle \\
& 2^3 \cdot C\left(3^{k-3}\right) + 2^2 \cdot 3^{k-2} + 2 \cdot 3^{k-1} + 3^k \\
= & \qquad \langle \text{ Suite } \rangle \\
& \ldots \\
= & \qquad \langle \ i^e \text{ induction } \rangle
\end{aligned}$$

$$
\begin{aligned}
& 2^i \cdot C\left(3^{k-i}\right) + 2^{i-1} \cdot 3^{k-(i-1)} + ... + 2^2 \cdot 3^{k-2} + 2 \cdot 3^{k-1} + 3^k \\
= & \quad \langle \text{ Suite } \rangle \\
& ... \\
= & \quad \langle \ k^e \text{ induction } \rangle \\
& 2^k \cdot C\left(3^{k-k}\right) + 2^{k-1} \cdot 3^{k-(k-1)} + ... + 2^2 \cdot 3^{k-2} + 2 \cdot 3^{k-1} + 3^k \\
= & \quad \langle \text{ Simplification } \rangle \\
& 2^k \cdot C\left(3^0\right) + 2^{k-1} \cdot 3^1 + ... + 2^2 \cdot 3^{k-2} + 2^1 \cdot 3^{k-1} + 2^0 \cdot 3^k \\
= & \quad \langle \text{ Simplification } \rangle \\
& 2^k \cdot C\left(3^0\right) + \sum_{i=1}^{k} \left(2^{k-i} \cdot 3^i\right) \\
= & \quad \langle \text{ Simplification } \rangle \\
& 2^k \cdot C\left(1\right) + \sum_{i=1}^{k} \left(2^{k-i} \cdot 3^i\right) \\
= & \quad \langle \text{ Valeur de base } \rangle \\
& 2^k \cdot 0 + \sum_{i=1}^{k} \left(2^{k-i} \cdot 3^i\right) \\
= & \quad \langle \text{ Simplification } \rangle \\
& \sum_{i=1}^{k} \left(2^{k-i} \cdot 3^i\right) \\
= & \quad \langle \text{ Règle exposant } \rangle \\
& \sum_{i=1}^{k} \left(\frac{2^k}{2^i} \cdot 3^i\right) \\
= & \quad \langle \text{ Séparer fraction } \rangle \\
& \sum_{i=1}^{k} \left(2^k \cdot \frac{1}{2^i} \cdot 3^i\right) \\
= & \quad \langle \text{ Simplification } \rangle \\
& \sum_{i=1}^{k} \left(2^k \cdot \frac{3^i}{2^i}\right) \\
= & \quad \langle \text{ Simplification } \rangle \\
& \sum_{i=1}^{k} \left(2^k \cdot \left(\frac{3}{2}\right)^i\right)
\end{aligned}
$$

$$
\begin{aligned}
&= && \langle \text{ Extraire produit constant de la sommation } \rangle \\
&2^k \cdot \sum_{i=1}^{k} \left(\left(\tfrac{3}{2}\right)^i \right) \\
&= && \langle \text{ Ajuster borne de la sommation } \rangle \\
&2^k \cdot \left(\sum_{i=0}^{k} \left(\left(\tfrac{3}{2}\right)^i \right) - \left(\tfrac{3}{2}\right)^0 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(\sum_{i=0}^{k} \left(\left(\tfrac{3}{2}\right)^i \right) - 1 \right) \\
&= && \langle \text{ Application de la règle de sommation } \rangle \\
&2^k \cdot \left(\frac{\left(\frac{3}{2}\right)^{k+1} - 1}{\frac{3}{2} - 1} - 1 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(\frac{\left(\frac{3}{2}\right)^{k+1} - 1}{\frac{1}{2}} - 1 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(2 \cdot \left(\left(\tfrac{3}{2}\right)^{k+1} - 1 \right) - 1 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(2 \cdot \left(\tfrac{3}{2}\right)^{k+1} - 2 - 1 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(2 \cdot \left(\tfrac{3}{2}\right)^{k+1} - 3 \right) \\
&= && \langle \text{ Extraire un } \tfrac{3}{2} \rangle \\
&2^k \cdot \left(2 \cdot \tfrac{3}{2} \cdot \left(\tfrac{3}{2}\right)^{k} - 3 \right) \\
&= && \langle \text{ Simplification } \rangle \\
&2^k \cdot \left(3 \cdot \left(\tfrac{3}{2}\right)^{k} - 3 \right) \\
&= && \langle \text{ Appliquer } k \text{ sur } \tfrac{3}{2} \rangle \\
&2^k \cdot \left(3 \cdot \frac{3^k}{2^k} - 3 \right) \\
&= && \langle \text{ Appliquer multiplication } 2^k \rangle \\
&2^k \cdot 3 \cdot \frac{3^k}{2^k} - 2^k \cdot 3
\end{aligned}
$$

$$
\begin{aligned}
&= && \langle \text{ Simplification } \rangle \\
&\quad 3 \cdot 3^k - 2^k \cdot 3 \\
&= && \langle \text{ Définition de } n \text{ et de } k \rangle \\
&\quad 3n - 2^{log_3(n)} \cdot 3 \\
&= && \langle \text{ Simplification } \rangle \\
&\quad 3\left(n - 2^{log_3(n)}\right) \\
&= && \langle \text{ Règle des log } \rangle \\
&\quad 3\left(n - 2^{log_3(2) log_2(n)}\right) \\
&= && \langle \text{ Règle des exposants } \rangle \\
&\quad 3\left(n - \left(2^{log_2(n)}\right)^{log_3(2)}\right) \\
&= && \langle \text{ Simplification } \rangle \\
&\quad 3\left(n - n^{log_3(2)}\right) \\
&\simeq && \langle \text{ Approximation du } log_3 2 \rangle \\
&\quad 3\left(n - n^{0.63093}\right)
\end{aligned}
$$

Notation asymptotique

Nous pouvons trouver la notation asymptotique :
Borne supérieure :

$$
\begin{aligned}
&\quad 3\left(n - n^{log_3(2)}\right) \\
&= && \langle \text{ Réécriture } \rangle \\
&\quad 3n - 3n^{log_3(2)} \\
&\leq && \langle \text{ En ajoutant } 3n^{log_3(2)} \rangle \\
&\quad 3n - 3n^{log_3(2)} + 3n^{log_3(2)} \\
&= && \langle \text{ Simplification } \rangle \\
&\quad 3n \\
&\in && \langle \text{ Définition du } \mathcal{O} \text{ avec } c_2 = 3 \text{ et } n_0 = 0 \rangle \\
&\quad \mathcal{O}(n)
\end{aligned}
$$

Borne inférieure :

$$
\begin{aligned}
& 3\left(n - n^{log_3(2)}\right) \\
= & \quad \langle \text{ Réécriture } \rangle \\
& 3n - 3n^{log_3(2)} \\
\geq & \quad \langle \text{ En retirant } n - 3n^{log_3(2)}. \text{ Vrai } \forall n \geq 20 \rangle \\
& 3n - 3n^{log_3(2)} - \left(n - 3n^{log_3(2)}\right) \\
= & \quad \langle \text{ Appliquer la négation sur la parenthèse } \rangle \\
& 3n - 3n^{log_3(2)} - n + 3n^{log_3(2)} \\
= & \quad \langle \text{ Appliquer la négation } \rangle \\
& 2n \\
\in & \quad \langle \text{ Définition de } \Omega \text{ avec } c_1 = 2 \text{ et } n_0 = 20 \rangle \\
& \Omega\left(n\right)
\end{aligned}
$$

Donc comme $3\left(n - n^{log_3(2)}\right) \in \Omega\left(n\right)$ et $3\left(n - n^{log_3(2)}\right) \in \mathcal{O}\left(n\right)$, nous pouvons conclure que $3\left(n - n^{log_3(2)}\right) \in \Theta\left(n\right)$.

Nous pouvons aussi conclure que $C\left(n\right) \in \Theta\left(n\right)$ pour $n = 3^k \; \forall k \in \mathbb{N}$

Démonstration de la validité pour tout entier n

Nous pouvons utiliser la règle de l'harmonie pour montrer que la notation asymptotique que nous avons trouvée pour les $n = 3^k$ est valide $\forall n \in \mathbb{N}$. Pour ce faire, nous devons montrer 3 choses :

1. $3\left(n - n^{log_3(2)}\right)$ doit être éventuellement non décroissante.
   C'est le cas, en particulier lorsque $n \geq 1$.
2. L'ordre de croissance, doit être une fonction harmonieuse, donc que $2n \in \Theta\left(n\right)$.
   C'est le cas puisque

$$
\begin{aligned}
& \lim_{x \to \infty} \frac{2n}{n} \\
= & \quad \langle \text{ Simplification } \rangle \\
& \lim_{x \to \infty} 2 \\
= & \quad \langle \text{ Application de la limite } \rangle \\
& 2 \quad \langle \text{ Valeur constante } > 0,\ 2n \in \Theta\left(n\right) \rangle
\end{aligned}
$$

3. On doit avoir $C(n) \in \Theta(n)$ pour $n = b^k \ \forall k \in \mathbb{N}$

   C'est le cas puisque nous l'avons démontré dans la section "Notation asymptotique" avec $b = 3$.

Puisque nous répondons aux trois critères, nous pouvons conclure que $C(n) \in \Theta(n) \ \forall n \in \mathbb{N}$.

Sylvain, Raphaël
111 124 564

Question 3

Algorithme 1

Nous allons séparer l'analyse de cet algorithme en deux blocs distincts. Le bloc A portera sur les boucles `for` imbriquées. Le bloc B portera sur la dernière boucle `for`. Les opérations restantes sont des opérations atomiques et ne changeront donc pas l'analyse asymptotique.

Bloc A

Taille de l'instance

La taille de l'instance est n, la valeur fournie en paramètre.

Opérateur de base

Nous allons prendre comme opérateur de base l'addition de c avec 1 (`c+1`).

Le nombre de fois que cet opérateur est atteint dépend uniquement de la valeur de n.

Le nombre de fois que l'opérateur de base est appelé peut être donné par la sommation suivante :

$C_A(n) = \sum_{i=1}^{n} \sum_{j=i^3}^{n^3} 1$

Résolvons cette équation.

$$
\begin{aligned}
& C_A(n) \\
= & \qquad \langle \text{ Définition de } C_A(n) \rangle \\
& \sum_{i=1}^{n} \sum_{j=i^3}^{n^3} 1 \\
= & \qquad \langle \text{ Sommation d'une constante } \rangle \\
& \sum_{i=1}^{n} ((n^3 - i^3 + 1) \cdot 1) \\
= & \qquad \langle \text{ Simplification } \rangle \\
& \sum_{i=1}^{n} (n^3 - i^3 + 1)
\end{aligned}
$$

Analyse asymptotique

Nous allons borner la sommation par une intégrale.
La sommation est non croissante.
Avec $f(i) = \sum_{i=1}^{n} (n^3 - i^3 + 1)$;
Avec $g(x) = \int_{1}^{n+1} (n^3 - x^3 + 1)\, dx$;
Avec $h(x) = \int_{0}^{n} (n^3 - x^3 + 1)\, dx$;
Résolvons $g(x) \leq f(i) \leq h(x)$, l'approximation par intégrale d'une sommation nulle part croissante :
Résolvons $g(x) = \int_{1}^{n+1} (n^3 - x^3 + 1)\, dx$, soit une borne inférieure de $\sum_{i=1}^{n} (n^3 - i^3 + 1)$.

$$
\begin{array}{ll}
& C_A(n) \\
= & \quad \langle \text{ Définition de } C_A(n) \rangle \\
& f(i) \\
\geq & \quad \langle \text{ Définition de } g(x) \rangle \\
& g(x) \\
= & \quad \langle \text{ Définition de } g(x) \rangle \\
& \int_{1}^{n+1} (n^3 - x^3 + 1)\, dx \\
= & \quad \langle \text{ Distribution de l'intégration } \rangle \\
& \int_{1}^{n+1} n^3 dx - \int_{1}^{n+1} x^3 dx + \int_{1}^{n+1} 1 dx \\
= & \quad \langle \text{ Intégration } \rangle \\
& [n^3 x]_1^{n+1} - \left[\frac{x^4}{4}\right]_1^{n+1} dx + [x]_1^{n+1} dx \\
= & \quad \langle \text{ Évaluation } \rangle \\
& (n^3(n+1) - n^3(1)) - \left(\frac{(n+1)^4}{4} - \frac{1^4}{4}\right) + (n + 1 - 1) \\
= & \quad \langle \text{ Simplification } \rangle \\
& (n^4 + n^3 - n^3) - \left(\frac{(n+1)^4 - 1}{4}\right) + n \\
= & \quad \langle \text{ Simplification } \rangle
\end{array}
$$

$$
\begin{aligned}
& n^4 - \frac{\left(n^2+2n+1\right)^2-1}{4} + n \\
= & \qquad \langle \text{ Simplification } \rangle \\
& n^4 - \frac{n^4+4n^3+6n^2+4n+1-1}{4} + n \\
= & \qquad \langle \text{ Simplification } \rangle \\
& n^4 - \frac{n^4+4n^3+6n^2+4n}{4} + n \\
= & \qquad \langle \text{ Mettre sur la même base } \rangle \\
& \frac{4n^4}{4} - \frac{n^4+4n^3+6n^2+4n}{4} + \frac{4n}{4} \\
= & \qquad \langle \text{ Ramener sur la même division } \rangle \\
& \frac{4n^4-\left(n^4+4n^3+6n^2+4n\right)+4n}{4} \\
= & \qquad \langle \text{ Distribuer la soustraction } \rangle \\
& \frac{4n^4-n^4-4n^3-6n^2-4n+4n}{4} \\
= & \qquad \langle \text{ Simplifier } \rangle \\
& \frac{3n^4-4n^3-6n^2}{4} \\
= & \qquad \langle \text{ Extraction de } \frac{n^2}{4} \rangle \\
& \frac{n^2}{4}\left(3n^2-4n-6\right) \\
\geq & \qquad \langle \ \forall n \geq 6 \ \rangle \\
& \frac{n^2}{4}\left(3n^2-4n-n\right) \\
= & \qquad \langle \text{ Simplification } \rangle \\
& \frac{n^2}{4}\left(3n^2-3n\right) \\
= & \qquad \langle \text{ Simplification } \rangle \\
& \frac{n^3}{4}\left(3n-3\right) \\
\geq & \qquad \langle \ \forall n \geq 3 \ \rangle \\
& \frac{n^3}{4}\left(3n-n\right) \\
= & \qquad \langle \text{ Simplification } \rangle \\
& \frac{n^3}{4}\left(2n\right) \\
= & \qquad \langle \text{ Simplification } \rangle
\end{aligned}
$$

$$
\begin{array}{ll}
 & \frac{2n^4}{4} \\
= & \qquad \langle \text{ Simplification } \rangle \\
 & \frac{n^4}{2} \\
\in & \qquad \langle \text{ Définition de } \Omega \rangle \\
 & \Omega\left(n^4\right)
\end{array}
$$

Résolvons $h(x) = \int_0^n \left(n^3 - x^3 + 1\right) dx$, soit une borne supérieure de $\sum_{i=1}^{n} \left(n^3 - i^3 + 1\right)$.

$$
\begin{array}{ll}
 & C_A\left(n\right) \\
= & \qquad \langle \text{ Définition de } C_A\left(n\right) \rangle \\
 & f(i) \\
\leq & \qquad \langle \text{ Définition de } h(x) \rangle \\
 & h(x) \\
= & \qquad \langle \text{ Définition de } h(x) \rangle \\
 & \int_0^n \left(n^3 - x^3 + 1\right) dx \\
= & \qquad \langle \text{ Distribution de l'intégration } \rangle \\
 & \int_0^n n^3 dx - \int_0^n x^3 dx + \int_0^n 1 dx \\
= & \qquad \langle \text{ Intégration } \rangle \\
 & \left[n^3 x\right]_0^n - \left[\frac{x^4}{4}\right]_0^n dx + \left[x\right]_0^n dx \\
= & \qquad \langle \text{ Évaluation } \rangle \\
 & \left(n^3(n) - n^3(0)\right) - \left(\frac{(n)^4}{4} - \frac{0^4}{4}\right) + (n - 0) \\
= & \qquad \langle \text{ Simplification } \rangle \\
 & n^4 - \frac{n^4}{4} + n \\
= & \qquad \langle \text{ Simplification } \rangle \\
 & \frac{3}{4}n^4 + n \\
\leq & \qquad \langle \forall n \geq 0 \rangle \\
 & \frac{3}{4}n^4 + n^4 \\
= & \qquad \langle \text{ Simplification } \rangle
\end{array}
$$

$$\begin{array}{ll} & \frac{7}{4}n^4 \\ \in & \qquad \langle \text{ Définition de } \mathcal{O} \rangle \\ & \mathcal{O}(n^4) \end{array}$$

Donc $C_A(n) \in \Theta(n^4)$, puisque $f(i) \in \Omega(n^4)$ et $f(i) \in \mathcal{O}(n^4)$

Bloc B

Taille de l'instance

La taille de l'instance est n, la valeur fournie en paramètre.

Opérateur de base

Nous allons prendre comme opérateur de base l'addition de c avec 1 (`c+1`).

Le nombre de fois que cet opérateur est atteint dépend uniquement de la valeur de n.

Le nombre de fois que l'opérateur de base est appelé peut être donné par la sommation suivante :

$C_B(n) = \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} 1$

Résolvons cette équation.

$$\begin{array}{ll} & C_B(n) \\ = & \qquad \langle \text{ Définition de } C_B(n) \rangle \\ & \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} 1 \\ = & \qquad \langle \text{ Sommation d'une constante } \rangle \\ & ((\lfloor\sqrt{n}\rfloor - 1 + 1) \cdot 1) \\ = & \qquad \langle \text{ Simplification } \rangle \\ & \lfloor\sqrt{n}\rfloor \end{array}$$

Analyse asymptotique

En utilisant la définition de la fonction plancher, nous avons :

$\sqrt{n} - 1 < C_B(n) \leq \sqrt{n}$

Utilisons ces inéquations pour trouver la notation asymptotique de $C_B(n)$

Trouvons Ω

$$
\begin{aligned}
& C_B(n) \\
= & \qquad \langle \text{ Définition de } C_B(n) \rangle \\
& \lfloor \sqrt{n} \rfloor \\
> & \qquad \langle \text{ Définition de la fonction plancher} \rangle \\
& \sqrt{n} - 1 \\
\geq & \qquad \langle \ \forall n \geq 4 \ \rangle \\
& \sqrt{n} - \frac{\sqrt{n}}{2} \\
= & \qquad \langle \text{ Simplification} \rangle \\
& \frac{\sqrt{n}}{2} \\
= & \qquad \langle \text{ Définition de la racine carrée} \rangle \\
& \frac{n^{\frac{1}{2}}}{2} \\
\in & \qquad \langle \text{ Définition } \Omega \rangle \\
& \Omega\left(n^{\frac{1}{2}}\right)
\end{aligned}
$$

Trouvons $\mathcal{O}$

$$
\begin{aligned}
& C_B(n) \\
= & \qquad \langle \text{ Définition de } C_B(n) \rangle \\
& \lfloor \sqrt{n} \rfloor \\
\leq & \qquad \langle \text{ Définition de la fonction plancher} \rangle \\
& \sqrt{n} \\
= & \qquad \langle \text{ Définition de la racine carrée} \rangle \\
& n^{\frac{1}{2}} \\
\in & \qquad \langle \text{ Définition } \mathcal{O} \rangle \\
& \mathcal{O}\left(n^{\frac{1}{2}}\right)
\end{aligned}
$$

Donc $C_B(n) \in \Theta\left(n^{\frac{1}{2}}\right)$

Résultat final

Nous avons pour le bloc A que $C_A(n) \in \Theta(n^4)$ et pour le bloc B que $C_B(n) \in \Theta\left(n^{\frac{1}{2}}\right)$.

Puisque chacun des blocs s'exécute une fois, nous pouvons conclure $C(n) \in \Theta(n^4) + \Theta\left(n^{\frac{1}{2}}\right)$.

En utilisant la règle du maximum, nous pouvons avoir : $C(n) \in \Theta(n^4)$.

Algorithme 2

Taille de l'instance

La taille de l'instance est n, le nombre d'éléments dans l'étendue, soit 1 plus la différence entre le paramètre r (position d'un élément à droite) et le paramètre l (position d'un élément à gauche).

$n = (r - l) + 1$

Opérateur de base

L'opérateur de base à utiliser est la comparaison `l < r` du premier `if`. C'est l'opération exécuter le plus souvent, car il n'y a aucune boucle dans la méthode et que c'est la première opération effectuée.

Définition de la récurrence

Le nombre $C(n)$ d'opérations de base effectuées sur un vecteur est donné par la récurrence suivante :

$$C(n) = \begin{cases} 1 & \text{si } n \leq 2 \\ 1 + 3 \cdot C\left(n - \left\lfloor \frac{n}{3} \right\rfloor\right) & \text{si } n > 2 \end{cases}$$

Explication de la récurrence :

1. Si $n = 1 + (l - r) \leq 1$, alors on effectue la comparaison (opérateur de base), puis on sort de la méthode directement.
2. Si $n = 1 + (l - r) = 2$, alors on effectue la comparaison (opérateur de base), puis on effectue une deuxième comparaison (des valeurs), en effectuant ou non un échange, on n'entre pas par la suite dans dernier `if`, puis on sort de la méthode.
3. Si $n = l - r \geq 3$, alors on effectue la comparaison (opérateur de base), puis on effectue une deuxième comparaison (des valeurs), en effectuant ou non un échange, on entre par la suite dans le dernier `if`, ou l'on va appeler trois fois la récursion. Deux fois en retirant $k = \left\lfloor \frac{n}{3} \right\rfloor$ de la valeur de r et une fois en l'ajoutant à l, ce qui a pour effet de retirer retirer k à n, d'où le $n - \left\lfloor \frac{n}{3} \right\rfloor$.

Résolution de la récurrence (pour notation asymptotique)

Supposons $n = 3^k \equiv log_3(n) = k$.
Nous pouvons redéfinir $C(n)$ ainsi :

$$
\begin{aligned}
& C(n) \\
= & \quad \langle \text{ Définition de } C(n) \rangle \\
& 1 + 3 \cdot C\left(n - \left\lfloor \tfrac{n}{3} \right\rfloor\right) \\
= & \quad \langle \text{ Définition de } n \rangle \\
& 1 + 3 \cdot C\left(3^k - \left\lfloor \tfrac{3^k}{3} \right\rfloor\right) \\
= & \quad \langle \text{ Règle des exposants } \rangle \\
& 1 + 3 \cdot C\left(3^k - \left\lfloor 3^{k-1} \right\rfloor\right) \\
= & \quad \langle \text{ Fonction plancher sur nombre entier } \rangle \\
& 1 + 3 \cdot C\left(3^k - 3^{k-1}\right) \\
= & \quad \langle \text{ Règle des exposants } \rangle \\
& 1 + 3 \cdot C\left(3 \cdot 3^{k-1} - 3^{k-1}\right) \\
= & \quad \langle \text{ Simplification } (3x - x = 2x) \rangle \\
& 1 + 3 \cdot C\left(2 \cdot 3^{k-1}\right) \\
= & \quad \langle \text{ Définition de } k \rangle \\
& 1 + 3 \cdot C\left(2 \cdot 3^{log_3(n)-1}\right) \\
= & \quad \langle \text{ Propriété des logs } \rangle \\
& 1 + 3 \cdot C\left(2 \cdot \tfrac{3^{log_3(n)}}{3^1}\right) \\
= & \quad \langle \text{ Simplification } \rangle \\
& 1 + 3 \cdot C\left(2 \cdot \tfrac{n}{3}\right) \\
= & \quad \langle \text{ Réécriture } \rangle \\
& 1 + 3 \cdot C\left(\tfrac{n}{\frac{3}{2}}\right) \\
= & \quad \langle \text{ Réécriture } \rangle \\
& 3 \cdot C\left(\tfrac{n}{\frac{3}{2}}\right) + 1
\end{aligned}
$$

$$
\begin{array}{ll}
\in & \langle \text{ Théorème général avec } r = 3,\ b = \frac{3}{2} \text{ et } d = 0 \\
 & \quad \text{et } r = 3 > b^d = \left(\frac{3}{2}\right)^0 = 1 \\
 & \quad d = 0, \text{ puisque } 1 \in \Theta\left(n^0\right) \rangle \\
 & \Theta\left(n^{log_{\frac{3}{2}}3}\right) \\
\simeq & \langle \text{ Évaluation de } log_{\frac{3}{2}}3 \rangle \\
 & \Theta\left(n^{2.709511}\right)
\end{array}
$$

Sylvain, Raphaël
111 124 564

Question 4

Analyse de `construire_noeud`

Notons que cette fonction possède une fonction privée récursive (`build_node`) et que la seule opération de `construire_noeud` est l'appel à cette fonction privée. Nous évaluons donc la fonction `build_node`.

Taille de l'instance

La taille de l'instance est n, le nombre d'éléments compris entre l'index `beginIndex` et l'index `endIndex` inclusivement. ($n =$`endIndex`$-$`beginIndex`$+1$)

Opération de base

Nous devons choisir comme opération de base l'appel à la fonction `fusion`.

Cette opération est l'opération la plus effectuée dans la fonction à une constante prêt, car il n'y a pas de boucles itératives suivant cette opération et qu'elle est imbriquée dans une conditionnels mais dont il n'y a pas d'autre traitement autour.

Le nombre d'opérations dépend uniquement dans la valeur de n.

Analyse de la récursion

La récurrence peut être donnée par la fonction suivante

$$C(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ C\left(\left\lceil \frac{n}{2} \right\rceil\right) + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C_{fusion}(n) & \text{si } n > 1 \end{cases}$$

Nous devons donc commencer par analyser la méthode de $C_{fusion}(n)$.

Analyse de `fusion`

Taille de l'instance

La taille de l'instance est n, le nombre de feuille dans l'arbre du noeud, soit la somme des feuilles de son noeud gauche et de son noeud droit.

Opération de base

Remarquons que tous les opérations dans cette méthodes sont de complexité constante (ou constante ammorti).

Nous prendrons comme opération de base la comparaison `currentParentIndex < nbLeafsParent` de la boucle `for`.

Cette opération est la plus appelé de la méthode, puisqu'il n'y a qu'une seule boucle et que c'est l'opérateur de sortie de cette boucle.

Il n'y a pas de meilleur ou de pire cas, le nombre de fois qu'est appelé cette opération peut être donné par la sommation suivante :

$C_{fusion}(n) = \sum_{i=0}^{n-1} 1$

Résolvons cette sommation

$$
\begin{aligned}
& C_{fusion}(n) \\
= & \qquad \langle \text{ Définition de } \texttt{fusion} \rangle \\
& \sum_{i=0}^{n-1} 1 \\
= & \qquad \langle \text{ Sommation d'une constante} \rangle \\
& (n-1-0+1)\cdot 1 \\
= & \qquad \langle \text{ Simplification} \rangle \\
& n
\end{aligned}
$$

Donc $C_{fusion}(n) \in \Theta(n)$

Retour à l'analyse de `construire_noeud`

Nous modifions la récurrence ainsi :

$$
\begin{aligned}
& C(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ C\left(\left\lceil \frac{n}{2} \right\rceil\right) + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C_{fusion}(n) & \text{si } n > 1 \end{cases} \\
\Rightarrow & \qquad \langle \text{ En supposant que } n = 2^k \rangle \\
& C\left(2^k\right) = \begin{cases} 0 & \text{si } k = 0 \\ C\left(\left\lceil \frac{2^k}{2} \right\rceil\right) + C\left(\left\lfloor \frac{2^k}{2} \right\rfloor\right) + C_{fusion}\left(2^k\right) & \text{si } k \geq 1 \end{cases} \\
\Rightarrow & \qquad \langle \text{ Puisque } \tfrac{2^k}{2} \text{ est toujours un entier} \rangle
\end{aligned}
$$

$$C\left(2^k\right) = \begin{cases} 0 & \text{si } k = 0 \\ 2 \cdot C(\frac{2^k}{2}) + C_{fusion}\left(2^k\right) & \text{si } k \geq 1 \end{cases}$$

$\Rightarrow$ $\langle$ Ramener sur n $\rangle$

$$C\left(n\right) = \begin{cases} 0 & \text{si } n \leq 1 \\ 2 \cdot C(\frac{n}{2}) + C_{fusion}\left(n\right) & \text{si } n > 1 \wedge n = 2^k \forall k \in \mathbb{N}^+ \end{cases}$$

$\Rightarrow$ $\langle$ Définition de $C_{fusion}\left(n\right)$ $\rangle$

$$C\left(n\right) = \begin{cases} 0 & \text{si } n \leq 1 \\ 2 \cdot C(\frac{n}{2}) + n & \text{si } n > 1 \wedge n = 2^k \forall k \in \mathbb{N}^+ \end{cases}$$

Nous avons ainsi une forme où le théorème général avec
$r = 2 \wedge b = 2 \wedge f\left(n\right) = n^1 \wedge d = 1$
Nous obtenons la forme 3 du théorème général
$r = 2 = 2 = 2^1 = b^d$
donc
$C\left(n\right) \in \Theta\left(n^1 \log n\right) \equiv C\left(n\right) \in \Theta\left(n \log n\right)$

Conclusion

La fonction `construire_noeud` s'exécute donc en temps $n \log n$ par rapport au nombre de points.