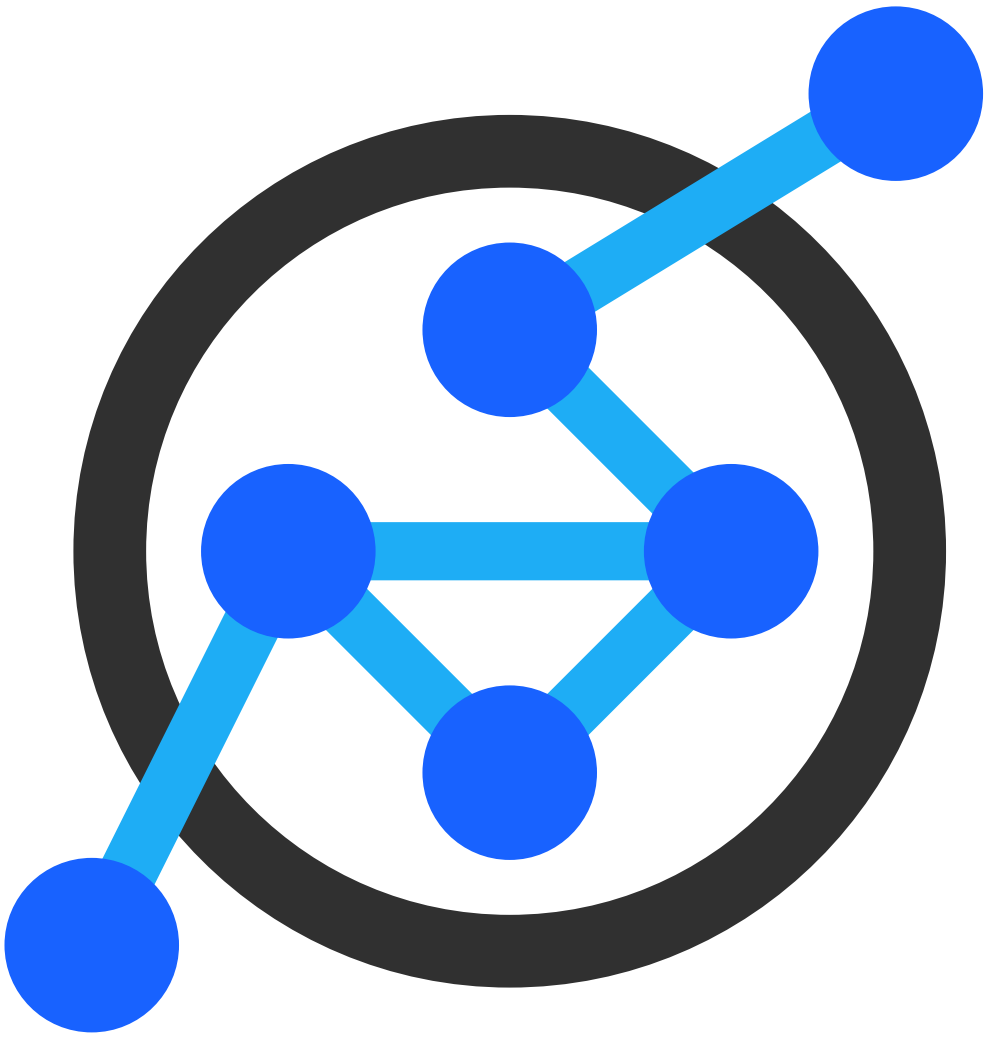
# Deuxième Tour Théorique

# Tâches



Swiss Olympiad in Informatics

6 et 7 mars 2021

# Instructions

— Tu as 5 heures et 30 minutes pour résoudre les tâches. Réserve suffisamment de temps pour envoyer tes réponses. Nous recommandons de réserver 30 minutes pour le scan et la mise en ligne. Si tu finis en avance, tu peux continuer à travailler sur les tâches et envoyer tes solutions à nouveau plus tard.
— Le concours est à livres ouverts : tu peux utiliser internet, des livres, d'anciennes solutions, etc.
— Si tu as des questions pendant le concours, pose-les par e-mail (`info@soi.ch`). Nous ne donnerons pas d'indices, donc essaie de formuler des questions claires auxquelles il est possible de répondre par oui ou par non.
— Ta solution doit être entièrement rédigée à la main sur du papier.
— Écris lisiblement.
— La solution doit se suffire à elle-même : tu ne peux pas faire références à des algorithmes connus, des blogs ou des articles, à l'exception des sections "Premier tour" et "Deuxième tour" du Wiki SOI.
— Envoie un PDF par tâche sur le grader, avec une assez bonne résolution pour que nous puissions tout lire. Quelques mégabytes par tâche devraient suffire (la limite est de 100 MiB).
— Nous ne consultons que la dernière soumission pour chaque tâche. Tu peux en envoyer autant que tu veux.
— Tu peux voir un aperçu de ce que tu as envoyé en cliquant sur "Detail", afin de vérifier que tu as envoyé le bon fichier.
— En cas de problème technique avec le scanner ou le grader, tu peux nous prouver que tu as fini à temps en remettant une photo à basse résolution. Envoie-la à Johannes Kapfhammer par e-mail, Discord ou Signal/WhatsApp.

## Évaluation

Les solutions sont évaluées selon des critères similaires à ceux du premier tour théorique. Les deux critères les plus importants sont la correction et le temps d'exécution asymptotique. La qualité de la description et les arguments prouvent la correction sont également pris en compte.

Tu peux toujours te référer à du contenu du Wiki SOI ou du tour 2H. Tu ne dois pas expliquer pourquoi, par exemple, l'algorithme de Dijkstra fonctionne, mais tu dois expliquer pourquoi et comment il peut être appliqué. Dans le cas de Dijkstra, tu devrais expliquer clairement sur quel graphe tu l'utilises et noter que les poids des arêtes de celui-ci ne sont pas négatifa.

L'algorithme doit être décrit en suffisamment de détail pour qu'il soit aisé de convertir la description en un programme. Écris également quelles structures de données tu utiliserais. Habituellement, le mieux est d'écrire un bref pseudocode.

Si un algorithme t'est demandé, tu devrais utiliser cette structure comme ligne directrice :

1. Décris l'idée derrière un algorithme qui résout le problème.
2. Donne du pseudocode ou explique comment on pourrait implémenter l'algorithme.
3. Prouve par des arguments la correction de l'approche utilisée.
4. Indique le temps d'exécution asymptotique ainsi que l'utilisation de mémoire (en les justifiant).

Si une partie de ta solution est utile pour plusieurs sous-tâches, il suffit de l'écrire une seule fois et de t'y référer à partir de là. Note, cependant, qu'un algorithme peut résoudre plusieurs sous-tâches mais pas nécessairement de façon optimale.

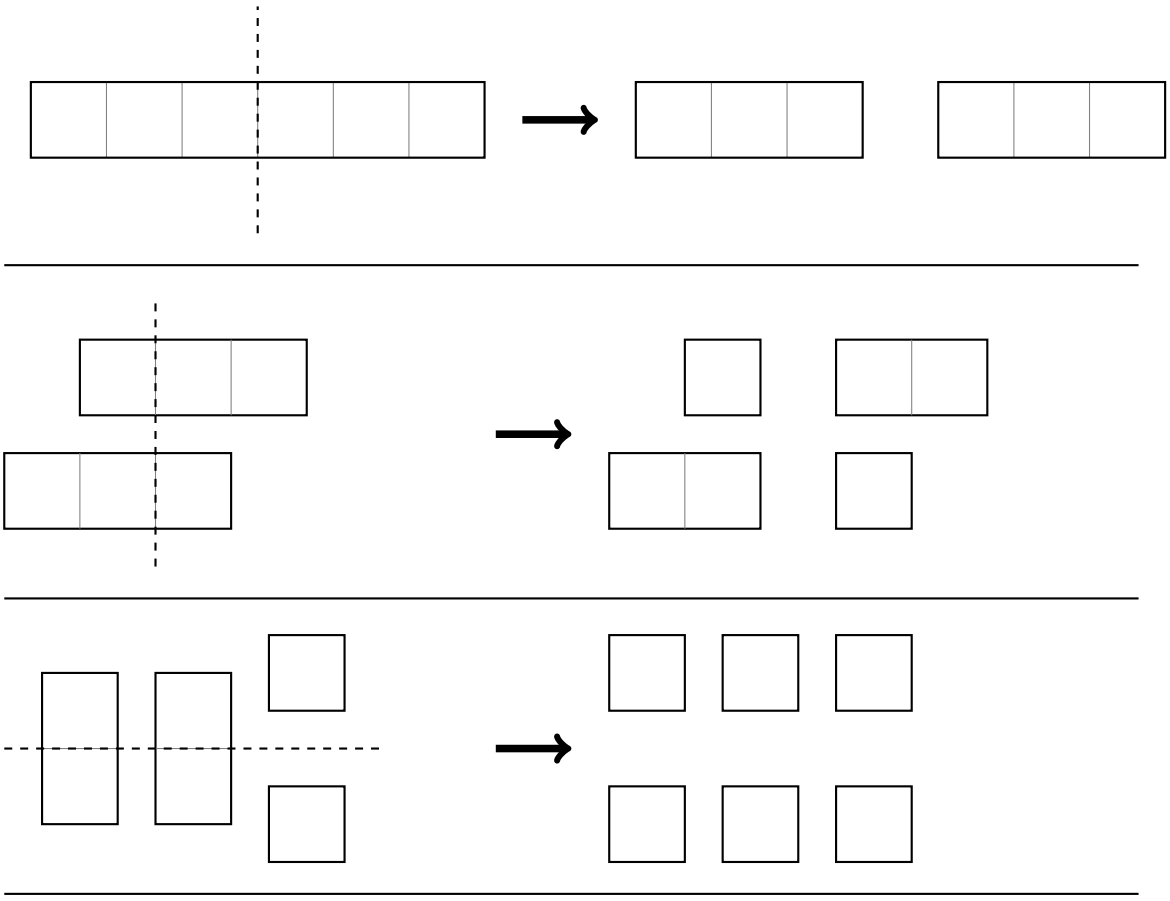
**Bonne chance !**

# Découpeur laser

La souris Binna a reçu récemment un découpeur laser. Grâce à ce dernier, elle aimerait couper une barre de chocolat en $n \cdot m$ carrés, chacun de taille $1 \times 1$. La machine fonctionne comme suit : Binna peut aligner un ou plusieurs morceaux de chocolat dans un plan. Elle peut les déplacer et/ou les tourner. Ensuite, le découpeur se déplace en ligne droite et coupe tous les morceaux par-dessus lesquels il passe. (En d'autres termes, en une étape, tu peux faire une coupe en ligne droite sur un nombre arbitraire de morceaux.) Quel est le nombre minimum de coupes que Binna doit utiliser ?

Formellement, tu reçois un rectangle de taille $n \times m$ et dois trouver le nombre minimal d'opérations pour le découper en carrés de taille $1 \times 1$. Avant chaque opération, tu peux d'abord orienter chaque morceau en le déplaçant et/ou en lui appliquant une rotation. Ensuite, tu choisis une ligne et chaque pièce qui a une intersection avec cette ligne est coupée en deux morceaux le long de la ligne. Cela conclut une opération.

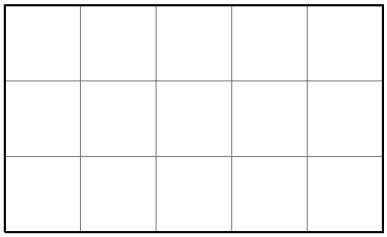
### Découper une barre de $1 \times 6$ en 3 coupes



— Le traitillé montre où la coupe est faite.
— D'une ligne à l'autre, les morceaux sont déplacés.

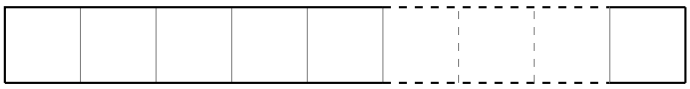
Le nombre total de coupes pour cette barre de $1 \times 6$ était 3, et il est possible de montrer que c'est optimal.

## Sous-tâche 1: Une barre de $3 \times 5$ (10 points)



Indique le nombre minimal de coupes dont Binna a besoin pour couper cette plaque de chocolat de dimensions $3 \times 5$ en 15 morceaux de taille $1 \times 1$.

## Sous-tâche 2: Une plaque longue et fine (35 points)



Pour une barre de chocolat de taille $1 \times m$, donne un algorithme qui calcule le nombre de coupes minimal. Prouve qu'il est correct et indique son temps d'exécution asymptotique et son utilisation de mémoire.

## Sous-tâche 3: Cas général (55 points)

Donne un algorithme qui calcule le nombre minimal de coupes pour une plaque de chocolat de dimensions $n \cdot m$. Prouve qu'il est correct et indique son temps d'exécution asymptotique et son utilisation de mémoire.

# Avion

Soit un avion avec $R \cdot C$ sièges, organisés en $R$ rangées de $C$ sièges. On peut assigner à chaque siège des coordonnées uniques $(r, c)$ (telles que $1 \leq r \leq R, 1 \leq c \leq C$). On dit que deux sièges sont voisins s'ils sont directement adjacents dans la même rangée ou si l'un est juste devant l'autre. Formellement, deux sièges $(r_i, c_i)$ et $(r_j, c_j)$ sont voisins si et seulement si $|r_i - r_j| + |c_i - c_j| = 1$.

La grande famille de Stofl, soit $R \cdot C$ souris numérotées de 1 à $R \cdot C$, organisent un voyage à Singapour. Comme c'est une grande famille, certains de ses membres pourraient ne pas s'entendre avec certains autres.

En particulier, on te donne deux fonctions : `likes(m)` retourne l'ensemble de souris que la souris $m$ apprécie ($1 \leq m \leq R \cdot C$), et `num(m)` retourne la taille de `likes(m)` (c'est-à-dire `num(m)` = |`likes(m)`|). Il est garanti que cette relation est symétrique : si la souris $a$ apprécie la souris $b$, alors la souris $b$ apprécie aussi la souris $a$, c'est-à-dire $a \in$ `likes(b)` $\iff b \in$ `likes(a)`. Toutes les autres paires de souris se détestent l'une l'autre.

Tu veux aider Stofl à faire de ce voyage un succès. Aide-le à trouver une répartition des sièges telle que les conditions suivantes soient satisfaites :

— Chaque souris est assise dans son propre siège.

— Toutes les paires de souris qui s'apprécient l'une l'autre occupent des sièges voisins l'un de l'autre.

— Aucune paire de souris qui se détestent l'une l'autrent occupent des sièges voisins l'un de l'autre.

S'il y a plusieurs répartitions des sièges qui conviennent, tu peux trouver n'importe laquelle. Appelle la fonction `assign(m, r, c)` pour assigner la souris $m$ au siège $(r, c)$. S'il n'y a pas de répartition de la sorte, annonce qu'il n'y en a aucune en appelant la fonction `impossible()` (cela ignorera les éventuels appels `assign(m, r, c)` antérieurs).

Tu peux supposer pour l'analyse que `num`, `assign` et `impossible` ont un temps d'exécution constant et que `likes` a un temps d'exécution et une utilisation de mémoire de $O($`num(m)`$)$.

## Sous-tâche 1: Exemple concret (20 points)

Soit $R = 3$ et $C = 3$ et les $N = 12$ paires de souris suivantes :

$$\{1,2\}, \{5,6\}, \{9,7\}, \{8,9\}, \{1,3\}, \{3,4\}, \{8,1\}, \{4,5\}, \{6,2\}, \{4,2\}, \{6,7\}, \{9,2\}.$$

Les souris peuvent-elles être placées dans l'avion sans contrevenir aux règles ci-dessus ?

## Sous-tâche 2: Un avion avec un siège par rangée (30 points)

Dans cette sous-tâche, suppose que $C = 1$ (c'est-à-dire qu'il n'y a qu'un siège par rangée), mais un nombre arbitraire de rangées $R$.

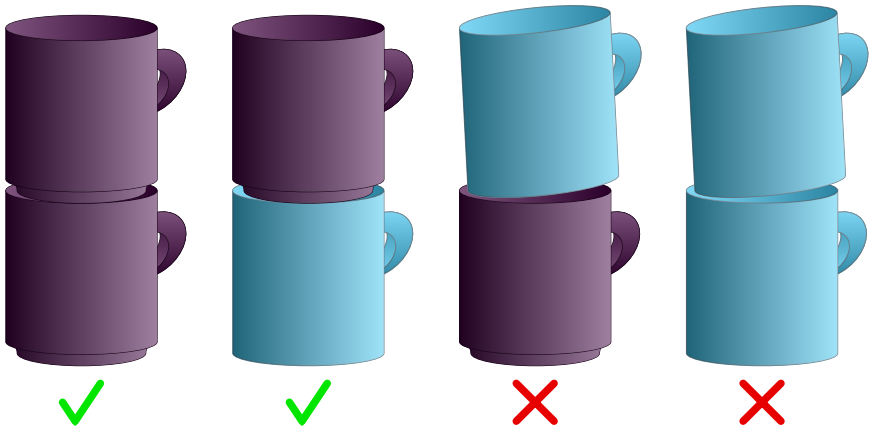
## Sous-tâche 3: Cas général (50 points)

Dans cette sous-tâche, il n'y a aucune contrainte sur $R$ ou $C$.

# Empilement de tasses

Souris Stofl a $2n$ tasses dans son armoire. Les tasses sont arrangées de gauche à droite sur une ligne et sont numérotés de 1 à $2n$. Comme souris Stofl manque de place horizontalement, il aimerait empiler ces tasses.

Quelques tasses ont une base parfaitement plate, tandis que d'autres sont courbés vers le centre. Des tasses plates ne peuvent pas être empilés sur d'autres tasses, car le tas résultant serait très instable. En revanche, des tasses courbés peuvent être empilés sur d'autres tasses sans problèmes. Dans l'image ci-dessous, les deux tas à gauche sont permis, alors que les deux tas à droite ne le sont pas.



Souris Stofl veut empiler ces tasses en $n$ tas de hauteur 2. Pour empiler la tasse $i$ sur la tasse $j$, souris Stofl a besoin de $|i - j|$ secondes. Trouve le temps minimal dont Stofl a besoin pour empiler ses tasses.

Pour l'analyse, tu peux supposer avoir une fonction type($i$) qui retourne le type de la tasse $i$ ($l \leq i \leq 2n$), en temps et espace constant.

## Sous-tâche 1: Résout un exemple (10 points)

Comment Stofl peut-il empiler les tasses de l'arrangement suivant en un temps minimal ? Tu ne dois pas prouver ta réponse.

## Sous-tâche 2: Association parfaite (40 points)

Exactement la moitié des tasses ont une base plate. Développe un algorithme qui trouve le coût minimal. Prouve sa justesse et donne le temps d'exécution et l'utilisation de mémoire asymptotique.

## Sous-tâche 3: L'armoire de Stofl (50 points)

L'armoire de Stofl ne satisfait pas nécessairement les contraintes additionnelles de la sous-tâche précédente. Néanmoins, il est garanti qu'au moins $n$ tasses ont une base courbée. Développe un

algorithme qui trouve le coût minimal pour le cas général. Prouve sa justesse et donne le temps d'exécution et l'utilisation de mémoire asymptotique.

# Astrophysique

Dr. Binna est une astrophysicienne expérimentale célèbre. Son domaine de recherche est la relativité générale. Les détails de ses derniers travaux sont si complexes que même les experts ont du mal à comprendre les détails, mais les travaux peuvent être simplifiés de telle manière : son but est de trouver les trous noirs et les trous blancs.

Intuitivement, un trou noir exerce une grande *force gravitationelle* sur son entourage ; de chaque point de l'univers qui n'est pas un trou noir, il existe un champ gravitationel qui fait parvenir lentement le point vers le trou noir. Aussi, une fois un point arrivé dans le trou noir, impossible d'en ressortir [1].

L'object astrophysique dual au trou noir est le trou blanc. Contrairement à un trou noir, un trou blanc est très *répulsif*. Intuitivement, pour chaque point de la galaxie qui n'est pas dans un trou blanc, et un deuxième point dans un trou blanc, il existe un champ gravitationel qui fait venir le deuxième point vers le premier. De plus, il est impossible de revenir dans un trou blanc [2].

Il est connu que la souris Binna s'intéresse à un ensemble de $n \geq 2$ points dans la galaxie, où il y a au plus un trou noir et au plus un trou blanc.

En utilisant un procédé expérimental et très théorique à la fois, la souris Binna peut, étant donné deux points $a$ et $b$, dire si il est possible de rejoindre simplement le point $b$ en commençant au point $a$ [3].

**Note importante :** La notion présentée au paragraphe précédent de *pouvoir être rejoint simplement depuis un point* est basée sur de l'astrophysique d'un niveau tellement avancé que personne ne peut vraiment comprendre. La seule règle que l'on pense comprendre à coup sûr est qu'un point est simplement joignable depuis lui-même. En particulier, la connaissance de la simple joignabilité de deux paires de points ne permet *pas* de conclure quelque chose sur la simple joignabilité d'une autre paire de points (par exemple, même si $b$ peut être rejoint simplement depuis $a$ et que $c$ peut être rejoint simplement depuis $b$, alors on n'a pas forcément que $c$ peut être rejoint simplement depuis $a$ ; cette relation n'est pas forcément transitive) !

Inspirée par sa méthode, la souris Binna a inventé le concept de trous noirs et blancs *généralisés*. Dans la suite, quand on parlera de trous noirs et trous blancs, on parlera implicitement de trous noirs et blancs généralisés.

Un trou noir est un ensemble de points où chaque point peut être rejoint simplement depuis partout, mais d'où on ne peut pas sortir. Plus précisément :

— Si $x$ est dans le trou noir, alors pour *tout* autre point $y$, $x$ peut être rejoint simplement depuis $y$.
— Si $x$ est dans le trou noir, mais $y$ n'est *pas* dans le trou noir, alors $y$ ne peut *pas* être rejoint simplement depuis $x$.

Un trou blanc est un ensemble de points depuis lesquels tous les autres points peuvent être rejoints simplement, mais les points du trou blanc ne peuvent pas être rejoints simplement depuis l'extérieur. En particulier :

— Si $x$ est dans un trou blanc, alors pour *tout* point $y$, $y$ peut être rejoint simplement depuis $x$.
— Si $x$ est dans un trou blanc mais $y$ ne l'est *pas*, alors $x$ ne peut *pas* être simplement rejoint depuis $y$.

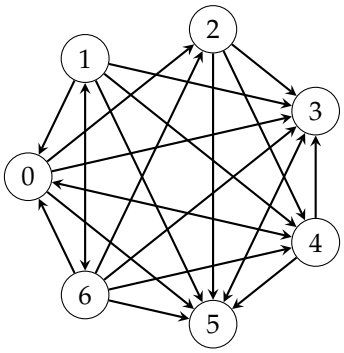
Comme vérifier si un point $y$ peut être rejoint simplement depuis un point $x$ est très difficile, la souris Binna te demande de minimiser le nombre de fois où tu vérifies cette propriété pour une paire de points $(x, y)$.

1. D'après la souris Binna, le domaine extérieur au trou noir ne correspond plus à un point dans l'espace, mais à un moment du passé. Bien évidemment, sortir d'un trou noir serait donc équivalent à voyager dans le temps.
2. La souris Binna pense que cela est dû au fait que l'intérieur d'un trou blanc correspond à un moment dans le passé (par exemple, le Big Bang est en fait simplement un trou blanc).
3. Selon les termes de Binna, il existe un champ gravitationnel dû à la constellation de trous noirs et blancs qui fait venir le point $a$ vers le point $b$.

## Sous-tâche 1: Une très petite galaxie (10 points)

Pour tester sa méthode, la souris Binna a déjà réussi à vérifier pour chaque paire ordonnée de points de la très petite galaxie si l'un peut être rejoint simplement depuis l'autre.

Si un point $b$ peut être rejoint simplement depuis un point $a$, il y a une flèche qui va de $a$ à $b$.




Quel(s) sont les point(s) dans le trou noir et quel(s) sont les points dans le trou blanc ?

## Sous-tâche 2: Un petit trou noir (15 points)

Dans cette sous-partie, il n'y a pas de trou blanc. De plus, il est garanti qu'il y a exactement un trou noir et qu'il y a exactement un point dans ce trou noir. Ton algorithme ne doit *pas* vérifier qu'il n'y a qu'un trou noir constitué d'un seul point. Ton algorithme peut partir du principe qu'il y a bien exactement un trou noir constitué d'exactement un seul point.

Tu peux demander à la souris Binna de mesurer pour un point $a$ et un point $b$ si $b$ peut être rejoint simplement depuis le point $a$. Ton objectif est de trouver le point dans le trou noir en demandant à Binna le plus petit nombre possible de mesures.

Dans cette partie, tu peux ignorer des termes de petit ordre (relativement à l'ordre du terme dominant), mais la constante du terme dominant reste importante. Par exemple, on considère qu'une solution qui utilise $4 \cdot n^2 + 3 \cdot n$ mesures est équivalente à une solution qui utilise $4 \cdot n^2 + 2 \cdot n$ mesures. Par contre, une solution qui utilise $3 \cdot n + 100$ mesures est strictement meilleure qu'une solution qui utilise $4 \cdot n$ mesures. Finalement, une solution où l'ordre du terme dominant est strictement plus grand que l'ordre du terme dominant d'une autre solution est strictement plus mauvaise (par exemple, $n^2$ est strictement plus mauvais que $100 \cdot n$).

Dans cette partie (et aussi les autres parties), on te demande de montrer que ton algorithme est correct et efficace. Tu dois ainsi prouver pourquoi ton algorithme va toujours trouver le bon ensemble de points et que le nombre de mesures est au plus ce que tu affirmes.

En revanche, il n'est *pas* nécessaire de prouver que la solution est optimale en termes de nombre de mesures, mais une solution avec moins de mesures dans le sens expliqué en haut obtiendra plus de points.

## Sous-tâche 3: Un petit trou noir et un petit trou blanc (20 points)

Dans cette partie, il y a toujours un trou noir et un trou blanc. De plus, il y a exactement un point qui est dans le trou noir, et exactement un point qui est dans le trou blanc. De plus, aucun point n'est à la fois dans le trou blanc et dans le trou noir.

Ton algorithme ne doit *pas* vérifier qu'il y a bien un trou noir et un trou blanc et que les deux comportent exactement un point distinct. Ton algorithme peut partir du principe que le trou noir et le trou blanc existent et que les deux consistent chacun exactement en un point distinct.

Ton but est de trouver le point du trou noir et le point du trou blanc avec un nombre de mesures le plus petit possible.

Comme avant, tu ne dois pas prouver que le nombre de mesures est optimal. Cependant, un nombre de mesures plus petit (dans le sens expliqué ci-dessus) donne plus de points.

## Sous-tâche 4: Un grand trou noir (25 points)

Dans cette partie, il n'y a pas de trou noir. Par contre, il existe exactement un trou noir et au moins un point est à l'intérieur du trou noir.

Ton but est de trouver tous les points qui sont dans le trou noir, avec un petit nombre de mesures.

Comme avant, tu ne dois pas prouver que le nombre de mesures est optimal. Cependant, un nombre de mesures plus bas (dans le sens expliqué ci-dessus) donne plus de points.

## Sous-tâche 5: Un grand trou noir et un grand trou blanc (30 points)

Dans cette partie, il y a exactement un trou noir et exactement un trou blanc. Au moins un point est dans le trou noir, et au moins un point est dans le trou blanc. Les points qui sont dans le trou noir sont distincts des points dans le trou blanc.

Ton but est de trouver les points qui sont dans le trou noir et les points qui sont dans le trou blanc avec le plus petit nombre possible de mesures.

Comme avant, tu ne dois pas prouver que le nombre de mesures est optimal. Cependant, un nombre de mesures plus petit (dans le sens expliqué ci-dessus) donne plus de points. Une solution avec au plus $4 \cdot n + O(1)$ mesures peut obtenir au plus 18 points pour cette partie.