**Composition d'Informatique (2 heures), Filière PC (XELC)**

Rapport de jury

# 1 Bilan général

À titre de rappel, cette épreuve n'est corrigée que pour les candidats admissibles. Le présent rapport ne concerne que la filière PC.

Cette année, le nombre total de candidats admissibles dans cette filière est 533. La note moyenne est 10.48 avec un écart-type de 3.59. Les tableaux ci-dessous donnent la répartition détaillée des notes par série, ainsi que la synthèse calculée sur l'ensemble des copies corrigées. La note minimale est 0.5 et la note maximale 20.

| | Série 1 | | Série 2 | | Série 3 | | Série 4 | | Synthèse | |
|---|---|---|---|---|---|---|---|---|---|---|
| $0 \leq N \leq 4$ | 4 | 2% | 5 | 3% | 7 | 5% | 5 | 3% | 21 | 3% |
| $4 < N \leq 8$ | 25 | 17% | 21 | 16% | 23 | 16% | 22 | 17% | 91 | 17% |
| $8 < N \leq 12$ | 73 | 52% | 64 | 48% | 54 | 39% | 60 | 47% | 251 | 47% |
| $12 < N \leq 16$ | 31 | 22% | 34 | 25% | 40 | 29% | 33 | 26% | 138 | 25% |
| $16 < N \leq 20$ | 7 | 5% | 5 | 3% | 9 | 6% | 6 | 4% | 27 | 5% |

| | Série 1 | Série 2 | Série 3 | Série 4 | Synthèse |
|---|---|---|---|---|---|
| Nombre de copies | 140 | 131 | 136 | 126 | 533 |
| Notes minimales | 1.90 | 0.50 | 2.20 | 1.20 | 0.50 |
| Notes maximales | 19.60 | 20.00 | 20.00 | 18.60 | 20.00 |
| Notes moyennes | 10.34 | 10.52 | 10.73 | 10.28 | 10.47 |
| Écart-type | 3.31 | 3.59 | 3.97 | 3.43 | 3.59 |

Les notes des candidats français se répartissent selon le tableau suivant :

| | | |
|---|---|---|
| 0≤N<4 | 16 | 3,19% |
| 4≤N<8 | 84 | 16,73% |
| 8≤N<12 | 242 | 48,21% |
| 12≤N<16 | 129 | 25,70% |
| 16≤N≤20 | 31 | 6,18% |
| Total : | 502 | 100% |
| Nombre de copies : | 502 | |
| Note moyenne : | 10,53 | |
| Écart-type : | 3,48 | |

Le langage de programmation PYTHON a été utilisé pour la totalité des copies.

# 2 Commentaires

Dans cette épreuve, les candidats ont étudié différentes implémentations d'opérateurs de l'algèbre relationnelle. Les opérateurs considérés sont (i) la sélection d'enregistrements dont l'un des champs vaut une constante donnée ; (ii) la sélection d'enregistrements dont deux champs sont égaux ; (iii) la

projection des enregistrements d'une table ; (iv) le produit cartésien de deux tables ; (v) la jointure entre deux tables ; et enfin (vi) la suppression des doublons d'une table.
Dans une première partie, on commence par implémenter ces opérateurs en supposant qu'une table est représentée comme une liste (non triée) d'éléments. Dans une seconde partie, on code des requêtes SQL à l'aide des opérateurs implémentés dans la première partie. Dans une troisième partie, on implémente de nouveau les opérateurs de la logique relationnelle en supposant d'abord que les tables sont représentées par des listes triées selon les données d'une colonne fixée et ne contiennent pas de doublon puis que les tables sont indexées par des dictionnaires PYTHON. Cette dernière partie intitulée "Amélioration des performances" devait mener à l'écriture d'algorithmes plus efficaces que ceux obtenus dans la première partie du sujet.
D'une manière générale, les aspects suivants sont pris en compte (par ordre d'importance) dans la correction :

1. la correction de l'*algorithme* proposé vis-à-vis de la spécification fournie ;
2. l'efficacité de la solution ;
3. la lisibilité du code (on favorise une solution simple devant une solution compliquée équivalente, on apprécie le code bien indenté et structuré, ...) ;
4. la correction des détails d'*implémentation* ;
5. les explications accompagnant le code sous la forme de commentaires.

Les correcteurs tiennent à faire remarquer les points suivants :

— En PYTHON, l'indentation est significative : ne pas indenter de façon suffisamment explicite peut donc rendre la réponse du candidat ambigüe.
— Il faut éviter d'introduire une fonction auxiliaire pour ne finalement pas l'utiliser dans la réponse finale. Le correcteur perd du temps bêtement à comprendre ce code inutile. Rayer proprement une partie inutile de la réponse est préférable dans cette situation.
— Pour améliorer la présentation, il est déconseillé de commencer à écrire un code source en bas de la page pour le poursuivre sur la page suivante. De même, avoir besoin d'écrire un programme sur plusieurs pages doit alerter le candidat : les réponses attendues dépassent très rarement la demi-page.
— Les candidats doivent être attentifs aux fonctions standards autorisées ou interdites par le sujet. Dans le présent sujet, l'ensemble des fonctions standards autorisées sur les listes et sur les dictionnaires était défini très clairement. L'usage de toute fonction extérieure à cet ensemble a été sanctionné.
— Attention à ne pas confondre `for x in l` qui permet à `x` d'itérer sur tous les éléments de `l` et `for i in range(len(l))` qui permet à `i` d'itérer sur tous les indices valides pour `l`.
— Enfin, il est conseillé de ne pas définir de listes par compréhension dans les copies car le coût de création de ces listes est souvent mal compris des candidats.

## 3 Commentaires détaillés

Pour chaque question, un tableau récapitule les taux de réussite avec les conventions suivantes :

— 0 signifie qu'aucun point n'a été donné pour la question (question non traitée ou totalement fausse) ;
— $< 0.5$ signifie que moins de la moitié des points ont été donnés ;
— $\geq 0.5$ signifie que plus de la moitié des points ont été donnés ;
— 1 signifie que la totalité des points ont été donnés.

**Question I.1**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 21 | 3% | 53 | 9% | 41 | 7% | 416 | 78% | 531 | 100% |

Dans cette question, le candidat devait écrire une fonction permettant de sélectionner l'ensemble des enregistrements d'une table dont l'un des champs est égal à une valeur prise en paramètre. Il s'agit d'itérer sur les enregistrements et d'utiliser une instruction conditionnelle pour inclure les enregistrements cherchés dans une *nouvelle* liste renvoyée par la fonction.

Remarques :

— Procéder en deux itérations (la première pour récolter les indices des enregistrements sélectionnés et la seconde pour construire effectivement la liste résultat) est une circonvolution inutile qui a été sanctionnée.

— De nombreuses étourderies ont été rencontrées dans les copies : indice mal formé, `constante` qui se transforme en `'constante'`, retour d'une liste d'indices plutôt que d'une liste d'enregistrements... Ces questions très simples rapportent peu de points mais il est néanmoins dommage de ne pas les valider.

**Question I.2**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 22 | 4% | 125 | 23% | 239 | 45% | 145 | 27% | 531 | 100% |

Cette question portait sur l'analyse de la complexité en pire cas de la fonction implémentée dans la question précédente.

Remarques :

— Une analyse de complexité doit *rigoureusement* suivre la structure du programme. Par exemple, il ne suffit pas de dire qu'un programme fait une itération sur une liste pour en déduire qu'il a une complexité linéaire : il faut encore préciser que le corps de cette boucle n'effectue que des opérations de complexité constantes.

— On s'attend à ce que la notation de Landau soit maîtrisée par les candidats : ainsi, conclure que la fonction a une complexité en $O(2 \cdot n)$ n'est pas satisfaisant.

**Question I.3**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 32 | 6% | 7 | 1% | 31 | 5% | 461 | 86% | 531 | 100% |

Il s'agissait ici de réaliser en PYTHON la sélection des enregistrements d'une table dont les champs de deux colonnes données sont égaux.

Remarques :

— Plusieurs candidats ont écrit une variante de cette fonction où les enregistrements qui ne vérifient pas le critère d'égalité sont remplacés par des enregistrements vides dans la table.

**Question I.4**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 47 | 8% | 10 | 1% | 47 | 8% | 427 | 80% | 531 | 100% |

Le candidat était invité à implémenter une fonction extrayant un sous-ensemble des champs d'un enregistrement. Ce sous-ensemble était spécifié par une liste d'indices.

Remarques :

— Une itération de la forme

```
for i in range(len(..)) if i in listeIndices
```

était à la fois interdite par le sujet et sous-optimale.

**Question I.5**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 4% | 5 | 0% | 27 | 5% | 474 | 89% | 531 | 100% |

Le sujet demandait ici au candidat d'étendre la fonction précédente aux tables d'enregistrements.
Remarques :

— On a sanctionné une réponse qui ne réutilisait par la fonction `ProjectionEnregistrement` obtenue pour la question précédente.

**Question I.6**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 4% | 9 | 1% | 26 | 4% | 471 | 88% | 531 | 100% |

Pour répondre à cette question, le candidat devait implémenter une fonction PYTHON réalisant le produit cartésien de deux tables.
Remarques :

— La concaténation de listes étant autorisée, il n'était pas nécessaire de la réimplémenter.
— Les candidats doivent être viligants aux types des valeurs manipulées par leur programme. Typiquement, dans cette question, certains candidats ont confondu `l += [ a + b ]` et `l += a + b`.

**Question I.7**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 115 | 21% | 126 | 23% | 156 | 29% | 134 | 25% | 531 | 100% |

On demandait ici au candidat d'implémenter une fonction réalisant la jointure de deux tables vis-à-vis de deux indices de colonnes donnés en argument. Il était suggéré aux candidats d'introduire une fonction auxiliaire réalisant la jointure entre deux enregistrements.
Remarques :

— Certaines copies introduisaient bien une fonction auxiliaire mais le test d'égalité entre les champs des enregistrements à joindre était réalisé à la fois dans cette fonction auxiliaire et la fonction principale. Cette redondance inutile des calculs a été sanctionnée.
— Le cas des couples d'enregistrements qui ne se joignent pas a souvent été traité incorrectement : dans un grand nombre de copies, on insère `None` dans le résultat à chaque fois que l'on rencontre de tels couples.
— On pouvait utiliser correctement un produit cartésien pour réaliser une jointure mais cette méthode a été légèrement sanctionnée car elle n'est pas très économe en mémoire.
— Dans la jointure entre enregistrements, il n'était pas correct de faire `r.pop (i2)` pour supprimer le champ redondant.

**Question I.8**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 161 | 30% | 25 | 4% | 123 | 23% | 222 | 41% | 531 | 100% |

Cette question portait sur la complexité en pire cas de la fonction introduite à la question précédente.
Remarques :

— L'arité d'au moins une des tables devait intervenir dans l'expression finale.
— La question *I.7* ne précisait pas si les enregistrements des deux tables d'entrées devaient être préservés par l'appel à la fonction de jointure. La complexité devait par contre être cohérente avec le choix fait par l'étudiant dans son implémentation.

**Question I.9**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 341 | 64% | 31 | 5% | 95 | 17% | 64 | 12% | 531 | 100% |

Pour finir cette série d'opérateurs, on s'attardait ici sur l'opération de suppression des enregistrements apparaissant plusieurs fois dans une table.

Remarques :

- Un grand nombre d'opérations interdites ont été trouvées dans les copies : typiquement, le prédicat d'appartenance à une liste `x in l`.
- Les cas limites ont souvent été éludés : certaines réponses traitaient le cas des tables vides de façon incorrecte.
- Des erreurs dans les indices ont souvent mené à des algorithmes supprimant toutes les occurrences des enregistrements apparaissant strictement plus d'une fois dans la table (au lieu de laisser exactement une occurrence).
- On a regretté l'absence de sortie prématurée lors des recherches de doublons.

**Question I.10**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 311 | 58% | 62 | 11% | 42 | 7% | 116 | 21% | 531 | 100% |

La première partie du sujet se concluait sur l'étude de la complexité de la fonction de suppression des doublons de la question I.9.

Remarques :

- Comme pour les questions de complexité précédentes, les correcteurs ont noté un certain manque de rigueur de l'argumentation. En particulier, la relation entre l'argumentaire et la structure du programme n'est pas toujours très claire.
- L'algorithme le plus souvent utilisé pour répondre à la question I.9 utilisait une boucle imbriquée dans une autre. Le nombre d'itérations de la boucle interne étant lié à la valeur de l'indice de la boucle externe, il était nécessaire d'effectuer une sommation utilisant ce même indice pour calculer le nombre d'opérations du programme. Très souvent, cette sommation a été oubliée et remplacée par une multiplication (incorrecte).

**Question II.1**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 37 | 6% | 16 | 3% | 265 | 49% | 213 | 40% | 531 | 100% |

Cette question (et plus généralement l'ensemble des questions de la seconde partie) demandait au candidat de coder une requête SQL à l'aide des fonctions PYTHON implémentées dans la partie I. On demandait ici d'implémenter une requête qui extrait tous les trajets dont la ville de départ est Rennes.

Remarques :

- Même si le sujet précisait bien que les réponses aux questions de la partie II devaient nécessairement utiliser uniquement les fonctions de la partie I *sans faire appel aux structures de contrôle de* PYTHON, certaines copies n'ont pas respecté cette contrainte et on obtenu une note nulle à la majorité des questions de cette partie.
- Il n'était pas nécessaire d'effectuer une projection de tous les champs à la fin du calcul pour simuler l'étoile de SQL. L'utilisation de cette opération inutile a été légèrement sanctionnée.
- Attention à ne pas confondre `Trajet`, la variable PYTHON introduite par le sujet pour représenter la table `Trajet`, et `'Trajet'`, une chaîne de caractère qui n'est donc pas une table. De même, `Rennes` et `'Rennes'` n'ont pas le même statut en PYTHON, le premier est un identificateur de variable tandis que le second est un litéral de type chaîne de caractère.

**Question II.2**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 66 | 12% | 2 | 0% | 5 | 0% | 458 | 86% | 531 | 100% |

On demandait ici de coder une requête SQL effectuant le produit cartésien de deux tables.
Remarques :

— Dans certaines copies, le programme PYTHON fait du zèle en supprimant les doublons. Ce n'était pas correct car la requête SQL n'utilise pas le mot-clé `DISTINCT`.

**Question II.3**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 276 | 51% | 32 | 6% | 8 | 1% | 215 | 40% | 531 | 100% |

La troisième requête SQL à implémenter sélectionnait les enregistrements du produit cartésien de deux tables à condition que les champs de deux colonnes données soient égaux.
Remarques :

— Utiliser une jointure n'était pas appropriée ici puisqu'elle supprime une colonne.
— Les indices des colonnes devaient prendre en compte le décalage des indices de la seconde table car les enregistrements de cette dernière se voient concaténés à ceux de la première table.

**Question II.4**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 54 | 10% | 30 | 5% | 103 | 19% | 344 | 64% | 531 | 100% |

Cette quatrième requête SQL était cette fois-ci la combinaison d'une jointure et d'une projection.
Remarques :

— La jointure supprimant une colonne, il y avait un décalage supplémentaire à prendre en compte dans le calcul des indices de la seconde table.

**Question II.5**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 91 | 17% | 41 | 7% | 192 | 36% | 207 | 38% | 531 | 100% |

Cette cinquième requête SQL mêlait produits cartésiens, diverses sélections et une projection finale.
Remarques :

— Deux produits cartésiens étaient suffisants, en utiliser trois a été pénalisé.
— Encore une fois, les erreurs d'indices ont été très discriminantes dans cette question. Il est conseillé aux candidats de prendre le temps de réaliser la forme des tables intermédiaires au brouillon pour trouver les bons indices.

**Question II.6**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 299 | 56% | 27 | 5% | 68 | 12% | 137 | 25% | 531 | 100% |

La dernière requête SQL s'appuyait sur le résultat de la requête de la question II.5 pour diriger le choix d'un enregistrement dans une table.
Remarques :

— Il était bien sûr attendu du candidat qu'il réutilise la réponse à la question précédente.
— L'opérateur `IN` a posé problème à certains candidats qui ont utilisé un produit cartésien et une sélection plutôt qu'une jointure pour le réaliser.

**Question III.1**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 203 | 38% | 50 | 9% | 106 | 19% | 172 | 32% | 531 | 100% |

La troisième partie débutait par l'écriture d'une fonction vérifiant si les enregistrements d'une table étaient bien triés vis-à-vis des valeurs d'un colonne donnée.

Remarques :

- La vérification qu'un prédicat est valide pour tous les éléments d'un ensemble peut arrêter son itération dès la première occurrence qui ne valide pas le prédicat en question. Si cette optimisation ne change pas la complexité asymptotique, elle est importante en pratique.
- La transitivité de l'ordre sur les valeurs comparées permet d'éviter de comparer un enregistrement avec tous ces successeurs dans la table, une comparaison avec l'enregistrement qui le suit immédiatement (s'il existe) suffit. Des candidats n'ont pas pris conscience de cette propriété en proposant un algorithme de complexité quadratique alors qu'un algorithme de complexité linéaire existe.
- Attention aux erreurs de dépassement des bornes : dans cet algorithme, on itère sur les paires d'enregistrements qui se suivent immédiatement donc il n'y a que $N - 1$ itérations à effectuer si la table contient $N$ enregistrements.

**Question III.2**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 252 | 47% | 69 | 12% | 89 | 16% | 121 | 22% | 531 | 100% |

La question III.2 s'intéresse à l'optimisation de la fonction de sélection des enregistrements dont un des champs vaut une constante donnée en profitant du fait que la table d'entrée est triée suivant la colonne correspond à ce champ.

Remarques :

- Toutes optimisations de l'algorithme étaient acceptées : bien sûr, la dichotomie était idéale mais le simple fait de sortir de l'itération dès que l'on croise un enregistrement trop grand suffisait à obtenir tous les points.
- En plus des habituelles erreurs de programmation, la plupart des erreurs à cette question consiste à renvoyer une table ne contenant qu'un seul enregistrement et pas tous les enregistrements contenant l'élément recherché.
- Les candidats ayant opté pour une dichotomie ont malheureusement souvent fait des erreurs dans les conditions d'arrêt ou dans le calcul des indices.

**Question III.3**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 446 | 83% | 17 | 3% | 32 | 6% | 36 | 6% | 531 | 100% |

On cherchait maintenant à optimiser l'opération de jointure en supposant que les deux tables d'entrées sont triées vis-à-vis de la colonne sur laquelle la jointure est faite.

Remarques :

- L'algorithme attendu suivait la même structure que l'algorithme de fusion de deux séquences triées : il fallait maintenir deux compteurs d'avancement dans les deux tables considérées en les mettant à jour au bon rythme. En général, les copies qui utilisaient deux boucles imbriquées ne suivaient pas ce schéma et n'obtenaient alors aucune amélioration de performances vis-à-vis de la solution naïve de la première partie.

**Question III.4**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 496 | 93% | 7 | 1% | 8 | 1% | 20 | 3% | 531 | 100% |

Dans cette question, le candidat étudiait la complexité de la fonction de jointure optimisée introduite dans la précédente question. On demandait aussi de discuter sous quelles conditions cette optimisation apporte effectivement un gain en performance.

Remarques :

- La complexité attendue était en $O((n_1 + n_2) \cdot (a_1 + a_2))$. Certains candidats ont oublié le terme en $(a_1 + a_2)$ même si le sujet précisait explicitement que les arités des deux tables devaient apparaître dans la réponse.
- La discussion devait se borner à comparer cette complexité à la complexité quadratique obtenue dans la question $I.8$. Certaines copies ont avancé des arguments informels peu convaincants.

**Question III.5**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 387 | 72% | 35 | 6% | 29 | 5% | 80 | 15% | 531 | 100% |

À partir de cette question, on supposait que les tables étaient accompagnées d'un index auxiliaire représenté par un dictionnaire PYTHON. Ainsi, une fois choisie la colonne d'indexation, un dictionnaire représentant un index peut associer efficacement pour chaque valeur possible du champ indexé les positions des enregistrements dont le champ indexé a cette valeur. La question III.5 demandait aux candidats d'écrire la fonction de construction du dictionnaire pour une table et une colonne d'indexation données.

Remarques :

- La plupart des erreurs viennent ici d'une mauvaise distinction des cas : il fallait traiter d'une part le cas où une valeur d'indexation n'a jamais été rencontrée et d'autre part le cas où il existe déjà une liste d'enregistrements associée à la valeur d'indexation considérée.
- Les algorithmes même corrects n'ayant pas une complexité linéaire en la taille de la table (en supposant que la recherche dans un dictionnaire est de coût de constant) n'ont pas obtenu la totalité des points.

**Question III.6**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 404 | 76% | 22 | 4% | 80 | 15% | 25 | 4% | 531 | 100% |

Dans la question III.6, il était demandé de fournir une version optimisée de la sélection des enregistrements par une constante. On s'appuie sur un dictionnaire auxiliaire qui sert d'index pour les valeurs d'une colonne donnée.

Remarques :

- Parcourir le dictionnaire était correct mais n'apportait aucune amélioration de performance par rapport à la version naïve de l'algorithme.
- Il fallait tester l'existence de la valeur recherchée dans le dictionnaire avant d'essayer d'accéder aux images de cette valeur. Cet oubli a coûté quelques points aux candidats.

**Question III.7**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 443 | 83% | 14 | 2% | 28 | 5% | 46 | 8% | 531 | 100% |

Il s'agissait ici d'établir la complexité de l'algorithme introduit par la question précédente. Le coût de cette fonction dépend de la taille de la sortie. Il existe donc un cas dégénéré où les enregistrements de la table ont toute la même valeur d'indexation et où le gain en performance est nul. Dans tous les autres cas, l'usage du dictionnaire est un avantage.

Remarques :

— Certains candidats ont oublié que le dictionnaire stocke les positions des enregistrements et non les enregistrements eux-mêmes : la complexité n'est donc pas constante mais bien linéaire en la sortie.

— L'analyse de complexité en pire cas menait à conclure que cet algorithme avait la même complexité que la fonction naïve. Il fallait donc mener une analyse plus fine qu'une analyse de complexité en pire cas, comme le sujet le suggérait.

**Question III.8**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 482 | 90% | 9 | 1% | 15 | 2% | 25 | 4% | 531 | 100% |

Dans la question III.8, on revenait une nouvelle fois sur l'algorithmede jointure mais en prenant cette fois-ci en argument un dictionnaire d'indexation pour la colonne de jointure de la seconde table. L'algorithme consiste alors à itérer sur la première table et pour chacun de ses enregistrements, à utiliser la table d'indexation de la seconde table pour obtenir les enregistrements avec lesquels le joindre.

Remarques :

— Les correcteurs ont apprécié la réutilisation de la fonction auxiliaire pour joindre deux enregistrements.

— Dans certaines copies, le candidat a oublié de tester l'existence de la valeur d'indexation dans le dictionnaire avant d'accéder à son image.

**Question III.9**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 503 | 94% | 2 | 0% | 9 | 1% | 17 | 3% | 531 | 100% |

Cette question portait sur la complexité de la fonction introduite par la question précédente.

Remarques :

— De nouveau, cette question demandait une analyse plus fine qu'une analyse en pire cas. Peu de candidats ont correctement formalisé les hypothèses sur la forme de la table permettant de conclure correctement.

**Question III.10**

| 0 | | $< 0.5$ | | $\geq 0.5$ | | 1 | | Total | |
|---|---|---|---|---|---|---|---|---|---|
| 502 | 94% | 13 | 2% | 6 | 1% | 10 | 1% | 531 | 100% |

Dans cette dernière question, on demandait aux candidats de discuter de la meilleure façon d'utiliser la fonction de jointure introduite dans la question III.8.

Remarques :

— Les candidats qui ont traité cette question ont en général eu la bonne intuition concernant les critères de choix de la table à indexer pour avoir une jointure performante, les réponses étaient trop peu argumentées : par exemple, on ne peut pas seulement répondre qu'il suffit d'indexer la plus grande des tables sans fournir de justification.