Composition d'Informatique (2 heures), Filière MP

(XC)

Rapport de M. Didier CASSEREAU, correcteur.

I. Bilan général

À titre de rappel, cette épreuve n'est corrigée que pour les candidats admissibles. Pour ma part, je n'ai corrigé que la filière MP.

Cette année le nombre total de candidats admissibles dans cette filière est de 296. La note moyenne est de 12,71 avec un écart-type de 3,52. 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 maximale 19,3/20. Deux copies ont obtenu une note éliminatoire de 2/20, et une copie a obtenu la note de 0/20, le candidat ne s'étant pas présenté à l'épreuve.

	Série 1		Série 2		Série 3		Série 4		**Synthèse**	
$0 \leq N < 4$	2	2,4%	0	0,0%	2	2,5%	0	0,0%	**4**	**1,4%**
$4 \leq N < 8$	10	12,2%	5	6,6%	5	6,2%	3	5,3%	**23**	**7,8%**
$8 \leq N < 12$	17	20,7%	23	30,3%	29	35,8%	14	24,6%	**83**	**28,0 %**
$12 \leq N < 16$	32	39,0%	38	50,0%	33	40,7%	26	45,6%	**129**	**43,6%**
$20 \leq N \leq 20$	21	25,6%	10	13,2%	12	14,8%	14	24,6%	**57**	**19,3%**
Total	82	100,0%	76	100,0%	81	100,0%	57	100,0%	**296**	**100,0%**
Epreuve complète*	27	32,9%	23	30,3%	26	32,1%	24	42,1%	**100**	**33,8%**

* *Épreuve complète* signifie ici que le candidat a abordé toutes les questions de l'énoncé et obtenu une note non nulle à chacune des 10 questions.

	Série 1	Série 2	Série 3	Série 4	**Synthèse**
Nombre de copies	82	76	81	57	**296**
Note minimale	2,0	4,7	0,0	6,7	**0,0**
Note maximale	19,3	19,0	19,3	18,3	**19,3**
Note moyenne	12,82	12,55	12,25	13,40	**12,71**
Ecart-type	4,16	2,94	3,61	3,04	**3,52**

Le langage de programmation choisi par les candidats est largement dominé par Maple (qui est cette année majoritairement bien orthographié, à la différence de l'an dernier) avec 58,8% des copies, suivi ensuite par Caml (25,7% des copies) et C/C++ (5,7% des copies). On trouve enfin quelques copies en Python (4,1%), Pascal (2,0%), Java (1,7%) et autres (1,7%, incluant par exemple Mathematica). Le tableau qui suit illustre cette répartition des langages choisis par les candidats.

Maple	Caml	C/C++	Python	Pascal	Java	Autres	**Total**
174	76	17	12	6	5	5	**296**
58,8%	25,7%	5,7%	4,1%	2,0%	1,7%	1,7%	**100,0%**

II. Commentaires

Cette année le sujet portait sur le problème dit du *sandwich au jambon*, encore appelé théorème de *Stone-Tukey*, qui stipule qu'un ensemble d'objets en dimension d peut toujours être séparé en deux par un hyperplan de dimension $d-1$. Le problème comportait 3 parties :

- la première partie, assez générale, permettait aux candidats de mettre en place les fonctions de base pour localiser la valeur maximale dans un tableau, ou déterminer le nombre de valeurs inférieures à une limite donnée,
- la deuxième partie consistait à écrire différentes fonctions visant à implémenter une méthode de tri rapide et efficace,
- la troisième et dernière partie permettait de généraliser l'approche numérique à un espace à 2 dimensions, avec un nuage de points caractérisés par leurs abscisses et ordonnées.

Les candidats étaient invités à passer outre certaines contraintes liées au langage de programmation (l'indexation des tableaux qui commence à 0 et non à 1 en C par exemple). Cette directive a été largement respectée par la grande majorité des candidats. Cependant certains candidats n'ont pas suivi cette recommandation, avec généralement des conséquences néfastes, parmi lesquelles des erreurs dans les indices.

J'invite donc explicitement les candidats à respecter ce genre de directive donnée dans l'énoncé.

L'objectif de cette épreuve est d'évaluer la capacité des candidats à écrire un code informatique permettant de résoudre un problème algorithmique donné. Il est important de proscrire tout ce qui relève du calcul formel ; dans le cas contraire le candidat prend le risque de se voir sanctionné par rapport à d'autres copies dans lesquelles on peut trouver un code complet et juste.

J'invite en particulier les candidats composant dans des langages tels que Mathematica à être très vigilants sur ce point.

L'évaluation d'un code informatique repose sur plusieurs éléments à mon avis essentiels, parmi lesquels

- évidemment le code doit être juste et donner le résultat correct,
- la clarté et la lisibilité du code sont également des éléments essentiels de l'évaluation : l'algorithme mis en œuvre doit être simple et clair, et un soin particulier doit être apporté dans la manière de présenter le code (indentation des boucles et tests, passages à la ligne,...),

- les commentaires explicatifs ne sont pas strictement indispensables, ils peuvent néanmoins aider à comprendre la démarche mise en œuvre, en particulier lorsque la méthode utilisée n'est pas simple ou ne fonctionne pas,
- l'efficacité intervient également dans l'évaluation du code, même si cela n'apparaît pas explicitement dans l'énoncé.

Quelques remarques générales à la lecture des codes :

- je sais bien que cette épreuve se déroule en temps limité, néanmoins j'attire l'attention des candidats sur l'importance de la présentation et de la lisibilité des copies ! en cas de doute ce n'est pas au correcteur d'évaluer ce que le candidat *aurait pu répondre* ; pour ma part, je juge exclusivement sur la base de ce qui est objectivement inscrit et lisible sur chaque copie.
- cette lisibilité est d'autant plus importante pour les codes que le candidat doit écrire ; le langage informatique étant un langage structuré, la copie se doit de refléter cette structure et la logique du langage, ainsi que la logique de l'algorithme implémenté,
- on trouve énormément d'instructions totalement inutiles, dans le genre `x=x`, ou des clauses `else` rattachées à un test `if` et qui ne font rien ; ces instructions ne servent à rien sinon à obscurcir le code, il serait préférable de les éviter,
- on observe souvent un usage abusif de la récursivité ; dans certains cas il est très pratique de pouvoir faire appel à cette spécificité de certains langages de programmation, il faut cependant bien avoir conscience que cela peut rendre le code plus complexe à appréhender, et que c'est plus consommateur en ressources machine qu'une simple boucle `for` ou `while` ; de manière générale je conseille de restreindre l'utilisation de code récursif aux situations pour lesquelles **cela apporte vraiment quelque chose** en terme de simplicité algorithmique,
- en complément de la remarque précédente, beaucoup de candidats définissent des fonctions intermédiaires (c'est plutôt bien), mais elles s'appellent toutes `aux` (et là c'est moins bien) ; cela ne coûte rien de choisir un nom plus représentatif de la tâche exécutée par la fonction, et cela aide beaucoup à comprendre le code. Cette remarque et la précédente visent essentiellement les candidats composant en Caml,
- on ne demande pas aux fonctions d'afficher vrai ou faux, ou plus généralement d'afficher le résultat obtenu ; les fonctions doivent résoudre le problème posé et retourner un résultat,
- on voit souvent des solutions qui sont justes, mais totalement tordues avec un code très difficile à lire et à comprendre ; clairement une telle situation doit être évitée,
- certaines notations n'ont pas de sens dans un langage informatique, telles que par exemple `t'`, θ, ou encore les accents dans les noms de variables ou fonctions,
- il faut faire attention de ne pas manipuler des variables qui ne sont pas initialisées, par exemple dans une fonction qui cherche à évaluer la valeur maximale d'un tableau,
- les éléments de structuration du code (boucles ou tests) ont un début et une fin qui sont bien marqués par les règles syntaxiques du langage, il est impératif que les candidats veillent à spécifier ces mots-clés de manière précise et systématique ; dans

le cas contraire, on a confusion totale sur ce qu'est supposé faire le code, et cette confusion n'est jamais à l'avantage du candidat,

- la syntaxe est parfois très approximative, avec un flou volontaire ou non ; rien ne doit être approximatif dans le domaine de l'informatique, il faut au contraire veiller à être très précis ; c'est en particulier souvent le cas des candidats qui composent en langage C/C++,
- je suis assez surpris de constater que certains candidats n'utilisent que des boucles `while`, alors que dans certaines circonstances la boucle `for` est plus intuitive et plus simple,
- de manière générale, on peut souvent considérer que les différentes questions sont à tiroir, il ne faut donc pas hésiter à utiliser les fonctions écrites dans les questions précédentes, plutôt que de tout refaire,
- il est impératif d'écrire les codes de manière complète et précise ; on ne peut absolument pas accepter des écritures partielles, par exemple en omettant les paramètres que l'on transmet lors d'un appel de fonction, parce que **c'est peut-être justement là** que l'on pourra faire la différence entre un code qui est correct et un code qui ne l'est pas,
- il faut éviter d'appeler la même fonction de manière itérative, par exemple dans une boucle ; il est préférable d'appeler la fonction une seule fois, stocker le résultat en mémoire, et ensuite réutiliser cette variable,
- dans une boucle, par exemple indicée par une variable entière `i`, on ne peut pas manipuler une variable nommée `ti` en espérant que le nom de cette variable va suivre l'évolution de `i` ; cette erreur est assez fréquente et aboutit inévitablement à un code qui n'a pas grand sens et qui n'a surtout aucune chance de fonctionner,
- une erreur très fréquente et basique : entre les indices `a` et `b`, on a `b-a+1` valeurs et non `b-a` !
- autre confusion fréquente cette année : beaucoup de candidats font la confusion entre les indices d'un tableau et les valeurs de ce tableau ; par ailleurs les indices sont tous à valeur entière, alors que les valeurs du tableau sont des quantités réelles.

III. Commentaires détaillés

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

- 0 signifie aucun point pour la question (question non traitée ou abordée mais totalement fausse, ce second cas de figure étant rare),
- $<$0,5 signifie moins de la moitié des points de la question,
- $\geq$0,5 signifie plus de la moitié des points de la question,
- 1 signifie la totalité des points de la question.

Question 1 :

	0		<0,5		≥0,5		1		Total	
Question 1	3	1,0%	42	14,2%	73	24,7%	178	60,1%	296	100,0%

Cette question consiste à calculer l'indice de la valeur maximale d'un tableau de valeurs. A noter ici qu'on ne cherche pas à examiner l'ensemble du tableau, mais uniquement les valeurs d'indice compris entre les deux paramètres `a` et `b`.

Cette question ne présente pas de difficulté particulière, il faut cependant éviter quelques pièges classiques et récurrents :

- la manipulation de variables non (ou mal) initialisées,
- la fonction qui retourne la valeur maximale, et non l'indice de cette valeur maximale,
- les algorithmes consistant à comparer chaque valeur du tableau à sa valeur suivante ou précédente, qui ne fonctionnent pas.

À titre de rappel, une instruction `return` signifie l'interruption de la fonction en cours, avec affectation immédiate du résultat. Il faut donc éviter d'utiliser `return` de manière anticipée dès qu'une condition logique de supériorité ou d'infériorité est rencontrée. Dans ce cas en effet, la fonction ne testera pas les autres valeurs du tableau.

Globalement les candidats ont bien noté que la recherche devait se faire entre les indices `a` et `b` et non sur l'ensemble du tableau. Comme indiqué ci-dessus, une bonne partie des erreurs provient d'une confusion entre indices et valeurs.

Question 2 :

	0		<0,5		≥0,5		1		Total	
Question 2	2	0,7%	14	4,7%	49	16,6%	231	78,0%	296	100,0%

Cette question consiste à calculer le nombre de valeurs du tableau inférieures à un paramètre transmis à la fonction. De même que pour la fonction précédente, cette exploration se fait entre des indices `a` et `b` et non sur l'ensemble du tableau.

Là encore cette question ne présentait pas de difficulté particulière, il fallait bien veiller à initialiser correctement la variable utilisée pour compter le nombre de valeurs du tableau satisfaisant la condition voulue. Une erreur d'inattention apparue un certain nombre de fois a consisté à calculer le nombre de valeurs supérieures et non inférieures au paramètre fourni à la fonction. Je recommande donc une lecture attentive de l'énoncé.

Cette question et la précédente sont des cas typiques pour lesquels une simple boucle `for` permet de résoudre le problème ; l'utilisation d'un code récursif dans ce genre de situation nuit beaucoup à la lisibilité et aux performances.

Question 3 :

	0		<0,5		≥0,5		1		Total	
Question 3	21	7,1%	97	32,8%	84	28,4%	94	31,8%	296	100,0%

Cette question consiste à partitionner le tableau, c'est-à-dire à isoler dans la partie gauche du tableau toutes les valeurs inférieures au pivot fourni, et dans la partie droite toutes les valeurs supérieures au pivot. A noter que l'on ne demandait pas que ces valeurs soient triées.

Cette question est clairement plus délicate que les deux précédentes et nécessitait un peu de réflexion de la part des candidats avant d'essayer de coder. Statistiquement on va du pire au meilleur, avec des codes d'une extrême complexité (très difficiles à relire, et souvent très mal présentés, ce qui rend la lecture encore plus délicate) et inversement des codes limpides et manifestement très réfléchis. Quelques erreurs fréquentes que j'ai pu observer :

- certains codes auraient pu être corrects, mais ne prennaient pas en compte le fait que le pivot pouvait être présent plusieurs fois dans le tableau,
- la fonction précédente a souvent été utilisée pour déterminer le nombre d'apparitions du pivot `p`, en calculant la différence `nombrePlusPetit(a,b,p)-nombrePlusPetit(a,b,p-1)` ; cela suppose qu'il ne peut pas y avoir de valeurs dans le tableau entre `p` et `p-1`, ce qui n'a absolument aucune raison d'être, les valeurs du tableau étant réelles,
- certains codes étaient beaucoup trop complexes, ou inversement beaucoup trop simples, et ne pouvaient manifestement pas fonctionner,
- certains candidats ont inversé la logique de la fonction, en mettant en tête du tableau les valeurs supérieures au pivot, et en fin de tableau les valeurs inférieures au pivot.

Question 4 :

	0		<0,5		≥0,5		1		Total	
Question 4	8	2,7%	50	16,9%	53	17,9%	185	62,5%	296	100,0%

Cette question consiste à trouver le `k`-ième élément d'un tableau. En théorie cette question n'aurait pas dû poser de difficulté particulière, toute la démarche logique à implémenter étant précisée, étape par étape, dans l'énoncé. Globalement cette question a été plutôt bien traitée par une majorité de candidats.

J'encourage là encore les candidats à lire attentivement l'énoncé qui suggérait l'écriture d'une fonction récursive.

Question 5 :

	0		<0,5		≥0,5		1		Total	
Question 5	85	28,7%	11	3,7%	4	1,4%	196	66,2%	296	100,0%

Cette question consiste à évaluer le coût en temps de calcul de l'algorithme implémenté dans la fonction précédente. On trouve une très grande variété de réponses, allant du $\propto n$ jusqu'à $\propto n!$ en passant par tous les intermédiaires possibles et imaginables.

Question 6 :

	0		<0,5		≥0,5		1		Total	
Question 6	55	18,6%	77	26,0%	143	48,3%	21	7,1%	296	100,0%

Cette question consiste à écrire un algorithme astucieux de détermination du pivot que l'on va ensuite utiliser en combinaison avec les fonctions précédentes. Le principe repose sur un découpage des données par blocs de 5 valeurs, puis à appeler la fonction de manière récursive.

Pas très simple, cette question nécessitait une certaine réflexion de la part des candidats, et une structuration rigoureuse et claire du code. Parmi les erreurs classiques que j'ai pu relever :

- entre les deux indices `a` et `b` inclus, on a `b-a+1` valeurs et non `b-a`,
- si je fais appel à la fonction `elementK()` précédente entre `a+5*k` et `a+5*(k+1)`, je traite en réalité les données par blocs de 6 valeurs et non 5, avec recouvrement entre les différents blocs,
- il n'était pas utile de réécrire une fonction de calcul du médian d'un tableau de 5 valeurs, la fonction `elementK()` précédente faisait parfaitement l'affaire,
- en programmation, on écrit `5*k` et non `5k`,
- dans ce découpage en blocs de 5 valeurs, il pouvait rester un dernier bloc incomplet en fin de tableau ; ce dernier bloc devait être traité avec soin, précisément parce qu'il ne contenait pas 5 valeurs mais moins, ce qui devait *de facto* affecter le calcul de l'élément médian,
- il ne fallait pas oublier l'appel récursif en fin de fonction ; certains candidats ont remplacé cet appel récursif par un appel unique à la fonction `elementK()`, ce qui limitait la portée de l'algorithme implémenté.

Là encore, j'ai pu voir des codes très compliqués, et inversement des codes très clairs et simples. Ces derniers sont bien entendu à privilégier absolument.

Question 7 :

	0		<0,5		≥0,5		1		Total	
Question 7	10	3,4%	58	19,6%	7	2,4%	221	74,7%	296	100,0%

Cette question consiste à trouver l'ordonnée d'une ligne horizontale séparant les points du plan en deux parties de même cardinal. Noter ici que l'énoncé faisait clairement apparaître l'existence d'une fonction retournant l'indice de la valeur médiane d'un tableau.

Compte tenu de cette dernière remarque, la réponse à cette question devait être très simple puisqu'elle consistait à faire appel à cette fonction dont on supposait l'existence, et à renvoyer ensuite la valeur du tableau `tabY` en mémoire à cet indice dans le tableau. Cette question a majoritairement été traitée correctement ; à noter cependant un nombre assez important de candidats ayant fait la confusion entre indice et valeur.

Question 8 :

	0		<0,5		≥0,5		1		Total	
Question 8	33	11,1%	59	19,9%	115	38,9%	89	30,1%	296	100,0%

Une fois localisée la droite horizontale précédente, cette question consiste à déterminer, parmi tous les angles possibles des points situés au-dessus, l'angle médian. Cet angle est supposé calculé entre la demi-droite horizontale passant par un point `(x,y)` donné et la demi-droite joignant ce point `(x,y)` au point courant dont les abscisse et ordonnée sont définies par les tableaux `tabX` et `tabY`.

Cette question ne posait normalement pas de véritable difficulté dans la mesure où le candidat disposait à ce niveau de tous les ingrédients nécessaires. Le principe de base consistait à explorer l'ensemble des points en ne sélectionnant que ceux localisés au-dessus de la demi-droite horizontale considérée. Pour ces points, il restait à calculer les angles (la fonction correspondante étant admise), puis l'indice médian du tableau des angles, et enfin l'angle médian.

Les erreurs ou remarques générales observées pour cette question sont les suivantes :

- dans cette question, il fallait bien penser que le tableau des angles serait de taille plus petite que les tableaux `tabX` et `tabY` ; dans la boucle d'exploration des points, il fallait donc impérativement indicer les angles éligibles avec une variable autre que celle utilisée pour l'exploration des points eux-mêmes ; dans le cas contraire on butait sur i) des angles non calculés dans le tableau, ii) un possible débordement de tableau, et iii) des angles éligibles non pris en compte dans le calcul de l'angle médian,
- la position de la droite horizontale pouvait s'obtenir avec la fonction `coupeY` précédente ; en revanche il fallait veiller à appeler cette fonction une seule et unique fois, stocker en mémoire la valeur retournée, et ensuite utiliser cette valeur dans la boucle ; il fallait en particulier éviter d'appeler cette fonction `coupeY` en boucle,
- une erreur très fréquente a été de retourner la position de l'angle médian dans le tableau des angles, et non l'angle médian lui-même ; d'une part cela n'était pas conforme à la question posée dans l'énoncé, et d'autre part cette position faisait référence à un tableau qui n'existait plus lorsque l'exécution de la fonction se terminait, de sorte que cette position n'était de toutes manières pas utilisable par la fonction appelante,
- autre erreur importante vue à plusieurs reprises : certains candidats ont souhaité faire appel à la fonction `partition` écrite dans la troisième question ; soit cette fonction `partition` est appliquée au seul tableau `tabY`, soit aux deux tableaux `tabX`

et `tabY` ; le premier scénario ne peut pas fonctionner dans la mesure où la fonction `partition` réordonne les éléments du tableau `tabY` sans toucher au tableau `tabX`, ce qui n'est pas correct car les valeurs contenues dans ces 2 tableaux sont indissociables (elles correspondent aux abscisses et ordonnées de nos points) ; procédant ainsi on modifie donc les ordonnées sans toucher aux abscisses, et la conséquence évidente est que l'on modifie les points ! le second scénario n'est pas correct non plus dans la mesure où les tableaux `tabX` et `tabY` sont *a priori* réordonnés indépendamment, et donc de manière différente.

Question 9 :

	0		<0,5		≥0,5		1		Total	
Question 9	63	21,3%	34	11,5%	128	43,2%	71	24,0%	296	100,0%

Cette question consiste à déterminer si l'angle médian précédemment trouvé est également médian pour les angles correspondant aux points en-dessous de la droite horizontale de référence. Cette question ne posait *a priori* pas de difficulté, en particulier pour les candidats qui avaient su traiter correctement la question précédente, dans la mesure où la démarche générale est très similaire.

Souvent les erreurs ou maladresses faites en question 8 ont été retrouvées à l'identique en question 9.

Question 10 :

	0		<0,5		≥0,5		1		Total	
Question 10	141	47,6%	59	19,9%	46	15,5%	50	16,9%	296	100,0%

Cette question consiste à utiliser un processus itératif afin de déterminer les deux droites qui séparent l'ensemble des points en 4 sous-ensembles comportant au plus $n/4$ points.

Là encore cette question ne posait pas de difficulté particulière dans la mesure où les candidats disposaient de tous les ingrédients nécessaires, auxquels s'ajoutait une description précise et détaillée de toutes les étapes de l'algorithme donnée dans l'énoncé. Certains candidats ont adopté une logique récursive, là où une simple boucle `while` pouvait parfaitement suffire. Etrangement un nombre assez important de candidats semblent ne pas savoir calculer le milieu d'un segment défini par les points (α, y) et (β, y) : l'abscisse de ce milieu est bien $(\beta + \alpha)/2$ et non $(\beta - \alpha)/2$.