Composition d'Informatique (2 heures), Filière MP (XLCR)

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 MP.

Cette année le nombre total de candidats admissibles dans cette filière est de 540 (pour les candidats X-ENS). La note moyenne est de 14,23 avec un écart-type de 2,79. 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 de 3/20 et la note maximale 19,7/20. Aucune copie n'a obtenu une note éliminatoire.

X - ENS

	Série 1		Série 2		Série 3		Série 4		**Synthèse**	
$0 \leq N < 4$	1	0,9%	1	0,7%	0	0,0%	0	0,0%	**2**	**0,4%**
$4 \leq N < 8$	8	6,8%	3	2,1%	3	2,2%	6	4,0%	**20**	**3,7%**
$8 \leq N < 12$	18	15,4%	15	10,7%	18	13,4%	15	10,1%	**66**	**12,2%**
$12 \leq N < 16$	67	57,3%	80	57,1%	74	55,2%	81	54,4%	**302**	**55,9%**
$16 \leq N \leq 20$	23	19,7%	41	29,3%	39	29,1%	47	31,5%	**150**	**27,8%**
Total	117	100,0%	140	100,0%	134	100,0%	149	100,0%	**540**	**100,0%**
Épreuve complète*	61	52,1%	69	49,3%	62	46,3%	74	49,7%	**266**	**49,3%**

* *Épreuve complète* signifie ici que le candidat a abordé toutes les questions de l'énoncé et obtenu une note non nulle à chacune des 13 questions.

	Série 1	Série 2	Série 3	Série 4	**Synthèse**
Nombre de copies	117	140	134	149	**540**
Note minimale	3,4	3,0	5,0	4,0	**3,0**
Note maximale	18,9	19,7	18,4	19,5	**19,7**
Note moyenne	13,61	14,42	14,31	14,47	**14,23**
Ecart-type	3,16	2,67	2,53	2,77	**2,79**

Cette année, le langage de programmation utilisé par les candidats est très majoritairement dominé par Python. On trouve de manière anecdotique quelques copies écrites en Caml ou C/C++. Cette situation traduit clairement le fait que le langage Python a été introduit dans les nouveaux programmes des classes préparatoires.

Les notes se répartissent selon le tableau suivant, avec une moyenne de 13,55 et un écart-type de 2,72 (pour les candidats français de l'École polytechnique) :

$0 \leqslant N < 4$	0	0,00 %
$4 \leqslant N < 8$	8	3,90 %
$8 \leqslant N < 12$	37	18,05 %
$12 \leqslant N < 16$	124	60,49 %
$16 \leqslant N \leqslant 20$	36	17,56 %
Total	205	100 %
Nombre de copies : 205		
Note moyenne : 13,55		
Écart-type : 2,72		

2. Commentaires

Cette année le sujet porte sur le calcul de l'enveloppe convexe d'un nuage de points dans le plan. Le problème comporte 3 parties :

- la première partie permet d'introduire certaines notions préliminaires,
- la deuxième partie traite d'une première méthode dite « du paquet cadeau »,
- la troisième partie décrit un « algorithme de balayage » qui permet de réaliser le calcul de l'enveloppe à coût moindre en temps de calcul.

Pour certaines questions, il est clairement demandé aux candidats d'expliciter la complexité de leur code. Pour ces questions particulières, cette analyse de la complexité est essentielle et fait explicitement partie de la notation.

L'évaluation d'un code informatique repose sur plusieurs éléments 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 :

- La lisibilité des copies est très importante. En cas de doute ce n'est pas au correcteur d'évaluer ce que le candidat *aurait pu répondre* ; seul ce qui est objectivement inscrit et lisible sur chaque copie est réellement pris en compte.

- 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 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.
- Attention à certaines notations qui peuvent avoir un sens à l'écrit, mais pas dans un langage informatique, comme par exemple `t'`, 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, il y 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.
- 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.
- Une erreur fréquente consiste à incrémenter le compteur d'une boucle `for` ; or le mécanisme de la boucle implique déjà cette incrémentation, de sorte que cela conduit à incrémenter deux fois, ce qui n'est en général pas ce que l'on cherche à faire ; inversement, les boucles `while` doivent explicitement contenir une instruction permettant d'incrémenter le compteur de la boucle, faute de quoi on risque de rentrer dans une boucle infinie.
- Certains opérateurs logiques sont utilisés de manière un peu approximative, comme par exemple >< à la place de <> ou ! =, ou encore =< à la place de <=. Attention à la confusion entre = et ==.
- De manière générale, il est très important, lorsqu'une fonction explore les valeurs d'un tableau, de vérifier que l'indice ne dépasse pas la taille du tableau ; il s'agit là d'une erreur grave de programmation, les candidats qui n'ont pas été vigilants sur ce point ont systématiquement été sanctionnés. A noter qu'en langage Python, l'indexation des tableaux commence à 0 et non à 1.

3. 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),
- <0,5 signifie moins de la moitié des points de la question,
- ≥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	5	0,9%	40	7,4%	374	69,3%	121	22,4%	540	100,0%

Cette question consiste à déterminer le point de plus petite ordonnée parmi les points du nuage. Dans le cas particulier où cette plus petite ordonnée est atteinte par plusieurs points du nuage, il fallait veiller à conserver le point de plus faible abscisse.

Cette question ne présente pas de vraie difficulté et elle a été plutôt bien traitée en moyenne. Parmi les erreurs les plus courantes, nous pouvons noter :

- la non-prise en compte de la directive de plus faible abscisse, dans le cas où plusieurs points ont l'ordonnée minimale,
- un test simultané sur abscisse et ordonnée : un tel test simultané ne fonctionne en général pas,
- calcul de l'ordonnée maximale et non minimale,
- quelques débordements de tableaux.

Question 2 :

	0		<0,5		≥0,5		1		Total	
Question 2	42	7,8%	140	25,9%	24	4,4%	334	61,9%	540	100,0%

Cette question est très simple, elle consiste simplement à appliquer la règle de calcul de l'orientation d'un triangle et d'en définir ainsi l'orientation positive ou négative. À noter ici qu'une réponse lapidaire n'était pas considérée comme acceptable, un minimum de justification était indispensable.

Question 3 :

	0		<0,5		≥0,5		1		Total	
Question 3	4	0,7%	12	2,2%	140	25,9%	384	71,1%	540	100,0%

Cette question n'est autre que l'écriture informatique du calcul réalisé lors de la question précédente pour des triplets de points spécifiques. Elle ne présentait pas vraiment

de difficulté, il fallait cependant veiller à explicitement différencier les orientations $+$, $-$ et 0.

Certains candidats ont observé qu'il n'était pas nécessaire de prendre en compte le facteur $1/2$ qui intervient dans l'aire d'un triangle, compte tenu du fait qu'on ne s'intéressait finalement qu'au signe du résultat. À ce sujet, il est important de noter qu'en Python, $1/2$ vaut ... 0.

Question 4 :

	0		<0,5		≥0,5		1		Total	
Question 4	0	0,0%	188	34,8%	330	61,1%	22	4,1%	540	100,0%

Cette question définit une relation binaire entre deux points du nuage, il fallait justifier que cette relation permettait de définir une relation d'ordre total. La réflexivité, l'antisymétrie et la totalité n'ont généralement pas posé de grandes difficultés. La transitivité était-elle plus délicate à établir. Certains candidats se sont lancés dans des calculs compliqués avec des inégalités auxquelles ils appliquent ensuite des divisions... cela devient rapidement très lourd et complexe, d'autant qu'il est alors indispensable de traiter tous les cas possibles, et ainsi éviter de diviser par 0 ou prendre en compte qu'une inégalité change de sens lorsqu'on la divise par une valeur négative. Majoritairement les candidats qui sont partis dans cette direction ne s'en sont pas sortis.

Par ailleurs, les aires ne s'additionnent pas, l'exemple des points $i(0,0)$, $j(-1,2)$, $k(0,1)$, $l(1,2)$ illustrant cette situation. Il fallait ici explicitement utiliser le fait que le point i se trouvait sur l'enveloppe.

A noter que la question de la co-linéarité de deux points n'a pas grand sens.

Question 5 :

	0		<0,5		≥0,5		1		Total	
Question 5	25	4,6%	61	11,3%	235	43,5%	219	40,6%	540	100,0%

Cette question a un peu perturbé les candidats qui ont composé en Caml ou C/C++ car c'est la seule qui mentionne explicitement le langage Python. Elle ne présentait pas de grande difficulté à partir du moment où l'on avait compris le sens de la relation d'ordre total précédente. Un point essentiel ici était que le code devait être à complexité linéaire, ce point a souvent été omis par les candidats.

À noter une erreur assez fréquente avec inversion du sens du test portant sur la fonction `orient()` précédente.

Question 6 :

	0		<0,5		≥0,5		1		Total	
Question 6	57	10,6%	72	13,3%	55	10,2%	356	65,9%	540	100,0%

Cette question consistait simplement à appliquer la logique de l'algorithme précédent et décrire son déroulement. Elle a été correctement traitée dans une grande majorité des cas. A noter que la clarté de la rédaction et des explications était très importante pour une question de ce genre.

Il fallait également veiller à parcourir l'ensemble des points du nuage, jusqu'au dernier.

Question 7 :

	0		<0,5		≥0,5		1		Total	
Question 7	6	1,1%	29	5,4%	201	37,2%	304	56,3%	540	100,0%

Cette question consistait simplement à itérer les appels de la fonction `écrire` à la question 5 pour parcourir l'ensemble des points de l'enveloppe convexe recherchée. L'idée ici était de partir du point le plus bas (question 1), puis d'itérer jusqu'à ce que l'on retombe sur le point de départ. Au fur et à mesure de l'itération, on construisait la liste des points formant l'enveloppe convexe recherchée.

Les erreurs ou défauts les plus fréquemment observés sont :
- une boucle infinie, par exemple parce que point de départ change en même temps que l'avancement de la boucle,
- des appels répétés aux fonctions précédentes, là où un appel unique et mise en mémoire est généralement préférable.

Question 8 :

	0		<0,5		≥0,5		1		Total	
Question 8	19	3,5%	7	1,3%	139	25,7%	375	69,4%	540	100,0%

Cette question consistait simplement à justifier du temps d'exécution de la boucle précédente, elle ne présentait aucune difficulté objective et a été majoritairement bien traitée.

Question 9 :

	0		<0,5		≥0,5		1		Total	
Question 9	37	6,9%	4	0,7%	10	1,9%	489	90,6%	540	100,0%

Il s'agit là d'une question de cours, à laquelle on pouvait répondre par le tri rapide / quicksort, ou encore le tri fusion.

À noter toutefois qu'il existe certes en théorie une méthode pour choisir efficacement un pivot parfait pour quicksort, mais elle est compliquée et peu efficace en pratique. Quicksort est donc accepté comme réponse, d'autant plus que cet algorithme est en pratique assez efficace, mais mergesort est plus approprié à la question posée. Bubblesort - tri à bulles - n'est par ailleurs notamment *pas* en $O(n \log n)$.

Question 10 :

	0		<0,5		≥0,5		1		Total	
Question 10	22	4,1%	51	9,4%	316	58,5%	151	28,0%	540	100,0%

Cette question consistait à écrire une fonction permettant l'ajout d'un nouveau point à l'enveloppe supérieure du nuage de points, à partir des fonctions de gestion de pile qui étaient supposées disponibles. Cela nécessitait de jouer avec les empilements/dépilements, elle nécessitait d'avoir bien compris le mécanisme dans son ensemble.

Cette question est clairement difficile, les propositions vont de 5 lignes à plusieurs dizaines de lignes ! C'est assez révélateur du recul que le candidat parvient à atteindre.

Les deux stratégies les plus courantes consistaient soit à implémenter une boucle `while` ou équivalente, soit à mettre en place un appel récursif de la fonction `majES()`. Les deux approches étaient effectivement envisageables.

Attention à n'utiliser que les fonctions fournies, il est explicitement dit que l'on n'est pas censé connaître la structure des piles (la fonction `len()` n'est notamment pas fournie, et les fonctions `push` et `pop` ne renvoient pas de pile). Il fallait donc veiller à construire la fonction `majES()` sans ce genre de fonction auxiliaire.

La même remarque s'applique pour la question suivante.

Le critère de la complexité constituait un élément important dans cette question.

Question 11 :

	0		<0,5		≥0,5		1		Total	
Question 11	40	7,4%	5	0,9%	9	1,7%	486	90,0%	540	100,0%

Cette question était la question symétrique de la précédente.

Question 12 :

	0		<0,5		≥0,5		1		Total	
Question 12	57	10,6%	28	5,2%	280	51,9%	175	32,4%	540	100,0%

Dans cette question, il fallait simplement utiliser les deux fonctions précédentes `majES()` et `majEI()` en boucle, et ensuite recoller les deux piles inférieure et supérieure.

Cette seconde partie a posé d'assez nombreuses difficultés. Il fallait voir ici qu'il n'était déjà pas nécessaire d'inverser l'ordre d'une des piles (il suffisait de dépiler `es` et réempiler au fur et à mesure dans `ei`, les points étant alors stockés dans l'ordre convenable). Il fallait également bien voir que le dernier point était présent à la fois dans `es` et `ei`, il fallait donc l'éliminer afin d'éviter un point en double au milieu de la liste.

Certains candidats ont également retiré le premier point qui pouvait sinon être présent en début ET en fin de liste. Dans ce cas particulier, le point doublon pouvait passer pour

acceptable.

Certains candidats ont ici inventés des méthodes très compliquées, allant jusqu'à gérer deux voire 3 piles auxiliaires totalement inutiles. Cela a généralement conduit à des codes très difficiles à relire.

Question 13 :

	0		<0,5		≥0,5		1		Total	
Question 13	219	40,6%	42	7,8%	186	34,4%	93	17,2%	540	100,0%

Cette question visait à justifier de la complexité globale de cette méthode de calcul de l'enveloppe convexe. Une première analyse, sans aller trop dans le détail, pouvait amener à conclure à une complexité en n^2, compte tenu de la complexité des fonctions `majES()` et `majEI()`, qui sont ensuite appelées en boucle par la fonction `convGraham()`. De nombreux candidats ont été bloqués à ce niveau, ne comprenant pas comment on pouvait finalement conclure en une complexité en $n \log n$.

En fait le sujet pouvait induire en erreur : à la question 10, il est demandé d'obtenir une complexité en $O(i)$, mais pour cette question, il fallait aller plus loin en faisant une analyse amortie : un sommet n'étant empilé/dépilé qu'un nombre constant de fois sur l'ensemble de l'exécution, la complexité totale de l'ensemble des appels aux fonctions `majES()` et `majEI()` est bien en $O(n)$.

Certains candidats sont tout de même allés jusqu'à écrire $O(n^2) < O(n \log n)$, ce qui était un peu abusé...