Correction de la question 6 :

> Trouvez un algorithme qui détermine en $O(n^2 log(n))$ si trois points quelconques d'un ensemble de n points sont alignés.

Je rappelle que l'espace considéré est en 2 dimensions, que les coordonnées des points sont connues (par exemple en coordonnées rectangulaires), qu'ils ne sont pas ordonnés selon un critère donné (surtout celui qui vous arrangerait le plus ;-)) et que les considérations de mémoire ne sont pas importantes. L'algorithme le plus naïf s'exécuterait en n^3 et consisterait à parcourir tous les triplets possibles et à tester si les trois points sont alignés ou non. Voyons si l'on peut mieux faire…

Algorithme n°1

On utilise la fonction EST_ALIGNE qui prend trois points et qui renvoie *vrai* si les trois points sont alignés, *faux* sinon. « ⊗ » est le produit scalaire de deux vecteurs.

```
1     EST_ALIGNE(A,B,C)
2           Si AC ⊗ AB = 0 renvoyer vrai
3           Sinon renvoyer faux
4     FIN
```

Complexité en temps : O(1)

La fonction TROUVE_ALIGNEMENT_ANGLE prend un ensemble E de n points et renvoie *vrai* s'il existe trois points dans cet ensemble qui sont alignés.

```
1    TROUVE_ALIGNEMENT_ANGLE
2         POUR chaque point P de l'ensemble E
3              POUR chaque point Q de l'ensemble E-{P}
4                   Calculer l'angle de (PQ) par rapport à la droite du
                    repère (Ox)
5              FIN POUR
6              Trier les points de l'ensemble E-{P} selon ces angles et les
               placer dans un tableau L
7              POUR i = 2 à Taille(L)
8                   SI EST_ALIGNE( P, L[i-1], L[i] ) renvoie vrai
9                         ALORS Renvoyer vrai
10                  FIN SI
11             FIN POUR
12        FIN POUR
13        Renvoyer faux
14   FIN
```

La boucle POUR principale teste pour un point P donné, s'il existe deux autres points qui sont alignés avec P. Comme ces deux autres points se trouvent toujours du même côté par rapport à P (à cause de l'algorithme), et comme on ne teste pas le cas où P se trouve entre ces points, il faut prendre pour P chacun des points de l'ensemble E. Optimisation : calculer les angles modulo π (plutôt que 2π).

Calcul de la complexité en temps :

- la boucle de la ligne 3 se fait en n-1 étapes
- le tri de la ligne 6 peut se faire en O(n.log(n))

- la boucle de la ligne 7 se fait en n-2 étapes
- la boucle principale à la ligne 2 se fait en n étapes

La complexité totale est donc de O(n*((n-1)+n*log(n)+(n-2))) = O(n*(2n+n*log(n)-3)) = O(n*n*log(n))

Algorithme n°2

L'algorithme n°2 a été proposé par J. Korczak pendant que vous composiez l'examen ! Il semblerait que son algorithme s'exécute en n*n… L'algorithme utilise un tableau à deux dimensions nommé TABLE de M*N entrées (M et N sont à déterminer à l'avance) dont chaque case est initialisée à *faux*.

```
1    TROUVE_ALIGNEMENT_DROITE
2         TANT QUE il y a encore un point P dans l'ensemble E
3              POUR chaque point Q dans l'ensemble E-{P}
4                   Trouver a et b tels que P et Q passent par la droite
                    y=ax+b
5                   Discrétiser a et b (les rendre entiers en les multipliant
                    par un scalaire adapté)
6                   SI TABLE[a][b] = vrai
7                         ALORS renvoyer vrai
8                   FIN SI
9                   Placer TABLE[a][b] = vrai
10             FIN POUR
11             Eliminer P de l'ensemble E
12        FIN TANT QUE
13        Renvoyer faux
14   FIN
```

Calcul de la complexité en temps :

- la boucle de la ligne 2 s'exécute n fois
- la boucle de la ligne 3 s'exécute au plus n-1 fois
- les autres opérations s'exécutent en temps constant

La complexité totale est donc de n*(n-1) = O(n*n)

Note : bien sûr, il faudra se débrouiller avec les coordonnées négatives (addition par un scalaire), ainsi que les droites du type x=c (stockage dans une table à part). Quant à l'optimisation du stockage de la table (par exemple en utilisant plutôt une table de hachage), si l'accès à la table TABLE[a][b] ne se fait plus en O(1), c'est perdu ;-)