Commentaires DS n°4

Généralités

- 3 exercices avec un énoncé ou algorithme un peu compliqué (sujets de concours ...), **à bien comprendre** avant de pouvoir traiter les questions, qui sont souvent progressives
- quelques questions "difficiles"
- gestion du temps : il faut consacrer du temps à chaque exercice
- plusieurs confusions C / OCaml (/ Python)

Exercice 1 : médiane des médianes en OCaml

- en OCaml, principalement sur les listes : récursivité, et filtrage
- **sans utiliser les aspects impératifs**
- pour les fonctions auxiliaires (qui étaient ici inutiles), il faut préciser leur spécification : rôle des paramètres et ce qui est renvoyé.
- ne pas oublier qu'un filtrage doit être exhaustif
- attention au parenthésage des paramètres : `f n+1` $\neq$ `f (n+1)`
- penser à utiliser les fonctions précédemment définies, si c'est pertinent
- trop d'erreurs sur les manipulations de listes
  - `::` et `@` ne sont pas équivalents
  - écrire `x::l` plutôt que `[x]@l` (qui réalise un appel de fonction)
  - écrire `[x;y;z]` est un peu plus concis que `x::y::z::[]` (mais c'est équivalent)
- **Q1 - Q2 - Q3**. déjà vu en DS ... attention à ne pas oublier un élément en cours de route ...
- **Q4**. `selection_n` doit déclencher une erreur si la liste est trop courte (et pas renvoyer `-1`
- **Q5**. `paquets_de_cinq`. Il y a une solution simple assez astucieuse. On renvoie une liste de listes.
- **Q6**. `medians`. on prend en paramètre une liste de listes (mais ça reste une liste). Ne pas oublier de trier chacune des listes
- **Q7**. `partage`. très facile récursivement, mais il faut récupérer le résultat dans un quadruplet
- **Q8**. `selection`. on implémente l'algorithme. L'opérateur de division est `/` en OCaml. Les noms de variable ne doivent pas commencer par une majuscule en OCaml
- **Q9**. Récurrence difficile pour majorer la complexité

Exercice 2 : élément majoritaire en C

- **Q1**. unicité : on attend une explication détaillée en tenant compte de la partie entière
- **Q2**. `occurrences`. facile
- **Q3**. `majoritaire`. facile. Optimisation possible en s'arrêtant à la moitié du tableau (attention à l'indice précis ...)
- **Q4**. complexité en $\mathcal{O}(n^2)$, à rédiger correctement
- **Q5-Q6**. invariant de boucle à rédiger correctement : initialisation + conservation
- **Q7**. pas facile. Conclusion : le seul l'élément en sommet de pile est potentiellement majoritaire
- **Q8**. `majoritaire_efficace`. On implémente avec le sommet de pile et le nombre de balles dans la corbeille. Ne pas oublier la vérification finale.
- **Q9**. complexité en $\mathcal{O}(n)$, facile

Exercice 3 : listes à enjambement

- attention à la lecture de l'énoncé : `INT_MIN` et `INT_MAX` sont les plus petit et plus grand entiers machine. On ne traitera que des entiers strictement compris entre ces deux valeurs "sentinelles"
- allocation mémoire : `malloc` lorsqu'on a besoin d'un nouveau maillon ; `free` lorsqu'on n'en a plus besoin
- **Q1**. `init`. on alloue 2 maillons dans le tas ; on initialise **tous** les champs
- **Q2**. `localise`. pas facile à comprendre ... on cherche le maillon qui serait juste avant la valeur `v` si elle était présente (cela facilitera l'insertion et la suppression). On parcourt la chaîne en s'arrêtant dès que possible : on avance tant que `t->suivant->donnee > v`.
- **Q3**. mise en place de tests. L'énoncé est très précis
- **Q4**. erreur de type. on corrige
- **Q5**. erreur cas oublié. on corrige
- **Q6**. `supprime`. Ne pas oublier le cas où `v` est absent. Ne pas oublier de libérer le maillon supprimé
- **Q7**. complexité $\mathcal{O}(n)$ **où** n **est la longueur de la liste** (n n'est pas défini par l'énoncé)
- **Q8**. régions mémoire : subtil ... voir corrigé
- **Q9-Q10**. listes à enjambement : plus compliqué, mais faisable si le temps le permet