Matthieu Boyer
**Planche A**



## Question de Cours

Rappelez la définition d'un arbre binaire et de son nombre de nœuds.

## Treap

On s'intéresse à une structure de données appelée *treap*. Un treap est un arbre binaire où tous les noeuds ont une clef et une valeur associées de sorte que l'ensemble des clefs des nœuds forme un arbre binaire de recherche et que l'ensemble des valeurs des nœuds forme un tas minimum.

1. Rappelez rapidement les propriétés vérifiées par les ABR et les tas minimum.
2. Donnez un algorithme de recherche de la valeur associée à une clef.
3. Donnez un algorithme pour l'insertion d'un élément.
4. Donnez un algorithme pour la suppression d'un élément.
5. Donnez des algorithmes pour la scission et la fusion de treaps.

Les complexités recherchées sont les suivantes :

**Proposition 0.1 — Complexity.** — Time for a successful search : $\mathcal{O}(\texttt{depth(v)})$ where $\texttt{key}(v) = k$.

— Time for an unsuccessful search : $\mathcal{O}(\max(\texttt{depth}(v^-), \texttt{depth}(v^+)))$ where $\texttt{key}(v^-)$ is the predecessor of the searched key.

— Insertion Time : $\mathcal{O}(\max(\texttt{depth}(v^-), \texttt{depth}(v^+)))$ where $\texttt{key}(v^-)$ is the predecessor of the searched key.

— Deletion Time : $\mathcal{O}(\text{tree depth})$

— Split/Merge Time : same as insertion / deletion

En supposant que les valeurs sont i.i.d. uniformément distribuées sur $\mathbb{R}$, on peut montrer que la complexité amortie de toutes les opérations est logarithmique. On définit

$$X(i,k) = \begin{cases} x_i, \dots, x_k & \text{si } i < k \\ x_k, \dots, x_i & \text{sinon} \end{cases}$$

et on note $x_1, \dots, x_n$ les noeuds de l'arbre par ordre croissant des clefs.

6. Montrer que pour tout $i \neq k$, $i$ est un ancêtre propre de $k$ si et seulement si $x_i$ a la plus faible priorité de $X(i,k)$.
7. En déduire le résultat.

Matthieu Boyer
**Planche B**



### Question de Cours

Rappelez la définition d'un arbre binaire équilibré. On détaillera notamment les notations utilisées.

### Tri par Comparaison

On s'intéresse la complexité minimale d'un algorithme de tri par comparaison, c'est-à-dire d'un algorithme qui trie un ensemble en ne pouvant que comparer ses éléments deux à deux.

1. Donnez la complexité la plus basse que vous connaissez d'un algorithme de tri.

Un arbre de décision pour un algorithme de tri est un arbre binaire qui montre les différentes exécutions possibles de l'algorithme sur différents ensembles.

2. Donnez un arbre de décision pour le tri par insertion (on ajoute un à un les éléments au bon endroit).
3. Montrez que la hauteur maximale d'une branche de l'arbre est la complexité dans le pire des cas de l'algorithme de tri.
4. Montrez que la hauteur $h$ d'un arbre de décision vérifie $n \log n = \mathcal{O}(h)$. En déduire le résultat.

L'enveloppe convexe d'un ensemble de points du plan est un polygone les reliant de sorte que tout point soit ou bien un sommet du polygone soit à l'intérieur du polygone. On représentera un tel polygone par une liste ordonnée de ses sommets.

5. Montrez que calculer l'enveloppe convexe d'un ensemble ne peut pas être fait en moins de $\mathcal{O}(n \log n)$ dans le pire des cas. On pourra introduire un ensemble de points bien choisis permettant de se ramener au début de l'exercice.

Matthieu Boyer
**Planche C**



### Question de Cours

Comparez les opérations suivantes entre un ensemble représenté par un tableau ordonné et un arbre binaire rouge-noir.

— Trouver le minimum.

— Ajouter un élément.

### LCA

Dans cette exercice on s'intéresse aux requêtes de plus bas ancêtre commun (LCA). Le LCA de deux noeuds est le noeud le plus profond d'un arbre qui a pour descendant les deux noeuds de départ.

1. Donnez un algorithme naïf qui résout la requête de LCA.

La décomposition lourd-léger d'un arbre $T$ est définie comme suit. Pour chaque nœud, on dit que l'arête amenant au plus gros sous-arbre est lourde (on règle les égalités arbitrairement). Ceci définit un chemin $P$ dit lourd d'arêtes lourdes de la racine à une feuille. On répète l'opération sur chaque arbre de $T \setminus P$. L'union de ces chemins est appelée la décomposition lourd-léger de l'arbre.

2. Montrer que tout chemin d'une racine à une feuille croise $\mathcal{O}(\log n)$ chemins lourds.
3. On organise les chemins lourds dans un arbre. Comment trouver le LCA de $u$ et $v$ à partir de cet arbre ? En déduire une structure de données en espace linéaire permettant de résoudre toute requête de LCA en $\mathcal{O}(\log mn)$.
4. Décrire une structure de données $\mathcal{O}(n \log n)$ en espace qui gère les requêtes de LCA en$\mathcal{O}(\log \log n)$. Que vaut $\log \log 10^9$ ?

Considérons le parcours préfixe de l'arbre. Lorsqu'on voit un noeud pour la première fois, on écrit 1. Lorsqu'on voit un noeud pour la dernière fois, on écrit 0 (en comptant les remontées dans le parcours, donc).

5. Montrer qu'il y a une injection entre l'ensemble des arbres et l'ensemble des chaînes de 0 et de 1 définie comme ci-dessus.
6. En déduire une borne supérieure sur le nombre d'arbres ordonnés à $x$ nœuds.
7. Utiliser cette borne supérieure pour améliorer votre structure de données en espace $\mathcal{O}(n)$ pour les arbres d'arité constante.