Algorithmique avancée: Contrôle Partiel du Jeudi 25 Octobre 2018.

Le barème est donné à titre indicatif. La compréhension du sujet faisant partie de l'épreuve, **on ne répondra à aucune question**. Si vous rencontrez des ambiguïtés, vous expliquerez **sur votre copie** comment vous les interprétez.

Durée 2h. Seul document autorisé: 1 feuille A4 recto-verso.

Dans les exercices I et II, on testera les algorithmes sur les tableaux suivants :

T_A:

33	27	39	2	8	34	13	24	19	15

et T_B:

144	21	25	20	11	18

I - Tas binaires minimaux pour remplir un sac (4 points)

On désire remplir un sac avec le plus d'éléments (en nombre) possible, mais la capacité du sac est limitée à un poids $c \geq 0$. On dispose de deux ensemble T et T' contenant les poids des éléments que l'on peut sélectionner. L'algorithme suivant renvoie la somme des poids des éléments sélectionnés :

Function RemplirSac(T,T',c) *T, T' tableaux d'entiers positifs de tailles $m + m' = n > 0$ et $c \geq 0$*

```
1  S ← 0; k ← 0; BuildHeap(T); BuildHeap(T′)
2  repeat
3      S ← S + k
4      if T.Size ≠ 0 and (T'.Size = 0 or T[1] ≤ T′[1]) then k ← Remove(T)
5      else k ← Remove(T′)
6  until (T.Size = 0 and T'.Size = 0) or (S + k > c)
7  return S
```

où BuildHeap(T) est la fonction vue en cours qui construit un tas binaire **minimal** en-place à partir d'un tableau T, Remove(T) supprime et renvoie la clé à la racine du tas binaire T et T.Size renvoie le nombre d'éléments présents dans le tas T.

1) **(3 pts)** On exécute RemplirSac(T_A,T_B,30). Dessinez les arbres décrivant les Tas Binaires obtenus après la ligne **1** puis décrivez les deux tableaux obtenus après le premier passage dans la boucle ainsi que la valeur de S retournée. Aucune justification n'est demandée.
2) **(1 pt)** Donnez les complexités temporelles et spatiales en pire cas $T_{RemplirSac}$ et $S_{RemplirSac}$ par une expression Θ fonction de n. Justifiez.

II - Tas binomiaux pour remplir un sac (3 points)

On traite le même problème avec l'algorithme suivant:

Function RemplirSac2(T,T',c) *T, T' tableaux d'entiers positifs de tailles $m+m' = n > 0$ et $c \geq 0$*

```
1  S ← 0; k ← 0; H_A ← CreateBinomHeap();H_B ← CreateBinomHeap()
2  For i = 1 to m do Add(H_A,T[i]); For j = 1 to m′ do Add(H_B,T′[j])
3  H ← Merge(H_A,H_B)
4  repeat
5      S ← S + k
6      k ← RemoveMin(H)
7  until (head(H) =NIL) or (S + k > c)
8  return S
```

où CreateBinomHeap() est la fonction qui crée un tas binomial vide, Add(H,e) ajoute l'élément e au tas H, Merge(H_1, H_2) retourne un tas résultant de la fusion de deux tas, RemoveMin(H) supprime et retourne l'élément minimum du tas H, *head*(H) est l'adresse du premier noeud du tas H.

1) **(2.5 pts)** On exécute RemplirSac2(T_A,T_B,30). Dessinez les Tas Binomiaux H_A et H_B après la ligne **2** puis le tas H obtenu après la ligne **3**, puis après le premier passage dans la boucle ainsi que la valeur S retournée. Aucune justification n'est demandée.
2) **(0.5 pt)** Donnez la complexité spatiale en pire cas $S_{RemplirSac2}$ de cet algorithme par une expression Θ fonction de n. Justifiez.

Le tableau T_1:

33	27	39	2	3	10	7	144	25	21	20	16	18

sera utilisé pour tester les algorithmes des exercices III et IV.

III- B-Arbres (7 points)

1) **(2 pts)** On considère un B-Arbre vide B_0 de degré $t = 3$, dessinez B_0 après lui avoir inséré successivement les **13** éléments du tableau T_1 (on ne demande pas de détailler la construction).

Sachant que la fonction AllocateNode qui crée un noeud vide dans un B-Arbre alloue un espace mémoire contenant la place maximum nécessaire au stockage des clés et des adresses pour ce noeud, considérant que chaque clé est codée sur 2 octets et chaque adresse aussi :

2) **(1 pt)** Quelle place mémoire occupe B_0? Justifiez.
3) **(1 pt)** Donnez une expression en Θ en fonction de t et de n de la complexité temporelle en pire cas de la recherche d'un élément dans un B-Arbre de degré t et de taille n. Justifiez.

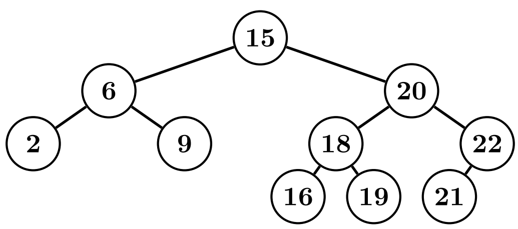
On peut rentabiliser le remplissage initial de l'arbre grâce à la méthode du "Bulk Loading" (chargement en masse). On classe d'abord les clés par ordre croissant puis on crée un noeud plein n_1 avec les premières clés, on insére la clé suivante dans un noeud n_0 parent de n_1, et on continue en remplissant un nouveau noeud n_2 (frère de n_1) avec les clés suivantes, etc. Plus généralement, lorsqu'un noeud du niveau le plus bas est plein on insère la clé suivante dans son premier ascendant qui a une place disponible (s'il n'en existe pas on crée un nouveau noeud racine), on redescend ensuite pour insérer les clés suivantes dans un nouveau noeud du niveau le plus bas (durant la descente on crée à chaque niveau un noeud sans clé contenant une seule adresse vers un fils).
Après ce "Bulk Loading", il peut exister des noeuds insuffisamment remplis à droite de l'arbre. Lorsqu'un noeud x est insuffisamment rempli, on réalise un transfert des dernières clés du noeud g (le noeud à gauche de x) vers x: la nouvelle répartition doit amener à avoir le même nombre de clés dans g et dans x à une clé prêt, cela amène aussi à échanger la clé du père de x avec une clé de g.

4) **(3 pts)** Construisez un B-Arbre B_1 de degré $t = 3$ contenant les éléments du tableau T_1 par la méthode du Bulk-Loading. Quelle place mémoire occupe-t'il? Pour chaque B-Arbre B_0 et B_1 donnez un élément dont la recherche demande le plus d'opérations et le nombre d'opérations nécéssaires.

IV- Arbres Binaires de Recherche (6 points)

1) **(1 pt)** On considère un arbre binaire de recheche B_2 vide, on y insère successivement les éléments du tableau T_1. Dessinez B_2 après ces insertions. Quelle est sa hauteur?
2) **(0.5 pt)** On considère la recherche d'un élément dans un arbre binaire de hauteur h et de taille n. Exprimez la complexité en pire cas en Θ de cette recherche. Quel est l'élément dont la recherche dans B_2 demande le plus d'opérations? (donnez l'élément et le nombre d'opérations nécessaires).
3) **(1.5 pts)** Proposez un tableau T_2 ayant les mêmes éléments que T_1 dans un autre ordre tel que l'insertion successive de ces éléments dans un arbre binaire de recherche vide B_3 donne un AVL. Quel est l'élément dont la recherche dans B_3 demande le plus d'opérations? (donnez l'élément et le nombre d'opérations nécessaires). Y a-t'il un gain en nombre d'opérations dans le pire des cas par rapport à la question précédente? Pourquoi?

A_1:



4) **(2 pts)** Pourquoi l'arbre A_1 ci-dessus est-il un AVL? Proposez un entier (non déjà présent) à ajouter à A_1 qui rompe sa propriété d'AVL, quel est le noeud le plus profond concerné par la perte de la propriété d'AVL? Comment peut-on rétablir la propriété d'AVL après l'insertion de cet élément? Précisez les noms des opérations à effectuer, dessinez les arbres résultants.
5) **(1 pt)** Donnez la complexité en Θ de l'ajout suivi du rétablissement de la propriété d'AVL dans le cas général d'un AVL de taille n (le noeud déséquilibré le plus profond étant repéré lors de l'ajout).