

Corrigé de l'examen final NF16 - Automne 2020

Exercice I (4 points)

1) Algorithme de tri par sélection :

```
Tri_Selection (T: tableau ; n: entier )
i,j,pos_min: entier
Pour i := 1 .. n -1
    pos_min:= i
    Pour j := i+1 .. n
        Si T[j]<T[ pos_min ]
            pos_min := j

    swap (T,i,pos_min )
```

Autres solutions acceptées : (tri pas sélection décroissant, tri par sélection en utilisant le maximum)

2) Complexité : $O(n^2)$, n étant le nombre d'élément dans le tableau

Justification brève : à chaque itération i , jusqu'à $n - 1$. On parcourt le tableau $(n - i)$ fois. Ce qui donne :


$$\sum_{i=1}^{n-1} (n - i) = \sum_{i=1}^{n-1} n = \frac{n(n-1)}{2}$$

3) Exécution de l'algorithme :

10	4	6	2	1
----	---	---	---	---

• Itération 1 :

- Sélectionne la case à l'indice 1
- Etape 1 : Parcourir le restant du tableau et trouver l'indice du minimum à partir de la case sélectionnée



10	4	6	2	1
----	---	---	---	---

10	4	6	2	1
----	---	---	---	---

- Etape 2 : Permuter les valeurs des deux cases sélectionnées

1	4	6	2	10
---	---	---	---	----

• Itération 2 :

- Sélectionne la case à l'indice 2
- Etape 1 : Parcourir le restant du tableau et trouver l'indice du minimum à partir de la case sélectionnée



1	4	6	2	10
---	---	---	---	----

1	4	6	2	10
---	---	---	---	----

- Etape 2 : Permuter les valeurs

1	2	6	4	10
---	---	---	---	----

- Itération 3 :

- Sélectionne la case à l'indice 3
- Etape 1 : Parcourir le restant du tableau et trouver l'indice du minimum à partir de la case sélectionnée

1	2	6	4	10
---	---	---	---	----

1	2	6	4	10
---	---	---	---	----

- Etape 2 : Permuter les valeurs des deux cases sélectionnées

1	2	4	6	10
---	---	---	---	----

- Itération 4 :

- Sélectionne la case à l'indice 4
- Etape 1 : Parcourir le restant du tableau et trouver l'indice du minimum à partir de la case sélectionnée

1	2	4	6	10
---	---	---	---	----

1	2	4	6	10
---	---	---	---	----

- Etape 2 : Permuter les valeurs des deux cases sélectionnées (ici le minimum est la case sélectionnée au départ donc on ne fait pas de permutation)

Exercice II (6 points)

Question 1. Lorsque l'on représente un tas sous forme de tableau, pour tout nœud d'indice i on a :

- son fils gauche à l'indice $gauche[i] = 2i + 1$;
- son fils droit à l'indice $droit[i] = 2i$;
- son père à l'indice $pere[i] = \lfloor \frac{i}{2} \rfloor$.

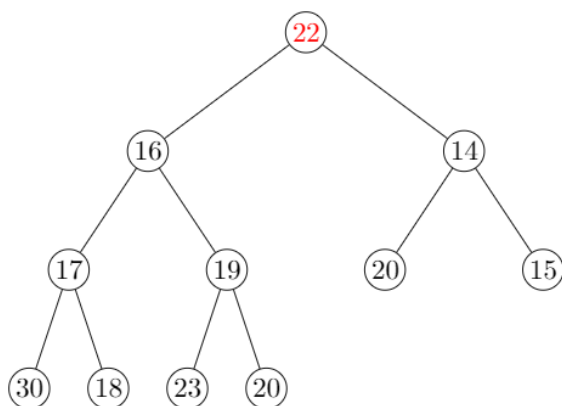
Le tas représenté par l'arbre de la Figure 1 a pour représentation sous forme de tableau :

12	16	14	17	19	20	15	30	18	23	20
----	----	----	----	----	----	----	----	----	----	----

Question 2. La fonction **Entasser** a une complexité en $O(\log_2 n)$, où n est le nombre de nœuds du tas. En effet, la fonction parcourt au pire l'arbre de la racine vers une feuille : sa complexité est donc en $O(h)$, avec h la hauteur du tas. Parce qu'un tas est un arbre parfait (ou complet, selon les définitions), on a $O(h) = O(\log_2 n)$. (*Note : cela a été prouvé en TD.*)

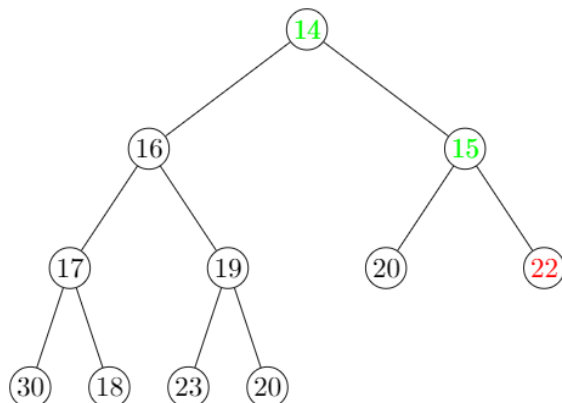
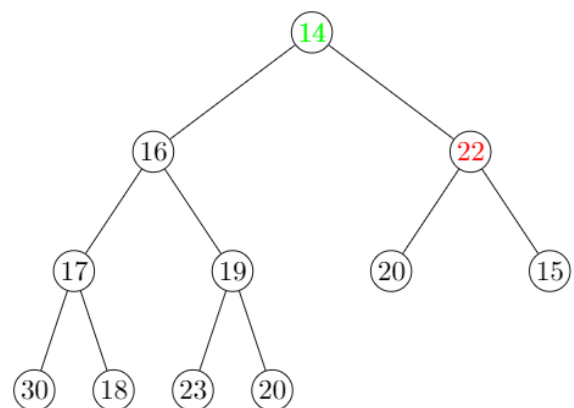
Remarque : On peut aussi utiliser la propriété d'arbre équilibré d'un tas pour justifier le lien entre la hauteur et $\log_2 n$, toutefois la démonstration n'a pas été vue en TD et n'est pas triviale (cf. Exercice 3 du final de Printemps 2010).

Question 3. a)



Etape 1 : On remplace la clé (12) du nœud d'indice 1 par sa nouvelle valeur augmentée 22. La structure de tas n'est plus respectée : la clé du nœud d'indice 1 est supérieure à celle de ses fils gauche et droit.

Etape 2 : On échange le nœud de clé 22 avec son plus petit fils, *i.e.* son fils droit de clé 14. La structure de tas n'est toujours pas respectée : la clé du nœud d'indice 3 (et de valeur 22, que l'on vient d'échanger) est supérieure à celle de ses fils gauche et droit.



Etape 3 : On échange le nœud de clé 22 avec son plus petit fils, *i.e.* son fils droit de clé 15. La structure de tas est désormais à nouveau respectée.

Remarque : On vient d'entasser le nœud dont on a augmenté la valeur.

```

Augmentation_Clé(T : tableau, i : entier, n : entier)
    Si n > T[i]
        T[i] := n
        Entasser(T, i)
    Fin Si
Fin

```

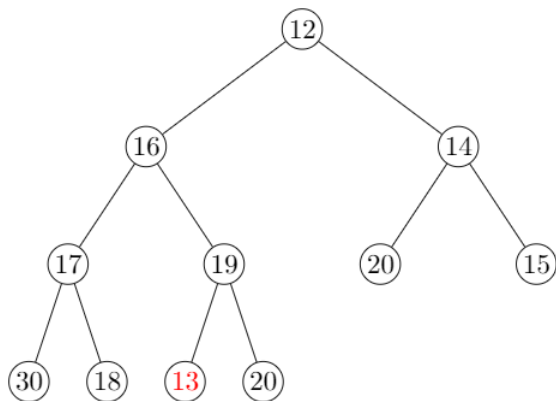
b)

Cet algorithme augmente la clé du nœud d'indice i à une nouvelle valeur n . Si cette nouvelle valeur ne représente pas une augmentation de la clé, on garde la valeur initiale de la clé. Sinon, on remplace la clé du nœud d'indice i par la valeur n , et on entasse le nœud modifié pour rétablir la structure de tas.

Remarque : Si l'on ne vérifie pas qu'il s'agit bien d'une augmentation de la valeur de la clé, on risque de diminuer la valeur de la clé : dans ce cas, entasser le nœud modifié ne permettrait pas de rétablir la structure de tas !

c) La complexité de cet algorithme équivaut à la complexité de la fonction Entasser, c'est-à-dire $O(\log_2 n)$.

Question 4. a)



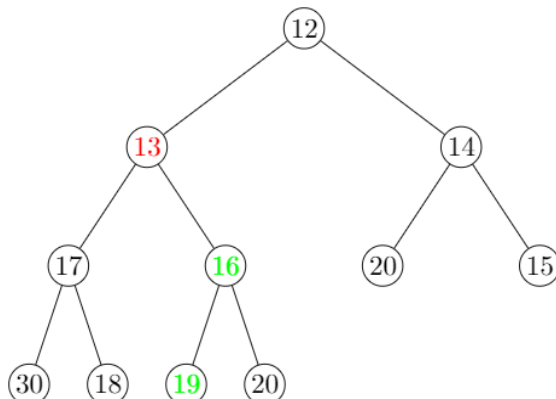
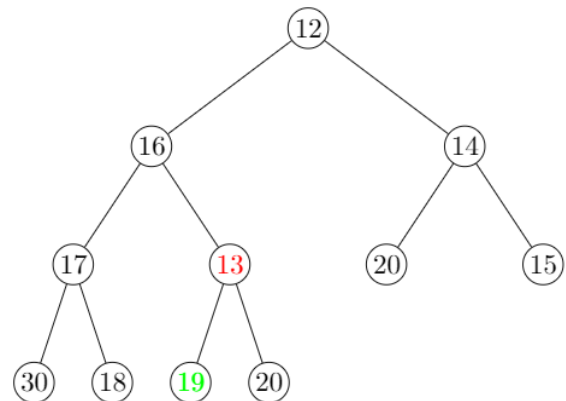
Etape 1 : On remplace la clé (23) du nœud d'indice 10 par sa nouvelle valeur diminuée 13.

La structure de tas n'est plus respectée : la clé du nœud d'indice 10 est plus petite que celle de son père.

Etape 2 : On échange le nœud de clé 13 avec son père, de clé 19.

La structure de tas n'est toujours pas respectée : la clé du nœud d'indice 5 (et de valeur 13, que l'on vient d'échanger) est inférieure à celle de son père.

Remarque : le sous-arbre de racine le nœud d'indice 5 est désormais un tas.



Etape 3 : On échange le nœud de clé 13 avec son père, de clé 16.

La structure de tas est désormais à nouveau respectée.

Remarque : Supposons que le nœud dont la clé est diminuée est le fils gauche (respectivement droit) de son père. Lorsque l'on échange ces deux nœuds, la relation d'ordre avec le fils droit (resp. gauche) du père est toujours respectée. De même, la relation d'ordre avec les fils du nœud fils est toujours respectée : il ne sera jamais nécessaire de descendre davantage le nœud père. Cela vient du fait que l'arbre était un tas avant la diminution de la clé du nœud.

b)

```

Diminution_Clé(T : tableau, i : entier, n : entier)
    Si n < T[i]
        T[i] := n
        Tant que i != 1 && T[i] > T[pere[i]]
            Echanger(T, i, pere[i])
            i := pere[i]
        Fin Tant Que
    Fin Si
Fin

```

Cet algorithme diminue la clé du nœud d'indice i à une nouvelle valeur n. Si cette nouvelle valeur ne représente pas une diminution de la clé, on garde la valeur initiale de la clé. Sinon, on remplace la clé du nœud d'indice i par la valeur n. On échange le nœud modifié et son père tant que la nouvelle clé du nœud a une valeur inférieure à celle de son père, et que l'on n'est pas arrivé à la racine.

c) Cet algorithme va, dans le pire des cas, faire remonter une feuille jusqu'à la racine. Sa complexité est donc, pour les mêmes raisons que la fonction **Entasser**, en $O(\log_2 n)$.

Question 5.

```

Modifier_Clé(T : tableau, i : entier, n : entier)
    Si n > T[i]
        Augmentation_Clé(T, i, n)
    Sinon Si n < T[i]
        Diminution_Clé(T, i, n)
    Fin Si
Fin

```

Cet algorithme modifie la clé du nœud d'indice i à une nouvelle valeur n, en conservant la structure de tas. Il fait appel aux fonctions **Augmentation_Clé** ou **Diminution_Clé** selon la valeur actuelle de la clé du nœud d'indice i. Ces fonctions conservent la structure de tas.

Exercice III (10 points)

Méthode 1 (Insérer les éléments d'un arbre dans l'autre):

R1. Insertion d'une clé dans un AVL (2 points)

```

Arbre insert_avl(Arbre A, int cle, int Delta)
{
    int D1,D2;
    Delta = 0;
    if(A == NULL)
    {
        Delta = 1;
        return creerNoeud(cle);
    }
    if (A->cle < cle)
    {
        A->gauche = insert_avl(A->gauche, cle, D1);
        if (A->equilibre > 0 || A->equilibre == 0 && D1 == 1) Delta = D1;
    }
}

```

```

        A->equilibre += D1;
    }
    if (A->cle > cle)
    {
        A->droit = insert_avl(A->droit, cle, D1);
        if (A->equilibre < 0 || A->equilibre == 0 && D1 == 1) Delta = D1;
        A->equilibre -= D1;
    }
    if(A->equilibre >= 2 || A->equilibre <= -2)
    {
        A = reequilibre(A, D2);
        Delta = Delta + D2;
    }
    return A;
}

```

R2. Première fonction de fusion (2 points)

```

Arbre fusion_1(Arbre A, Arbre B) {
    if (B == NULL) {
        return A;
    }

    int delta = 0;
    A = insert(A, B->cle, &delta);
    A = fusion_1(A, B->gauche);
    A = fusion_1(A, B->droit);

    return A;
}

```

Méthode 2 (Utiliser un tableau intermédiaire trié):

R3. a. Fonction arbre_vers_tab (1 point)

```

void arbre_vers_tab(Arbre A, int tab[], int* taille)
{
    if(A!=NULL)
    {
        arbre_vers_tab(A->gauche, tab, taille);
        tab[*taille]=A->cle;
        (*taille)++;
        arbre_vers_tab(A->droit, tab, taille);
    }
}

```

R3. b. Fonction tab_vers_arbre (0.75 points)

```

Noeud* tab_vers_arbre(int tab[], int debut, int fin)
{
    if (debut > fin) return NULL;
    int mil = (debut + fin)/2;
    Noeud *noeud = creerNoeud(tab[mil]);
    noeud->gauche = tab_vers_arbre(tab, debut, mil-1);
    noeud->droit = tab_vers_arbre(tab, mil+1, fin);
    noeud->hauteur = 1 + max(hauteur(noeud->gauche), hauteur(noeud->droit));
    return noeud;
}

```

R4. Deuxième fonction de fusion (2 points)

```
Arbre fusion_2(Arbre A, Arbre B)
{
    int taille=0, i=0,j=0;
    int tab1[MAX]; int tab2[MAX];
    int taille1=0; int taille2=0;
    arbre_vers_tab(A,tab1,&taille1);
    arbre_vers_tab(B,tab2,&taille2);
    while(i<taille1 && j<taille2)
    {
        if(tab1[i]<tab2[j]) tab[taille++]=tab1[i++];
        else if (tab1[i]>tab2[j]) tab[taille++]=tab2[j++];
        else{ tab[taille++]=tab2[j++]; i++; }
    }
    while(i<taille1) tab[taille++]=tab1[i++];
    while(j<taille2) tab[taille++]=tab2[j++];
    return tab_vers_arbre(tab,0, taille-1);
}
```

Complexité et comparaison :

Soient n et m les nombres d'éléments dans le premier et le deuxième arbre, respectivement.

R5. Complexité temporelle (0.75 points)**Méthode 1 : $O(m \cdot \log(n))$**

On insère les m éléments du premier arbre dans le deuxième. La complexité temporelle de l'insertion dans un AVL est $O(\log(n))$.

Méthode 2 : $O(n+m)$

On effectue des parcours d'arbre et de tableaux. Le parcours le plus long est celui du tableau contenant les éléments des deux arbres.

R6. Complexité spatiale (0.75 points)**Méthode 1 : $O(1)$**

À part les arbres on n'utilise qu'un nombre constant de variables.

Méthode 2 : $O(n+m)$

On a besoin d'un tableau pouvant contenir les éléments des deux arbres.

R7. Comparaison (0.75 points)

La première méthode est plus intéressante lorsqu'on a des contraintes d'espace et la deuxième quand on a des contraintes de temps.