TD d'algorithmique avancée

TD1 : Recherche du max et min

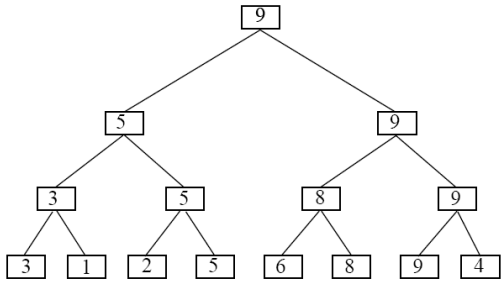
Recherche du maximum

1. Ecrivez un algorithme qui permet de trouver le maximum dans un ensemble à ***n*** éléments sous forme d'une fonction appelée MAXIMUM qui prend en argument un tableau et retourne la valeur maximale de ses éléments.
2. Calculez la complexité de votre algorithme en nombre de comparaisons avec les éléments du tableau.

Recherche du deuxième plus grand élément

Soit un ensemble d'éléments dont les valeurs ne sont pas doublées.

1. Ecrivez un algorithme simple qui permet de retrouver le deuxième plus grand élément sous forme d'une fonction appelée DEUXIEME_PLUS_GRAND() qui prend en paramètre un tableau de valeurs et retourne la valeur du deuxième plus grand élément.
2. Calculez est la complexité de votre algorithme simple en nombre de comparaisons avec les éléments du tableau.
3. Récrivez votre algorithme de recherche du maximum sous la forme d'un tournoi. Les éléments sont comparés entre eux deux à deux. Les gagnants seront comparés entre eux de nouveau jusqu'à ce qu'il reste un seul élément. Pour faciliter la tâche, on suppose que le nombre des éléments ***n*** est de la forme $n=2^x$. La figure suivante montre le déroulement de l'algorithme.



4. Calculer la complexité de cet algorithme en nombre de comparaison avec les éléments du tableau.
5. Proposez un nouvel algorithme de recherche du deuxième plus grand élément en utilisant une méthode de tournoi.
6. Calculez la complexité de votre nouvel algorithme en nombre de comparaisons avec les éléments du tableau.

Recherche du maximum et du minimum

Soit un ensemble E de taille paire de valeurs ne contenant pas deux fois la même valeur.

1. Ecrivez un algorithme simple qui permet de retrouver le maximum et le minimum de l'ensemble E de ***n*** éléments.
2. Calculez sa complexité en nombre de comparaisons avec les éléments du tableau.
3. Proposez un algorithme plus optimal.

Indications : Comparez les éléments par paire dans une première phase, en les triant les *max* dans les cases pairs et les *min* dans les cases impairs.

4. Calculez la complexité de cet algorithme en nombre de comparaisons avec les éléments du tableau.

TD d'algorithmique avancée

Corrigé du TD1 : Recherche du max et min

Recherche du maximum

1. Ecrivez un algorithme qui permet de trouver le maximum dans un ensemble à n éléments sous forme d'une fonction appelée MAXIMUM qui prend en argument un tableau et retourne la valeur maximale de ses éléments.

```
MAXIMUM(A)
    max A[1]
    pour i ← 2 à n faire
       si max < A[i] alors max A[i]
    renvoyer max
```

2. Calculez la complexité de votre algorithme en nombre de comparaisons.

   La complexité de notre algorithme en nombre de comparaison est $T(n-1)$.

Recherche du deuxième plus grand élément

Soit un ensemble d'éléments dont les valeurs ne sont pas doublées.

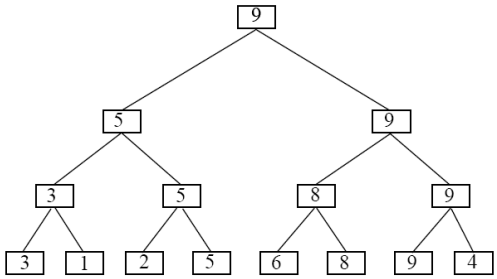
1. Ecrivez un algorithme simple qui permet de retrouver le deuxième plus grand élément sous forme d'une fonction appelée DEUXIEME_PLUS_GRAND() qui prend en paramètre un tableau de valeurs et retourne la valeur du deuxième plus grand élément.

```
DEUXIEME_PLUS_GRAND(A)
      rang_max ← 1
      pour i ← 2 à n faire
            si A[rang_max] < A[i] alors rang_max ← i
      si rang_max ≠ 1   alors rang_second ← 1
                        sinon rang_second ← 2
      pour i ← rang_second+1 à n faire
            si i ≠ rang_max et A[rang_second] < A[i] alors
                  rang_second ← i
      renvoyer A[rang_second]
```

2. Calculez est la complexité de votre algorithme simple en nombre de comparaisons ?

   La recherche du maximum coûte $n-1$ comparaisons. La boucle qui recherche le deuxième plus grand élément effectue $n-2$ comparaisons. D'où un coût total de $2n-3 \Rightarrow T(n)=(2n-3)$ comparaisons.

3. Récrivez votre algorithme de recherche du maximum sous la forme d'un tournoi. Les éléments sont comparés entre eux deux à deux. Les gagnants seront comparés entre eux de nouveau jusqu'à ce qu'il reste un seul élément. Pour faciliter la tâche, on suppose que le nombre des éléments n est de la forme $n=2^x$. La figure suivante montre le déroulement de l'algorithme.




```
MAXIMUM_ELEM(a, b)
      si a < b alors retourner b
      sinon retourner a
MAXIMUM_BINAIRE(A)
      pour i ← 2 à n , i ← i+2 faire
            B[i/2] = MAXIMUM_ELEM(A[i-1], A[i])
      retourner B
MAXIMUM(A)
      //On suppose que la taille de A = 2^x
      tant que ( taille de A > 1 ) faire
            A ← MAXIMUM_BINAIRE(A)
      retourner A[1]
```

4. Calculer la complexité de cet algorithme en nombre de comparaison.

   Remarquez que le nombre de niveau dans l'arbre binaire est $\log_2(n)$ ce qui donne qu'on a $\log_2(n)$ appels à la fonction MAXIMUM_BINAIRE(). Cette dernière appel la fonction MAXIMUM_ELEM() ($n/2$) fois dans le 1er niveau, ($n/4$) fois dans le 2ème niveau… et en général ($n/2^k$) fois dans le $k$ème niveau.

   Donc la complexité en nombre de comparaison est de l'ordre de: $\frac{n}{2}+\frac{n}{2^2}+...+\frac{n}{2^k}$

   avec $k=\log_2(n) \Rightarrow T(n)=\sum_{k=1}^{\log_2 n}\frac{n}{2^k}=n-1$ (exp : pour n=8, 7 comparaisons).

5. Proposez un nouvel algorithme de recherche du deuxième plus grand élément en utilisant une méthode de tournoi.

```
MINIMUM_ELEM(a, b)
      si a > b alors retourner b
      sinon retourner a
MAXIMUM_ELEM(a, b)
      si (a=max1 ou b=max1) alors retourner MINIMUM_ELEM(a, b)
      sinon si a < b alors retourner b
            sinon retourner a
MAXIMUM_BINAIRE(A)
      pour i ← 2 à n , i ← i+2 faire
            B[i/2] = MAXIMUM_ELEM(A[i-1], A[i])
      retourner B
DEUXIEME_PLUS_GRAND(A)
      //On suppose que la taille de A = 2^x
      tant que ( taille de A > 2 ) faire
            A ← MAXIMUM_BINAIRE(A)
      retourner MINIMUM_ELEM(A[1], A[2])

//Exécution de l'algorithme
max1 = -1 //Initialisation de max1 à -1 avant le premier appel
max1 = DEUXIEME_PLUS_GRAND(A) //phase 1 : On cherche le max1
max2 = DEUXIEME_PLUS_GRAND(A) //phase 2 : On cherche le max2
```

6. Calculez la complexité de votre nouvel algorithme en nombre de comparaisons.

Remarquez que le nombre de niveau dans l'arbre binaire est $\log_2(n)$ ce qui donne qu'on a $\log_2(n)$ appels à la fonction MAXIMUM_BINAIRE(). Cette dernière appel la fonction MAXIMUM_ELEM() ($n/2$) fois dans le 1er niveau, ($n/4$) fois dans le 2ème niveau… et en général ($n/2^k$) fois dans le $k$ème niveau.

Chaque appel à la fonction MAXIMUM_ELEM() effectue au maximum 3 comparaisons avec les éléments du tableau.

Donc la complexité en nombre de comparaisons est de l'ordre de: $3\cdot\frac{n}{2}+3\cdot\frac{n}{2^2}+...+3\cdot\frac{n}{2^k}$ avec k = $\log_2(n)$ $\Rightarrow$ la complexité d'une phase est $T'(n)=3\cdot\sum_{k=1}^{\log_2 n}\frac{n}{2^k}$

Donc $T(n)=2\cdot\left(3\cdot\sum_{k=1}^{\log_2 n}\frac{n}{2^k}\right)=6\cdot\sum_{k=1}^{\log_2 n}\frac{n}{2^k}=6\cdot(n-1)$ (exp : pour n=8, 42 comparaisons).

Recherche du maximum et du minimum

Soit un ensemble E de taille paire de valeurs ne contenant pas deux fois la même valeur.

1. Ecrivez un algorithme simple qui permet de retrouver le maximum et le minimum de l'ensemble E de n éléments.

```
MAXIMUM_ET_MINIMUM(A)
   max ← A[1]
   pour i ← 2 à n faire
         si max < A[i] alors max ← A[i]
   min ← A[1]
   pour i ← 2 à n faire
         si min > A[i] alors min ← A[i]
   retourner max et min
```

2. Calculez sa complexité en nombre de comparaisons ?

   T(n) = $2n$-2 comparaisons.

3. Proposez un algorithme plus optimal.

   Indications : Comparez les éléments par paire dans une première phase, en les triant les *max* dans les cases pairs et les *min* dans les cases impairs.

   L'algorithme se décompose en trois phases :

   a) On compare les éléments de l'ensemble par paire. On met dans les cases paires les plus grands éléments et dans les cases impaires les plus petits.

   b) On recherche le minimum parmi tous les plus petits éléments.

   c) On recherche le maximum parmi tous les plus grands éléments.

```
Maximum-et-Minimum(A)
      pour i ← 1 à n-1 faire par pas de 2
            si A[i] > A[i+1] alors échanger A[i] et A[i+1]
      min ← A[1]
      pour i ← 3 à n faire par pas de 2
            si A[i] < min alors min ← A[i]
      max ← A[2]
      pour i ← 4 à n faire par pas de 2
```

```
            si A[i] > max alors max ← A[i]
        retourner max et min
```

4. Calculez la complexité de cet algorithme en nombre de comparaisons ?

   Observons indépendamment le coût des trois boucles :

   Dans la première boucle il existe $\frac{n}{2}$ paires, ce qui fait $\frac{n}{2}$ comparaisons.

   Dans la 2ème boucle, parmi ***n*** éléments on a $\frac{n}{2}$ éléments de rangs impairs. Dans cette phase on effectue donc $\frac{n}{2}-1$ comparaisons.

   Dans la 3ème boucle on effectue aussi $\frac{n}{2}-1$ comparaisons.

   D'où la complexité totale est :

   $$T(n)=\frac{n}{2}+2\cdot\left(\frac{n}{2}-1\right)=\frac{n}{2}+n-2=3\cdot\frac{n}{2}-2$$