EXAMEN (Corrigé)

| **Module:** Algorithmique avancée | **Niveau:** 4Info B1, 4Info B2 |
| --- | --- |
| **Documents:** Non autorisés | **Durée:** 1h30 min |
| **Professeur :** Hamrouni K. | **Date :** 15 Avril 2010 |

***Exercice-1 :*** *(2 points)*

*Considérer la fonction placée dans l’encadré-1. Donner en fonction de N le nombre de mots « BONJOUR » imprimés par l’appel « Afficher(N) » (expliquer votre réponse).*

**Solution :**

La boucle FOR a une longueur N et la boucle WHILE a une longueur log(N) Le nombre de « Bonjour » affiché est donc nlog(n).

***Exercice-2 :*** *(5 points)*

1. *Citer deux méthodes de tri ayant une complexité en O(N2) et expliquer (deux lignes maximum) le principe de chacune.*
2. *Citer deux méthodes de tri (autre que la méthode de « tri rapide » (Quicksort)) ayant une complexité en O(Nlog(N)). Expliquer en deux lignes maximum le principe de chacune.*
3. *La méthode de « tri rapide » a une complexité au meilleur O(Nlog(N)) et une complexité au pire O(N2). Expliquer dans quelle situation, elle peut avoir chacune des deux complexités.*

**Solution :**

1. Méthodes de tri ayant une complexité *O(N2)* : Méthode de tri par sélection : Sélection le minimum et le déplacer en tête du tableau. Méthode de tri à bulles : parcourir le tableau, comparer chaque paire d’éléments adjacents et les permuter si le premier est plus grand. Refaire le parcours et les permutations tant que des permutations ont été effectuées.
2. Méthodes de tri ayant une complexité O(nlogn) : tri par insertion binaire et tri par fusion. Le tri par insertion binaire : on suppose qu’un partie du tableau a été triée et l’autre non triée. Prendre le premier élément de la partie non triée, chercher son emplacement dans la partie triée et le déplacer. La recherche se fait d’une façon binaire. Tri par fusion : diviser le tableau en deux parties, trier chaque partie et fusionner.
3. La méthode de tri rapide opère en partitionnant le tableau en deux parties. Si les deux parties ont presque la même longueur, la complexité est O(nlogn). Par contre si une partie a une longueur 1 et l’autre (n-1) alors la complexité est *O(N2).*

***Exercice-3 :*** *(7 points)*

*Soit un tableau X composé de N entiers.*

1. *Ecrire une fonction « int Max1(int X[ ], int N) » récursive simple permettant de déterminer la position du maximum du tableau X. Calculer sa complexité en fonction de N et en nombre de comparaisons entre les éléments du tableau.*
2. *Ecrire une fonction « int Max2(int X[ ], int Left, int Right) » récursive opérant selon la démarche « diviser pour règner » permettant de déterminer la position du maximum des éléments du tableau X entre Left et Right. La fonction doit renvoyer la position du maximum. Calculer sa complexité en fonction de N et en nombre de comparaisons entre les éléments du tableau.*
3. *Ecrire une fonction « void Trier (int X[ ], int N) » permettant de trier le tableau X par ordre croissant. Vous devez opérer en sélectionnant le maximum et en le plaçant à la fin du tableau. La fonction doit appeler la fonction Max2. Calculer la complexité de Trier en fonction de N et en nombre de comparaisons entre les éléments du tableau.*
4. *Proposer une version récursive de la fonction « Trier » élaborée dans la question précédente. Calculer sa complexité.*

**Solution :**

1. Méthode récursive simple :

Int Max1 (int \*X, int N)

{ int a ;

if (N==1) return X[0] ;

a = Max1 (X, N-1);

if (a > X[N-1]) return a;

else return X[N-1];

}

Complexité:



1. Méthode récursive opérant selon la démarche « diviser régner » :

Int Max2 (int \*X, int left, int right)

{ int m, y1, y2 ;

if (left == right) return left ;

m= (left+right)/2;

y1= Max2 (X, left, m);

y2= Max2 (X, m+1, right);

if (X[y1]>X[y2]) return y1;

else return y2;

}

Complexité:



1. Méthode de tri:

void Trier ( int \*X, int N)

{ int i, k, w ;

for ( i=N-1 ; i > 0 ; i-- )

{ k = Max2 ( X , 0, i ) ;

w = X[k] ; X[k] = X[i] ; X[i] = w ; // permuter

}

}

Complexité:



1. Méthode de tri récursive :

void Trier ( int \*X, int N)

{ int k, w ;

If ( N < 2 ) return;

k = Max2 ( X , 0, N-1) ;

w = X[k] ; X[k] = X[N-1] ; X[N-1] = w ; // permuter

Trier ( X , N-1);

}

Complexité:



***Exercice-4 :*** *( 6 points)* 

*Etant donné un arbre binaire de recherche A. Chaque nœud contient : X : une valeur entière, G : un pointeur sur le fils gauche et D : un pointeur sur le fils droit. La structure « arbre » contient un seul pointeur « racine » pointant sur le premier nœud de l’arbre (voir encadré-2).*

*Ecrire une fonction « void AfficherDescendant (arbre \*A, int V) » permettant d’afficher par ordre croissant tous les descendants du nœud ayant la valeur V. Si ce nœud n’existe pas, elle n’affiche rien.*

**Solution :**

Pour afficher les descendants du nœud ayant la valeur V, il faut d’abord le chercher et s’il est trouvé, il faut effectuer un parcours infixe de tout l’arbre ayant ce nœud comme racine.

Node \* Chercher(Arbre A, int V)

{ return find (A.Racine,V) ;

}

Node \*find(Node \*Père, int V)

{ if (Père == NULL ) return NULL ;

If(Pere -> X == V)return Pere;

If(V < Pere->X) return Find (Pere->G, V);

Return Find (Pere-> D, V);

}

void Afficher (arbre A, int V)

{ node \*P ;

P = Chercher (A.racine,V) ;

If (p != NULL) Infixe(P) ;

}

void Infixe (node \*P)

{ if ( p != NULL)

{ Infixe ( p -> G) ;

Printf( « %d » , p->X) ;

Infixe ( p-> D) ;

}

}