Université Cadi Ayad- Marrakech

Faculté des Sciences – Semlalia

Département Informatique

TD4

Algorithmique II SMI3

Exercice 1

1. Version itérative recherche dichotomique

```
recher-dicho (T,k)
i <- 0, j <- (n-1)
tant que i<j faire
        mid <- (i+j)/2
        si k>T[mid] alors
                i <- mid+1
       sinon j <- mid
      fin si
 retourner i
```

```
int recher_dicho(int T[],int n, int k){
   int i,j;
 i =0, j=(n-1) ;
 while( i<=j) {
 int mid=(i+j)/2 ;
   if (k>T[mid])
```

```
            i=mid+1 ;
      else j = mid-1;
    }
    return i;
    }
    int main(){
       int k=10;
       int T[]={2,5,7,8,9};
       printf("%d",recher_dicho(T,5, k));
    }
```

2. Version récursive recherche dichotomique

```
Recherc_dich_recursive(T,k,i,j)
      si i=j alors retourne i
       sinon mid <- (i+j)/2
            si k>T[mid] alors retourne Recherc_dich_recursive(T,k,mid+1,j)
            sinon retourne Recherc_dich_recursive (T,k,i,mid)
           fin si
     fin si
```

```
#include <stdio.h>
int Recherc_dich_recursive(int T[],int k, int i,int j) {
   int mid;
      if (i=j)
      return i;
      else
      mid =(i+j)/2 ;
            if(k>T[mid])
```

```
            return Recherc_dich_recursive(T,k,mid+1,j) ;
             else
             return Recherc_dich_recursive (T,k,i,mid) ;

}
int main(){
   int k ;
   int T[]={2,5,7,8,9};
   printf("%d",Recherc_dich_recursive(T,k,0,5));
}
```

Exercice 2 :

```
#include<stdio.h>
struct article
{int numero;
char nom[20];
int qte_stock;
float prix;
} ;
void SaisieArticle (struct article * art)
{
    printf (" numero ? \n");
    scanf (" %d",&art->numero);
    printf (" nom ? \n ");
    scanf("%s",art->nom);
    printf (" quantité en stock ? \n ") ;
    scanf (" %d" , &art->qte_stock);
    printf (" prix ?\n ");
```

```
scanf (" %f" ,&art->prix);
}
void AfficheArticle (struct article art)
{
    printf (" Cet article a pour : \n ");
    printf (" \t numéro : %d \n ", art.numero);
    printf (" \t nom : %s \n ", art.nom);
    printf (" \t stock : %d \n ", art. qte_stock);
    printf (" \t prix : %f \n ", art.prix);
}
        void SaisieTabArticle(struct article T[ ], int n)
{     int i;
        for(i=0;i<n;i++)
          {   printf (" saisie de l'article %d \n " ,i+1);
              SaisieArticle (&T[i]);
}
}
        void AfficheTabStock(struct article *T, int n, int q)
{ int i;
printf (" Les articles ayant un stock >=%d sont : \n " ,q);
for(i=0;i<n;i++)
if(T[i].qte_stock>=q)  AfficheArticle(T[i]);
}
        int main( )
{ struct article T[3];
SaisieTabArticle(T, 3);
AfficheTabStock(T, 3, 2);
```

}

Exercice 3 :

1. L'analyse commence comme toujours par la boucle la plus interne, ici celle de lignes 3 à 5. La quantité d'opérations réalisé dans un tour de la boucle (ligne 4 essentiellement) ne dépend ni des valeurs des variables i et j, ni de la taille du tableau tab (ou de son contenu). On a en effet :

– une affectation x= ;

– trois additions et une soustraction j+1, i-1+j et x+ ;

– un accès à une case du tableau tab ;

– une multiplication *(j+1) ;

– une incrémentation i++ ;

– une comparaison j < 3.

Globalement, un tour de cette boucle réalise donc Θ(1) opérations. De plus, la boucle s'exécute exactement trois fois pour j prenant les valeurs 0, 1 et 2. Donc le nombre d'opérations réalisées par la boucle complète est un Θ(1). On constate ensuite que l'exécution complète de la boucle des lignes 3 à 5 constitue l'essentiel de l'exécution d'un tour de la boucle des lignes 2 à 5 (la boucle externe). Un tour de cette boucle prend donc Θ(1) opérations. Enfin, cette boucle externe s'exécute pour les valeurs de i allant de 1 à tab.length-2, bornes incluses. On a donc tab.length-2 exécutions du corps de la boucle, c'est-à-dire Θ(tab.length).

2. Le programme est constitué d'une phase d'initialisation (lignes 1 et 2), puis d'une boucle. On détermine d'abord le nombre d'exécution de la boucle. On remarque que la boucle s'exécute à chaque fois que i < n. Or, i est initialisé à 0 (ligne 1) et chaque exécution (tour) de la boucle incrémente i de 1 (ligne 9). La boucle s'exécute donc pour les valeurs de i 0, 1, etc. jusqu'à n-1, soit donc n fois. On constate ensuite que le corps de la boucle (lignes 4 à 9), contient un nombre d'instruction qui ne dépend ni de la valeur de i ni de celle de j. En effet, les deux lignes 5 et 7 contiennent chacune deux instructions (un calcul et une affectation). Les autres lignes (4 et 9) sont toujours exécutées. De ce fait, le temps d'exécution du corps de la boucle est donc un Θ(1). Comme la boucle s'exécute n fois, le temps d'exécution du programme est alors en Θ(n).

3. 2 affectations n fois 1 affectation 1 multiplication + 1 affectation + 1 addition, 1 multiplication, 1 affectation ! donc Nombre d'opérations : f2(n) = 2 + n*6 = 6n +2 L'algo 2 est donc en O(n).