Examen d'algorithmique (401 I)

Aucun document autorisé

1 Complexité d'algorithme récursif

On utilisera la fonction (dans $\Theta(1)$) $max_2(n_1, n_2)$ qui renvoie le maximum entre les deux nombres n_1 et n_2.

Soit l'algorithme $Max(T, deb, fin)$ qui renvoie le maximum d'un tableau T situé entre les indices (valides pour T) deb exclu et fin inclus (avec $deb < fin$)

si $fin - deb = 1$ **alors retourner** $T[fin]$;
si $fin - deb = 2$ **alors retourner** $max_2(T[fin-1], T[fin])$;
$tiers = deb + ((fin - deb)\ div\ 3)$;
retourner $max_2(Max(T, deb, tiers), Max(T, tiers, fin))$

Question 1

En fonction de quel paramètre n va-t-on définir la fonction f qui donne la complexité de cet algorithme ?

Question 2

Ètablir l'équation de récurrence de f.

Question 3

Résoudre cette équation. A quelle classe de complexité appartient Max ? On pourra admettre que pour tout k on a $\lfloor \frac{k}{3} \rfloor + \lceil \frac{2k}{3} \rceil = k$ [1].

Question 4

On coupe maintenant le tableau par moitié (et non plus $\frac{1}{3}$, $\frac{2}{3}$).
On décide de ne pas s'ennuyer avec les indices, mais d'utiliser deux fonctions, $RecDeb(T)$ et $RecFin(T)$, qui à partir d'un tableau T de n éléments fabriquent chacune un tableau, la première des $\lfloor \frac{n}{2} \rfloor$ éléments les plus à gauche de T, la deuxième des $\lceil \frac{n}{2} \rceil$ éléments les plus à droite de T. La complexité de chacune des ces deux fonctions est évidente.
Écrire l'algorithme $Max_b(T)$ qui utilise ces deux fonctions et renvoie le maximum du tableau T (on pourra utiliser $taille(T)$ qui renvoie la taille du tableau T).
Établir l'équation de récurrence de sa complexité. Montrer que cette complexité appartient $\theta(nlog(n))$.

1. $\lfloor \frac{p}{q} \rfloor$ désigne la partie entière par défaut de $\frac{p}{q}$ et $\lceil \frac{p}{q} \rceil$ sa partie entière par excès.

2 Arbres binaires de recherche contenant des clés égales

L'égalité entre clés pose un problème pour l'implémentation des arbres binaires de recherche.

```
void Sommet::InsererValeur(Valeur v){
  if (Cle(racine)>Cle(v))
    {if (SAG) SAG->InsererValeur(v) ; else GrefferSAG(new Sommet(v));}
  else
    {if (SAD) SAD->InsererValeur(v) ; else GrefferSAD(new Sommet(v));}
}
```

Remarque : on suppose que la complexité du calcule explicite de la clé (`cle(v)`) à partir de l'élément `v` est de complexité dans $\Theta(1)$.

Question 5

Quel ABR construit cette méthode quand on l'utilise pour insérer successivement n éléments dont les clés sont identiques ?

Question 6

Quel est la classe de complexité de l'insertion successive de n éléments dont les clés sont identiques ?

On veut modifier ce comportement en utilisant l'une des stratégies suivantes

1. Gérer une liste d'éléments ayant des clés égales à celle de x.
   Chaque sommet de l'ABR pointe alors sur une liste d'éléments.
2. Gérer un indicateur booléen b dans le sommet correspondant à un élément de clé x. Descendre à gauche ou à droite selon la valeur de b, qui alterne entre FAUX et VRAI chaque fois que le sommet est visité pendant l'insertion d'un sommet ayant la même clé que x.

Question 7

Que devient, pour chacune de ces deux stratégies, la complexité de l'insertion quand on l'utilise pour insérer successivement n éléments qu'on sait être différents mais dont les clés sont identiques ?

Question 8

Que devient, pour chacune de ces deux stratégies, la complexité de la recherche d'un élément `v` *?*