Algorithmique et Base de la programmation

Durée : *1h30*
Documents autorisés : **AUCUN**

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier sur votre copie.

Préambule

L'objectif de cet examen est d'étudier l'algorithme de tri par tas. Sa particularité est d'utiliser une structure de données particulière qu'est le tas, qui peut-être représenté à l'aide d'un tableau. Comme dans le cours, le tri qui nous intéresse ici, sera un tri croissant sur des entiers.

Les exercices de cet examen forme un tout. Mais il est possible de répondre à chaque question indépendemment en supposant résolues les questions précédentes.

1 Qu'est ce qu'un tas ? (1 point)

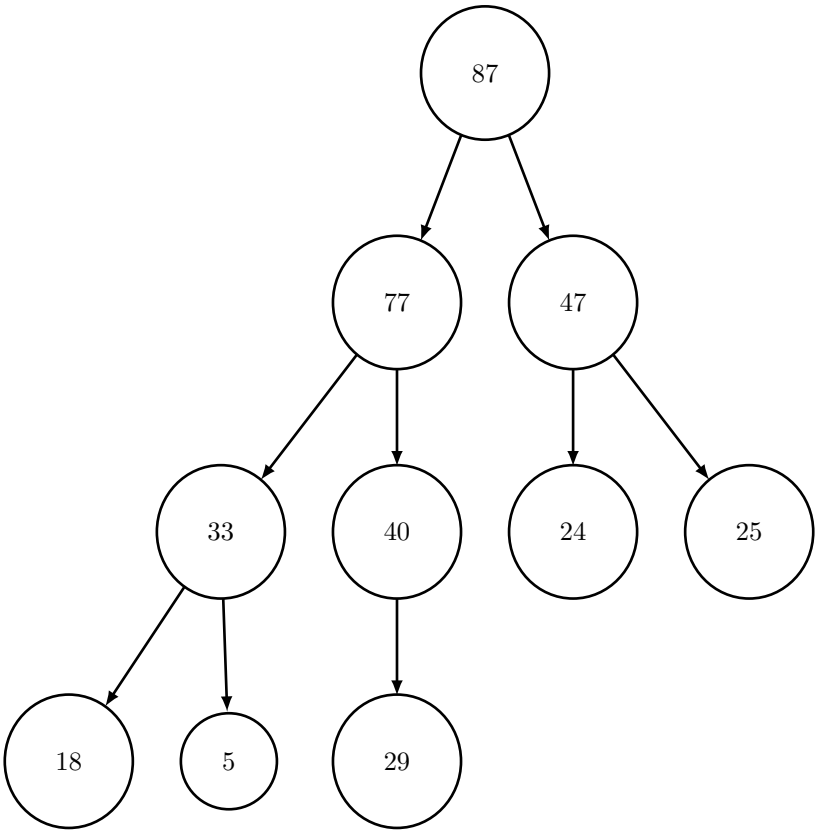
Un tas est un arbre binaire particulier : la valeur de chaque noeud est supérieure aux valeurs contenues dans ses sous-arbres et l'arbre est rempli par niveau (de gauche à droite), un nouveau niveau n'étant commencé que lorsque le précédent est complet.

Un tas peut être représenté l'aide d'un tableau t de telle sorte que les fils gauche et droit de $t[i]$ sont respectivement $t[2 * i]$ et $t[2 * i + 1]$.

Dessinez l'arbre binaire représenté par le tableau t suivant :

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
t	87	77	47	33	40	24	25	18	5	29

Solution proposée :



2 Fonction *estUnTas* (5 points)

Donnez l'algorithme récursif de la fonction suivante qui permet de savoir si un tableau t de n éléments significatifs représente un tas à partir de la racine de position i :

- **fonction** estUnTas (t : **Tableau**[1..MAX] **d'Entier** , i,n : **Naturel) : Booleen**
  ⌊**précondition(s)** i≤n

Solution proposée :

```
fonction estUnTas (t : Tableau[1..MAX] d'Entier , i,n : Naturel) : Booleen
   ⌊précondition(s)  i≤n
debut
   si 2*i > n alors
      retourner VRAI
   sinon
      si 2*i+1 > n alors
         retourner t[i]≥t[2*i]
      sinon
         si t[i]≥max(t[2*i],t[2*i+1]) alors
            retourner estUnTas(t,2*i,n) et estUnTas(t,2*i+1,n)
         sinon
            retourner FAUX
         finsi
      finsi
   finsi
fin
```

3 Procédure *faireDescendre* (6 points)

À l'issue de l'appel à cette procédure *faireDescendre*, l'arbre (représenté par un tableau) dont la racine est en position i sera un tas. On présuppose que les deux arbres dont les racines sont positionnées en $2i$ et $2i + 1$ sont des tas.

La signature de cette procédure est :

– **procédure** faireDescendre **(E/S** t : **Tableau**[1..MAX] **d'Entier** , **E** i,n : **Naturel)**

⌊**précondition(s)** (2*i≤n et estUnTas(t,2*i,n)) ou (2*i+1≤n et estUnTas(t,2*i,n) et estUnTas(t,2*i+1,n))

1. En supposant que la première valeur du tableau t de la partie 1 ne soit pas 87 mais 30. Donnez les valeurs de t après l'appel `faireDescendre(t,1,10)`.
2. Proposez l'algorithme de la procédure *faireDescendre*.
3. Donnez la complexité dans le pire des cas de votre algorithme. Justifiez.

Solution proposée :

1. (1 point) On obtient alors le tableau :

77	40	47	33	30	24	25	18	5	29

2. (4 points) L'algorithme est :

fonction indiceMax **(**t : **Tableau**[1..MAX] **d'Entier** , n,i,j : **Naturel) : Naturel**

⌊**précondition(s)** i≤n et i≤j

debut
  si j≤n **alors**
    si t[i]>t[j] **alors**
      retourner i
    sinon
      retourner j
    finsi
  sinon
    retourner i
  finsi
fin

Version itérative

procédure faireDescendre **(E/S** t : **Tableau**[1..MAX] **d'Entier** , **E** i,n : **Naturel)**

⌊**précondition(s)** (2*i≤n et estUnTas(t,2*i,n)) ou (2*i+1≤n et estUnTas(t,2*i,n) et estUnTas(t,2*i+1,n))

Déclaration elementBienPositionne :**Booleen**
posMax :**Naturel**

debut
  elementBienPositionne ← FAUX
  tant que non elementBienPositionne **faire**
    si 2*i≤n **alors**
      posMax ← indiceMax(t,n,2*i,2*i+1)
      si t[i]≥t[posMax] **alors**
        echanger(t[i],t[posMax])
        i ← posMax
      sinon
        elementBienPositionne ← VRAI
      finsi
    sinon
      elementBienPositionne ← VRAI
    finsi
  fintantque
fin

Version récursive

procédure faireDescendre **(E/S** t : **Tableau**[1..MAX] **d'Entier** , **E** i,n : **Naturel)**

⌊**précondition(s)** (2*i≤n et estUnTas(t,2*i,n)) ou (2*i+1≤n et estUnTas(t,2*i,n) et estUnTas(t,2*i+1,n))

Déclaration posMax :**Naturel**

debut
  si 2*i≤n **alors**
    posMax ← indiceMax(t,n,2*i,2*i+1)
    si t[i]≥t[posMax] **alors**
      echanger(t[i],t[posMax])

```
            faireDescendre(t,posMax,n)
        finsi
    finsi
fin
```

3. (1 point) À chaque itération l'indice de i est multiplié par 2 (à un près) jusqu'à ce que i soit plus grand que n, la complexité est donc en $log_2(n)$.

4 Procédure *tamiser* (4 points)

L'objectif de cette procédure est de transformer un tableau de n éléments significatifs quelconque en un tas. Pour ce faire on part du milieu du tableau en remontant jusqu'au premier élément du tableau pour qu'à l'issue de chaque itération l'arbre représenté par le tableau dont la racine est à la position i soit un tas.

1. Soit le tableau t suivant :

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
t	33	77	25	18	40	24	47	87	5	29

   Donnez les valeurs de ce tableau à l'issue de chaque itération.
2. Proposez l'algorithme de la procédure *tamiser*.
3. Donnez la complexité dans le pire des cas de votre algorithme. Justifiez.

Solution proposée :

1. (2 points) A l'issue de chaque itération on a :

i=5	33	77	25	18	40	24	47	87	5	29
i=4	33	77	25	87	40	24	47	18	5	29
i=3	33	77	47	87	40	24	25	18	5	29
i=2	33	87	47	77	40	24	25	18	5	29
i=1	87	77	47	30	40	24	25	18	5	29

2. (1 point) On obtient l'algorithme :

```
procédure tamiser ( E/S t : Tableau[1..MAX] d'Entier , E n : Naturel )
    Déclaration  i : Naturel
debut
    pour i ←n div 2 à 1 pas de -1 faire
        faireDescendre(t,i,n)
    finpour
fin
```

3. (1 point) L'algorithme est une boucle déterministe dont l'une des bornes est fonction de la taille du problème n, la complexité est donc n fois la complexité du corps de cette itération. On a donc une complexité dans le pire des cas qui de $n * log_2(n)$.

5 Procédure *trierParTas* (4 points)

Le principe du tri par tas est simple. Après avoir transformé le tableau t composé de n éléments significatifs en un tas, cet algorithme est composé d'itérations i (allant de n jusqu'à 2) qui :
– échange $t[1]$ et $t[i]$;
– s'assure que le tableau de $i-1$ éléments significatifs soit un tas.

Voici les différentes étapes de cet algorithme une fois que le tableau t de la partie 4 ait été transformé en tas (tableau de la partie 1) :

1	77	33	47	29	40	24	25	18	5	87
2	47	40	25	33	29	24	5	18	77	87
3	40	33	25	18	29	24	5	47	77	87
4	33	29	25	18	5	24	40	47	77	87
5	29	24	25	18	5	33	40	47	77	87
6	25	24	5	18	29	33	40	47	77	87
7	24	18	5	25	29	33	40	47	77	87
8	18	5	24	25	29	33	40	47	77	87
9	5	18	24	25	29	33	40	47	77	87

1. Dessinez l'analyse descendante *a posteriori* de ce problème.
2. Proposez l'algorithme de la procédure *trierParTas*.
3. Donnez la complexité dans le pire des cas de votre algorithme. Justifiez.

Solution proposée :

1. (1 point) à faire
2. (2 points)
   procédure trierParTas **(E/S** t : **Tableau**[1..MAX] **d'Entier** , **E** n : **Naturel)**
   debut
   tamiser(t,n)
   pour i ←n **à** 2 **pas de** -1 **faire**
   echanger(t[1],t[i])
   faireDescendre(t,1,i-1)
   finpour
   fin
3. (1 point) L'algorithme est composé d'un schéma séquentiel à deux instructions : tamiser est en $n * log_2(n)$ et la deuxième instruction (le pour) est aussi en $n * log_2(n)$. La complexité dans le pire des cas est en $n * log_2(n)$.