Informatique PYTHON. Structure de tas et tri par tas. Corrigé

1) a) Il y a 2^p sommets de hauteur p, puisque chaque sommet admet deux fils.
Donc il y a $1+2+...+2^p = 2^{p+1}-1$ sommets de hauteur $\leq p$.

b) Dans un tas de hauteur p, on a donc $2^p - 1 < n \leq 2^{p+1}-1$, c'est-à-dire $2^p \leq n < 2^{p+1}$.
Ce qui s'écrit aussi $p \leq \log n < p+1$, c'est-à-dire $p = \lfloor \log n \rfloor$.

2)

```
def G(i) :  return 2*i+1
def D(i) :  return 2*i+2
def P(i) :
     if i > 0 :  return (i-1)//2
     else :  return 0
```

3)

```
def ajout(L,x) :
     L.append(x) ; i = len(L)-1
     while i>0 and L[P(i)] > L[i] :    # en fait (i > 0) est inutile car P(0) = 0
          L[P(i)],L[i] = L[i],L[P(i)]
          i = P(i)
```

4)

a)

```
def minFils(i,L) :
     n = len(L)
     if G[i] >= n :  return i
     if G[i] == n-1 :  return G(i)
     if L[G(i)] <= L[D(i)] :  return G(i)
     else :  return D(i)
```

b)

```
def supprime(L) :
     x = L.pop() ; n = len(L)
     if n == 0 :  return x
     racine = L[0] ; L[0] = x ; i = 0 ; j = minFils(i,L)
     while L[j] < L[i] :     # important : s'arrête notamment lorsque  i = j
          L[j],L[i] = L[i],L[j]
          i = j ; j = minFils(i,L)
     return racine
```

5) Les procédures `ajout(L,a)` et `supprime(L)` demandent $O(p) = O(\log n)$ opérations,où p est la longueur maximale des branches (c'est-à-dire la hauteur de l'arbre).

6) a)

```
def creerTas(M) :
```

```
    L = []
    for x in M : ajout(L,x)
    L
```

La complexité est en $O(n \log n)$, puisque chaque ajout est en $O(\log n)$.

b)

```
def tri(M) :
    L = creerTas(M) ; M2 = []
    while L :
        M2.append(supprime(L))
    return M2
```

La complexité est en $O(n \log n)$, puisque chaque ajout et chaque suppression est en $O(\log n)$.

7) a)

```
def dictionnaire(L) :
    D = {} ; n = len(L)
    for i in range(n) :  D[L[i]] = i
    return D
```

b)

```
def ajout(x,L,D) :
    if (x in D) : return None    # on ne fait rien si x est un élément du tas
    L.append(x) ; n = len(L) ; i = len(L)-1 ; D[x] = i ;
    while i>0 and L[P(i)] > L[i] :
        y = L[P(i)]
        L[P(i)],L[i] = x,y
        D[x],D[y] = P(i),i
        i = P[i]
```

La complexité est toujours en $O(\log n)$.

c)

```
def modif(x,y,L,D) :
    n = len(tas) ; i = D[x] ; del D[x] ; L[i] = y ; D[y] = i
    if y < x :        # si y < x, il faut faire une percolation ascendante
        while i>0 and L[P(i)] > L[i] :
            z = L[P(i)]
            L[P(i)],L[i] = y,z
            D[x],D[z] = P(i),i
            i = P[i]
    elif y > x :        # si y > x, il faut faire une percolation descendante
        j = minFils(i,L)
        while L[j] < y :
            z = L[j]
            L[j],L[i] = y,z
```

```
        D[y],D[z] = D[z],D[y]
        i = j ; j = minFils(i,L)
```

8) a) La fonction auxiliaire `percol(M,i)` peut être définie en récursif (et aussi en itératif) :

```
def percol(M,i) :
    n = len(M) ; j = minFils(M,i)
    if M[j] < M[i] :  M[j],M[i] = M[i],M[j] ; percol(M,j)
```

b)

```
def tasPartiel(M,i) :
    if i < len(M) :
        tasPartiel(M,G(i)) ; tasPartiel(M,D(i)) ; percol(M,i)
```

Remarque : Si $i \geq n$, où n est la longueur M, alors `tasPartiel(M,i)` ne fait rien.

```
def creerTas(M) :
    if M == [] :  return []
    tasPartiel(M,0) ; return M
```

c) On a $c(n_p) \leq 2c(n_{p-1}) + Kp$, où $n_p = 2^p - 1$.

Donc $c(n_p) \leq 2^p c(n_0) + K(p + 2(p-1) + 4(p-2)... + 2^{p-1})$.

Or, $\sum_{k=0}^{p-1}(p-k)2^k = \sum_{j=1}^{p} j 2^{p-j} \leq \lambda 2^p$, où $\lambda = \sum_{j=1}^{+\infty} \dfrac{j}{2^j}$ série convergente, car $\dfrac{j}{2^j} = O_{+\infty}\left(\dfrac{1}{j^2}\right)$.

Remarque : En fait, pour $|x| < 1$, $\left(\dfrac{1}{1-x}\right)' = \sum_{n=0}^{+\infty} (x^n)'$, donc $\lambda = \dfrac{2}{(1-1/2)^2} = 2$.

On en conclut que $c(n) \leq (1+\lambda)2^p = 3n$.

Remarque culturelle : En résumé, en comparant les structures de listes triées et de tas, on obtient :

Structure	**liste triée**	**tas**
accès au min	$O(1)$	$O(1)$
suppression du min	$O(1)$	$O(\log n)$
ajout d'un élément	$O(n)$	$O(\log n)$