FTP_Alg_Week 2: Exercises with solutions

jungkyu.canci@hslu.ch

23 September 2024

Exercise 1 *Viewing a heap as a tree, we define the height of a node in a heap to be the number of edges on the longest simple downward path from the node to a leaf, and we define the height of the heap to be the height of its root.*

What are the minimum and maximum numbers of elements in a heap of height h?

Solution: An heap structure is build on a complete binary tree. We can define the depth or level k, that is the set of all nodes of the tree, which can be reached with exactly k edges moving down from the root. If the k–th level is the last one of the tree, than the height is k and we denote it with the letter h. Note that an intermediate level k–th level (which is full because the tree is complete and the level is an intermediate one) contains exactly 2^k nodes (one can prove it by induction). The last level, the h–th one with h the height, has a number o nodes in the range 1 to 2^h. Therefore the minimal number of nodes is

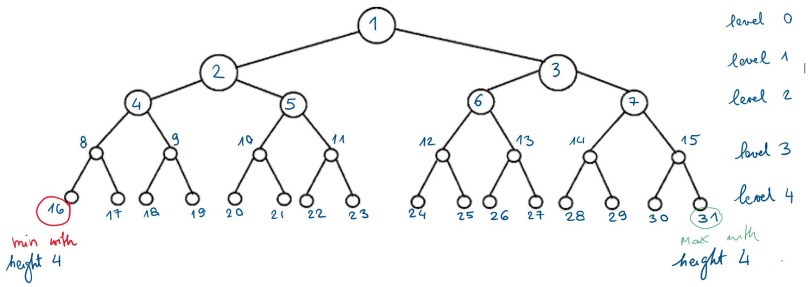
$$1 + 2 + \ldots + 2^{h-1} + 1 = 2^h - 1 + 1 = 2^h$$

We have used the formula

$$\sum_{i=0}^{n} a^n = \frac{a^{n+1} - 1}{a - 1}$$

where $a \neq 1$ (otherwise the sum is trivially $n + 1$). We have used it with $a = 2$. The maximal number of nodes is

$$1 + 2 + \ldots + 2^{h-1} + 2^h = 2^{h+1} - 1.$$



Exercise 2 *Show that an n-element heap has height* $\lfloor \lg n \rfloor$. *(Where* $\lg(\cdot)$ *denotes logarithm in base 2).*

Solution: From the previous exercise (in particular its solution) we have seen that

$$2^h \leq n \leq 2^{h+1} - 1$$

where h is the height of the complete binary tree. By taking the logarithm lg in base 2 at each term of the above sequence of inequalities, since lg is a monotone increasing function we have

$$h \leq \ln n \leq \ln(2^{h+1} - 1)$$

Now it is enough to note that

$$h = \ln 2^h \leq \ln n \leq \ln(2^{h+1} - 1) < h + 1$$

Thus we have

$$h = \lfloor h \rfloor \leq \lfloor \ln n \rfloor \leq \lfloor \ln(2^{h+1} - 1) \rfloor = h$$

Exercise 3 (*) *Show that, with the array representation for storing an* n*–element heap, the leaves are the nodes indexed by* $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

Solution: In the array representation of a complete binary tree supporting an heap, the root is in the position 1, its left child is in position 2, and its right child is in position 3. We can prove by induction that a node in the position k in the array, if it has a left child, this is in position $2k$ in the array and if it has a right child, this latter is in position $2k + 1$. It is clear that it is enough to prove the condition on the left child. Use the figure above to have an evidence of this fact. But, we formulate it in a lemma that we prove.

Lemma: *Let* v *be a node of a complete tree, which is not a leaf. Let* k *the index of* v*, then its left child* $lc(v)$ *has index* $2 \cdot k$.
Proof: Let us prove it by induction. The statement is clearly true for the root having index 1. Let's assume that the statement is true for m, then we prove that it is true for $m + 1$. If the node with index $m + 1$ is not a leaf (we denote it by u), also the node with index m, let us say the node w, is not a leaf. More precisely the node w have two children, because u has at least the left children (and recall that the tree is complete). By inductive hypothesis $lc(w)$ has index $2 \cdot m$. Therefore $rc(w)$ has index $2 \cdot m + 1$ and so $lc(u)$ has index $2 \cdot m + 1 + 1 = 2 \cdot (m + 1)$. This latter proves the lemma.

By using the previous lemma one can easily prove the following:

Lemma: *The parent of a node of index* n *has index* $\lfloor n/2 \rfloor$.
Proof: It is a straightforward consequence of previous lemma.

Now we can continue the solution of Exercise 3. If the last level (the one of the height h) is full, this means that the leaves are in the positions

$$2^h, 2^h + 1, \ldots, n = 2^{h+1} - 1$$

Note that in this case we have

$$\lfloor n/2 \rfloor + 1 = \lfloor (2^{h+1} - 1)/2 \rfloor + 1 = \lfloor 2^h - 1/2 \rfloor + 1 = 2^h - 1 + 1 = 2^h$$

If the h–th level is not full, we have leaves also at the level $h-1$. By the second lemma above the last parent is the one at level $h-1$ with index $\lfloor n/2 \rfloor$. Thus the first leave must have index $\lfloor n/2 \rfloor + 1$.

Exercise 4 *1. We consider the running time of a recursive algorithm $y(n)$. Suppose that $y(n)$ verifies the following:*

$$\begin{cases} y(1) = 0 \\ y(n) = y\left(\frac{n}{2}\right) + 1 \quad n \geq 1 \end{cases}$$

If possible calculate the running time.

2. We consider the running time of a recursive algorithm $y(n)$. Suppose that $y(n)$ verifies the following:

$$\begin{cases} y(1) = 0 \\ y(n) = 3y\left(\frac{n}{4}\right) + n^2 \log_2 n \quad n \geq 1 \end{cases}$$

If possible calculate the running time

3. We consider the running time of a recursive algorithm $y(n)$. Suppose that $y(n)$ verifies the following:

$$\begin{cases} y(1) = 0 \\ y(n) = 5y\left(\frac{n}{3}\right) + \log_2 n \quad n \geq 1 \end{cases}$$

If possible calculate the running time is

Solution: We apply

Master Theorem *Let $a \geq 1$, $b > 1$ be constants, $f(n)$ be a function and $T(n)$ defined by the following recurrence*

$$T(n) = aT(n/b) + f(n).$$

where n/b is interpreted as either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then

I) $T(n) = \Theta\left(n^{\log_b a}\right)$ if $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$.

II) $T(n) = \Theta\left(n^{\log_b a} \ln n\right)$ if $f(n) = \Theta\left(n^{\log_b a}\right)$.

III) $T(n) = \Theta\left(f(n)\right)$ if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$for some $\epsilon > 0$, and if $af(n/b) \leq c \cdot f(n)$ for all $n \geq n_0$, where c and n_0 are positive constant with $c < 1$.

1. We have $a = 1$, $b = 2$ and $f(n) = 1$. Since $n^{\log_b a} = n^0 = 1$, we have $f(n) = \Theta(1)$. Thus we are in the case II of Master Theorem. So $T(n) = \Theta(\ln n)$.

2. We have $a = 3$, $b = 4$ and $f(n) = n^2 \log_2 n$. Since $n^{\log_b a} = n^{\log_4 3}$, we have $f(n) = \Omega(n^{\log_4 3+\epsilon})$ for some $\epsilon > 0$ (for example take $\epsilon = \frac{1-\log_4 3}{2}$). Furthermore the condition

$$af(n/b) < cf(n)$$

for $n \geq 1$, with $c = 3$ (and $n_0 = 1$). Actually we have

$$3f(n/b) < 3f(n)$$

because the function $f(n) = n^2 \ln n$ is strictly increasing. Therefore for Master Theorem (in the case 3) we have $T(n) = \Theta(n^2 \ln n)$.

3. We have $a = 5$, $b = 3$ and $f(n) = \log_2 n$. Since $n^{\log_b a} = n^{\log_3 5} > n$, because $\log_3 5 > 1$, we have $f(n) = \log_2 n = O(n^{\log_3 5-\epsilon})$ for some $\epsilon > 0$ (actually for all ϵ such that $\log_3 5 - \epsilon > 0$). Therefore we are in the case 1 of Master Theorem and so $T(n) = \Theta\left(n^{\log_3 5}\right)$.