Course CSCE 750

By Y. MENG *email* : $y(dot)meng201011(at)gmail(dot)com$

exe. 6.1.1 What are the minimum and maximum numbers of elements in a heap of height h?

Since a heap is an almost-complete binary tree (complete at all levels except possible the lowest one). So, it has at most $(2^{h+1} - 1)$ elements (if it is a complete binary tree) and at least $((2^h - 1) + 1) = 2^h$ elements (if there is only one element in the lowest level).

MAXIMUM:

$$\begin{bmatrix} & & x & & \\ & x & & x & \\ \vdots & & \vdots & & \vdots \\ x & & \cdots & & x \end{bmatrix}_{\leftarrow 2^h \rightarrow}$$

$$number = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

MINIMUM:

$$\begin{bmatrix} & & x & & \\ & x & & x & \\ \vdots & & \vdots & & \vdots \\ & & \cdots & & \\ x & & \cdots & & x \end{bmatrix}_{\leftarrow 2^{h-1} \rightarrow [x \qquad]}$$

$$number = \sum_{i=0}^{h-1} 2^i + 1 = 2^h - 1 + 1 = 2^h$$

exe. 6.1.4 Where is a max-heap might be smallest element reside, assuming that all elements are distinct?

It could be any of the leaves, that is, elements with index between $\lfloor \frac{n}{2} \rfloor + 1$ and n. And the probability of each index is $\frac{1}{(\frac{n}{2})}$ (i.e., $\frac{2}{n}$).

exe. 6.1.5 Is an array that is in sorted order a min-heap?

Yes.

According to the definition, for any index i in the heap, both values of LEFT(i) and RIGHT(i) are larger than the value of NODE(i). And since the array is sorted, for any index i in array, the element $A[i]$ is less than $A[i * 2]$ and $A[i * 2 + 1]$, which represent LEFT(i) and RIGHT(i) respectively.

exe. 6.2.1 Using Fig. 6.2 as a model, illustrate the operation of MAX-HEAPIFY(A, 3) on the array A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]

MAX-HEAPIFY(A, 3)

$$\begin{vmatrix}
 & & & & & & & 27 & & & & & & \\
 & & & & & \swarrow & & & & \searrow & & & & \\
 & & & 17 & & & & & & & & [3] & & \\
 & & \swarrow & & \searrow & & & & & & \swarrow & & \searrow & \\
 & 16 & & & & 13 & & & & 10 & & & & 1 \\
\swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & \\
5 & & 7 & & 12 & & 4 & & 8 & & 9 & & 0 &
\end{vmatrix}$$

MAX-HEAPIFY(A, 6)

$$\begin{vmatrix}
 & & & & & & & 27 & & & & & & \\
 & & & & & \swarrow & & & & \searrow & & & & \\
 & & & 17 & & & & & & & & 10 & & \\
 & & \swarrow & & \searrow & & & & & & \swarrow & & \searrow & \\
 & 16 & & & & 13 & & & & [3] & & & & 1 \\
\swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & \\
5 & & 7 & & 12 & & 4 & & 8 & & 9 & & 0 &
\end{vmatrix}$$

MAX-HEAPIFY(A, 13)

$$\begin{vmatrix}
 & & & & & & & 27 & & & & & & \\
 & & & & & \swarrow & & & & \searrow & & & & \\
 & & & 17 & & & & & & & & 10 & & \\
 & & \swarrow & & \searrow & & & & & & \swarrow & & \searrow & \\
 & 16 & & & & 13 & & & & 9 & & & & 1 \\
\swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow & \\
5 & & 7 & & 12 & & 4 & & 8 & & [3] & & 0 &
\end{vmatrix}$$

---

exe.6.2.5 The code for *MAX-HEAPIFY* is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient *MAX-HEAPIFY* that uses an iterative control construct (a loop) instead of recursion.

MAX-HEAPIFY-ITERATIVE(A, i)

```
largest = -1
while (largest != i)
    l = LEFT(i)
    r = RIGHT(i)
    if l < A.heap_size and A[i] < A[l]
        largest = l
    else
        largest = i
    if r < A.heap_size and A[largest] < A[r]
        largest = r
    if largest != i
        SWAP A[i] and A[largest]
        i = largest
        largest = -1
```

---

exe.6.2.6 Show that the worst-case running time of *MAX-HEAPIFY* on a heap of size n is $\Omega(\lg n)$. (Hint: For a heap with n nodes, give node values that cause *MAX-HEAPIFY* to be called recursively at every node on a simple path from the root down to a leaf.)

Take the leftmost path in given heap, let the smallest element be the root and left-child is larger than right-child for every element, then

MAX-HEAPIFY will be called for h many times (h is the height), since it is called at each level in order to sink the smallest element to the leftmost leaf. Since $h = \lfloor \lg n \rfloor$, the worst-case running time of the procedure is $\Omega(\lg n)$.

Following solution is from "Algs, Instructor's Manual".

*If you put a value at the root that is less than every value in the left and right subtrees, then **MAX-HEAPIFY** will be called recursively until a leaf is reached. To make the recursive calls travers the longest path to a leaf, choose values that make **MAX-HEAPIFY** always recurse on the left child. It follows the left branch when the left-child $\geq$ right-child, so putting 0 at the root and 1 at all the other nodes, for example, will accomplish taht. With such values, **MAX-HEAPIFY** will be called h times (where h is the height of heap, which is the number of edges in the longest path from the root to a leaf), so its running time will be $\Theta(h)$ (since each call does $\Theta(1)$ work), which is $\Theta(\lg n)$. Since we have a case in which **MAX-HEAPIFY**'s running time is $\Theta(\lg n)$, its worst-case running time is $\Omega(\lg n)$.*

---

exe. 6.3.1 Using Fig. 6.3 as a model, illustrate the operation of *BUILD-MAX-HEAP* on the array A = [5, 3, 17, 10, 84, 19, 6, 22, 9].

BUILD-MAX-HEAP(A, 4), MAX-HEAPIFY(A, 4)

$$\begin{vmatrix} & & & & & & 5 & & & & & \\ & & & & & \swarrow & & \searrow & & & & \\ & & & 3 & & & & & & 17 & & \\ & & \swarrow & & \searrow & & & & \swarrow & & \searrow & \\ & [10] & & & & 84 & & 19 & & & & 6 \\ \swarrow & & \searrow & & & & & & & & & \\ 22 & & 9 & & & & & & & & & \end{vmatrix}$$

MAX-HEAPIFY(A, 8)

$$\begin{vmatrix} & & & & & & 5 & & & & & \\ & & & & & \swarrow & & \searrow & & & & \\ & & & 3 & & & & & & 17 & & \\ & & \swarrow & & \searrow & & & & \swarrow & & \searrow & \\ & 22 & & & & 84 & & 19 & & & & 6 \\ \swarrow & & \searrow & & & & & & & & & \\ [10] & & 9 & & & & & & & & & \end{vmatrix}$$

BUILD-MAX-HEAP(A, 3), MAX-HEAPIFY(A, 3)

$$\begin{vmatrix} & & & & & & 5 & & & & & \\ & & & & & \swarrow & & \searrow & & & & \\ & & & 3 & & & & & & [17] & & \\ & & \swarrow & & \searrow & & & & \swarrow & & \searrow & \\ & 22 & & & & 84 & & 19 & & & & 6 \\ \swarrow & & \searrow & & & & & & & & & \\ 10 & & 9 & & & & & & & & & \end{vmatrix}$$

MAX-HEAPIFY(A, 6)

$$\begin{vmatrix} & & & & & & 5 & & & & & \\ & & & & & \swarrow & & \searrow & & & & \\ & & & 3 & & & & & & 19 & & \\ & & \swarrow & & \searrow & & & & \swarrow & & \searrow & \\ & 22 & & & & 84 & & [17] & & & & 6 \\ \swarrow & & \searrow & & & & & & & & & \\ 10 & & 9 & & & & & & & & & \end{vmatrix}$$

BUILD-MAX-HEAP(A, 2), MAX-HEAPIFY(A, 2)

5

↙ ↘

[3] 19

↙ ↘ ↙ ↘

22 84 17 6

↙ ↘

10 9

MAX-HEAPIFY(A, 5)

5

↙ ↘

84 19

↙ ↘ ↙ ↘

22 [3] 17 6

↙ ↘

10 9

BUILD-MAX-HEAP(A, 1), MAX-HEAPIFY(A, 1)

[5]

↙ ↘

84 19

↙ ↘ ↙ ↘

22 3 17 6

↙ ↘

10 9

MAX-HEAPIFY(A, 2)

84

↙ ↘

[5] 19

↙ ↘ ↙ ↘

22 3 17 6

↙ ↘

10 9

MAX-HEAPIFY(A, 4)

84

↙ ↘

22 19

↙ ↘ ↙ ↘

[5] 3 17 6

↙ ↘

10 9

MAX-HEAPIFY(A, 8)

$$\left|\begin{array}{ccccccccc}
 & & & & 84 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & 22 & & & & 19 & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 & 10 & & & 3 & 17 & & & 6 \\
\swarrow & & \searrow & & & & & & \\
[5] & & 9 & & & & & &
\end{array}\right|$$

Terminate (i = largest = 1)

$$\left|\begin{array}{ccccccccc}
 & & & & 84 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & 22 & & & & 19 & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 & 10 & & & 3 & 17 & & & 6 \\
\swarrow & & \searrow & & & & & & \\
5 & & 9 & & & & & &
\end{array}\right|$$

---

exe.6.4.1Using Fig. 6.4 as a model, illustrate the operation of *HEAPSORT* on the array A = [5, 13, 2, 25, 7, 17, 20, 8, 4].

Build the max-heap

$$\left|\begin{array}{ccccccccc}
 & & & & 5 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & 13 & & & & 2 & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 & [25] & & & 7 & 17 & & & 20 \\
\swarrow & & \searrow & & & & & & \\
8 & & 4 & & & & & &
\end{array}\right|$$

$$\left|\begin{array}{ccccccccc}
 & & & & 5 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & 13 & & & & [2] & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 & 25 & & & 7 & 17 & & & 20 \\
\swarrow & & \searrow & & & & & & \\
8 & & 4 & & & & & &
\end{array}\right|$$

$$\left|\begin{array}{ccccccccc}
 & & & & 5 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & [13] & & & & 20 & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 & 25 & & & 7 & 17 & & & 2 \\
\swarrow & & \searrow & & & & & & \\
8 & & 4 & & & & & &
\end{array}\right|$$

$$\left|\begin{array}{cccccccccccc}
&&&&&&& [5] &&&& \\
&&&&& \swarrow &&& \searrow &&& \\
&&&& 25 &&&&& 20 && \\
&&& \swarrow && \searrow &&& \swarrow && \searrow & \\
&& 13 &&&& 7 & 17 &&&& 2 \\
& \swarrow && \searrow &&&&&&&& \\
8 &&&& 4 &&&&&&&
\end{array}\right|$$

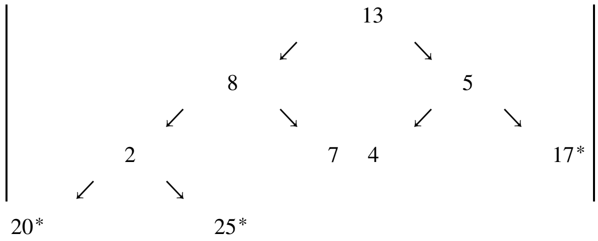
$$\left|\begin{array}{cccccccccccc}
&&&&&&& 25 &&&& \\
&&&&& \swarrow &&& \searrow &&& \\
&&&& 13 &&&&& 20 && \\
&&& \swarrow && \searrow &&& \swarrow && \searrow & \\
&& 8 &&&& 7 & 17 &&&& 2 \\
& \swarrow && \searrow &&&&&&&& \\
5 &&&& 4 &&&&&&&
\end{array}\right|$$

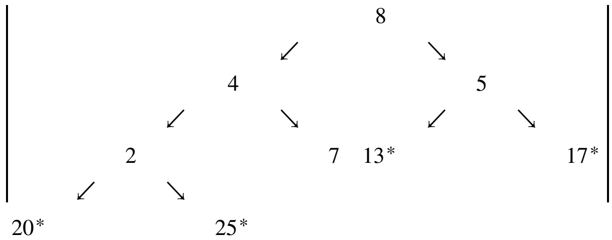
Heapsort

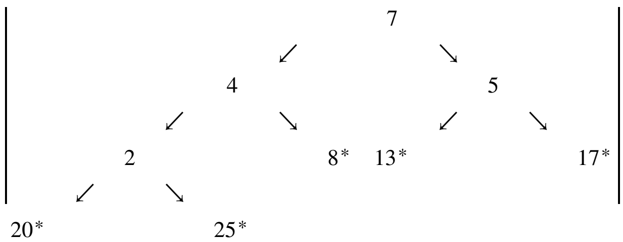
$$\left|\begin{array}{cccccccccccc}
&&&&&&& 25 &&&& \\
&&&&& \swarrow &&& \searrow &&& \\
&&&& 13 &&&&& 20 && \\
&&& \swarrow && \searrow &&& \swarrow && \searrow & \\
&& 8 &&&& 7 & 17 &&&& 2 \\
& \swarrow && \searrow &&&&&&&& \\
5 &&&& 4 &&&&&&&
\end{array}\right|$$

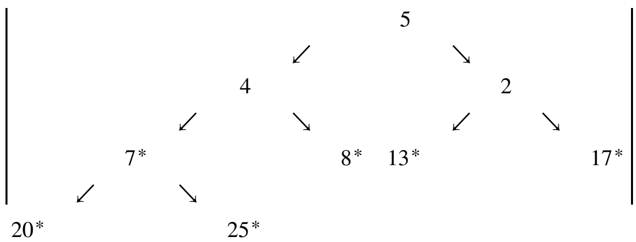
$$\left|\begin{array}{cccccccccccc}
&&&&&&& 20 &&&& \\
&&&&& \swarrow &&& \searrow &&& \\
&&&& 13 &&&&& 17 && \\
&&& \swarrow && \searrow &&& \swarrow && \searrow & \\
&& 8 &&&& 7 & 4 &&&& 2 \\
& \swarrow && \searrow &&&&&&&& \\
5 &&&& 25^* &&&&&&&
\end{array}\right|$$

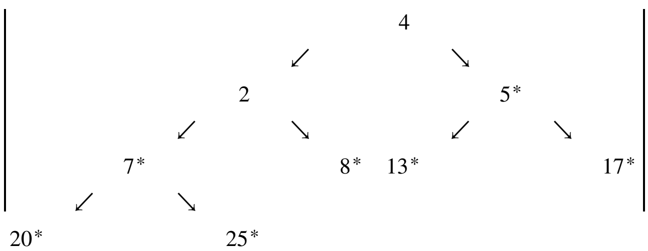
$$\left|\begin{array}{cccccccccccc}
&&&&&&& 17 &&&& \\
&&&&& \swarrow &&& \searrow &&& \\
&&&& 13 &&&&& 5 && \\
&&& \swarrow && \searrow &&& \swarrow && \searrow & \\
&& 8 &&&& 7 & 4 &&&& 2 \\
& \swarrow && \searrow &&&&&&&& \\
20^* &&&& 25^* &&&&&&&
\end{array}\right|$$











Terminate (only one element in the tree)

$$\left|\begin{array}{cccccccccccc}
 & & & & & & & 2 & & & & \\
 & & & & & \swarrow & & & \searrow & & & \\
 & & & & 4^* & & & & & 5^* & & \\
 & & & \swarrow & & \searrow & & & \swarrow & & \searrow & \\
 & & 7^* & & & & 8^* & 13^* & & & & 17^* \\
 & \swarrow & & \searrow & & & & & & & & \\
20^* & & & & 25^* & & & & & & &
\end{array}\right|$$

---

exe. 7.3.2 When *RANDOMIZED-QUICKSORT* runs, how many calls are made to the random-number generator *RANDOM* in the worst case? How about in the best case? Give your answer in terms of Θ-notation. (Hint: Write a recurrence for the number of random number generator, then solve that recurrence via the substitution method.)

Worst case:

$T(n) = T(n-1) + 1 = n = \Theta(n)$.

Best case:

$T(n) = 2T(\frac{n}{2}) + 1$

$n^{log_b a} = n^{log_2 2} = n^1$,

$f(n) = 1 = n^0 = O(n)$, case 3 of the Master Theorem applies. Thus,

$T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 2}) = \Theta(n)$.

---

exe.7.4.2 Show that quicksort's best-case running time is $\Omega(n\lg n)$.

Average

$T(n) = 2T(\frac{n}{2}) + \Theta(n)$

$n^{log_b a} = n^{log_2 2} = n$,

$f(n) = n = \Theta(n)$, case 2 of the Master Theorem applies. Thus,

$T(n) = \Theta(n^{log_b a}\lg n) = \Theta(n\lg n)$.

Therefore, $T(n) = \Omega(n\lg n)$.

In [ ]: