Learn DAA : From B K Sharma

TCS-503: Design and Analysis of Algorithms

Heapsort

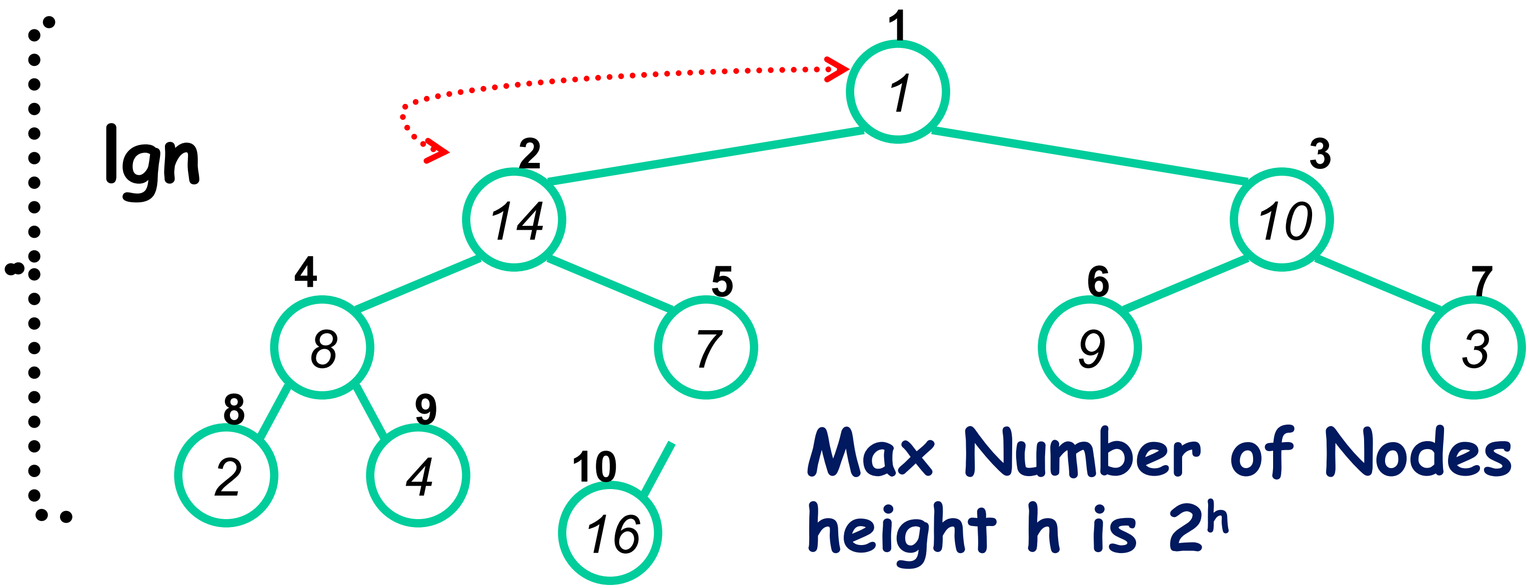
Analyzing MAX-HEAPIFY() Algorithm

```
MAX-HEAPIFY(A, i)
    l=LEFT(i)
    r=RIGHT(i)
    If l ≤ heap-size[A] and A[l] > A[i]   then
            largest ← l
    else
            largest ← r
    If r ≤ heap-size[A] and A[r] > A[largest]  then
            largest ← r
    If largest ≠ i   then
            Exchange A[i] ↔ A[largest]
            MAX-HEAPIFY(A, largest)
```

Analyzing HEAPSORT()

A = {1, 14, 10, 8, 7, 9, 3, 2, 4, 16}

MAX-HEAPIFY(A,1)



Max Number of Nodes n at height h is 2^h

$n=2^h$

$\lg n=\lg(2^h)$

$\lg n=h$

The running time on a node of height h is

$T(n)=O(\lg n)$

Analyzing BUILD-MAX-HEAP()

BUILD-MAX-HEAP(A)

1 heap-size[A]$\leftarrow$length[A]

2 For i$\leftarrow\lfloor$length[A]/2$\rfloor$ downto 1

3 do MAX-HEAPIFY(A,i) O(lgn)

O(n)

Specifically $\lfloor n/2 \rfloor$

Analyzing HEAPSORT()

```
HEAPSORT(A)

BUILD-MAX-HEAP(A)                              O(n)
For i← length[A] downto 2
    do exchange A[1] ↔ A[i]
        heap-size[A] ←heap-size[A] -1
        MAX-HEAPIFY(A,1)                       O(lg n)
```

n-1 times

BUILD-MAX-HEAP takes O(n)

Each of the n-1 calls to MAX-HEAPIFY takes time O(lg n)

Total time is O(n lg n)

$$\begin{aligned} T(n) &= O(n) + (n - 1)\, O(\lg n) \\ &= O(n) + n.O(\lg n) - O(\lg n) \\ &= O(n) + O(n \lg n) \\ &= O(n \lg n) \end{aligned}$$

Analyzing HEAPSORT()

The O(n log n) run time of heap-sort is much better than the O(n^2) run time of selection and insertion sort

Although, it has the same run time as Merge sort, but it is better than Merge Sort regarding memory space

Heap sort is in-place sorting algorithm

But not stable

Does not preserve the relative order of elements with equal keys

Summary

We can perform the following operations on heaps:

MAX-HEAPIFY	$O(\lg n)$
BUILD-MAX-HEAP	$O(n)$
HEAP-SORT	$O(n\lg n)$