Data Structures and Algorithms: Lecture 6

Barbara Morawska

August 28, 2018

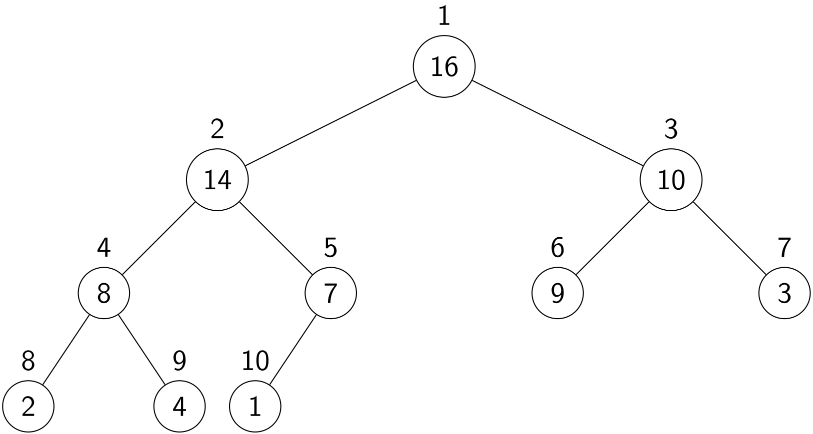
Heapsort

- running time: $O(n \lg n)$. Hence like MERGE-SORT, but not like INSERTION-SORT.
- sorts in place. Hence like INSERTION-SORT not like MERGE-SORT

Heap

- Not a "garbage-collected storage"
- Definition: **(binary) heap is a nearly complete binary tree**
- Nearly complete means all levels of the tree are filled except the last one which is filled from the left to the right to some point.

Example



Stored as an array:

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Example

Stored as an array:

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

The array has two attributes:

- *A.length*: number of all elements in the array,
- *A.heap−size*: number of elements of the heap stored in the array
  $A.heap{-}size \leq A.length$

The root of the tree is $A[1]$.

Height of a node in a heap:

the number of edges on a longest path from the node to the leaf

Height of the heap:

the height of the root

If the heap has n elements, its height is $\Theta(\lg(n))$.

Example

Stored as an array:

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Given an index i it is easy to compute the parent and children of i in the tree:

- $\textsc{Parent}(i)$

  1. **return** $\lfloor \frac{i}{n} \rfloor$

- $\textsc{Left}(i)$

  1. **return** $2i$

  $\backslash\backslash$ shifting the binary representation of i to the left adding 0 at the end.

- $\textsc{Rightt}(i)$

  1. **return** $2i + 1$

  $\backslash\backslash$ shifting the binary representation of i to the left adding 1 at the end.

Two kinds of heaps

There are two kinds of heaps

- max-heaps: satisfy max-heap property
- min-heaps: satisfy min-heap property

Max-heap property:

$$\forall (1 \leq i \leq A.heap{-}size) \quad A[\textsc{Parent}(i)] \geq A[i]$$

Maximal element is at the root.

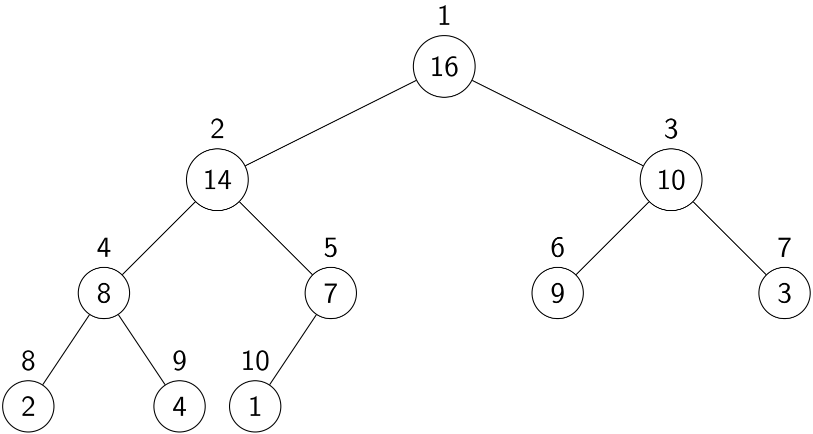
Min-heap property:

$$\forall (1 \leq i \leq A.heap{-}size) \quad A[\textsc{Parent}(i)] \leq A[i]$$

Minimal element is at the root.

We will use max-heap for sorting.

Example



Maintaining the max-heap property

Procedure MAX-HEAPIFY(A, i)

```
l = Left(i)
r = Right(i)
if l ≤ A.heap−size and A[l] > A[i] then
    largest = l
else
    largest = i
if r ≤ A.heap−size and A[r] > A[largest] then
    largest = r
if largest ≠ i then
    exchange values A[i] and A[largest]
    MAX-HEAPIFY(A, largest)
```

Trace it on: $A =< 16, 4, 10, 14, 7, 9, 3, 2, 8, 1 >, i = 2$.

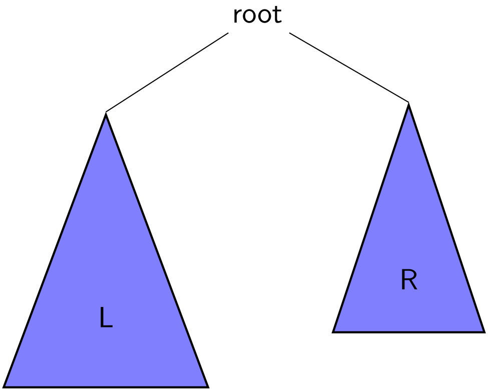
Running time

- All lines of the algorithm take constant time, except the last line, which contains the recursive call
- What is the size of the input?
- The size is the size of the subtree rooted at i, call it n.
- The size of the input in the recursive call is the size of the subtree rooted at *largest*.

$$T(n) = T(\text{size of the subtree}) + \Theta(1)$$

In the worst case the subtree rooted at *largest* is as big as possible.

Worst case



- Assume the number of nodes in $R = k$
- Number of nodes in $L = k + (k + 1)$
- $n = 1 + k + (2k + 1) = 3k + 2.$
- Size of L is approx $2/3 \cdot n$

Recurrence for MAX-HEAPIFY

The recurrence for MAX-HEAPIFY is:

$$T(n) \leq T(\frac{2n}{3}) + \Theta(1)$$

Solution

$a = 1, \quad b = 3/2, \quad f(n) = \Theta(1)$

- $n^{\log_{3/2} 1} = n^0 = 1$
- $f(n) = \Theta(1) = \Theta(n^{\log_{3/2} 1})$

Case 2 of Master Theorem:

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

If h is the height of i, we can write it also $O(h)$.

Building a max-heap

Important observation

In any heap-array, the elements at the indexes:

$$\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots, n$$

are leaves! Hence each of them is a heap of the height 0.

Procedure BUILD-MAX-HEAP(A)

$A.heap{-}size = A.length$
for $i = \lfloor \frac{A.length}{2} \rfloor$ *down to* 1 **do**
    MAX-HEAPIFY(A, i)

Running time:

- Each call to MAX-HEAPIFY costs $O(\lg n)$
- BUILD-MAX-HEAP makes $O(n)$ such calls.
- Hence $O(n \lg n)$ not tight!

Better running time analysis

Notice that:

- Heap has height $\lfloor \lg n \rfloor$
- At each height h there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes.
- Question: How many times MAX-HEAPIFY is called starting with nodes at height h (going up the tree)?
- Answer: $\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil$ times.

Hence the running time is:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) = O(n \sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{h}{2^{h+1}} \rceil)$$

Notice: $\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = \frac{1/2}{(1/2)^2} = \frac{4}{2} = 2$

Hence $O(n \sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{h}{2^{h+1}} \rceil) = O(n)$