Computer Science 331

Operations on Binary Heaps

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #25

Outline

Max-Heapify: Introduction

Recall that an array can be used to represent a binary heap.

Observation: An array can be used to represent any *binary tree* with the same shape as a heap — "heap order" is not used to define this representation.

The "Max-Heapify" algorithm, described next, is used to take an array representation of a binary tree that is "almost" a heap, and convert it into a heap storing the same multiset.

This is a useful "subroutine" for a variety of more interesting operations that will be described later.

Max-Heapify: Specification of Requirements

Pre-Condition:

- A is an array representing a binary tree (with the same shape as a heap).
- i is an integer; $0 \leq i < \text{heap-size}(A) \leq \text{length}(A)$.
- A satisfies *all* the properties of an array representation of a max-heap, *except* that $A[i]$ might be less than
  - $A[\text{left}(i)]$ (if $\text{left}(i) < \text{heap-size}(A)$), as well as
  - $A[\text{right}(i)]$ (if $\text{right}(i) < \text{heap-size}(A)$).
- In particular, if $i > 0$ then
  - if $\text{left}(i) < \text{heap-size}(A)$ then $A[\text{parent}(i)] \geq A[\text{left}(i)]$ and
  - if $\text{right}(i) < \text{heap-size}(A)$ then $A[\text{parent}(i)] \geq A[\text{right}(i)]$.

Max-Heapify: Specification of Requirements

Post-Condition:

- The elements stored in A have been reordered but otherwise unchanged.
- Furthermore, $A[j]$ is unchanged for every integer j such that heap-size$(A) \leq j <$ length(A).
- A represents a max-heap.

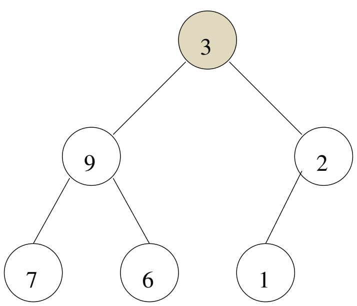
Max-Heapify: Pseudocode

Max-Heapify(A, i)
  {A array, $0 \leq i <$ heap-size(A)}
  $\ell =$ left(i); $r =$ right(i); $largest = i$
  if ($\ell <$ heap-size(A)) **and** ($A[\ell] > A[i]$) **then**
    $largest = \ell$
  end if
  if ($r <$ heap-size(A)) **and** ($A[r] > A[largest]$) **then**
    $largest = r$
  end if
  if $largest \neq i$ **then**
    Swap: $tmp = A[i]$; $A[i] = A[largest]$; $A[largest] = temp$
    Max-Heapify(A, $largest$)
  end if

Example

Suppose A represents the following binary tree and $i = 0$.

Note: The pre-condition for "Max-Heapify(A, i)" is satisfied.

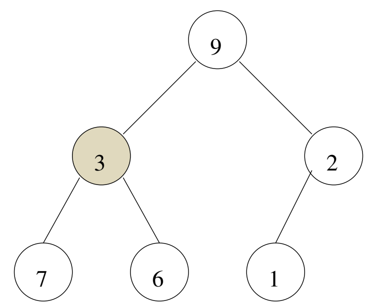


After the initial tests, $largest =$ left(i) $= 1$.

- Values are exchanged and procedure is called with $i = 1$.

Example ($i = 1$)

Note: Pre-condition for "Max-Heapify(A, i)" is satisfied before this procedure is called again.

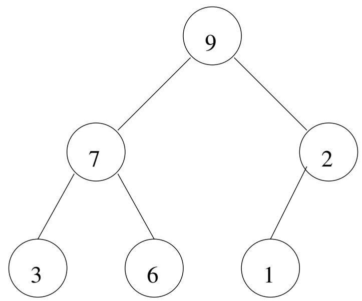


After the initial tests, $largest =$ left(i) $= 3$.

- Values are exchanged and procedure is called with $i = 3$.

Example ($i = 3$)

Note: Pre-condition for "Max-Heapify(A, i)" is satisfied before this procedure is called again.



The subtree with root at index 3 satisfies the max-heap order property. A now represents a max-heap.

Partial Correctness

Theorem 1

Suppose Max-Heapify is called with an array A and integer i such that the precondition for Max-Heapify is satisfied. Then either Max-Heapify does not terminate at all, or the following properties are satisfied on termination:

- *A stores the values it did before Max-Heap was called. However, the ordering of these values might have been changed.*
- *$A[j]$ has not been changed for any integer j such that $\textit{heap-size}(A) \leq j \leq \textit{length}(A)$.*
- *$\textit{heap-size}(A)$ has not been changed*
- *A represents a max-heap.*

Proof (induction on height(i))

Proof.

Base case (height$(i) = 0$):

-
-

Inductive case: assume that height$(i) = h$ and that **Max-Heapify** is partially correct for all sub-heaps of height $< h$

-
-
-
-

Thus, **Max-Heapify** is partially correct by induction. □

Termination and Efficiency

Theorem 2

*Suppose Max-Heapify is called with an input array A and an integer i such that the precondition of **Max-Heapify** is satisfied. Then Max-Heap terminates after performing $O(\textit{height}(i))$ operations.*

Let $T(h)$ be the number of steps used by **Max-Heapify**(A, i) in the worst case when height$(i) = h$.

$$T(h) =$$

The following lemma implies the theorem.

Termination and Efficiency (proof)

Lemma 3

For all $h \geq 0$, $T(h) \leq ch + c$ *where* $c = \max(c_0, c_1, c_2)$.

Proof (Induction on h).

Base case ($h = 0$): $T(0) = c_0 \leq c(0) + c = c$
Base case ($h = 1$): $T(1) = c_1 \leq c(1) + c = 2c$
Assume that the lemma holds for all $j < h$. We have

$$\begin{aligned} T(h) &= \max\left[T(h-1), T(h-2)\right] + c_2 \\ &\leq \max\left[c(h-1) + c, c(h-2) + c\right] + c2 \\ &< c(h-1) + c + c_2 = ch + c2 \leq ch + c \ . \end{aligned}$$

Thus, the result follows by induction. □

Procedure Build-Max-Heap

Objective: Reorganize the elements stored in an array A to produce a representation of a Max-Heap

Precondition:

- A is an array of size $n \geq 1$, containing values from some ordered type

Postcondition:

- A represents a heap of size n
- Entries of A are reordered but otherwise unchanged

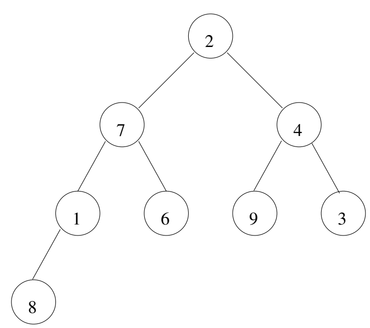
Pseudocode

Idea: Use Max-Heapify to impose max heap order on subtrees:

- start at last non-leaf node
- move up to the root

Build-Max-Heap(A)
  {Note length(A) = heap-size(A)}
  n = length(A)
  $i = \lfloor n/2 \rfloor - 1$
  while $i \geq 0$ **do**
    Max-Heapify(A, i)
    $i = i - 1$
  end while

Example

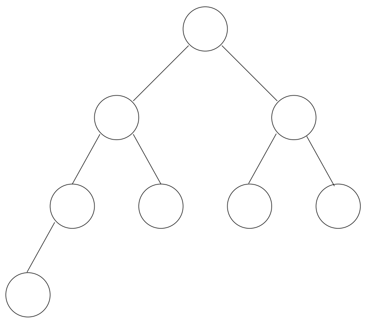


0	1	2	3	4	5	6	7
2	7	4	1	6	9	3	8

Example: $i = 3$

Max-Heapify(A, 3):

-

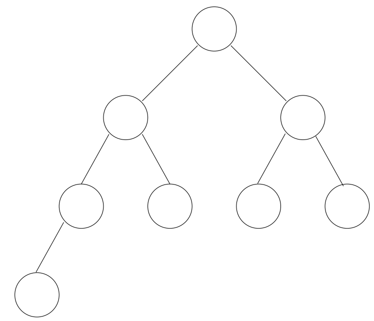


0	1	2	3	4	5	6	7

Example: $i = 2$

Max-Heapify(A, 2):

-

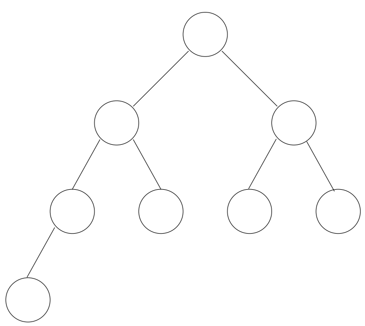


0	1	2	3	4	5	6	7

Example: $i = 1$

Max-Heapify(A, 1):

-

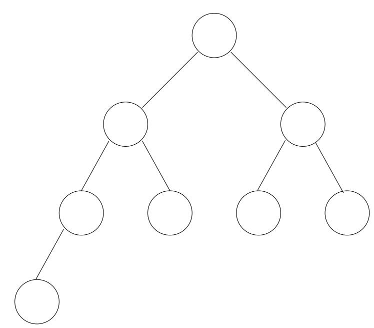


0	1	2	3	4	5	6	7

Example: $i = 0$

Max-Heapify(A, 0):

-
-



0	1	2	3	4	5	6	7

Partial Correctness

Loop Invariant: If the loop is executed at least k times then after the k^{th} execution of the loop body,

- $\text{length}(A) = \text{heap-size}(A) = n$.
- $i = \lfloor n/2 \rfloor - 1 - k$, so that $i \in \mathbb{Z}$, and $-1 \leq i \leq \lfloor n/2 \rfloor - 1$.
- for every integer j such that $i + 1 \leq j \leq n - 1$
  - if $\text{left}(j) < n$ then $A[j] \geq A[\text{left}(j)]$ and
  - if $\text{right}(j) < n$ then $A[j] \geq A[\text{right}(j)]$
- The entries of A have been reordered but are otherwise unchanged.

Partial Correctness: A Complication

Complication: The pre-condition we have used for “Max-Heapify” is not satisfied when it is called by “Build-Max-Heap.”

Solution: Notice that “Max-Heapify” also solves a *different* problem than the one we first discussed.

The proof that Max-Heapify solves the different (related) problem (that we need here) is a modification of the original proof of correctness.

Revised Requirements for Max-Heapify

Pre-Condition (for **Max-Heapify**(A, j)):

- A is an array representing a binary tree (with the same shape as a heap)
- i is an integer such that $-1 \leq i \leq n - 1$
- j is an integer such that $i + 1 \leq j < n = \text{heap-size}(A) = \text{length}(A)$
- for every integer k such that $i + 1 \leq k < n$ and such that $k \neq j$:
  - if $\text{left}(k) < n$ then $A[k] \geq A[\text{left}(k)]$, and
  - if $\text{right}(k) < n$ then $A[k] \geq A[\text{right}(k)]$.
- if $\text{parent}(j) \geq i + 1$ then
  - if $\text{left}(j) < n$ then $A[\text{parent}(j)] \geq A[\text{left}(j)]$, and
  - if $\text{right}(j) < n$ then $A[\text{parent}(j)] \geq A[\text{right}(j)]$.

Partial Correctness: Max-Heapify

Post-Condition:

- The elements stored in A have been reordered but otherwise unchanged.
- For *every* integer k such that $i + 1 \leq k < n$:
  - if $\text{left}(k) < n$ then $A[k] \geq A[\text{left}(k)]$, and
  - if $\text{right}(k) < n$ then $A[k] \geq A[\text{right}(k)]$.

Theorem 4

Suppose that the revised pre-conditions are satisfied when Max-Heapify is called with input array A and an integer input j. Then either Max-Heapify does not terminate or the postconditions are satisfied.

Method of Proof: Induction on $\text{height}(j)$

Termination and Efficiency: Max-Heapify

Theorem 5

Suppose that the revised pre-conditions are satisfied when Max-Heapify is called with input array A and an integer input j. Then Max-Heapify terminates and the number of steps used by this algorithm is in $O(\mathit{height}(j))$ in the worst case.

Method of Proof: This proof is virtually identical to the proof of termination and efficiency of "Max-Heapify" for the original pre-condition.

Partial Correctness: Build-Max-Heap

Exercises:

1. Modify the original proofs concerning the correctness and efficiency of "Max-Heapify" to establish the claims concerning the correctness and efficiency of "Max-Heapify" (with a different pre-condition) that are given above.
2. Prove the correctness of the loop invariant for "Build-Max-Heap" that is stated above.
3. Show that $i = -1$ when the loop for "Build-Max-Heap" terminates. Use this, with the loop invariant, to prove the partial correctness of this program.

Termination and Efficiency

Loop Variant: $f(n, i) = i + 1$

Cost of Loop Body for a Given i:

-

Number of iterations:

-
-

Worst-Case Cost of Build-Max-Heap:

-
-

Procedure Delete-Max

Objective: Remove the largest element from a heap and return its value.

Precondition:

- A is an array of size $n \geq 1$ that represents a nonempty Max-Heap

Postcondition:

- Largest entry in the heap has been returned as output
- A now represents a heap including all of the original elements except for the one that has been returned

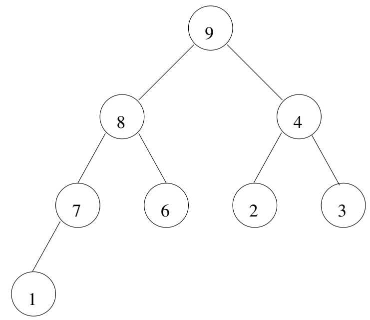
Exception: `EmptyHeapException`

Idea and Pseudocode

Idea: Copy the value from the node that must be deleted to the root, and use **Max-Heapify** to restore heap-order. Return the value that was initially at the root.

Delete-Max(A)
  if heap-size(A) > 1 **then**
    largest $= A[0]$; $A[0] = A[\text{heap-size}(A) - 1]$
    heap-size(A) $=$ heap-size(A) $- 1$; **Max-Heapify**(A, 0)
    return *largest*
  else if heap-size(A) $= 1$ **then**
    heap-size(A) $= 0$
    return $A[0]$
  else
    throw `EmptyHeapException`
  end if

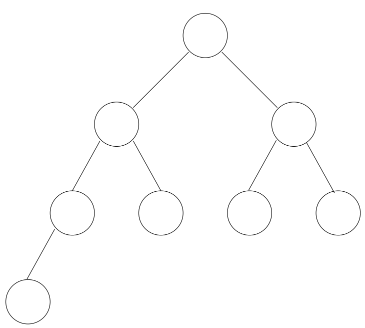
Example



0	1	2	3	4	5	6	7
9	8	4	7	6	2	3	1

heap-size(A) $= 8$

Example: Output and Resulting Heap



0	1	2	3	4	5	6	7

heap-size(A) $=$

Analysis

Partial Correctness:

- if heap-size(A) $= 0$, correct output is returned
- precondition implies that A is a Max-Heap, so $A[0]$ is the largest element
- two cases:

Termination and Efficiency: